# Getting Started

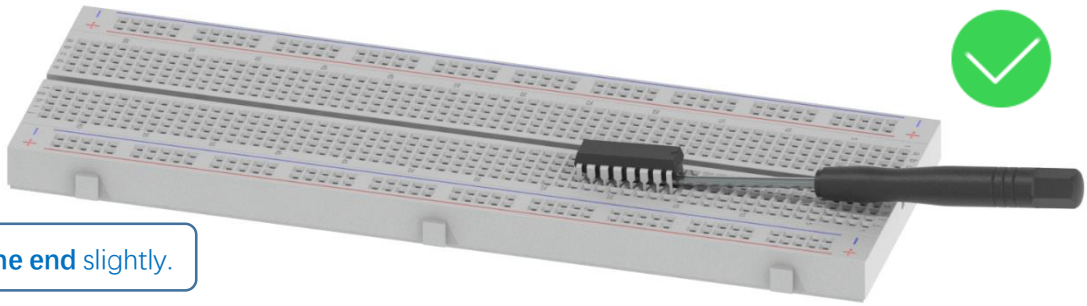
**Thank you for choosing Freenove products!**
After you download the ZIP file we provide. Unzip it and you will get a folder contains several files and folders. There are three PDF files:

- **Tutorial.pdf**
  It contains basic operations such as installing system for Raspberry Pi.
  **The code in this PDF is in C.**
- **Tutorial_GPIOZero.pdf**
  It contains basic operations such as installing system for Raspberry Pi.
  **The code in this PDF is in Python.**
- **Processing.pdf** in Freenove_Complete_Starter_Kit_for_Raspberry_Pi\Processing
  **The code in this PDF is in Java.**

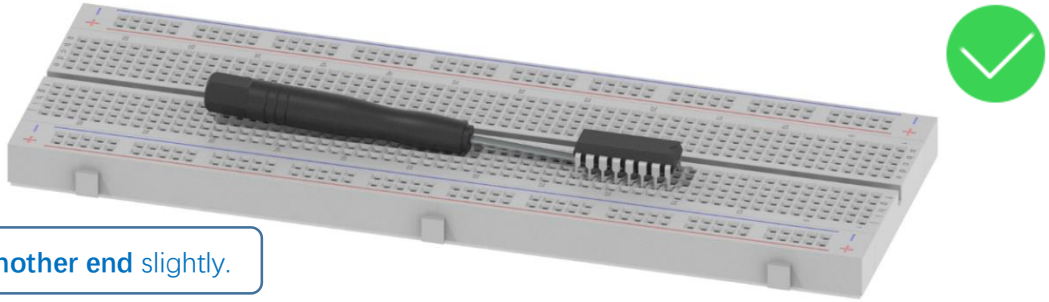
We recommend you to start with **Tutorial.pdf** or **Tutorial_GPIOZero.pdf** first.
If you want to start with Processing.pdf or skip some chapters of Tutorial.pdf, you need to finish necessary steps in **Chapter 7 AD/DA** of **Tutorial.pdf** first.

# Remove the Chips

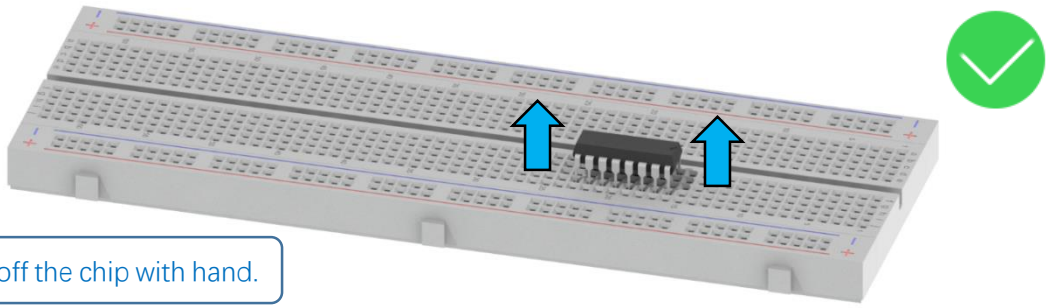
Some chips and modules are inserted into the breadboard to protect their pins.
You need to remove them from breadboard before use. (There is no need to remove GPIO Extension Board.)
Please find a tool (like a little screw driver) to handle them like below:



Step 1, lift **one end** slightly.

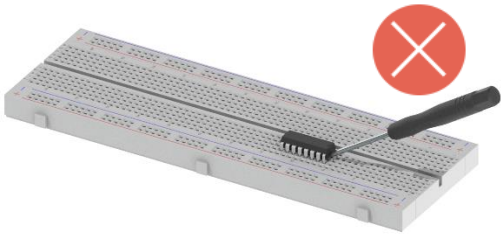


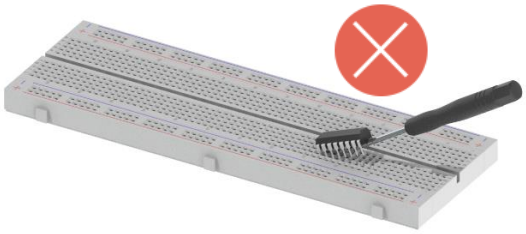
Step 2, lift **another end** slightly.



Step 3, take off the chip with hand.

**Avoid lifting one end with big angle directly.**





## Get Support and Offer Input

Freenove provides free and responsive product and technical support, including but not limited to:

- Product quality issues
- Product use and build issues
- Questions regarding the technology employed in our products for learning and education
- Your input and opinions are always welcome
- We also encourage your ideas and suggestions for new products and product improvements

For any of the above, you may send us an email to:

**support@freenove.com**

## Safety and Precautions

Please follow the following safety precautions when using or storing this product:

- Keep this product out of the reach of children under 6 years old.
- This product should be used only when there is adult supervision present as young children lack necessary judgment regarding safety and the consequences of product misuse.
- This product contains small parts and parts, which are sharp. This product contains electrically conductive parts. Use caution with electrically conductive parts near or around power supplies, batteries and powered (live) circuits.
- When the product is turned ON, activated or tested, some parts will move or rotate. To avoid injuries to hands and fingers, keep them away from any moving parts!
- It is possible that an improperly connected or shorted circuit may cause overheating. Should this happen, immediately disconnect the power supply or remove the batteries and do not touch anything until it

cools down! When everything is safe and cool, review the product tutorial to identify the cause.
- Only operate the product in accordance with the instructions and guidelines of this tutorial, otherwise parts may be damaged or you could be injured.
- Store the product in a cool dry place and avoid exposing the product to direct sunlight.
- After use, always turn the power OFF and remove or unplug the batteries before storing.

## About Freenove

Freenove provides open source electronic products and services worldwide.

Freenove is committed to assist customers in their education of robotics, programming and electronic circuits so that they may transform their creative ideas into prototypes and new and innovative products. To this end, our services include but are not limited to:

- Educational and Entertaining Project Kits for Robots, Smart Cars and Drones
- Educational Kits to Learn Robotic Software Systems for Arduino, Raspberry Pi and micro: bit
- Electronic Component Assortments, Electronic Modules and Specialized Tools
- **Product Development and Customization Services**

You can find more about Freenove and get our latest news and updates through our website:

http://www.freenove.com

## Copyright

# Contents

# Preface

Raspberry Pi is a low cost, **credit card sized computer** that plugs into a computer monitor or TV, and uses a standard keyboard and mouse. It is an incredibly capable little device that enables people of all ages to explore computing, and to learn how to program in a variety of computer languages like Scratch and Python. It is capable of doing everything you would expect from a desktop computer, such as browsing the internet, playing high-definition video content, creating spreadsheets, performing word-processing, and playing video games. For more information, you can refer to Raspberry Pi official website. For clarification, this tutorial will also reference Raspberry Pi as RPi, RPI and RasPi.

In this tutorial, most chapters consist of **Components List**, **Component Knowledge**, **Circuit**, and **Code**. We provide C code for each project in this tutorial. After completing this tutorial, you can learn Java by reading Processing.pdf.

This kit does not contain **Raspberry and its accessories**. You can also use the components and modules in this kit to create projects of your own design.

Additionally, if you encounter any issues or have questions about this tutorial or the contents of kit, you can always contact us for free technical support at:

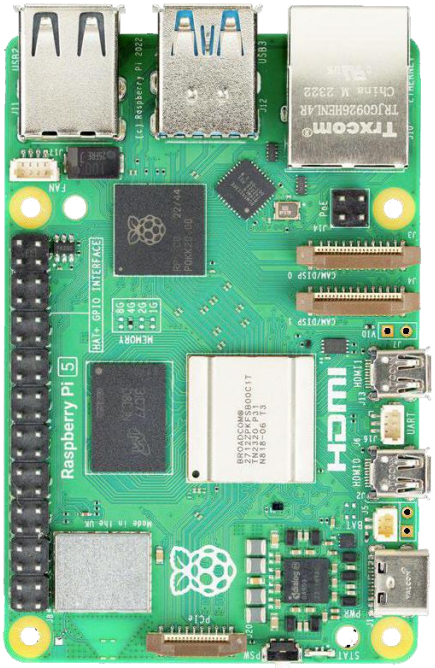
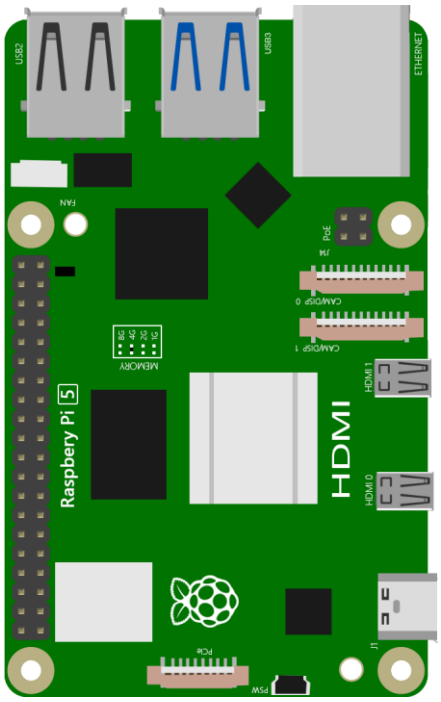
**support@freenove.com**

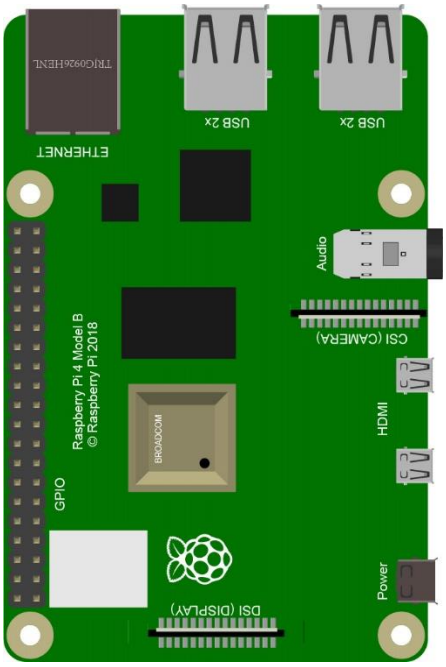
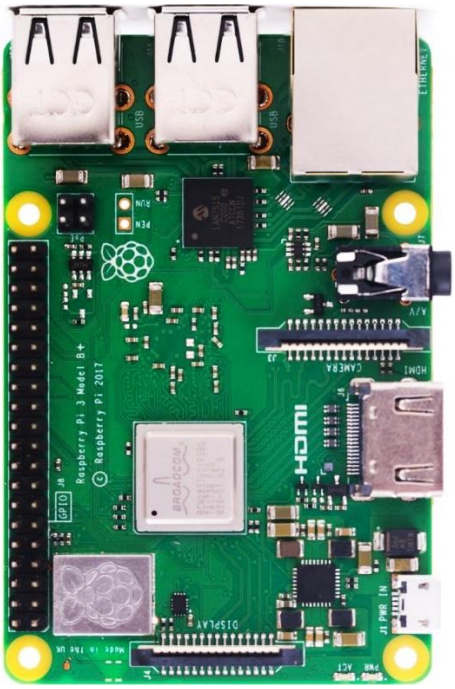
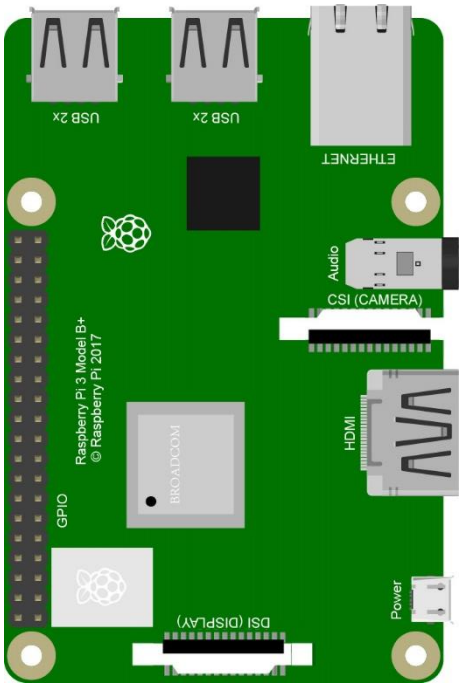
# Raspberry Pi

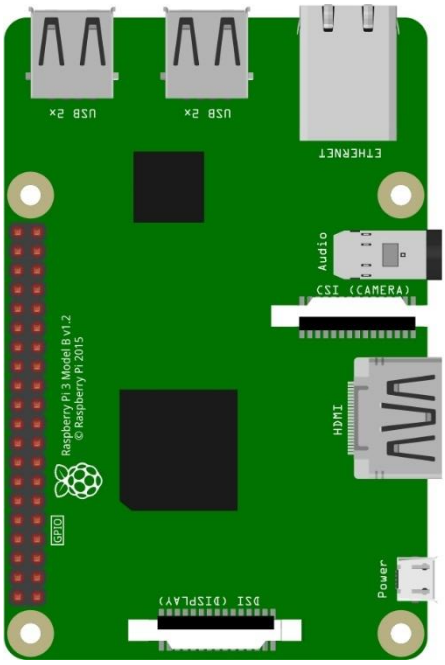
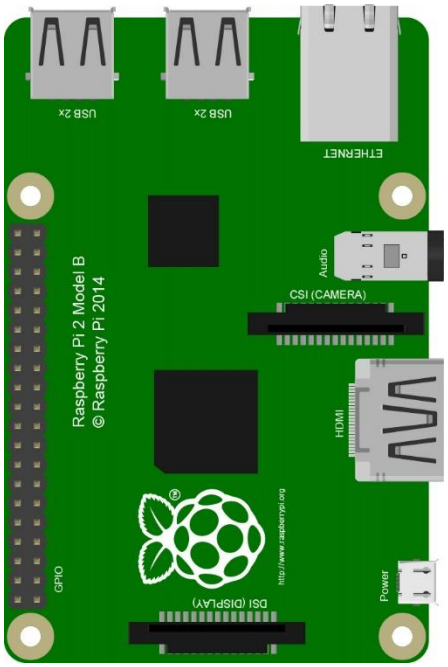
So far, at this writing, Raspberry Pi has advanced to its fifth generation product offering. Version changes are accompanied by increases in upgrades in hardware and capabilities.

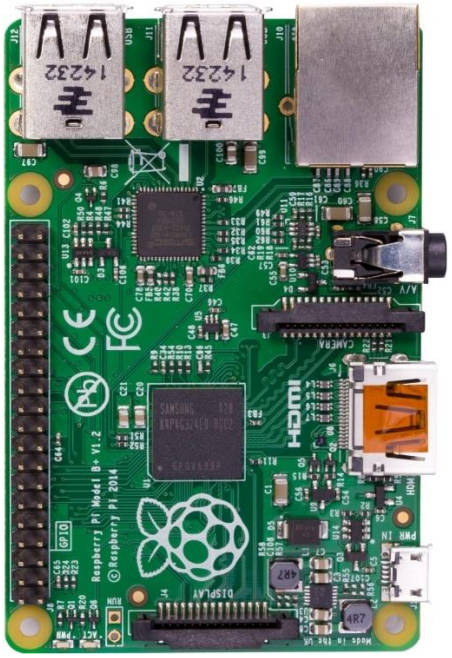
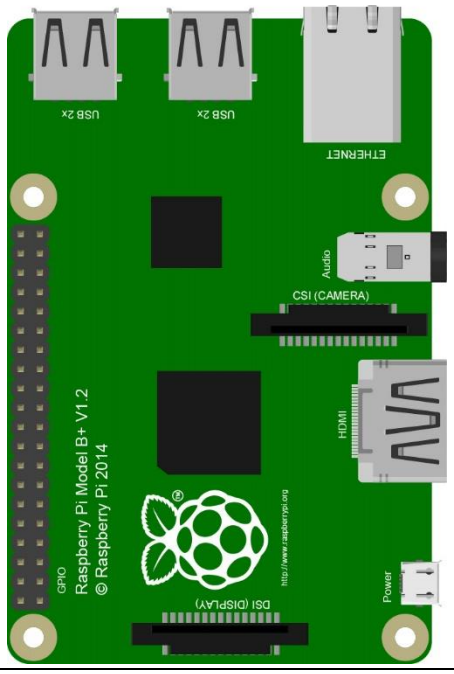
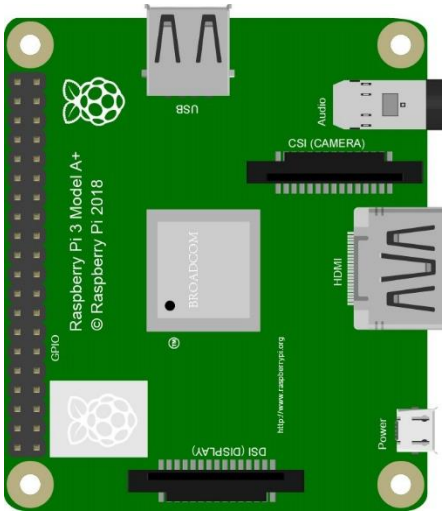
The A type and B type versions of the first generation products have been discontinued due to various reasons. What is most important is that other popular and currently available versions are consistent in the order and number of pins and their assigned designation of function, making compatibility of peripheral devices greatly enhanced between versions.

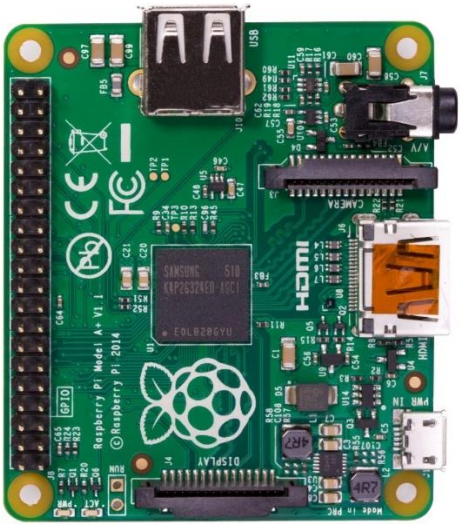
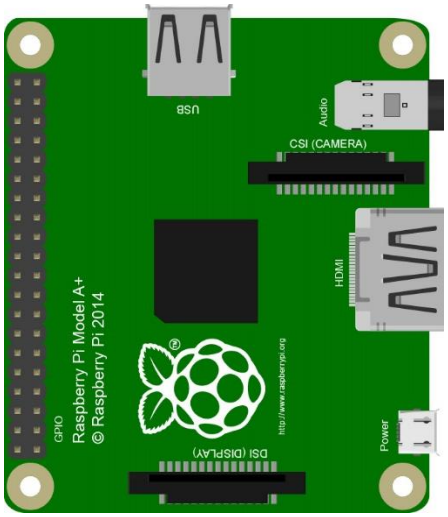
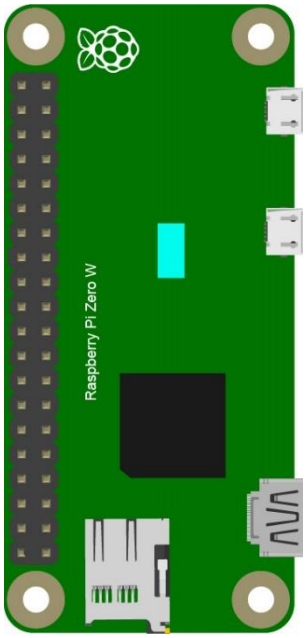
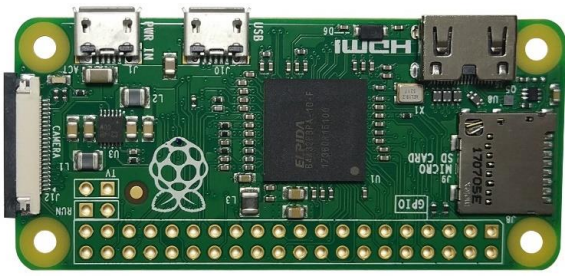
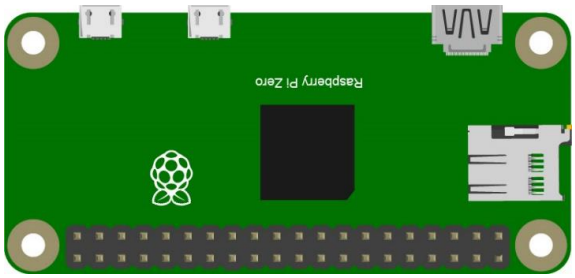
Below are the raspberry pi pictures and model pictures supported by this product. They have 40 pins.

| Practicality picture of Raspberry Pi 5: | Model diagram of Raspberry Pi 5: |
| --- | --- |
|  |  |

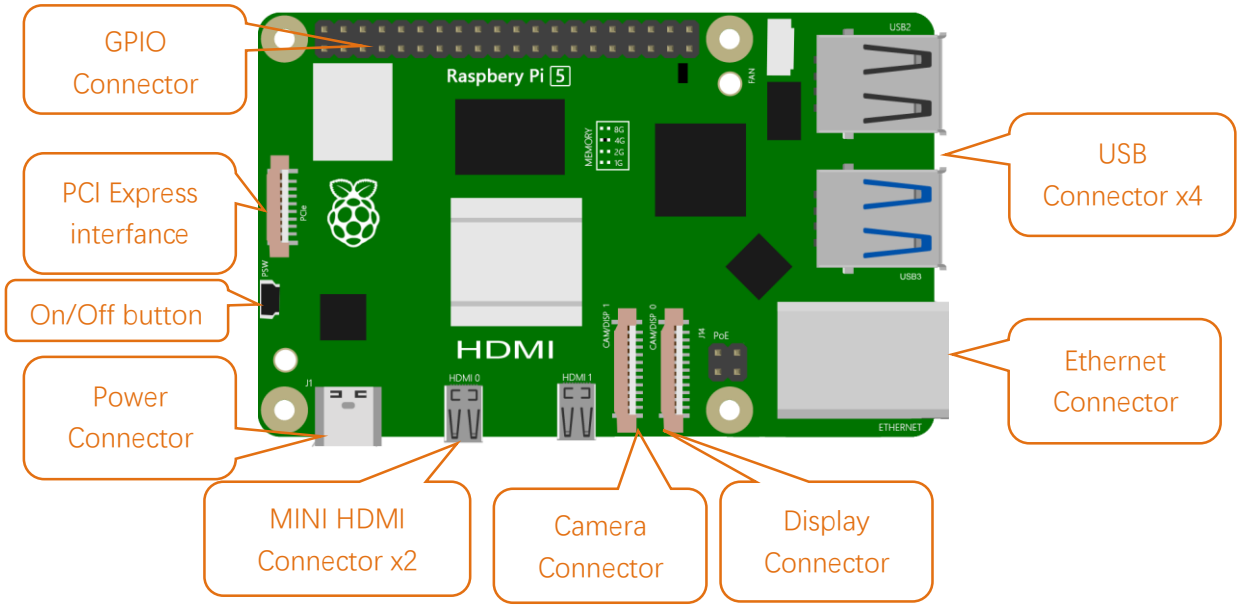
| | |
|---|---|
| Actual image of Raspberry Pi 4 Model B:  | CAD image of Raspberry Pi 4 Model B:   |
| Actual image of Raspberry Pi 3 Model B+:  | CAD image of Raspberry Pi 3 Model B+:   |

| | |
|---|---|
| Actual image of Raspberry Pi 3 Model B:<br>  | CAD image of Raspberry Pi 3 Model B:<br>  |
| Actual image of Raspberry Pi 2 Model B:<br>  | CAD image of Raspberry Pi 2 Model B:<br>  |

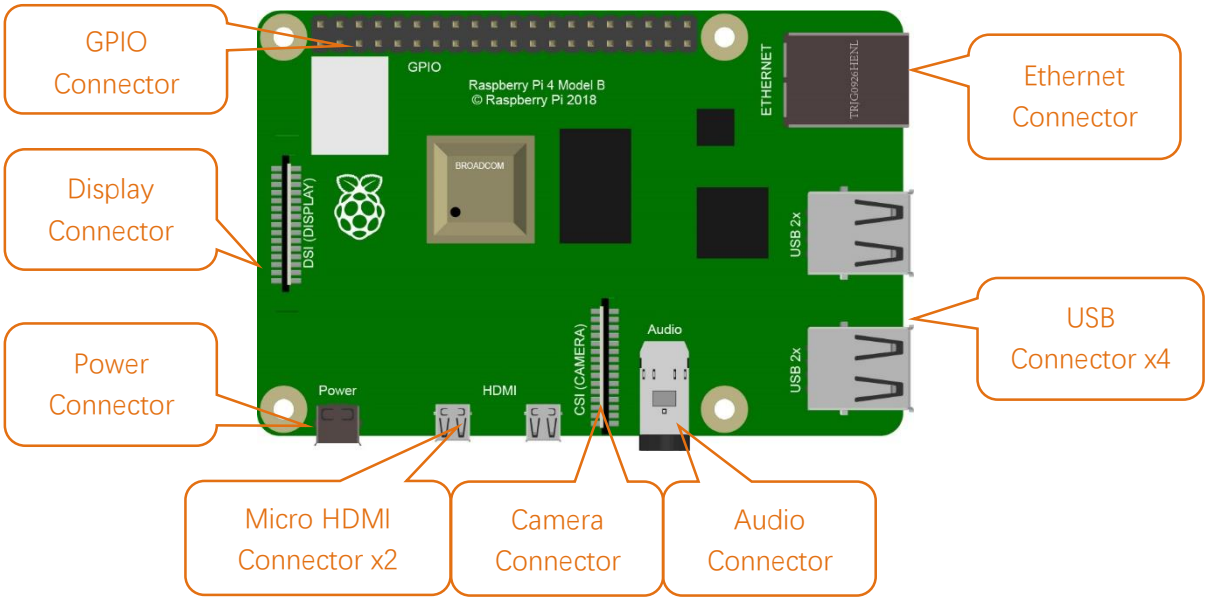
| Actual image of Raspberry Pi 1 Model B+: | CAD image of Raspberry Pi 1 Model B+: |
| --- | --- |
|  |  |
| Actual image of Raspberry Pi 3 Model A+: | CAD image of Raspberry Pi 3 Model A+: |
|  |  |

| | |
|---|---|
| Actual image of Raspberry Pi 1 Model A+: | CAD image of Raspberry Pi 1 Model A+: |
|  |  |
| Actual image of Raspberry Pi Zero W: | CAD image of Raspberry Pi Zero W: |
|  |  |
| Actual image of Raspberry Pi Zero: | CAD image of Raspberry Pi Zero: |
|  |  |

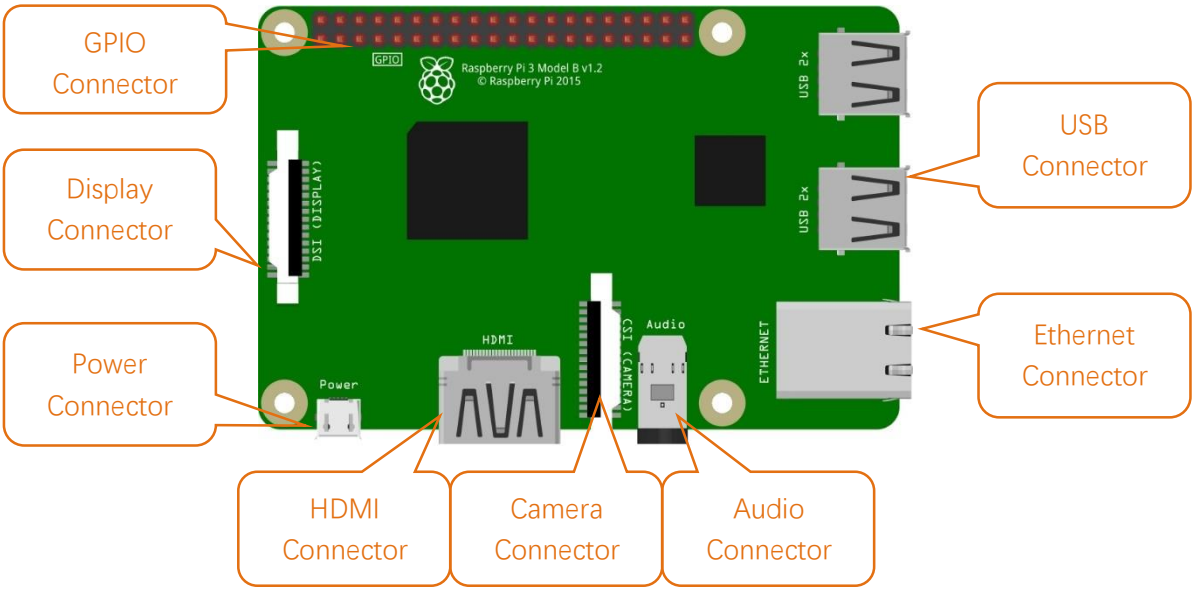
Below are the raspberry pi pictures and model pictures supported by this product. They have 40 pins.
Hardware interface diagram of RPi 5 is shown below:



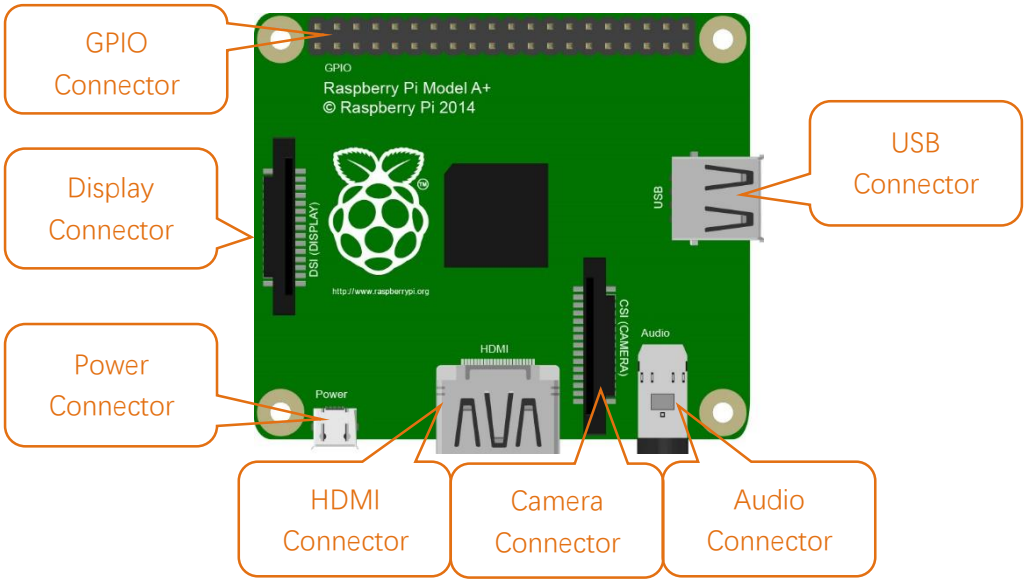

Hardware interface diagram of RPi 4B is shown below:



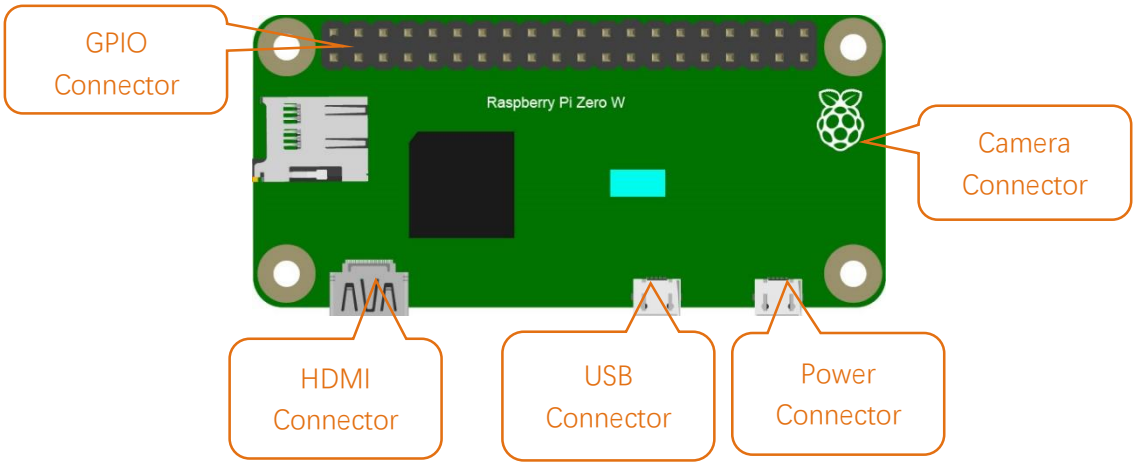
Hardware interface diagram of RPi 3B+/3B/2B/1B+:




Hardware interface diagram of RPi 3A+/A+:




Hardware interface diagram of RPi Zero/Zero W:

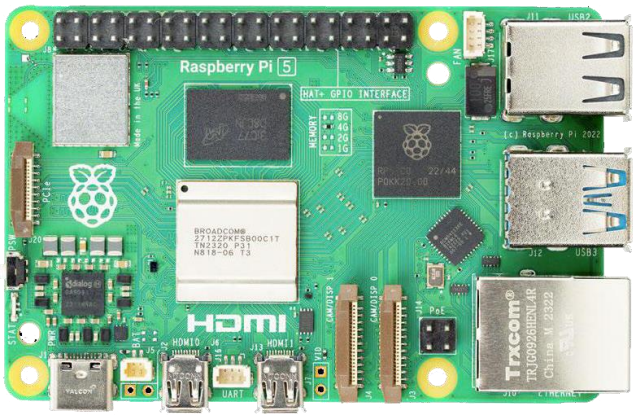
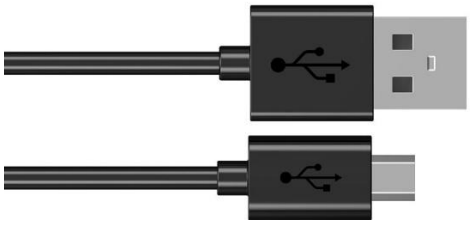


# Installing an Operating System

The first step is to install an operating system on your RPi so that it can be programmed and function. If you have installed a system in your RPi, you can start from Chapter 0 Preparation.

## Component List

### Required Components

| | |
|---|---|
| Any Raspberry Pi with 40 GPIO  | 5V/3A Power Adapter. Note: Different versions of Raspberry Pi have different power requirements (please check the power requirements for yours on the chart in the following page.)  |
| Micro or Type-C USB Cable x1  | Micro SD Card (TF Card) x1, Card Reader x1  |

Power requirements of various versions of Raspberry Pi are shown in following table:

| Product | Recommended PSU current capacity | Maximum total USB peripheral current draw | Typical bare-board active current consumption |
|---|---|---|---|
| Raspberry Pi 1 Model A | 700mA | 500mA | 200mA |
| Raspberry Pi 1 Model B | 1.2A | 500mA | 500mA |
| Raspberry Pi 1 Model A+ | 700mA | 500mA | 180mA |
| Raspberry Pi 1 Model B+ | 1.8A | 1.2A | 330mA |
| Raspberry Pi 2 Model B | 1.8A | 1.2A | 350mA |
| Raspberry Pi 3 Model B | 2.5A | 1.2A | 400mA |
| Raspberry Pi 3 Model A+ | 2.5A | Limited by PSU, board, and connector ratings only. | 350mA |
| Raspberry Pi 3 Model B+ | 2.5A | 1.2A | 500mA |
| Raspberry Pi 4 Model B | 3.0A | 1.2A | 600mA |
| Raspberry Pi 5 | 5.0A | 1.6A (600mA if using a 3A power supply) | 800mA |
| Raspberry Pi 400 | 3.0A | 1.2A | 800mA |
| Raspberry Pi Zero | 1.2A | Limited by PSU, board, and connector ratings only | 100mA |
| Raspberry Pi Zero W | 1.2A | Limited by PSU, board, and connector ratings only. | 150mA |
| Raspberry Pi Zero 2 W | 2A | Limited by PSU, board, and connector ratings only. | 350mA |

For more details, please refer to https://www.raspberrypi.org/help/faqs/#powerReqs

In addition, RPi also needs an Ethernet network cable used to connect it to a WAN (Wide Area Network).

The Raspberry Pi 5 provides 1.6A of power to downstream USB peripherals when connected to a power supply capable of 5A at +5V (25W). When connected to any other compatible power supply, the Raspberry Pi 5 restricts downstream USB devices to 600mA of power.

# Optional Components

Under normal circumstances, there are two ways to login to Raspberry Pi: 1) Using a stand-alone monitor. 2) Using a remote desktop or laptop computer monitor "sharing" the PC monitor with your RPi.

## Required Accessories for Monitor

If you choose to use an independent monitor, mouse and keyboard, you also need the following accessories:
1. A display with a HDMI interface
2. A Mouse and a Keyboard with an USB interface

As to Pi Zero and Pi Zero W, you also need the following accessories:
1. A Mini-HDMI to HDMI Adapter and Cable.
2. A Micro-USB to USB-A Adapter and Cable (Micro USB OTG Cable).
3. A USB HUB.
4. USB to Ethernet Interface or USB Wi-Fi receiver.

For different Raspberry Pi Modules, the optional items may vary slightly but they all aim to convert the interfaces to Raspberry Pi standards.

<table>
<tr><th></th><th>Pi Zero</th><th>Pi A+</th><th>Pi Zero W</th><th>Pi 3A+</th><th>Pi B+/2B</th><th>Pi 3B/3B+</th><th>Pi 4B</th><th>Pi 5</th></tr>
<tr><td>Monitor</td><td colspan="8">Yes (All)</td></tr>
<tr><td>Mouse</td><td colspan="8">Yes (All)</td></tr>
<tr><td>Keyboard</td><td colspan="8">Yes (All)</td></tr>
<tr><td>Micro-HDMI to HDMI Adapter & Cable</td><td>Yes</td><td>No</td><td>Yes</td><td>No</td><td>No</td><td>No</td><td>No</td><td>No</td></tr>
<tr><td>Micro-HDMI to HDMI Adapter & Cable</td><td colspan="6">No</td><td colspan="2">Yes</td></tr>
<tr><td>Micro-USB to USB-A Adapter & Cable (Micro USB OTG Cable)</td><td>Yes</td><td>No</td><td>Yes</td><td colspan="5">No</td></tr>
<tr><td>USB HUB</td><td>Yes</td><td>Yes</td><td>Yes</td><td>Yes</td><td>No</td><td>No</td><td>No</td><td>No</td></tr>
<tr><td>USB to Ethernet Interface</td><td rowspan="2" colspan="2">select one from two or select two from two</td><td colspan="2">optional</td><td>Internal Integration</td><td rowspan="2" colspan="3">Internal Integration</td></tr>
<tr><td>USB Wi-Fi Receiver</td><td colspan="2">Internal Integration</td><td>optional</td></tr>
</table>

## Required Accessories for Remote Desktop

If you do not have an independent monitor, or if you want to use a remote desktop, you first need to login to Raspberry Pi through SSH, and then open the VNC or RDP service. This requires the following accessories.

| | Pi Zero | Pi Zero W | Pi A+ | Pi 3A+ Pi B+/2B Pi 3B/3B+/4B/5 |
|---|---|---|---|---|
| **Micro-USB to USB-A Adapter & Cable (Micro USB OTG Cable)** | Yes | Yes | No | NO |
| **USB to Ethernet interface** | Yes | Yes | Yes | |

# Raspberry Pi OS

**Without Screen - Use Raspberry Pi - under Windows PC:** https://youtu.be/YND0RUuP-to

**With Screen - Use Raspberry Pi - under Windows PC:** https://youtu.be/HEywFsFrj3I

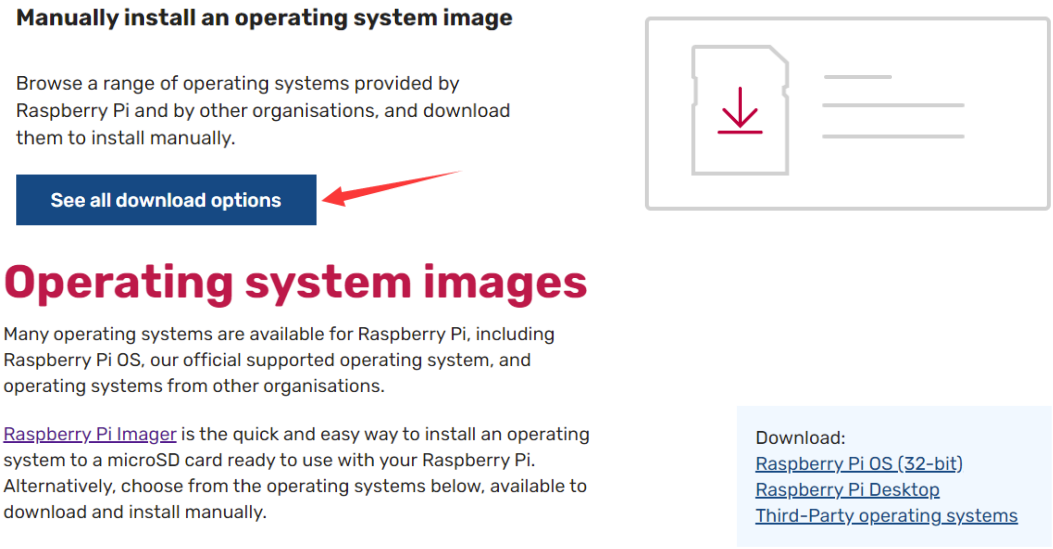
## Automatically Method

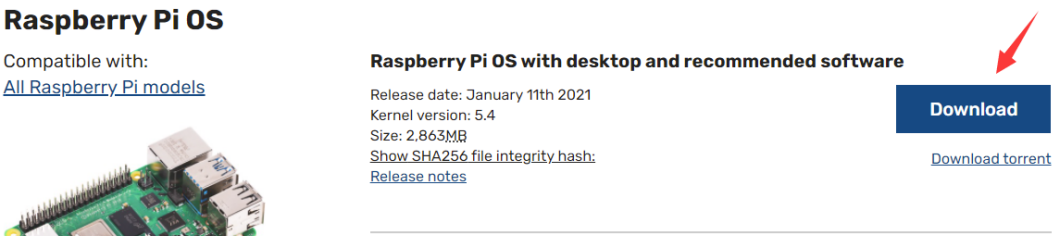
You can follow the official method to install the system for raspberry pi via visiting link below:
https://projects.raspberrypi.org/en/projects/raspberry-pi-setting-up/2
In this way, the system will be downloaded **automatically** via the application.

## Manually Method

After installing the Imager Tool in the **link above**. You can **also** download the system **manually** first.

Visit https://www.raspberrypi.org/downloads/



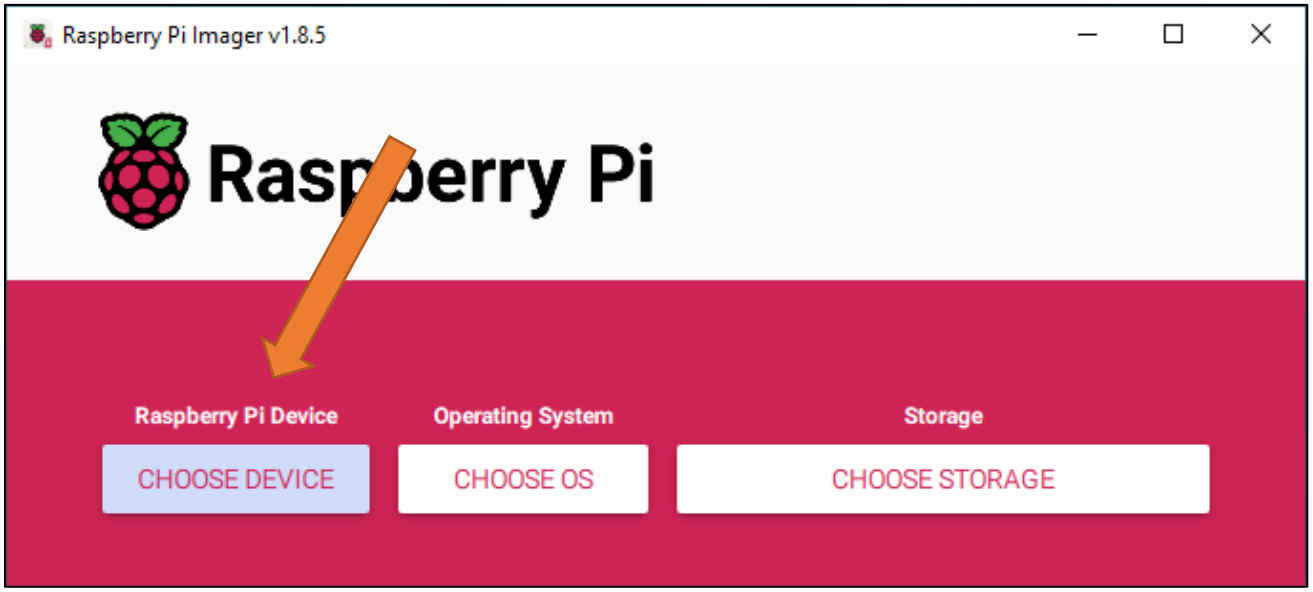




And then the zip file is downloaded.

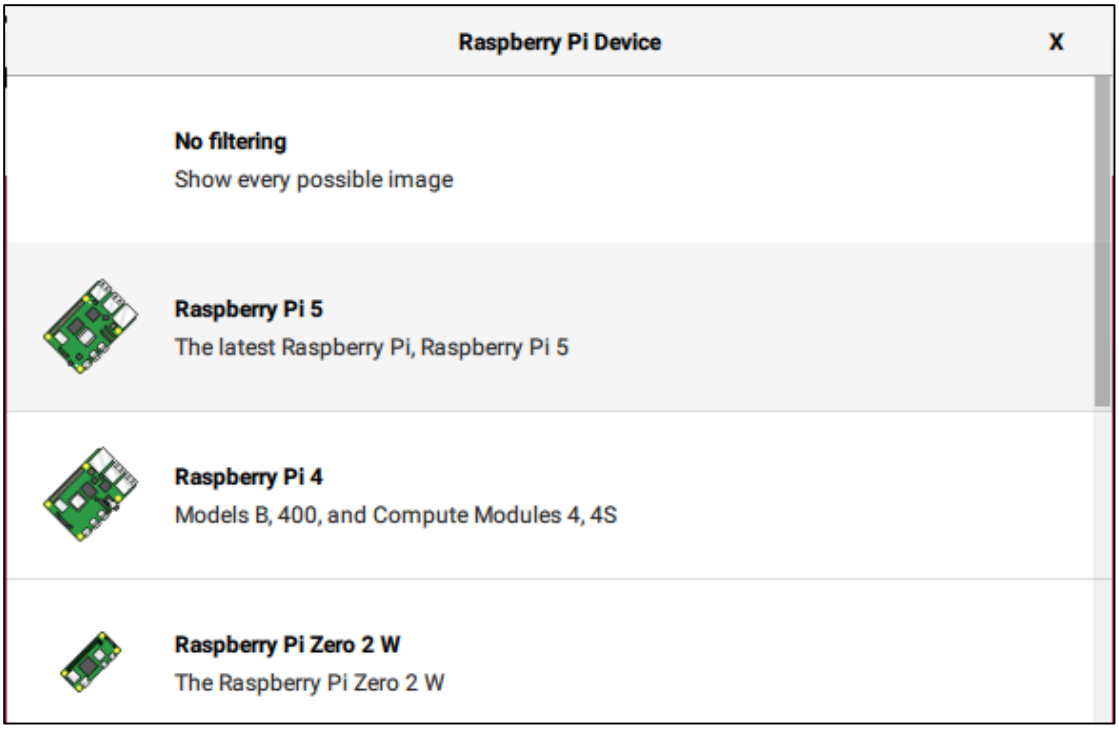
Write System to Micro SD Card

First, put your Micro **SD card** into card reader and connect it to USB port of PC.

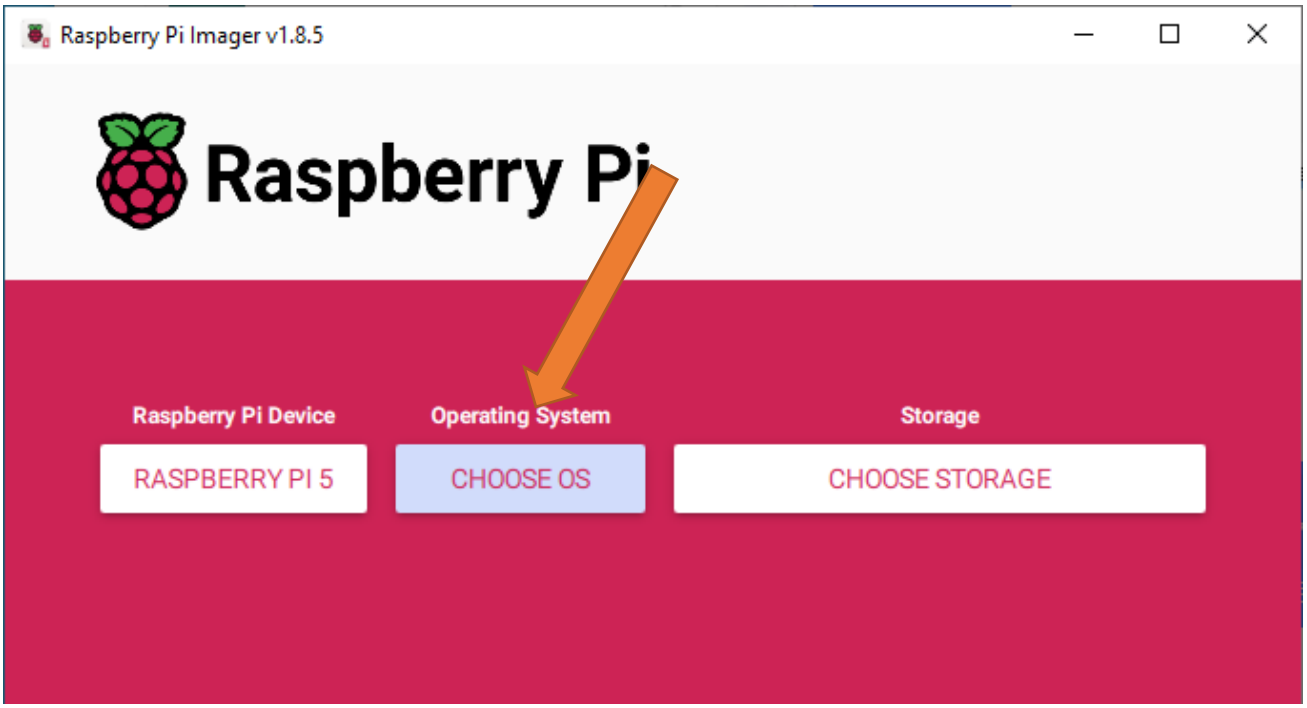

Then open imager toll. Clicked Choose Device.



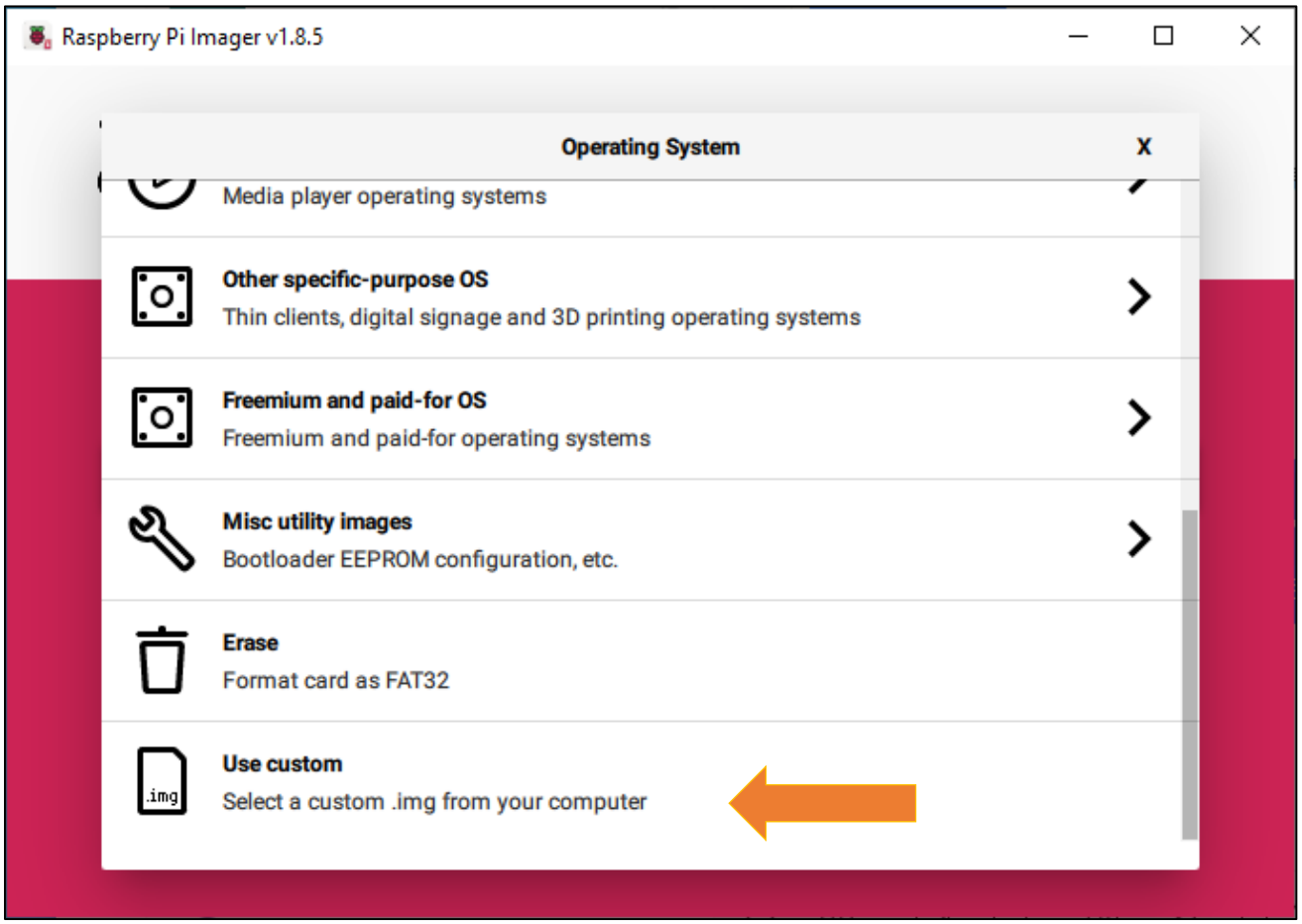

Select a Raspberry PI Device based on your Raspberry PI version. It will help us filter out the right version of the system for the Raspberry PI.



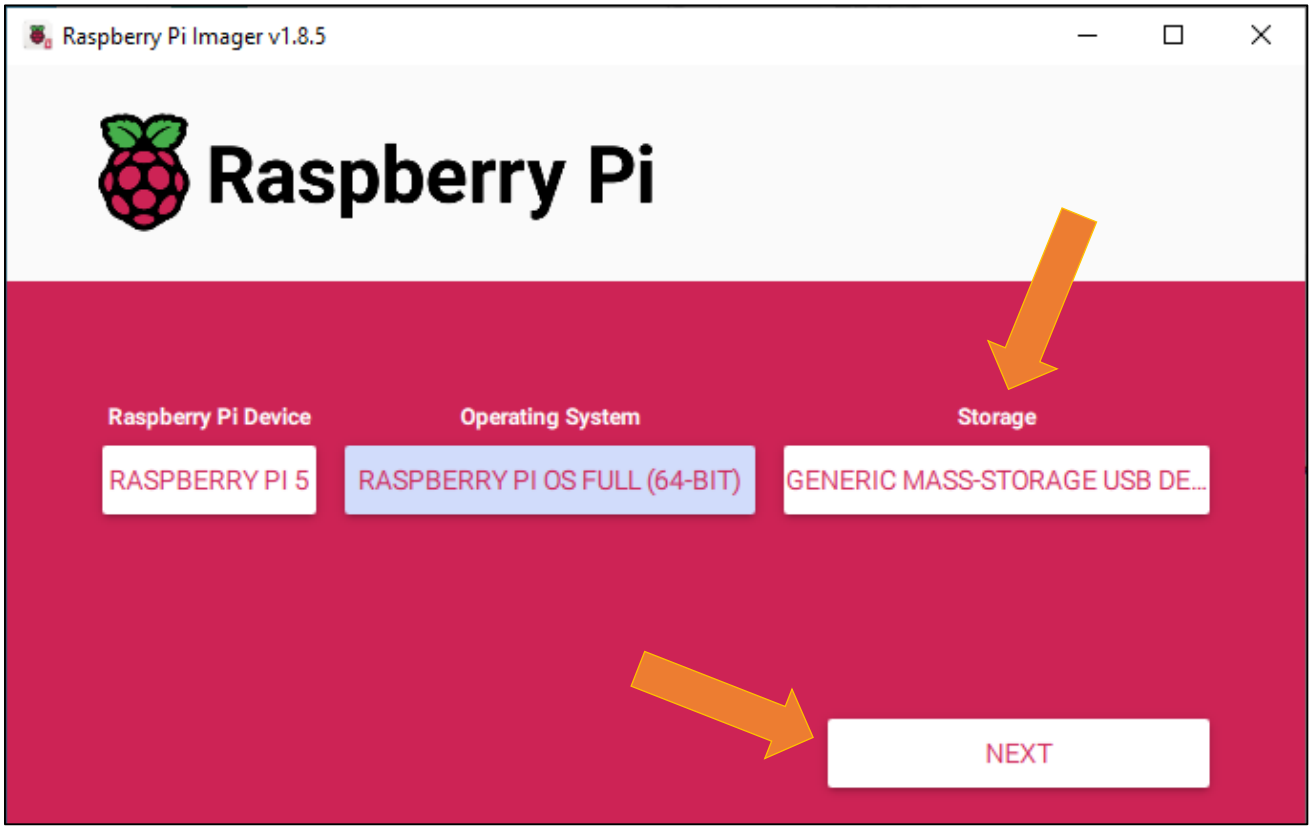
Clicked Operating System.



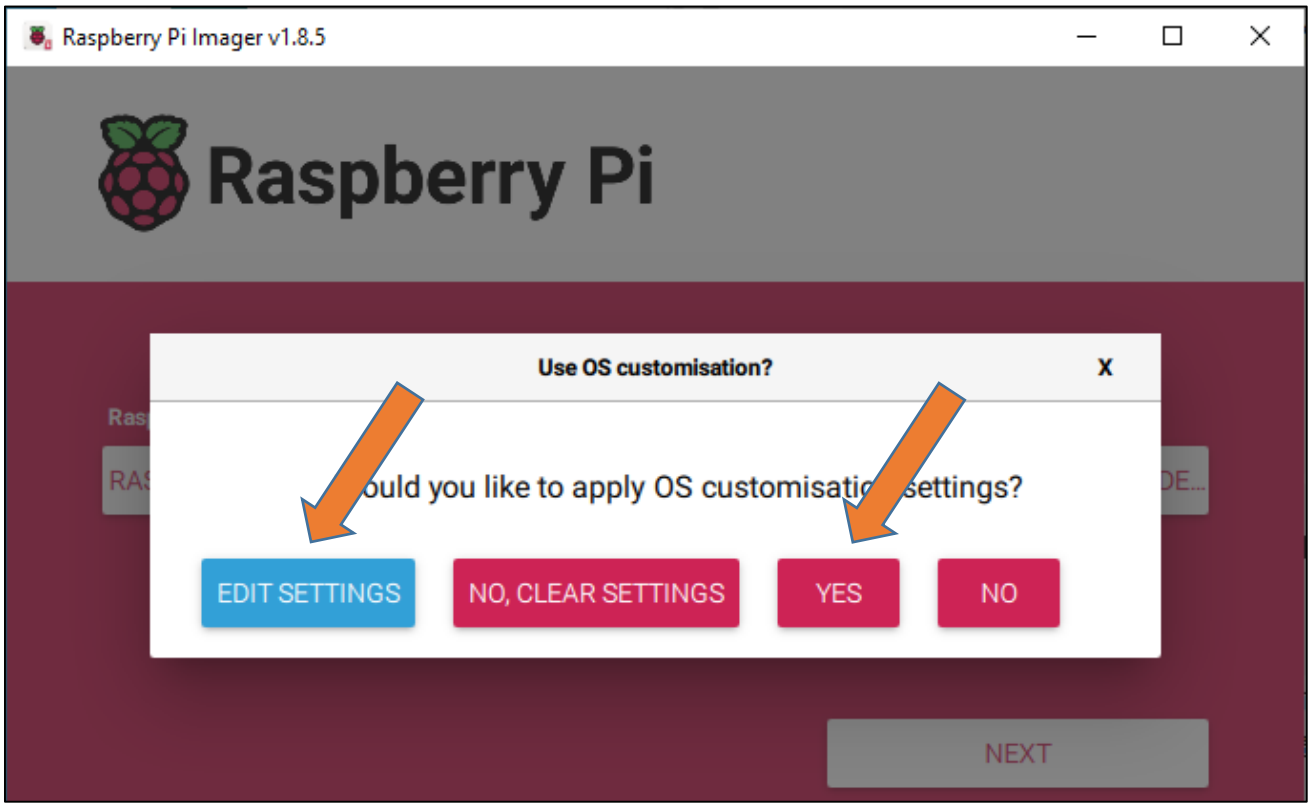

Choose system that you just downloaded in Use custom.



Choose the SD card. Then click "Next".

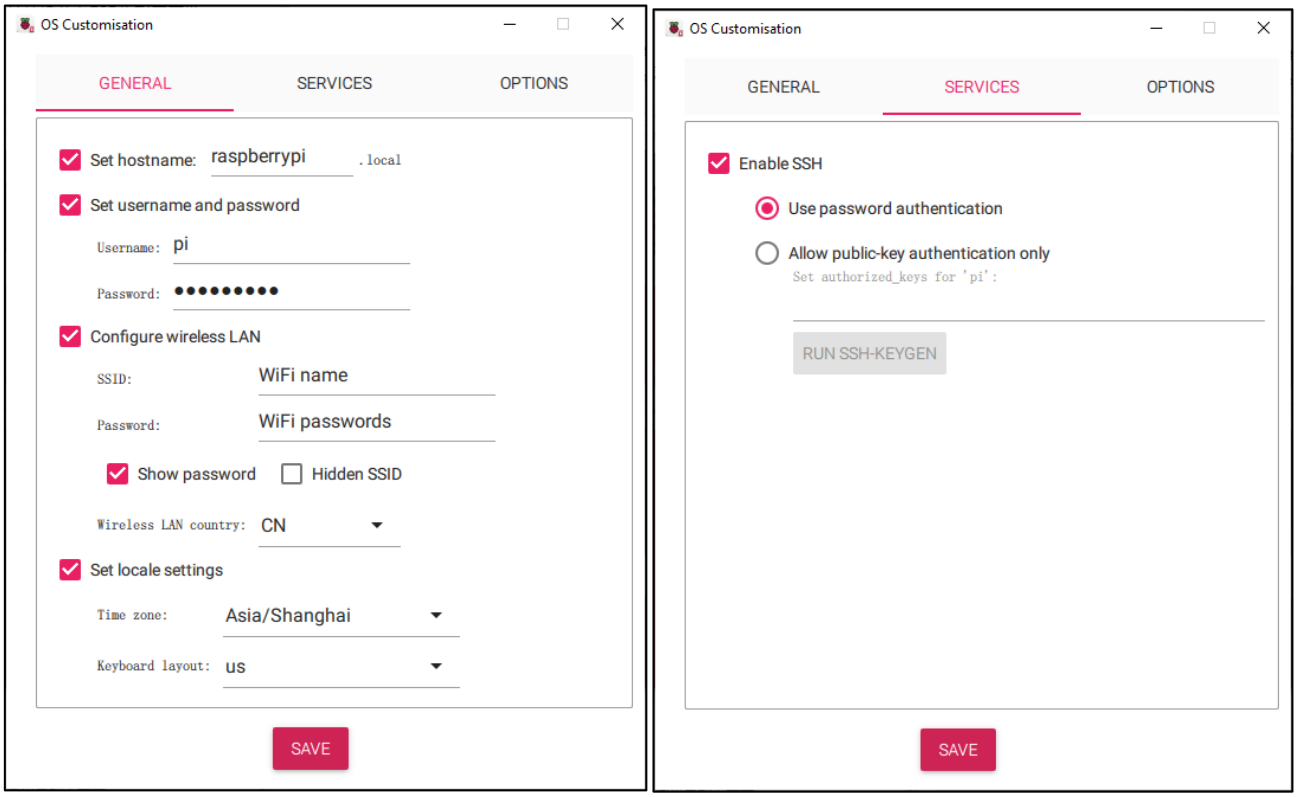



You can configure the Raspberry PI according to your needs.

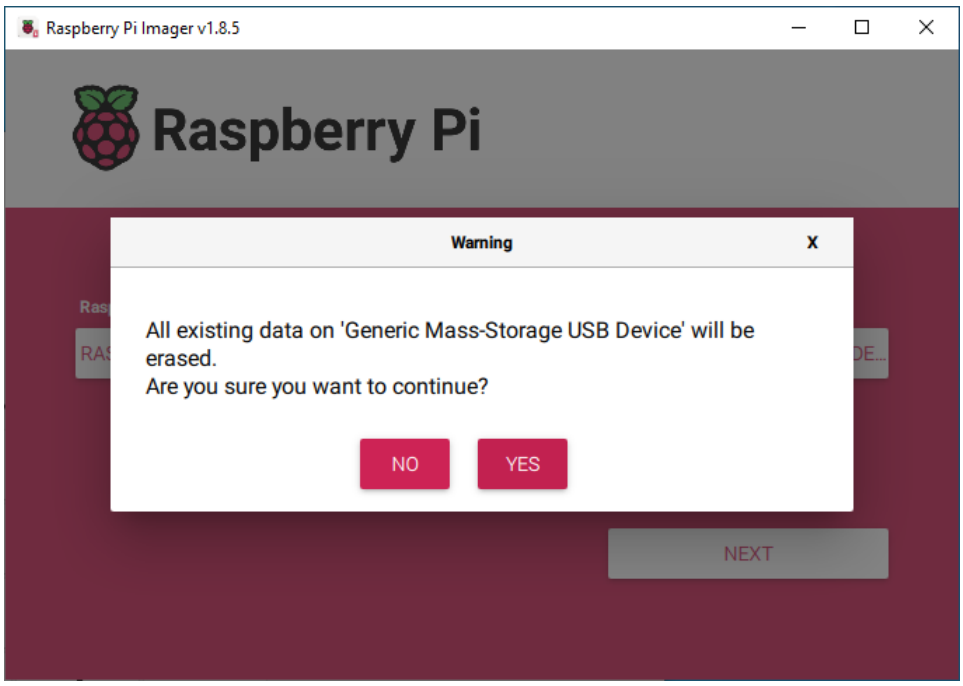


# Enable ssh and configure WiFi

**On the GENERAL screen, configure your information based on your actual situation.**
**Enable SSH on the SERVICES page.**

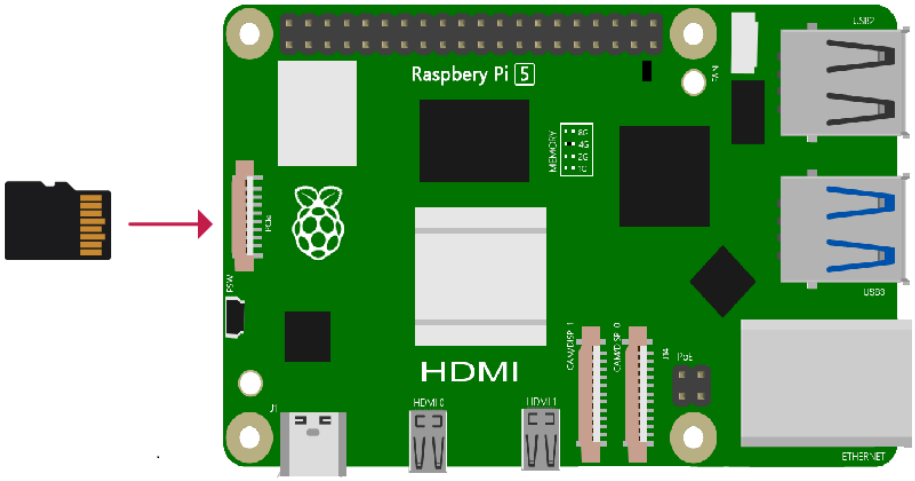



Click Save, in the new screen, click Yes, wait for SD to brush into the Raspberry system.



## Insert SD card

Then remove SD card from card reader and insert it into Raspberry Pi.



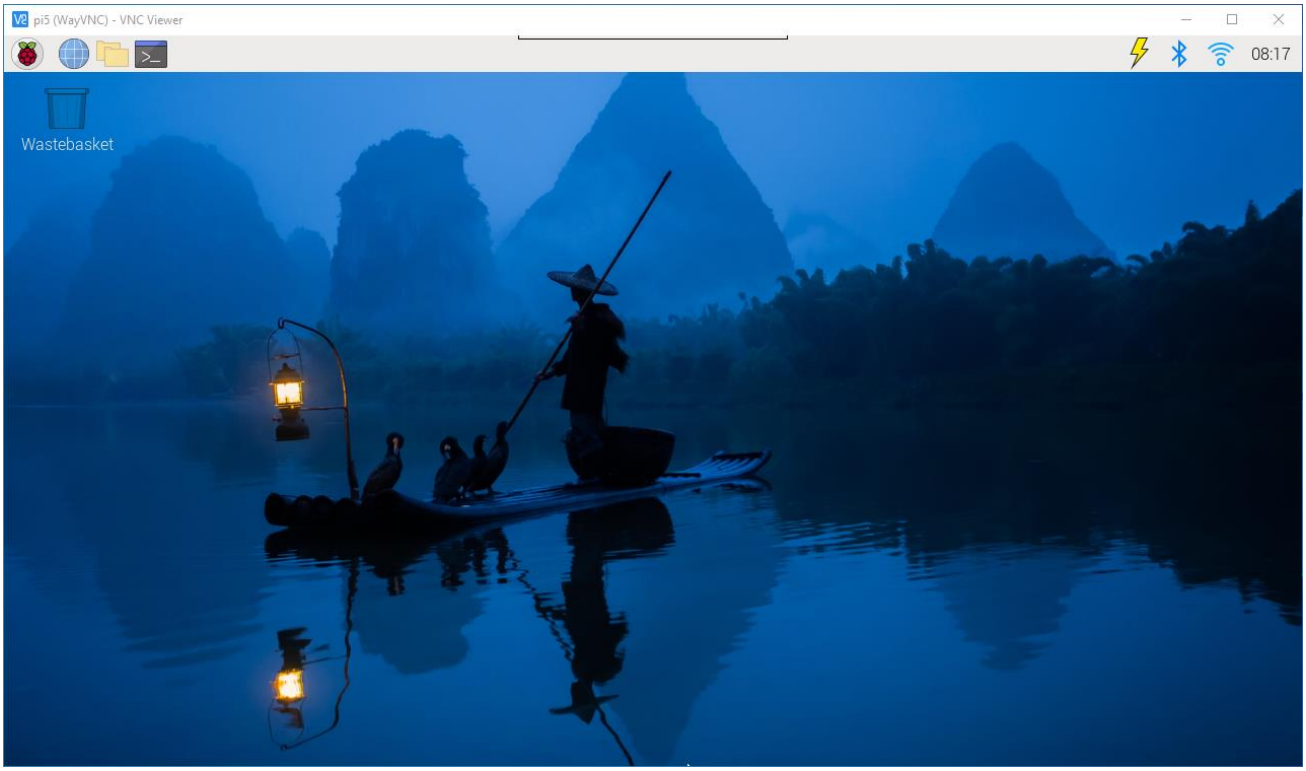

Connect to the power supply and wait for the Raspberry PI to turn on.

# Getting Started with Raspberry Pi

## Monitor desktop

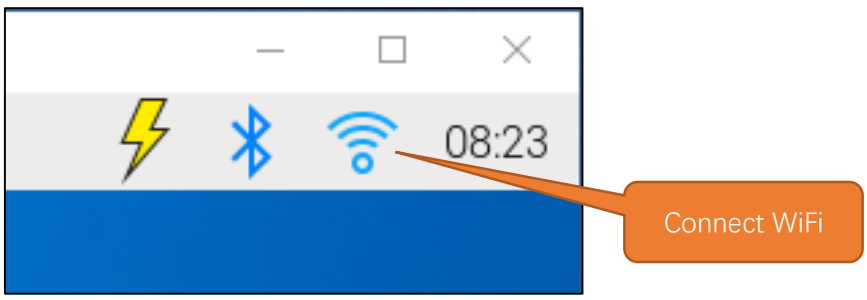
If you do not have a spare monitor, please skip to next section Remote desktop & VNC. If you have a spare monitor, please follow the steps in this section.

After the system is written successfully, take out Micro SD Card and put it into the SD card slot of RPi. Then connect your RPi to the monitor through the HDMI port, attach your mouse and keyboard through the USB ports, attach a network cable to the network port and finally, connect your power supply (making sure that it meets the specifications required by your RPi Module Version. Your RPi should start (power up). Later, after setup, you will need to enter your user name and password to login. The default user name: pi; password: raspberry. After login, you should see the following screen.




Congratulations! You have successfully installed the RASPBERRY PI OS operating system on your RPi.

Raspberry Pi 5, 4B, 3B+/3B integrates a Wi-Fi adaptor. You can use it to connect to your Wi-Fi. Then you can use the wireless remote desktop to control your RPi. This will be helpful for the following work. Raspberry Pi of other models can use wireless remote desktop through accessing an external USB wireless card.



# Remote desktop & VNC

**If you have logged in Raspberry Pi via display, you can skip to VNC Viewer.**

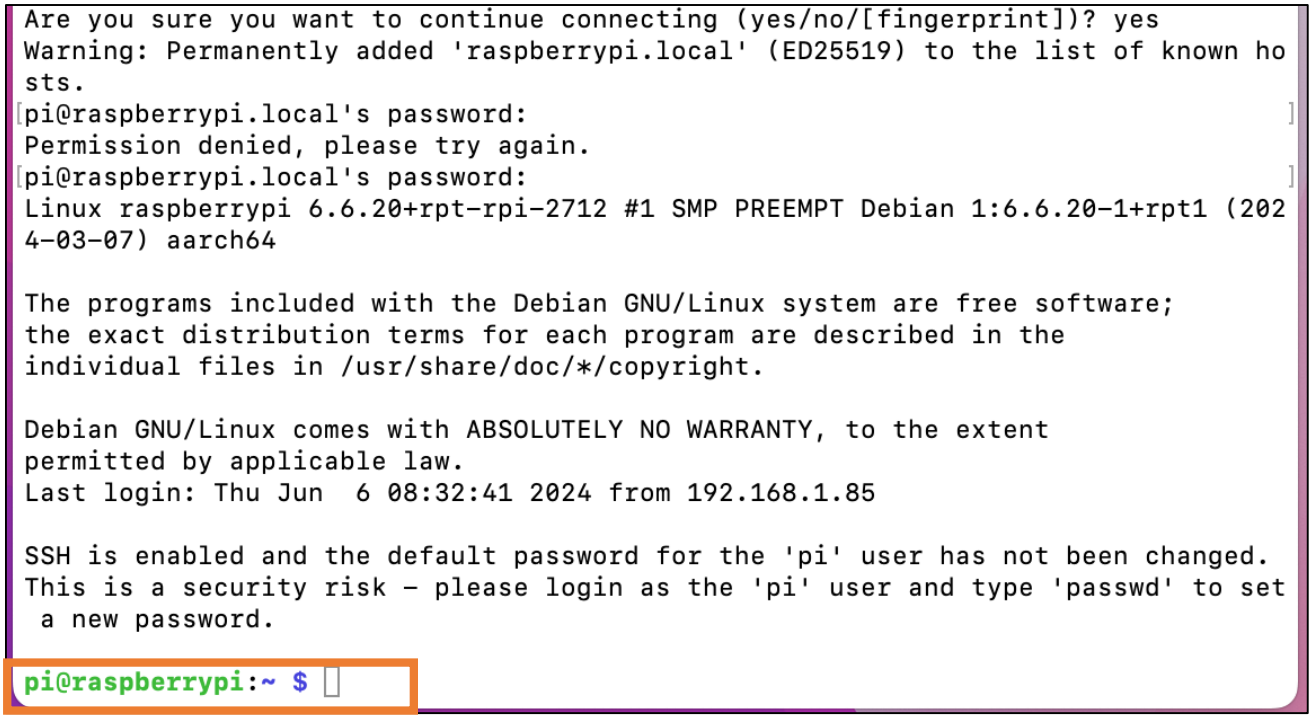
If you don't have a spare display, mouse and keyboard for your RPi, you can use a remote desktop to share a display, keyboard, and mouse with your PC. Below is how to use:
MAC OS remote desktop and Windows OS remote desktop.

## MAC OS Remote Desktop

Open the terminal and type following command. **If this command doesn't work, please move to next page.**

```
ssh pi@raspberrypi.local
```

The password is **raspberry** by default, case sensitive. You may need to type **yes** during the process.

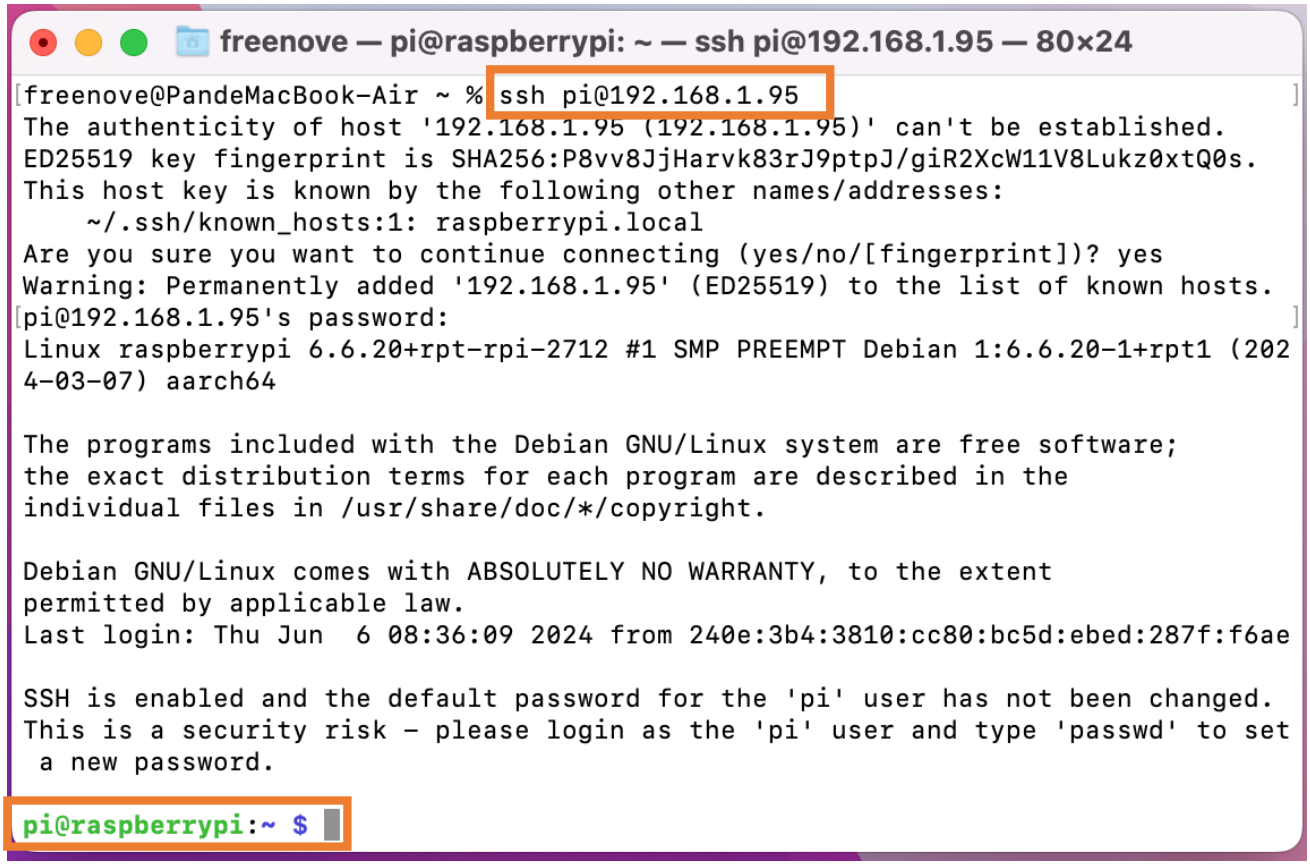
You can also use the IP address to log in Pi.

Enter **router** client to **inquiry IP address** named "raspberry pi". For example, I have inquired to **my RPi IP address, and it is "192.168.1.95".**

Open the terminal and type following command.

```
ssh pi@192.168.1.95
```

When you see pi@raspberrypi:~ $, you have logged in Pi successfully. Then you can skip to next section.



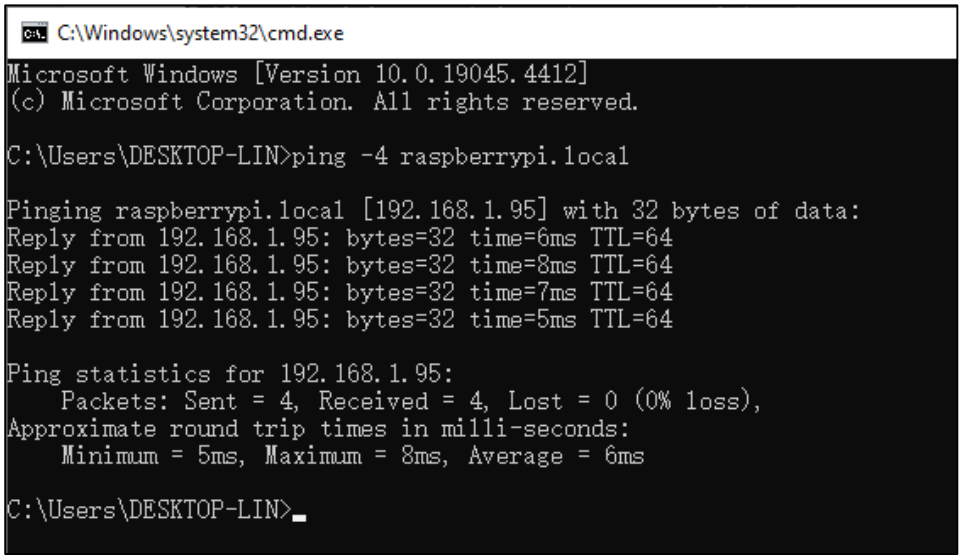

**Then you can skip to VNC Viewer.**

## Windows OS Remote Desktop

**If you are using win10, you can use follow way to login Raspberry Pi without desktop.**

Press **Win+R**. Enter **cmd**. Then use this command to check IP:

```
ping -4 raspberrypi.local
```



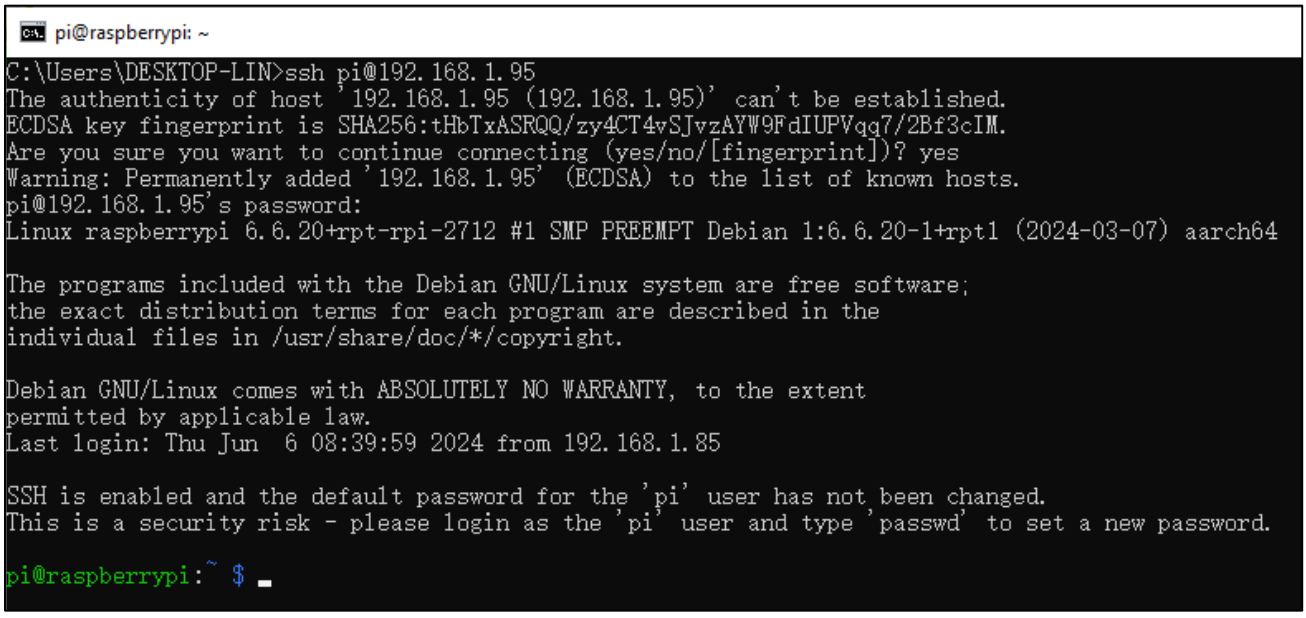

Then 192.168.1.147 is my Raspberry Pi IP.

Or enter **router** client to **inquiry IP address** named "**raspberrypi**". For example, I have inquired to **my RPi IP address, and it is "192.168.1.95".**

```
ssh pi@xxxxxxxxxxx(IP address)
```

Enter the following command:

```
ssh pi@192.168.1.95
```



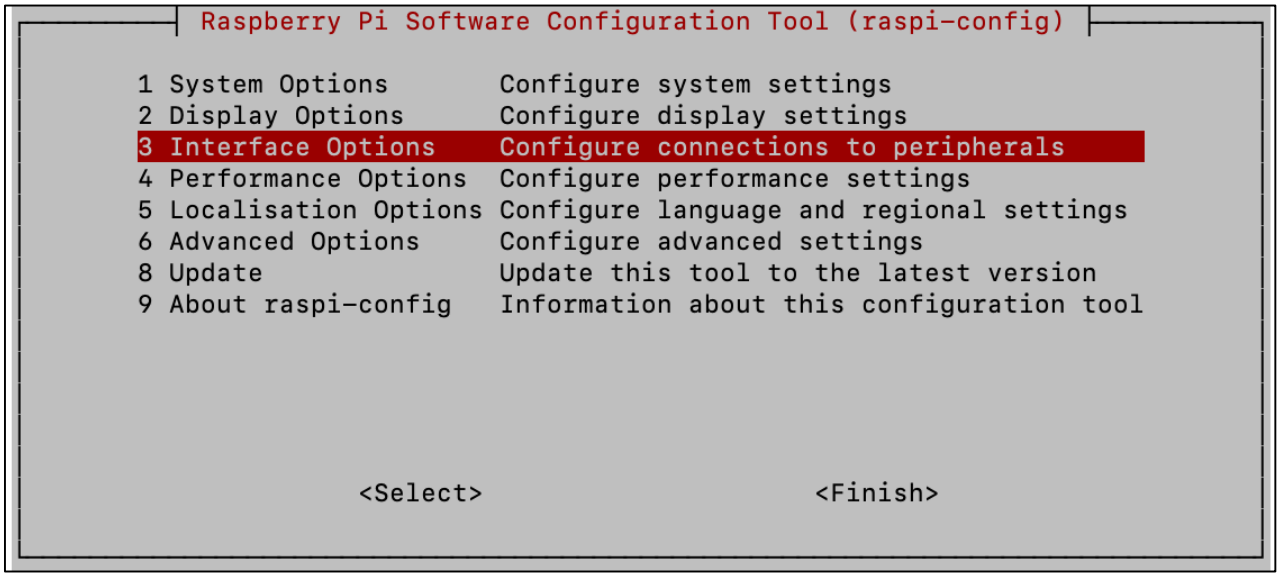
## VNC Viewer & VNC

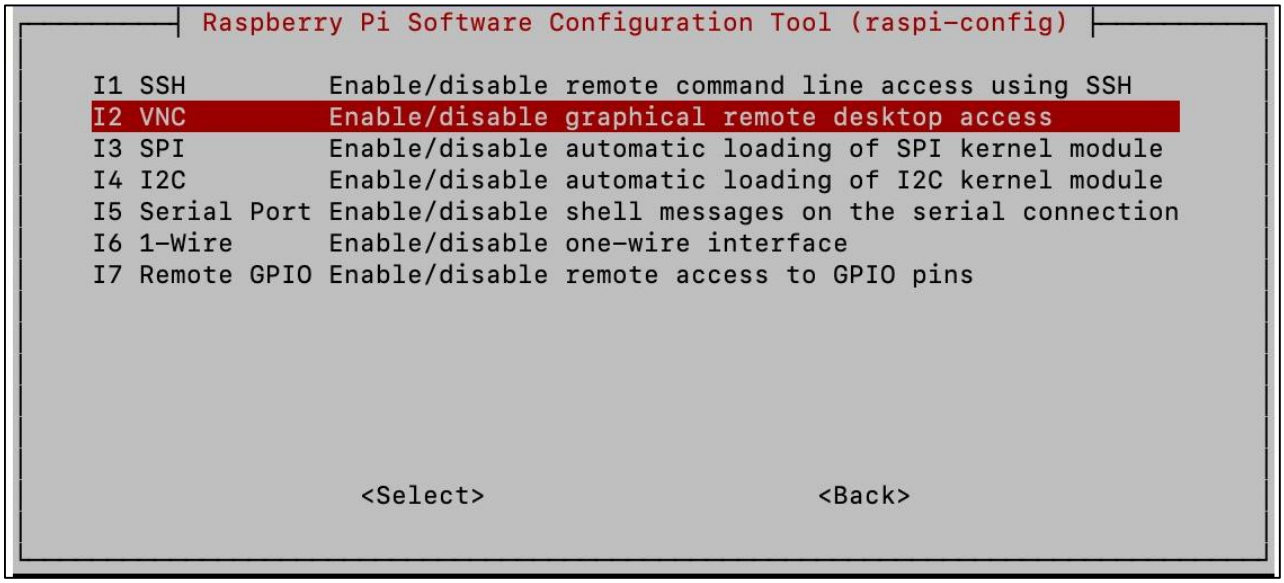
### Enable VNC

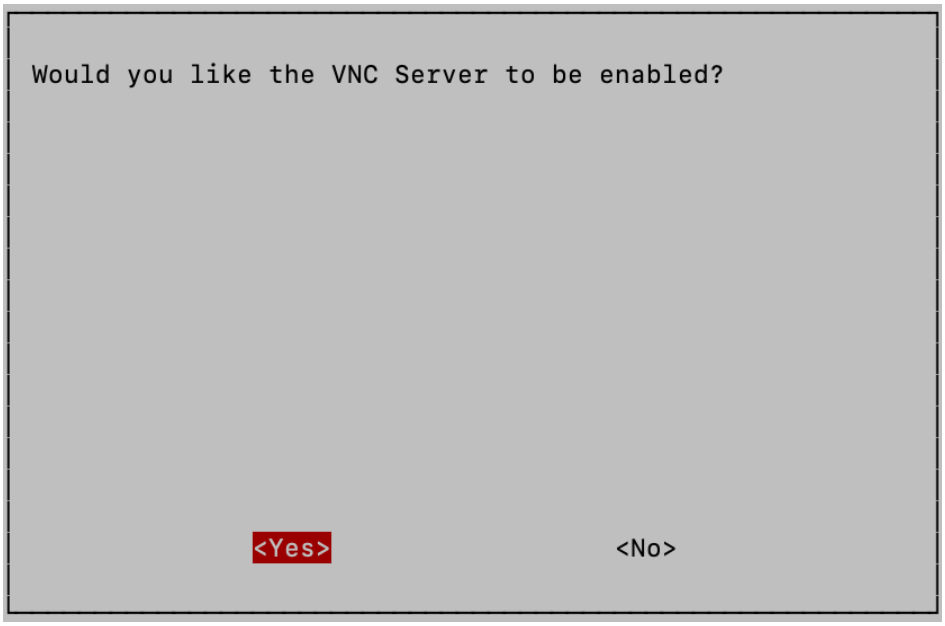
Type the following command. And select Interface Options→P5 VNC → Enter→Yes→OK. Here Raspberry Pi may need be restarted, and choose ok. Then open VNC interface.

```
sudo raspi-config
```

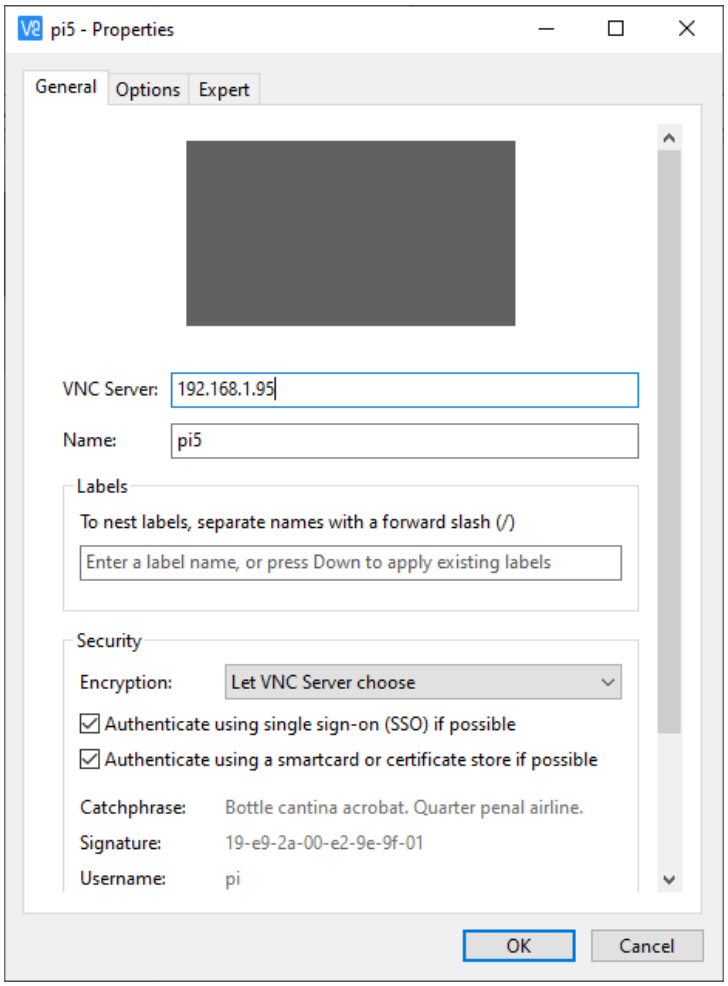
```
SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set
 a new password.

pi@raspberrypi:~ $ sudo raspi-config
```



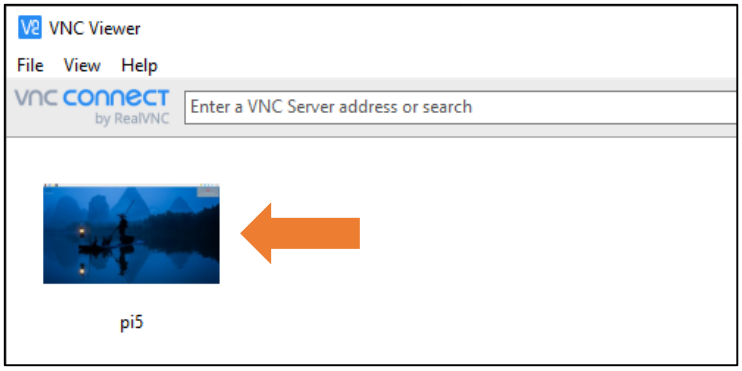



Then download and install VNC Viewer according to your computer system by click following link:
https://www.realvnc.com/en/connect/download/viewer/
After installation is completed, open VNC Viewer. And click File → New Connection. Then the interface is shown below.

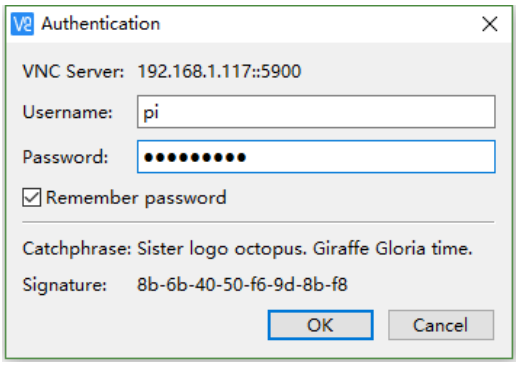



Enter ip address of your Raspberry Pi and fill in a name. Then click OK.

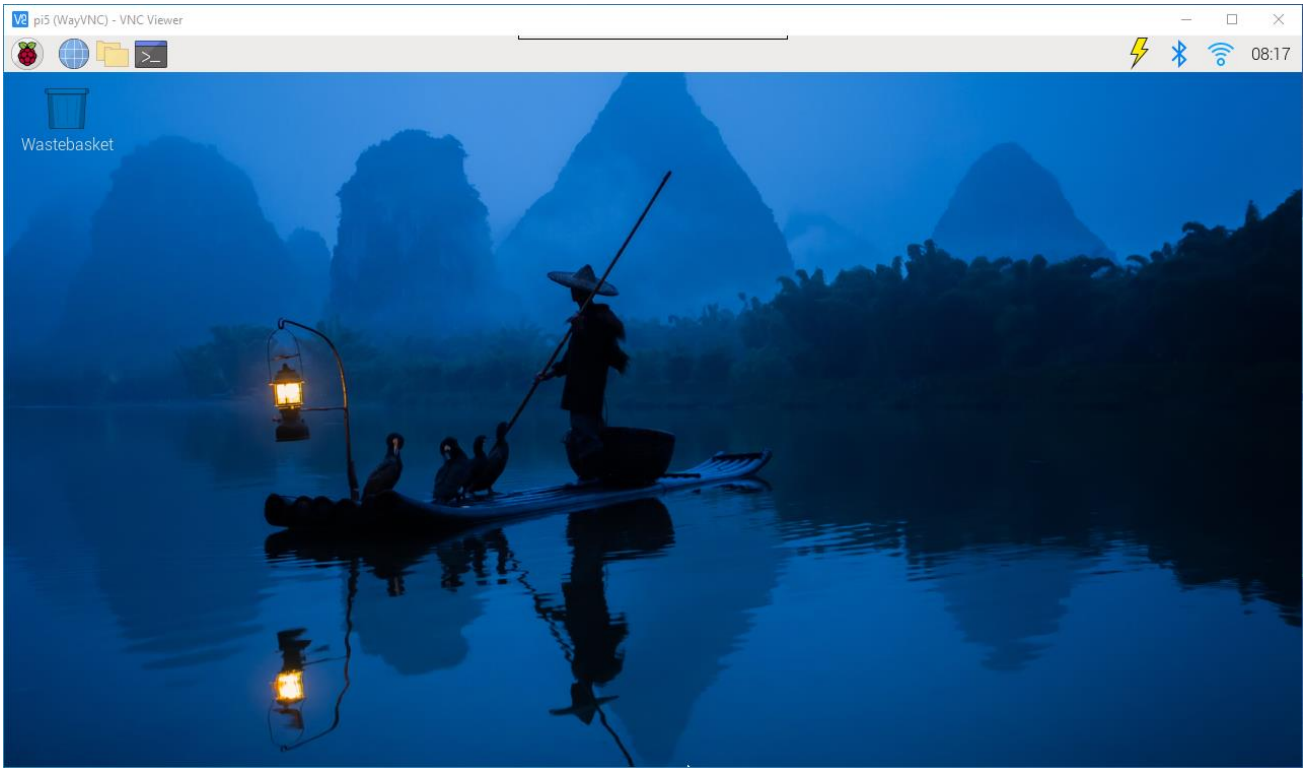
Then on the VNC Viewer panel, double-click new connection you just created,




and the following dialog box pops up.

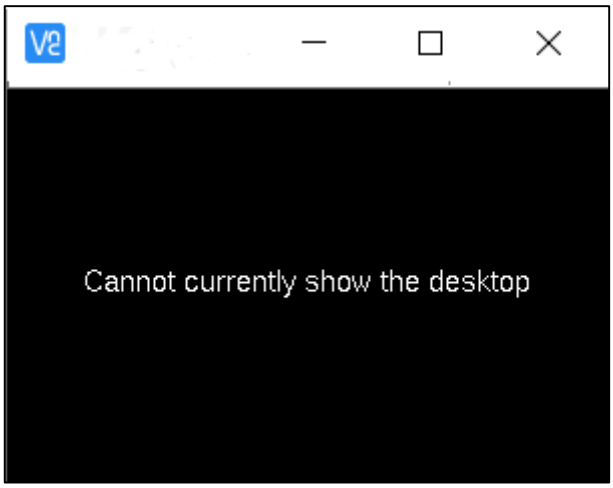



Enter username: **pi** and Password: **raspberry**. And click OK.



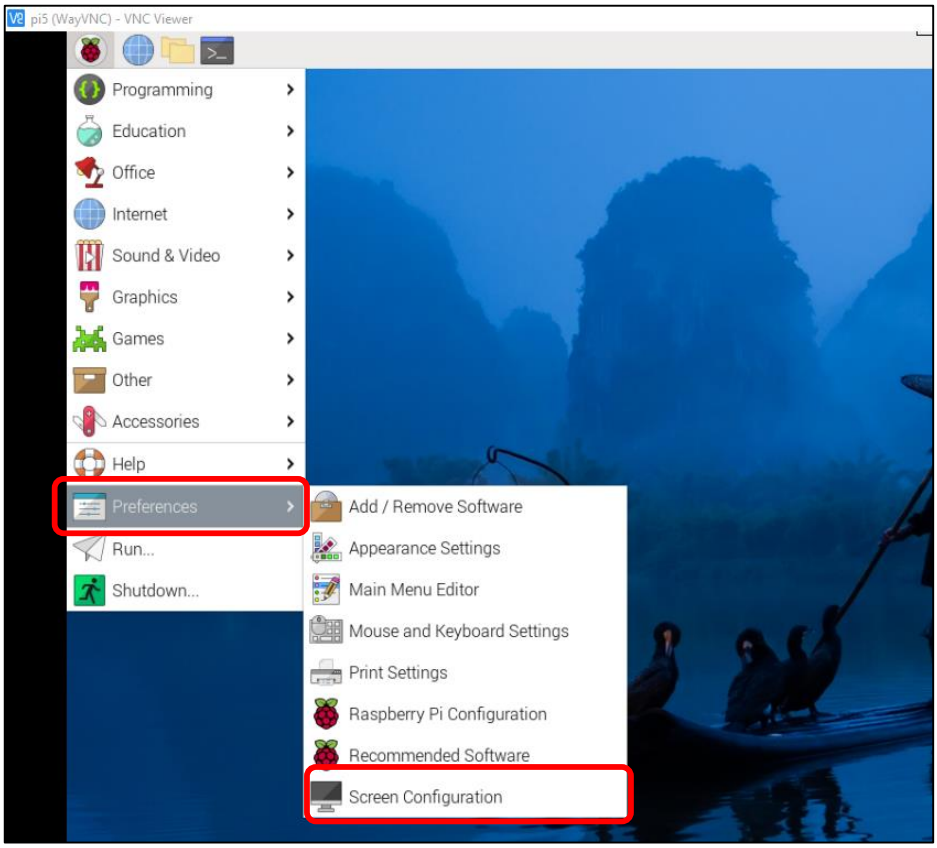

Here, you have logged in to Raspberry Pi successfully by using VNC Viewer

If there is black window, please set resolution.

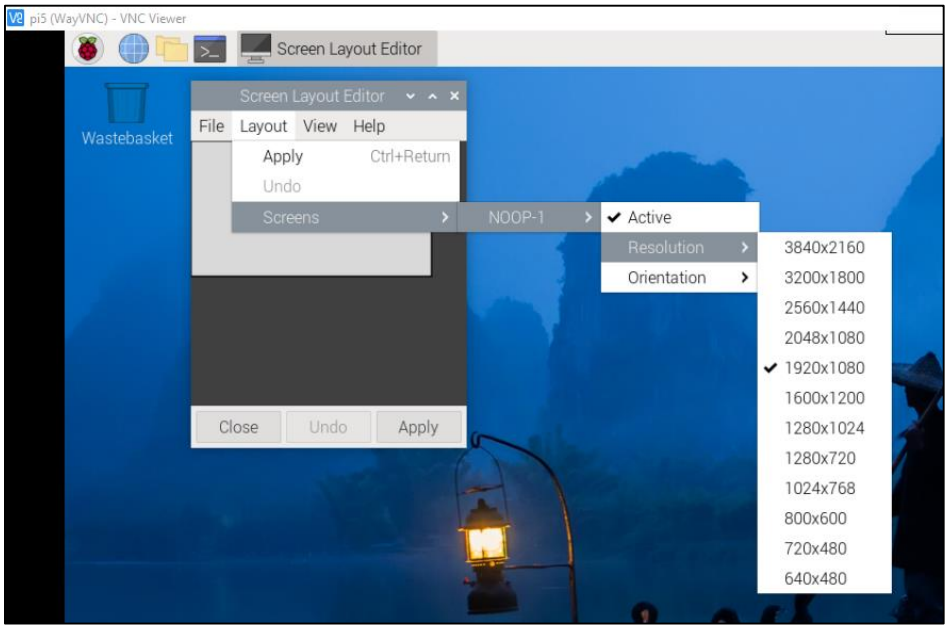



**Set Resolution**

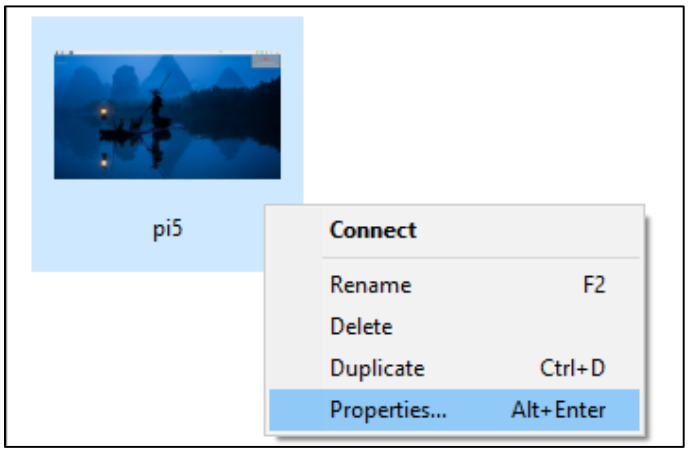
You can also set other resolutions.

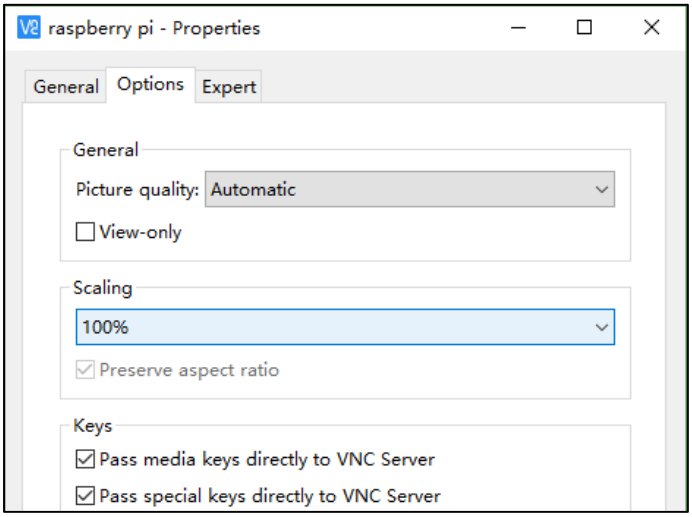


If you don't know what resolution to set properly, you can try 1920x1080.



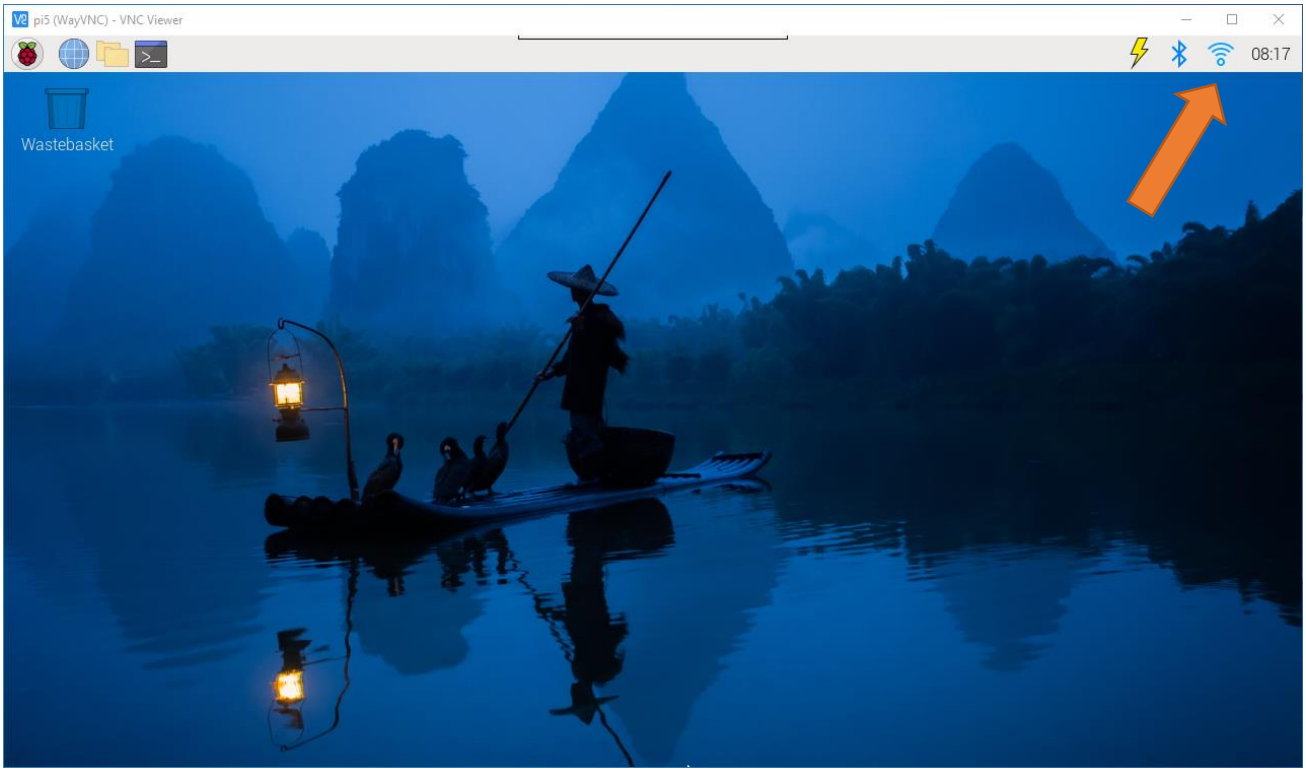

In addition, your VNC Viewer window may zoom your Raspberry Pi desktop. You can change it. On your VNC View control panel, click right key. And select Properties->Options label->Scaling. Then set proper scaling.



Here, you have logged in to Raspberry Pi successfully by using VNC Viewer and operated proper setting.

Raspberry Pi 5/4B/3B+/3B integrates a Wi-Fi adaptor.If you did not connect Pi to WiFi. You can connect it to wirelessly control the robot.

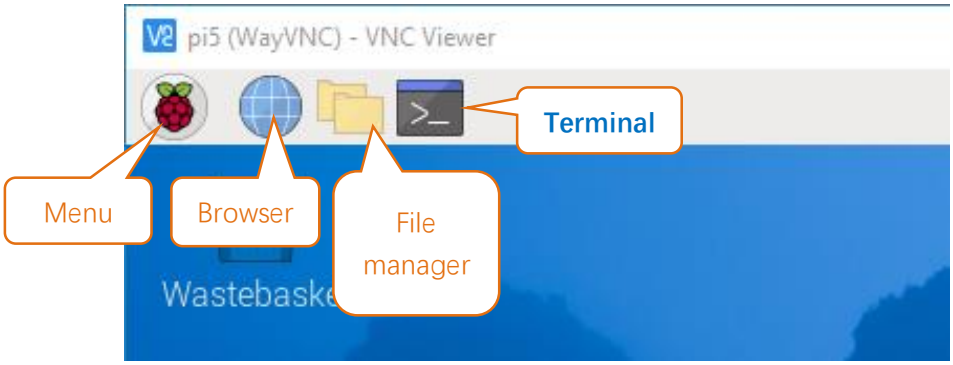


# Chapter 0 Preparation

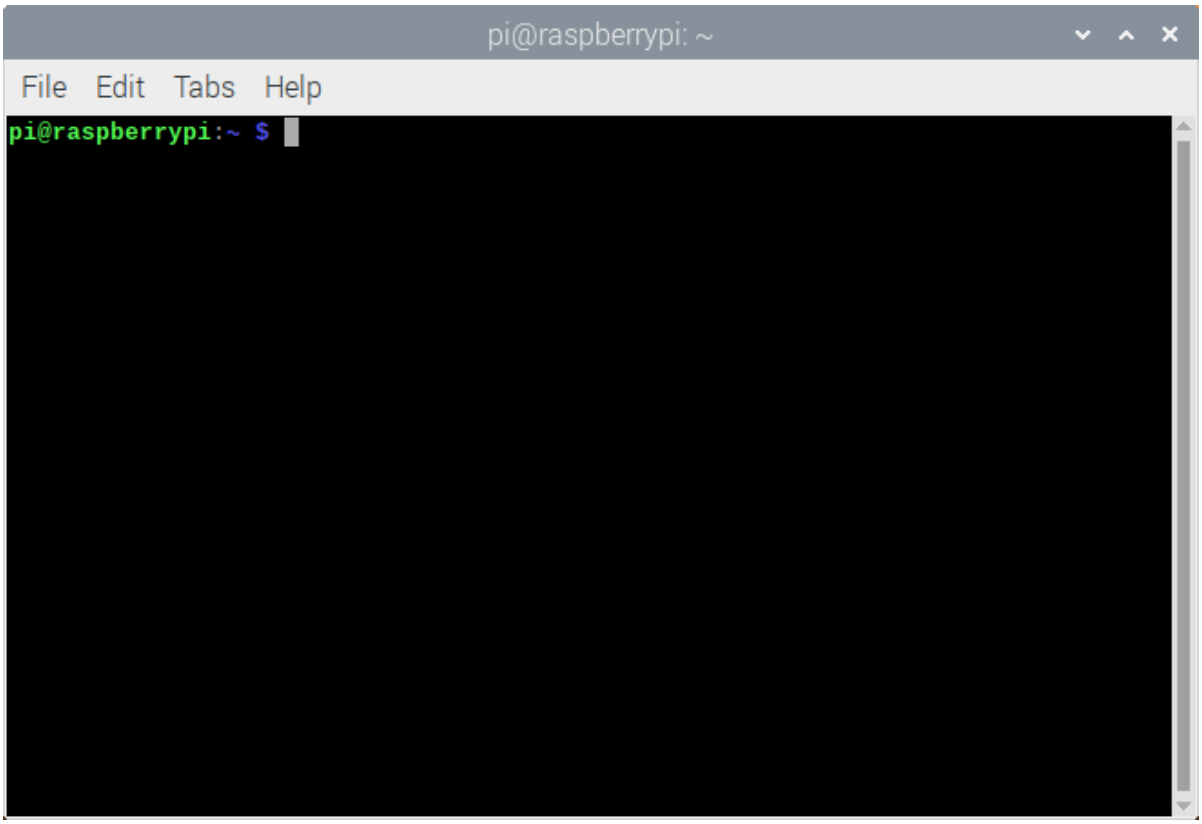
Why "Chapter 0"? Because in program code the first number is 0. We choose to follow this rule. In this chapter, we will do some necessary foundational preparation work: Start your Raspberry Pi and install some necessary libraries.

## Linux Command

Raspberry Pi OS is based on the Linux Operation System. Now we will introduce you to some frequently used Linux commands and rules.
First, open the Terminal. All commands are executed in Terminal.

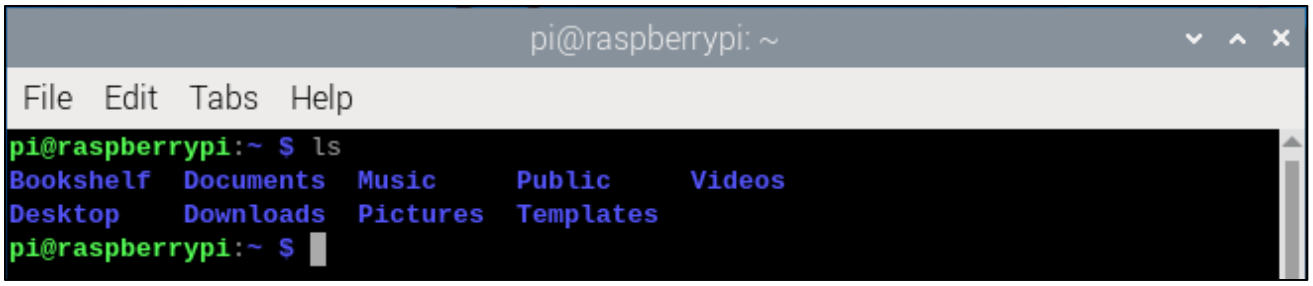



When you click the Terminal icon, following interface appears.



**Note: The Linux is case sensitive.**

First, type "ls" into the Terminal and press the "Enter" key. The result is shown below:




The "ls" command lists information about the files (the current directory by default).

Content between "$" and "pi@raspberrypi:" is the current working path. "~" represents the user directory, which refers to "/home/pi" here.

```
pi@raspberrypi:~ $ pwd
/home/pi
```

"cd" is used to change directory. "/" represents the root directory.

```
pi@raspberrypi:~ $ cd /usr
pi@raspberrypi:/usr $ ls
bin  games  include  lib  local  man  sbin  share  src
pi@raspberrypi:/usr $ cd ~
pi@raspberrypi:~ $
```

Later in this Tutorial, we will often change the working path. Typing commands under the wrong directory may cause errors and break the execution of further commands.

Many frequently used commands and instructions can be found in the following reference table.

| Command | instruction |
|---|---|
| **ls** | Lists information about the FILEs (the current directory by default) and entries alphabetically. |
| **cd** | Changes directory |
| **sudo + cmd** | Executes cmd under root authority |
| **./** | Under current directory |
| **gcc** | GNU Compiler Collection |
| **git clone URL** | Use git tool to clone the contents of specified repository, and URL in the repository address. |

There are many commands, which will come later. For more details about commands. You can refer to:
http://www.linux-commands-examples.com

## Shortcut Key

Now, we will introduce several commonly used shortcuts that are very useful in Terminal.

1. **Up and Down Arrow Keys**: Pressing "↑" (the Up key) will go backwards through the command history and pressing "↓" (the Down Key) will go forwards through the command history.

2. **Tab Key**: The Tab key can automatically complete the command/path you want to type. When there is only one eligible option, the command/path will be completely typed as soon as you press the Tab key even you only type one character of the command/path.

As shown below, under the '~' directory, you enter the Documents directory with the "cd" command. After typing "cd D", pressing the Tab key (there is no response), pressing the Tab key again then all the files/folders that begin with "D" will be listed. Continue to type the letters "oc" and then pressing the Tab key, the "Documents" is typed automatically.

```
pi@raspberrypi:~ $ cd D
Desktop/   Documents/ Downloads/
pi@raspberrypi:~ $ cd Doc
```

```
pi@raspberrypi:~ $ cd D
Desktop/   Documents/ Downloads/
pi@raspberrypi:~ $ cd Documents/
```

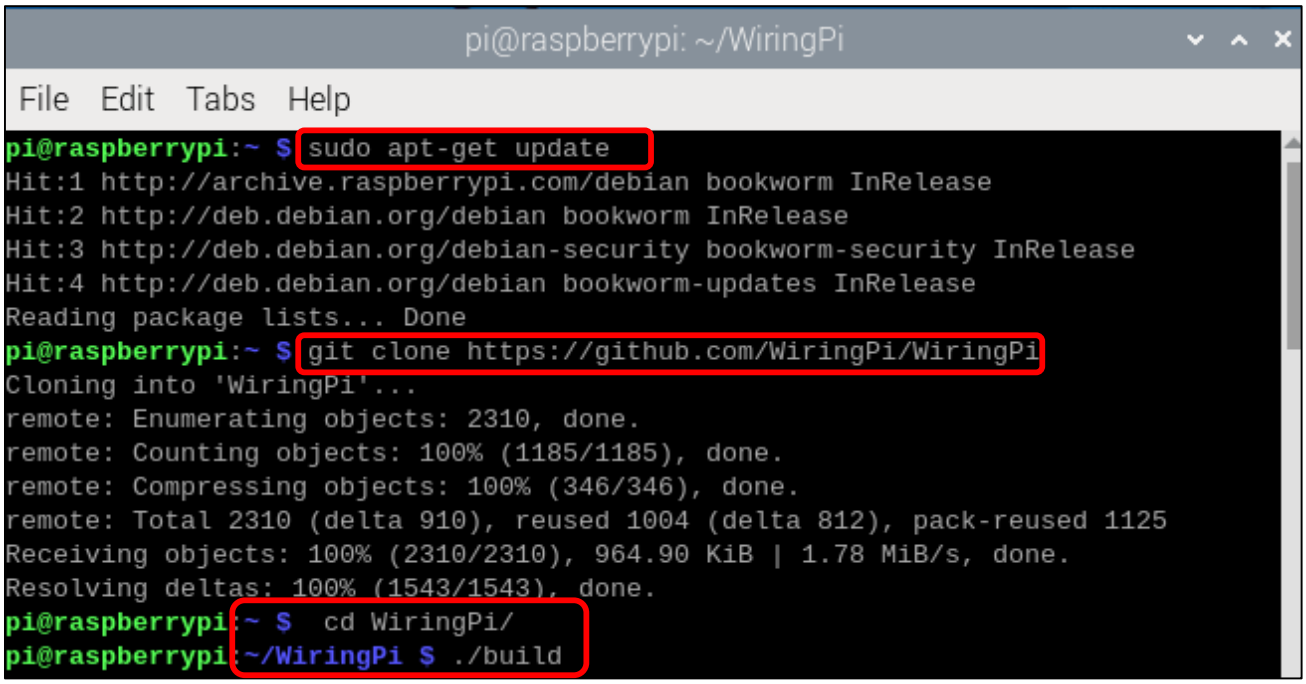
# Install WiringPi

WiringPi is a GPIO access library written in C language for the used in the Raspberry Pi.

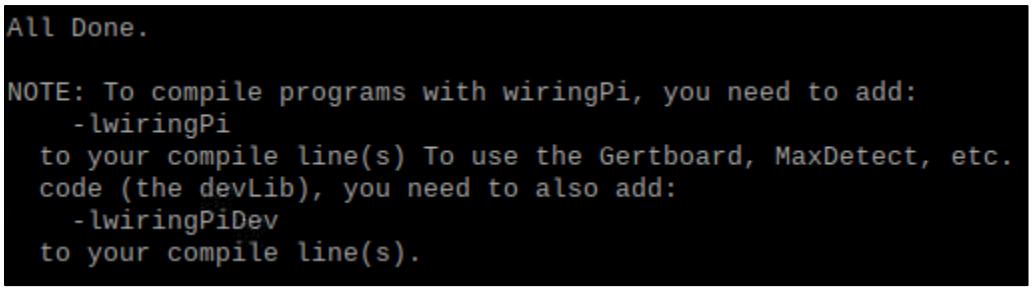
## WiringPi Installation Steps

To install the WiringPi library, please open the Terminal and then follow the steps and commands below.
Note: For a command containing many lines, execute them one line at a time.
Enter the following commands **one by one** in the **terminal** to install WiringPi:

```
sudo apt-get update
git clone https://github.com/WiringPi/WiringPi
cd WiringPi
./build
```



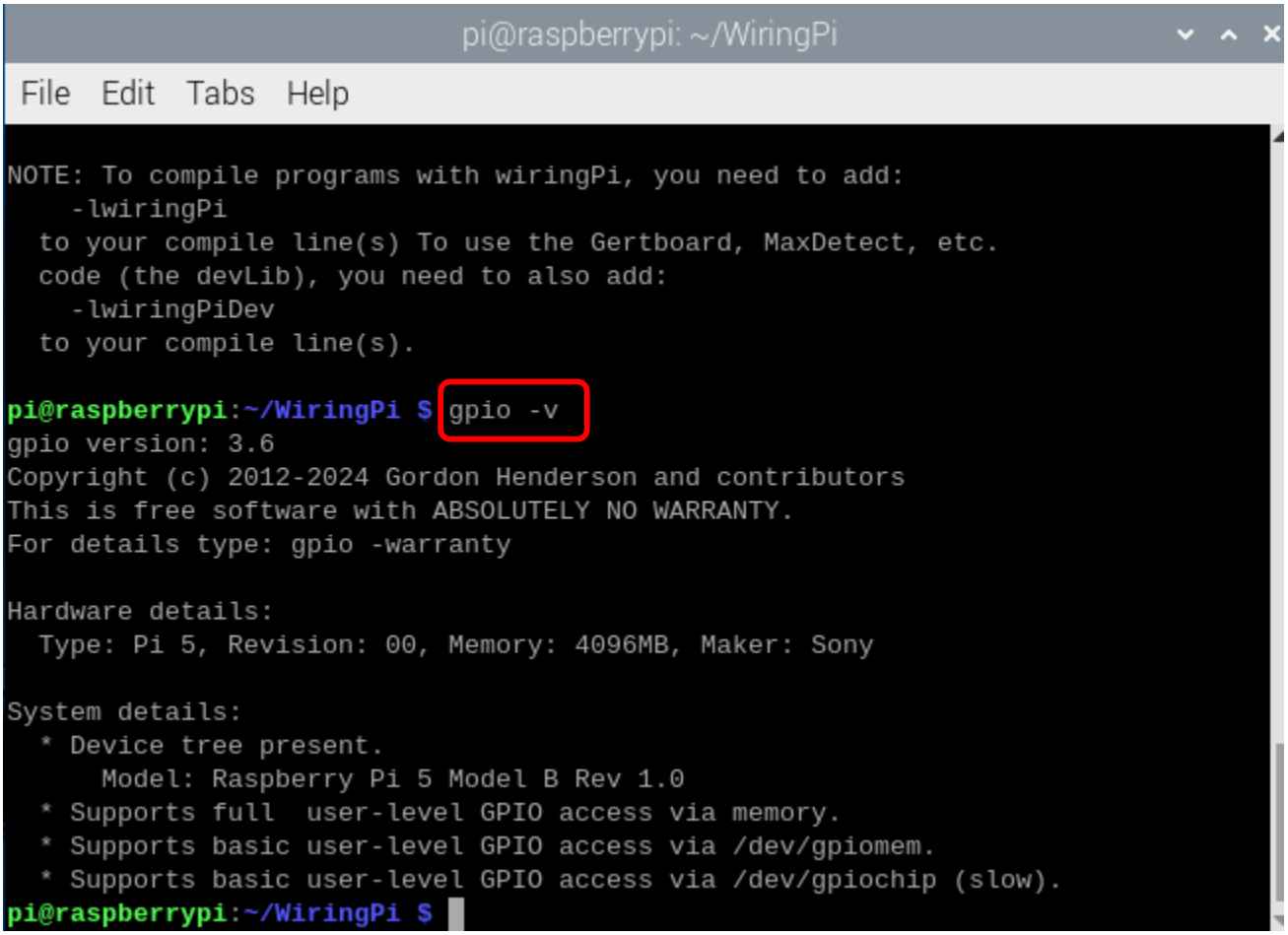

The following figure shows the successful installation.



Run the gpio command to check the installation:

```
gpio -v
```

That should give you some confidence that the installation was a success.



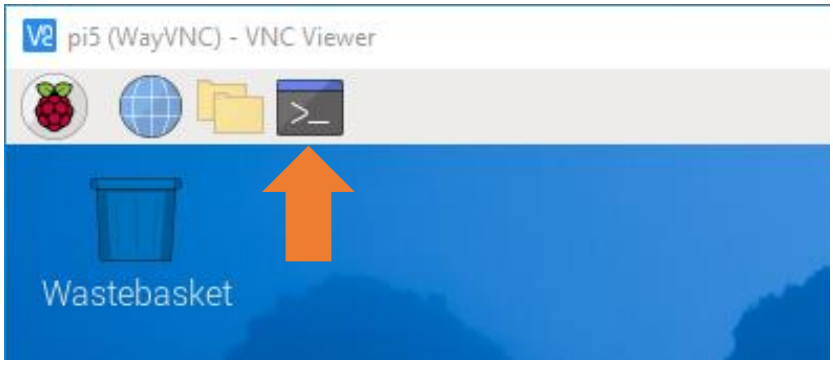
# Obtain the Project Code

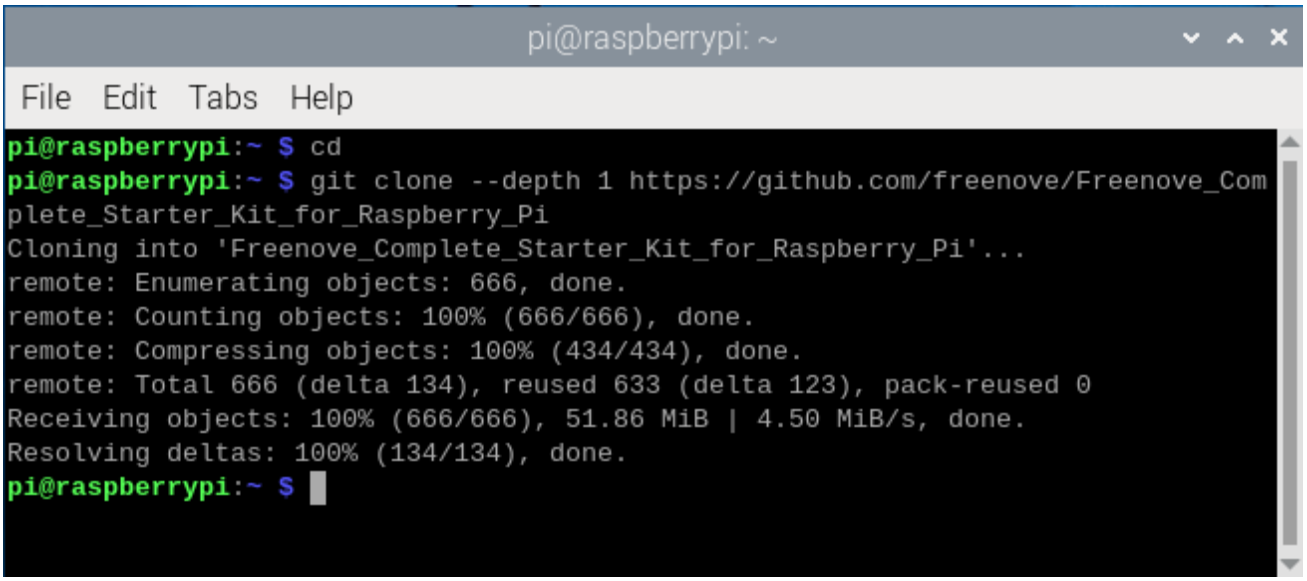
After the above installation is completed, you can visit our official website (http://www.freenove.com) or our GitHub resources at (https://github.com/freenove) to download the latest available project code. We provide both **C** language and **Python** language code for each project to allow ease of use for those who are skilled in either language.

This is the method for obtaining the code:
In the pi directory of the RPi terminal, enter the following command.

```
cd
git clone --depth 1 https://github.com/freenove/Freenove_Complete_Starter_Kit_for_Raspberry_Pi
```

**(There is no need for a password. If you get some errors, please check your commands.)**



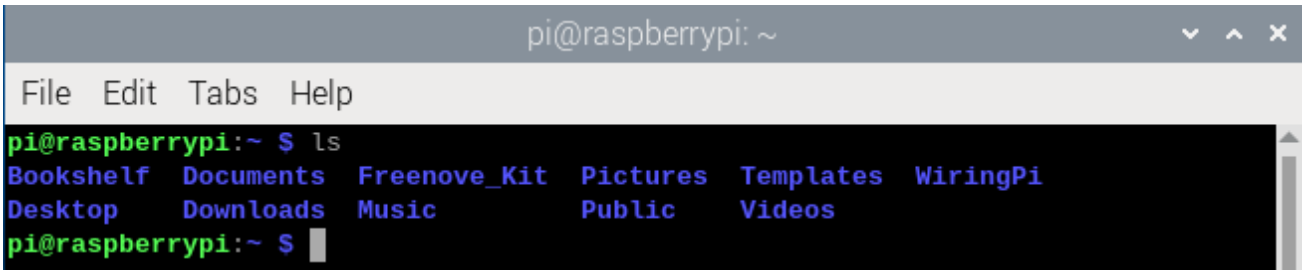



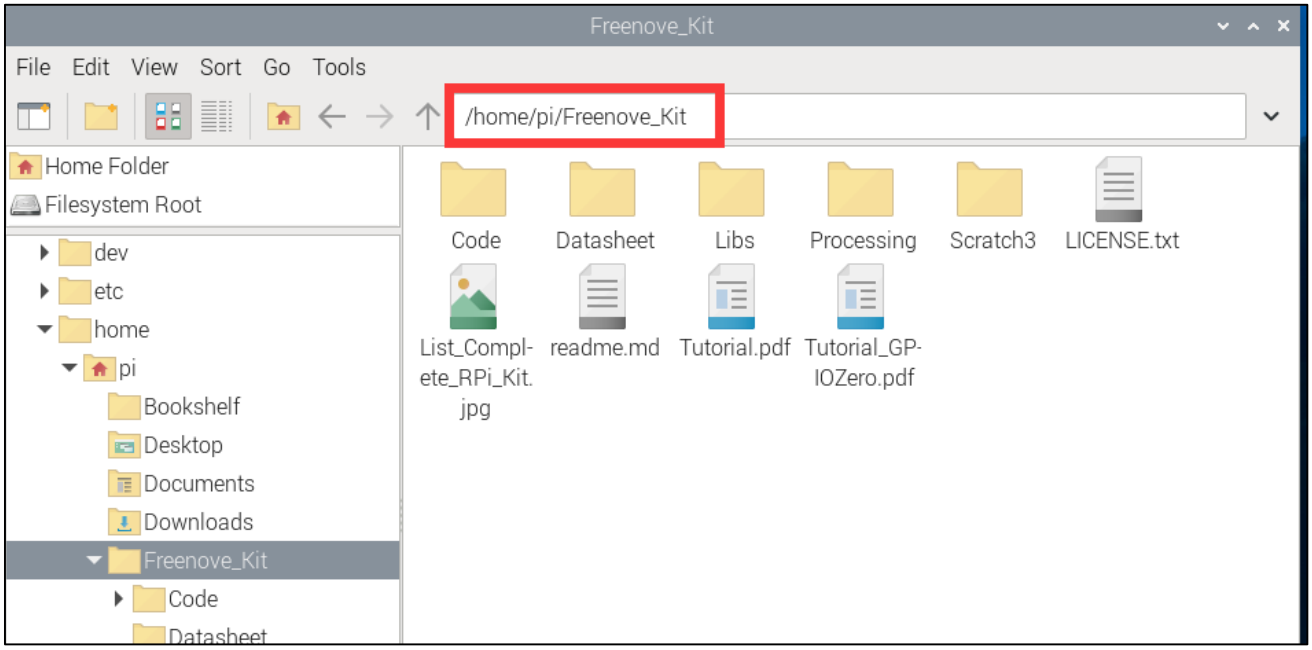

After the download is completed, a new folder "Freenove_Complete_Starter_Kit_for_Raspberry_Pi" is generated, which contains all of the tutorials and required code.

This folder name seems a little too long. We can simply rename it by using the following command.

```
mv Freenove_Complete_Starter_Kit_for_Raspberry_Pi/ Freenove_Kit/
```

"Freenove_Kit" is now the new and much shorter folder name.



# Chapter 1 LED

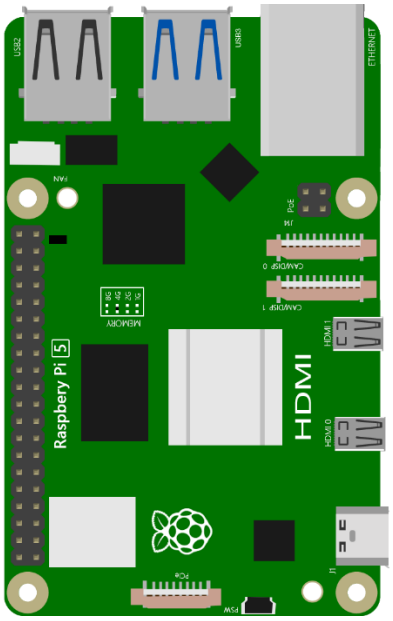
This chapter is the Start Point in the journey to build and explore RPi electronic projects. We will start with simple “Blink” project.

## Project 1.1 Blink

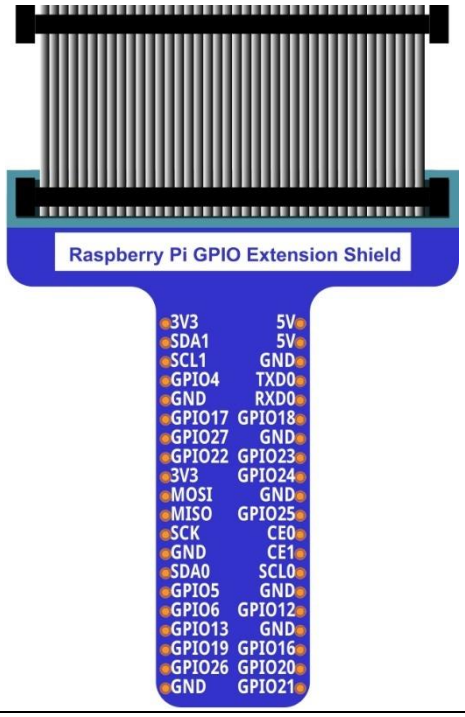
In this project, we will use RPi to control blinking a common LED.

### Component List

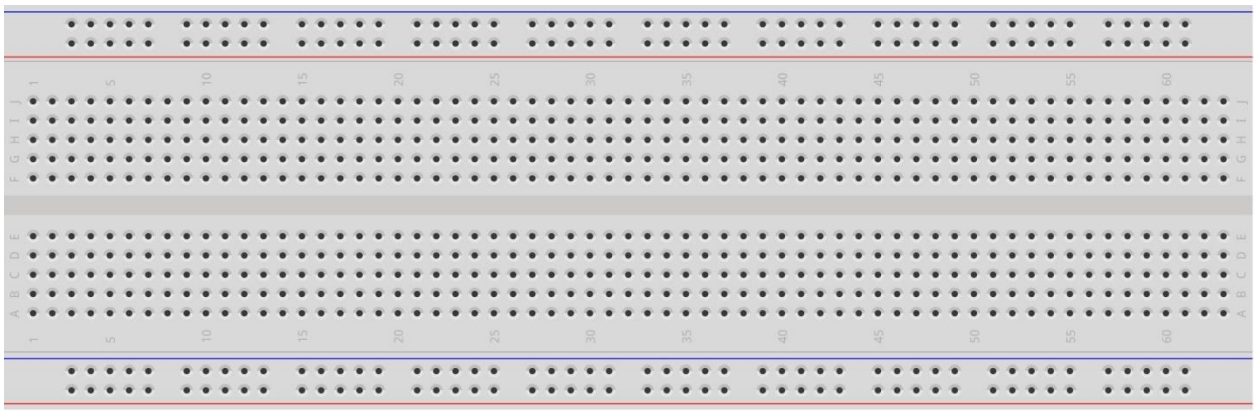
Raspberry Pi
(Recommended: Raspberry Pi 5 / 4B / 3B+ / 3B
Compatible: 3A+ / 2B / 1B+ / 1A+ / Zero W / Zero)




GPIO Extension Board & Ribbon Cable




Breadboard x1



| LED x1 | Resistor 220Ω x1 | Jumper<br>**Specific quantity depends on the circuit.** |
|---|---|---|
|  |  |  |

In the components list, 3B GPIO, Extension Shield Raspberry and Breadboard are necessary for each project. Later, they will be reference by text only (no images as in above).

# GPIO

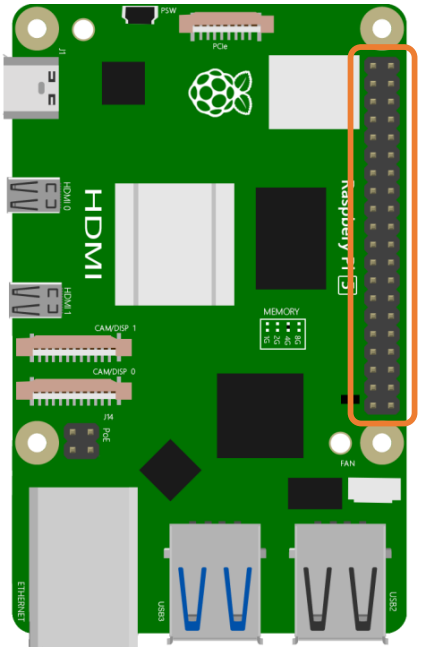
GPIO: General Purpose Input/Output. Here we will introduce the specific function of the pins on the Raspberry Pi and how you can utilize them in all sorts of ways in your projects. Most RPi Module pins can be used as either an input or output, depending on your program and its functions.

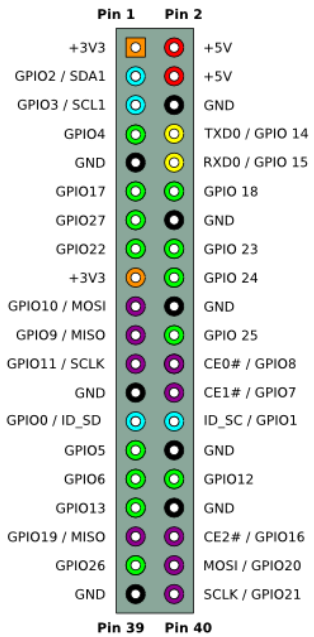
When programming GPIO pins there are 3 different ways to reference them: GPIO Numbering, Physical Numbering and WiringPi GPIO Numbering.

## BCM GPIO Numbering

The Raspberry Pi CPU uses Broadcom (BCM) processing chips BCM2835, BCM2836 or BCM2837. GPIO pin numbers are assigned by the processing chip manufacturer and are how the computer recognizes each pin. The pin numbers themselves do not make sense or have meaning as they are only a form of identification. Since their numeric values and physical locations have no specific order, there is no way to remember them so you will need to have a printed reference or a reference board that fits over the pins.

Each pin's functional assignment is defined in the image below:

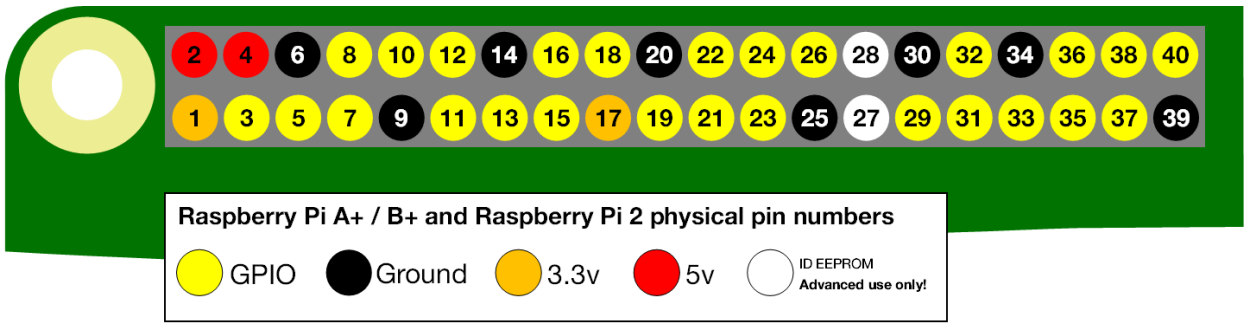



For more details about pin definition of GPIO, please refer to http://pinout.xyz/

## PHYSICAL Numbering

Another way to refer to the pins is by simply counting across and down from pin 1 at the top left (nearest to the SD card). This is 'Physical Numbering', as shown below:




## WiringPi GPIO Numbering

Different from the previous two types of GPIO serial numbers, RPi GPIO serial number of the WiringPi are numbered according to the BCM chip use in RPi.

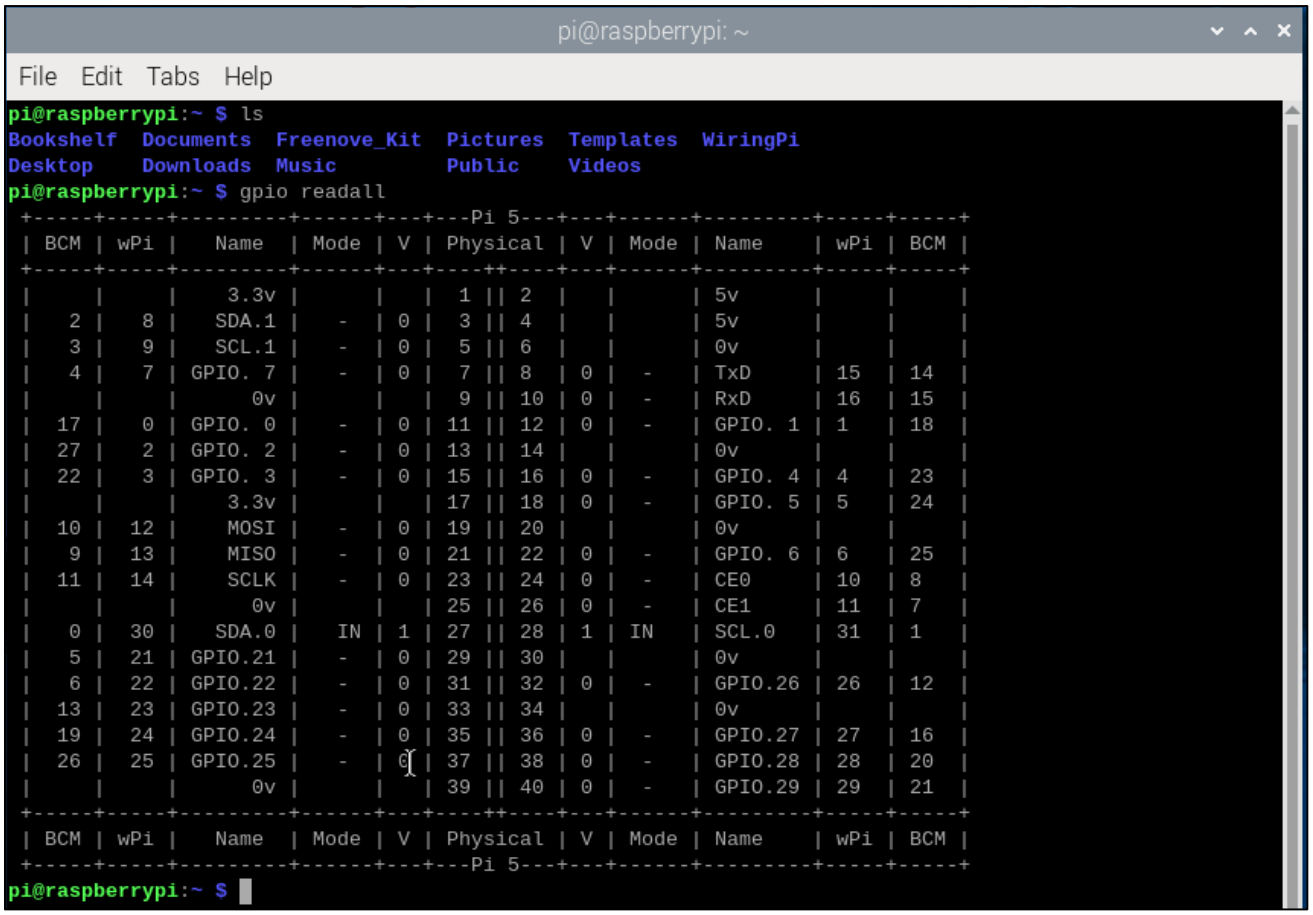
| wiringPi Pin | BCM GPIO | Name | Header | Name | BCM GPIO | wiringPi Pin | |
|---|---|---|---|---|---|---|---|
| — | — | 3.3v | 1 \| 2 | 5v | — | — | For Pi B |
| 8 | R1:0/R2:2 | SDA | 3 \| 4 | 5v | — | — | |
| 9 | R1:1/R2:3 | SCL | 5 \| 6 | 0v | — | — | |
| 7 | 4 | GPIO7 | 7 \| 8 | TxD | 14 | 15 | |
| — | — | 0v | 9 \| 10 | RxD | 15 | 16 | |
| 0 | 17 | GPIO0 | 11 \| 12 | GPIO1 | 18 | 1 | |
| 2 | R1:21/R2:27 | GPIO2 | 13 \| 14 | 0v | — | — | |
| 3 | 22 | GPIO3 | 15 \| 16 | GPIO4 | 23 | 4 | |
| — | — | 3.3v | 17 \| 18 | GPIO5 | 24 | 5 | |
| 12 | 10 | MOSI | 19 \| 20 | 0v | — | — | |
| 13 | 9 | MISO | 21 \| 22 | GPIO6 | 25 | 6 | |
| 14 | 11 | SCLK | 23 \| 24 | CE0 | 8 | 10 | |
| — | — | 0v | 25 \| 26 | CE1 | 7 | 11 | |
| 30 | 0 | SDA.0 | 27 \| 28 | SCL.0 | 1 | 31 | |
| 21 | 5 | GPIO.21 | 29 \| 30 | 0V | | | |
| 22 | 6 | GPIO.22 | 31 \| 32 | GPIO.26 | 12 | 26 | |
| 23 | 13 | GPIO.23 | 33 \| 34 | 0V | | | |
| 24 | 19 | GPIO.24 | 35 \| 36 | GPIO.27 | 16 | 27 | |
| 25 | 26 | GPIO.25 | 37 \| 38 | GPIO.28 | 20 | 28 | |
| | | 0V | 39 \| 40 | GPIO.29 | 21 | 29 | |
| wiringPi Pin | BCM GPIO | Name | Header | Name | BCM GPIO | wiringPi Pin | |

For A+, B+, 2B, 3B, 3B+, 4B, Zero

(For more details, please refer to https://projects.drogon.net/raspberry-pi/wiringpi/pins/ )

You can also use the following command to view their correlation.

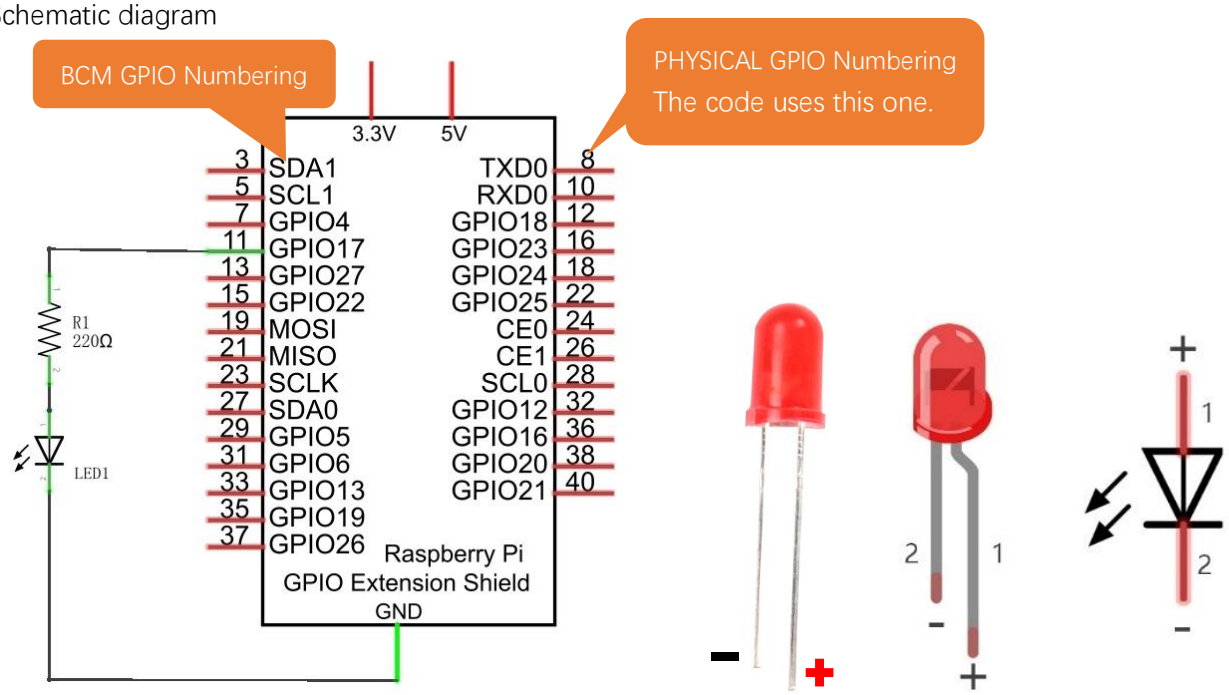
```
gpio readall
```



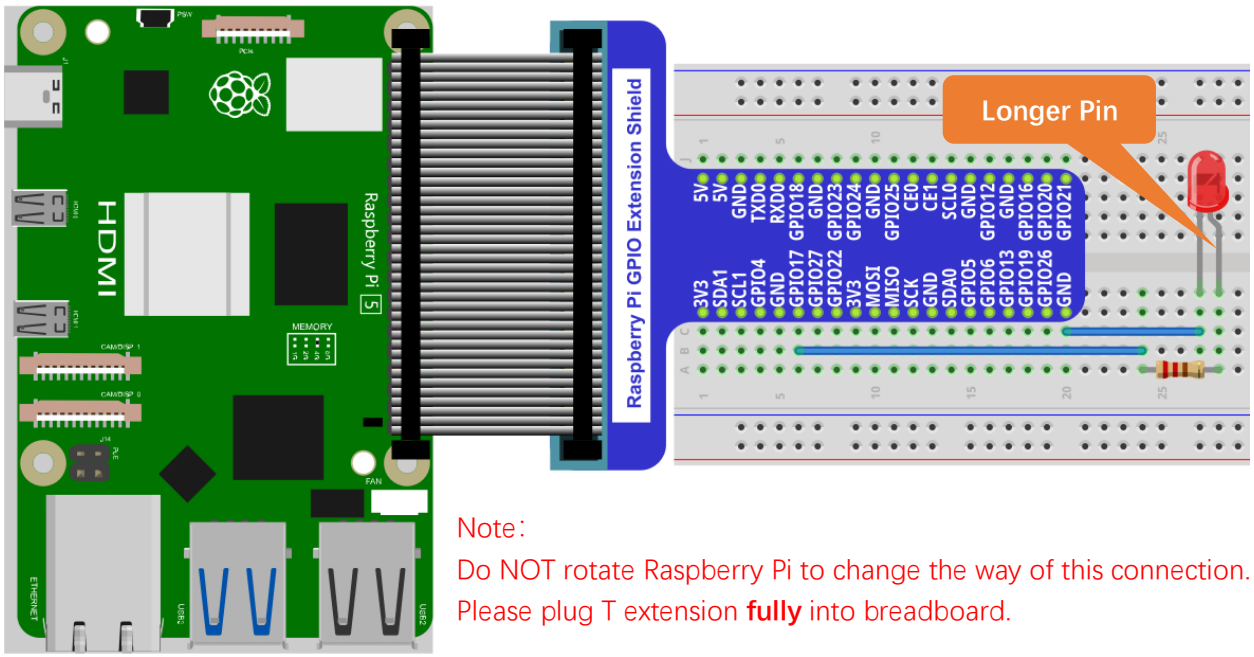
# Circuit

First, disconnect your RPi from the GPIO Extension Shield. Then build the circuit according to the circuit and hardware diagrams. After the circuit is built and verified correct, connect the RPi to GPIO Extension Shield.
CAUTION: Avoid any possible short circuits (especially connecting 5V or GND, 3.3V and GND)!
WARNING: A short circuit can cause high current in your circuit, create excessive component heat and cause permanent damage to your RPi!

Schematic diagram



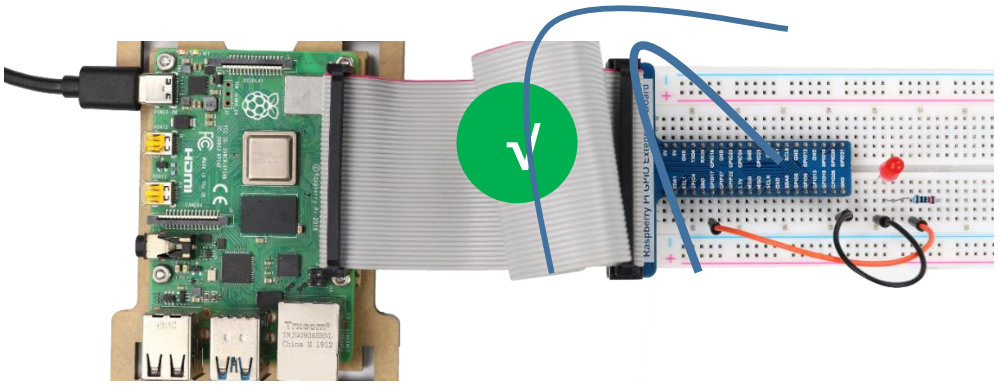

Hardware connection. **If you need any support, please contact us via:** support@freenove.com

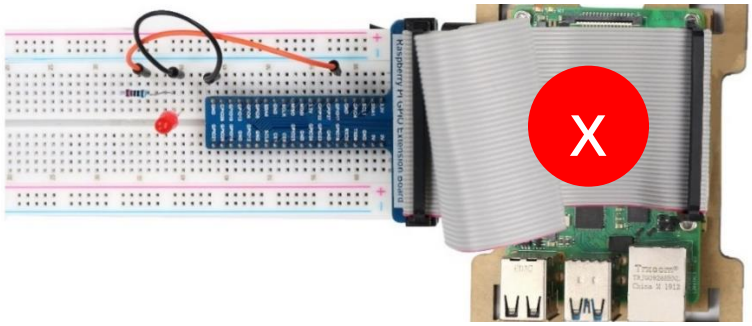


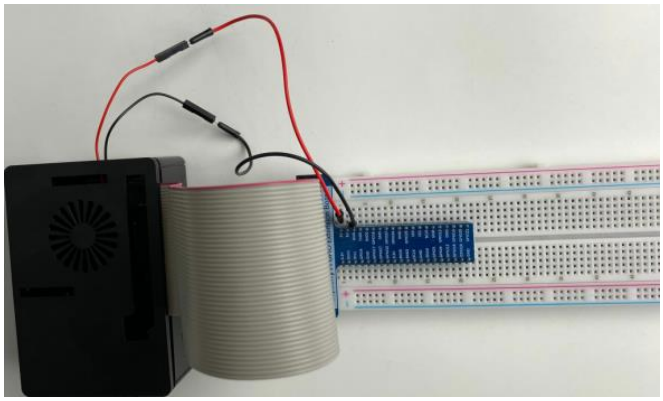

Note:
Do NOT rotate Raspberry Pi to change the way of this connection.
Please plug T extension **fully** into breadboard.

**Youtube video** https://youtu.be/hGQtnxsr1L4

**The connection of Raspberry Pi T extension board is as below. Don't reverse the ribbon.**



If you have a fan, you can connect it to 5V GND of breadboard via jumper wires.

**How to distinguish resistors?**

There are only three kind of resistors in this kit.

The one with 1 red ring is 10KΩ 

The one with 2 red rings is 220Ω 

The one with 0 red ring is 1KΩ 

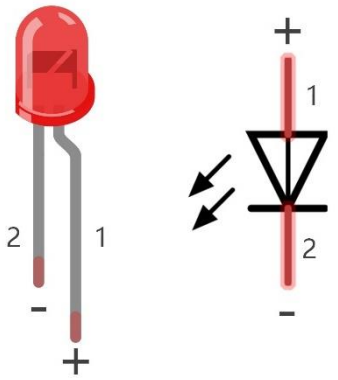
Future hardware connection diagrams will only show that part of breadboard and GPIO Extension Shield.

# Component knowledge

## LED

An LED is a type of diode. All diodes only work if current is flowing in the correct direction and have two Poles. An LED will only work (light up) if the longer pin (+) of LED is connected to the positive output from a power source and the shorter pin is connected to the negative (-) output, which is also referred to as Ground (GND). This type of component is known as “Polar” (think One-Way Street).

All common 2 lead diodes are the same in this respect. Diodes work only if the voltage of its positive electrode is higher than its negative electrode and there is a narrow range of operating voltage for most all common diodes of 1.9 and 3.4V. If you use much more than 3.3V the LED will be damaged and burnt out.



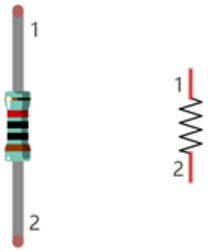

| LED | Voltage | Maximum current | Recommended current |
|---|---|---|---|
| Red | 1.9-2.2V | 20mA | 10mA |
| Green | 2.9-3.4V | 10mA | 5mA |
| Blue | 2.9-3.4V | 10mA | 5mA |
| Volt ampere characteristics conform to diode | | | |

Note: LEDs cannot be directly connected to a power supply, which usually ends in a damaged component. A resistor with a specified resistance value must be connected in series to the LED you plan to use.

## Resistor

Resistors use Ohms (Ω) as the unit of measurement of their resistance (R). 1MΩ=1000kΩ, 1kΩ=1000Ω.

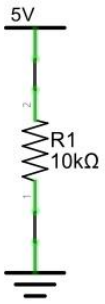
A resistor is a passive electrical component that limits or regulates the flow of current in an electronic circuit. On the left, we see a physical representation of a resistor, and the right is the symbol used to represent the presence of a resistor in a circuit diagram or schematic.




The bands of color on a resistor is a shorthand code used to identify its resistance value. For more details of resistor color codes, please refer to the card in the kit package.

With a fixed voltage, there will be less current output with greater resistance added to the circuit. The relationship between Current, Voltage and Resistance can be expressed by this formula: I=V/R known as Ohm’s Law where I = Current, V = Voltage and R = Resistance. Knowing the values of any two of these allows you to solve the value of the third.

In the following diagram, the current through R1 is: I=U/R=5V/10kΩ=0.0005A=0.5mA.

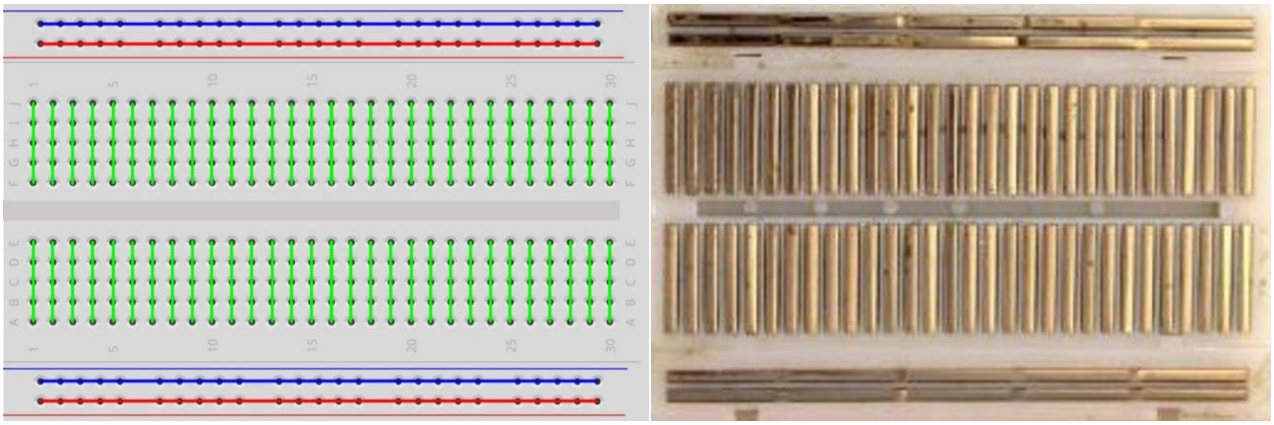



WARNING: Never connect the two poles of a power supply with anything of low resistance value (i.e. a metal object or bare wire) this is a Short and results in high current that may damage the power supply and electronic components.

Note: Unlike LEDs and Diodes, Resistors have no poles and re non-polar (it does not matter which direction you insert them into a circuit, it will work the same)

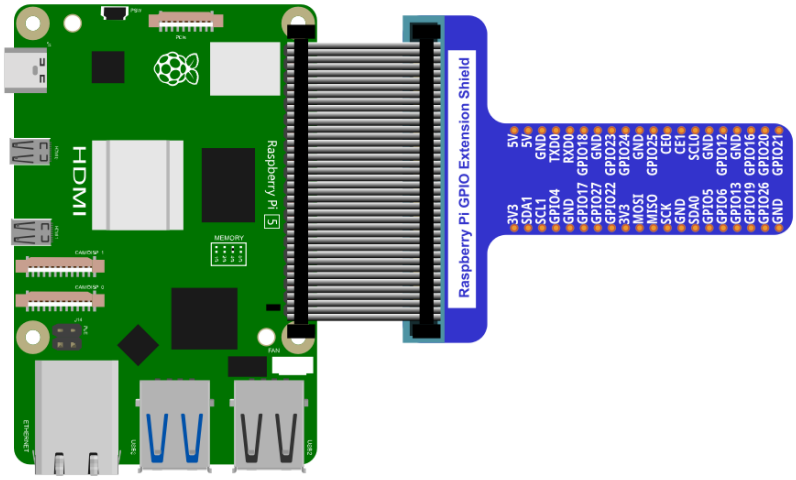
## Breadboard

Here we have a small breadboard as an example of how the rows of holes (sockets) are electrically attached. The left picture shows the ways the pins have shared electrical connection and the right picture shows the actual internal metal, which connect these rows electrically.



## GPIO Extension Board

GPIO board is a convenient way to connect the RPi I/O ports to the breadboard directly. The GPIO pin sequence on Extension Board is identical to the GPIO pin sequence of RPi.



# Code

According to the circuit, when the GPIO17 of RPi output level is high, the LED turns ON. Conversely, when the GPIO17 RPi output level is low, the LED turns OFF. Therefore, we can let GPIO17 cycle output high and output low level to make the LED blink. We will use both C code to achieve the target.

## C Code 1.1.1 Blink

First, enter this command into the Terminal one line at a time. Then observe the results it brings on your project, and learn about the code in detail.

If you want to execute it with editor, please refer to section Code Editor to configure.

**If you have any concerns, please contact us via: support@freenove.com**

It is recommended that to execute the code via command line.

1. If you did not update wiring pi, please execute following commands **one by one**.

```
sudo apt-get update
git clone https://github.com/WiringPi/WiringPi
cd WiringPi
./build
```

2. Use cd command to enter 01.1.1_Blink directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/01.1.1_Blink
```

3. Use the following command to compile the code “Blink.c” and generate executable file “Blink”.

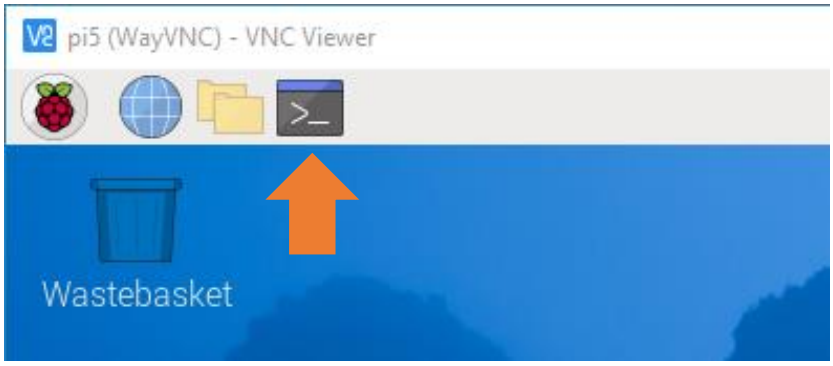
**“l” of “lwiringPi” is low case of “L”.**

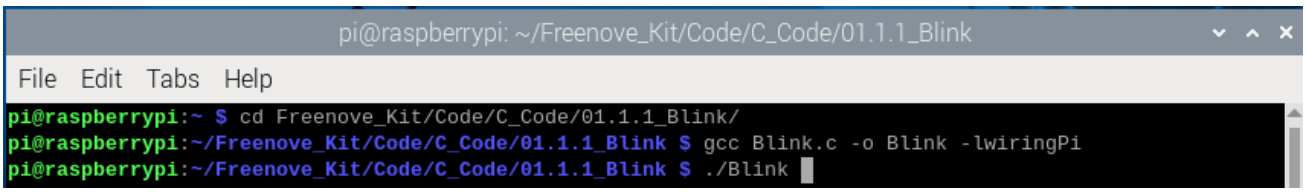
```
gcc Blink.c -o Blink -lwiringPi
```

4. Then run the generated file “blink”.

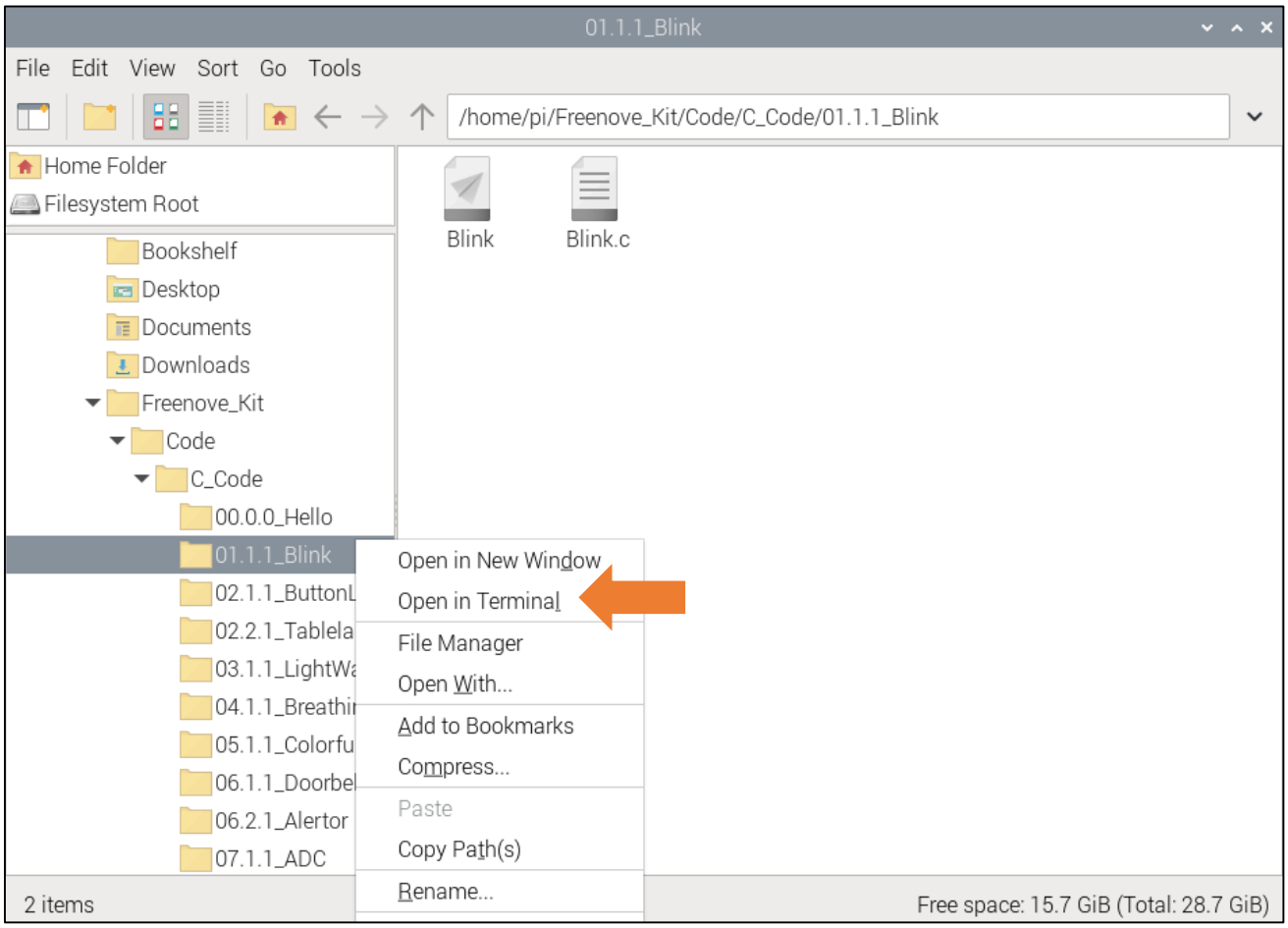
```
sudo ./Blink
```

Now your LED should start blinking! CONGRATUALTIONS! You have successfully completed your first RPi circuit!



You can also use the file browser. On the left of folder tree, right-click the folder you want to enter, and click "Open in Terminal".




You can press “Ctrl+C” to end the program. The following is the program code:

```
#include <wiringPi.h>
#include <stdio.h>

#define  ledPin    0  //define the led pin number

void main(void)
{
    printf("Program is starting ... \n");

    wiringPiSetup();  //Initialize wiringPi.

    pinMode(ledPin, OUTPUT);              //Set the pin mode
    printf("Using pin%d\n",%ledPin);      //Output information on terminal
    while(1){
        digitalWrite(ledPin, HIGH);       //Make GPIO output HIGH level
        printf("led turned on >>>\n");    //Output information on terminal
        delay(1000);                      //Wait for 1 second
```

```
18          digitalWrite(ledPin, LOW);          //Make GPIO output LOW level
19          printf("led turned off <<<\n");     //Output information on terminal
20          delay(1000);                        //Wait for 1 second
21      }
22  }
```

In the code above, the configuration function for GPIO is shown below as:

| **void pinMode(int pin, int mode);** |
|---|
| This sets the mode of a pin to either INPUT, OUTPUT, PWM_OUTPUT or GPIO_CLOCK. Note that only wiringPi pin 1 (BCM_GPIO 18) supports PWM output and only wiringPi pin 7 (BCM_GPIO 4) supports CLOCK output modes.<br>This function has no effect when in Sys mode. If you need to change the pin mode, then you can do it with the gpio program in a script before you start your program |
| **void digitalWrite (int pin, int value);** |
| Writes the value HIGH or LOW (1 or 0) to the given pin, which must have been previously set as an output. |

For more related wiringpi functions, please refer to https://github.com/WiringPi/WiringPi

GPIO connected to ledPin in the circuit is GPIO17 and GPIO17 is defined as 0 in the wiringPi numbering. So ledPin should be defined as 0 pin. You can refer to the corresponding table in Chapter 0.

```
    #define  ledPin    0 //define the led pin number
```

GPIO Numbering Relationship

| WingPi | BCM(Extension) | Physical | | BCM(Extension) | WingPi |
|---|---|---|---|---|---|
| 3.3V | 3.3V | 1 | 2 | 5V | 5V |
| 8 | GPIO2/SDA1 | 3 | 4 | 5V | 5V |
| 9 | GPIO3/SCL1 | 5 | 6 | GND | GND |
| 7 | GPIO4 | 7 | 8 | GPIO14/TXD0 | 15 |
| GND | GND | 9 | 10 | GPIO15/RXD0 | 16 |
| 0 | GPIO17 | 11 | 12 | GPIO18 | 1 |
| 2 | GPIO27 | 13 | 14 | GND | GND |
| 3 | GPIO22 | 15 | 16 | GPIO23 | 4 |
| 3.3V | 3.3V | 17 | 18 | GPIO24 | 5 |
| 12 | GPIO10/MOSI) | 19 | 20 | GND | GND |
| 13 | GPIO9/MOIS | 21 | 22 | GPIO25 | 6 |
| 14 | GPIO11/SCLK | 23 | 24 | GPIO8/CE0 | 10 |
| GND | GND | 25 | 26 | GPIO7/CE1 | 11 |
| 30 | GPIO0/SDA0 | 27 | 28 | GPIO1/SCL0 | 31 |
| 21 | GPIO5 | 29 | 30 | GND | GND |
| 22 | GPIO6 | 31 | 32 | GPIO12 | 26 |
| 23 | GPIO13 | 33 | 34 | GND | GND |
| 24 | GPIO19 | 35 | 36 | GPIO16 | 27 |
| 25 | GPIO26 | 37 | 38 | GPIO20 | 28 |
| GND | GND | 39 | 40 | GPIO21 | 29 |

In the main function main(), initialize wiringPi first.

```
    wiringPiSetup();  //Initialize wiringPi.
```

After the wiringPi is initialized successfully, you can set the ledPin to output mode and then enter the while loop, which is an endless loop (a while loop). That is, the program will always be executed in this cycle, unless it is ended because of external factors. In this loop, use digitalWrite (ledPin, HIGH) to make ledPin output high level, then LED turns ON. After a period of time delay, use digitalWrite(ledPin, LOW) to make ledPin output low level, then LED turns OFF, which is followed by a delay. Repeat the loop, then LED will start blinking.

```
    pinMode(ledPin, OUTPUT);//Set the pin mode
    printf("Using pin%d\n",%ledPin);    //Output information on terminal
    while(1){
        digitalWrite(ledPin, HIGH);  //Make GPIO output HIGH level
        printf("led turned on >>>\n");      //Output information on terminal
        delay(1000);                        //Wait for 1 second
        digitalWrite(ledPin, LOW);  //Make GPIO output LOW level
        printf("led turned off <<<\n");          //Output information on terminal
        delay(1000);                        //Wait for 1 second
    }
```

# Other Code Editors (Optional)

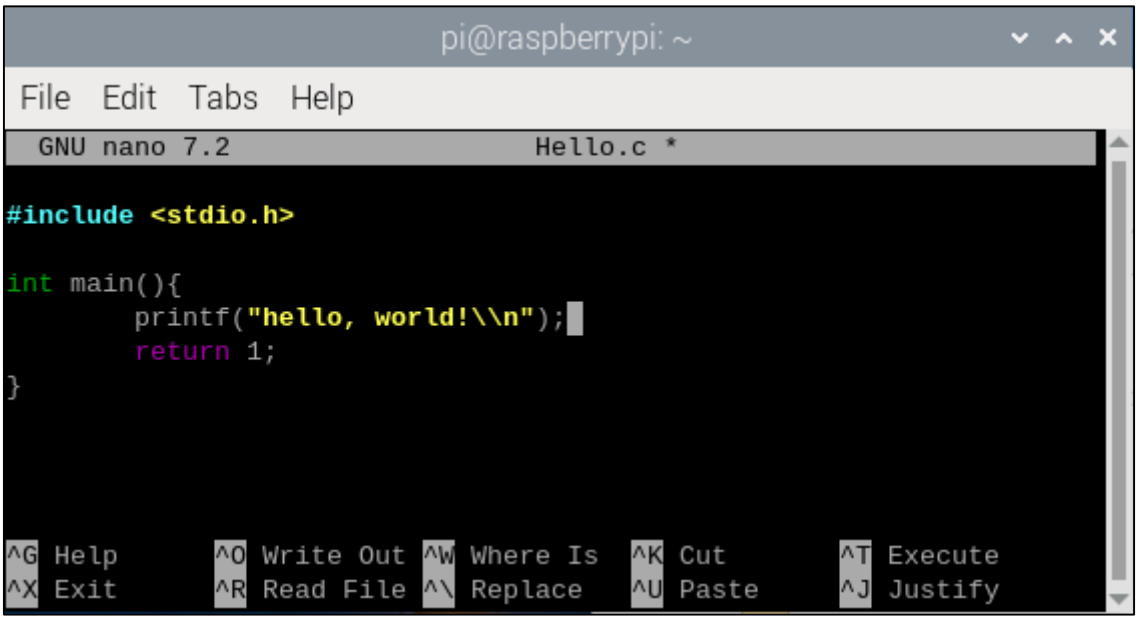
If you want to use other editor to edit and execute the code, you can learn them in this section.

## nano

Use the nano editor to open the file "Hello.c", then press " Ctrl+X " to exit.

```
nano Hello.c
```

As is shown below:



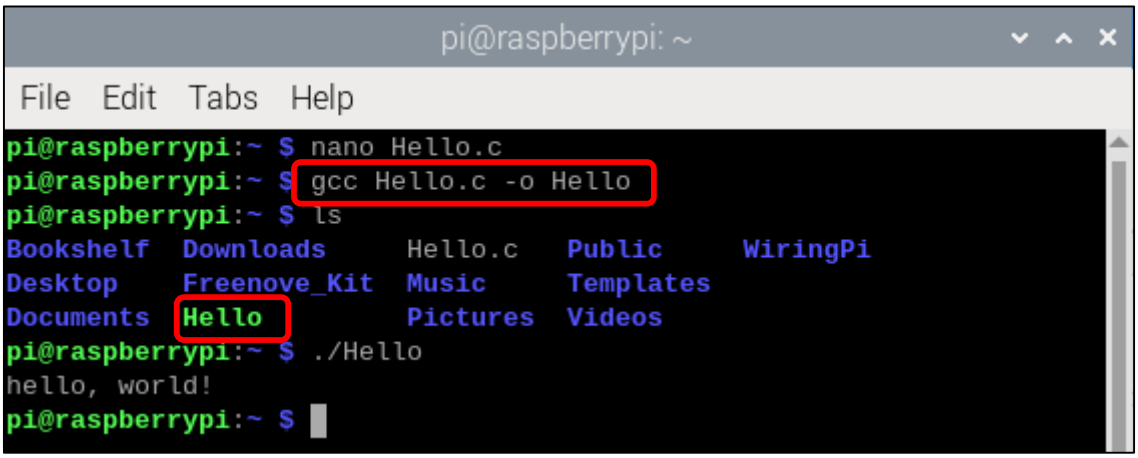

Use the following command to compile the code to generate the executable file “Hello”.

```
gcc Hello.c -o Hello
```

Use the following command to run the executable file “Hello”.

```
sudo ./Hello
```

After the execution, "Hello, World!" is printed out in terminal.

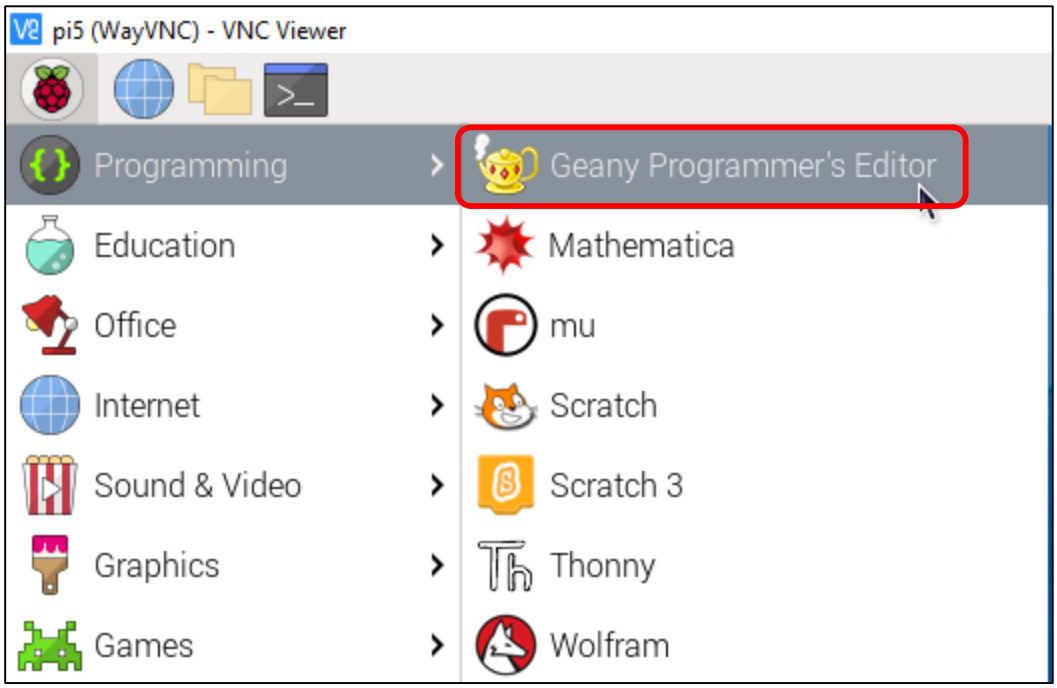


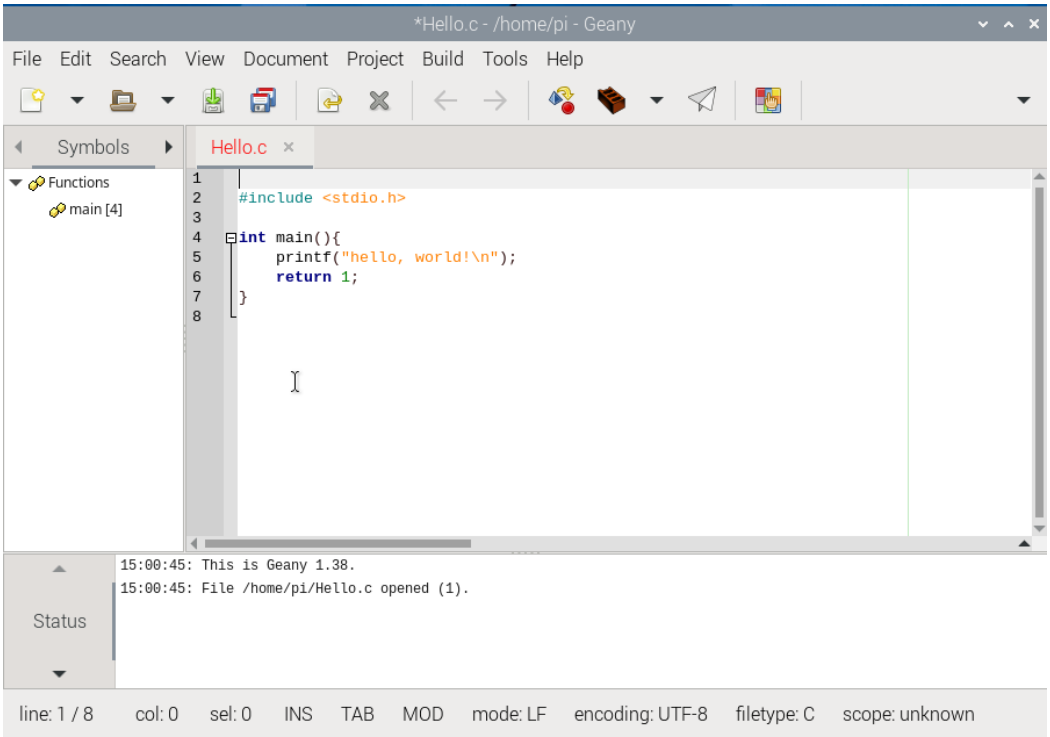
## geany

Next, learn to use the Geany editor. Use the following command to open the Geany in the sample file "Hello.c" file directory path.

```
geany Hello.c
```

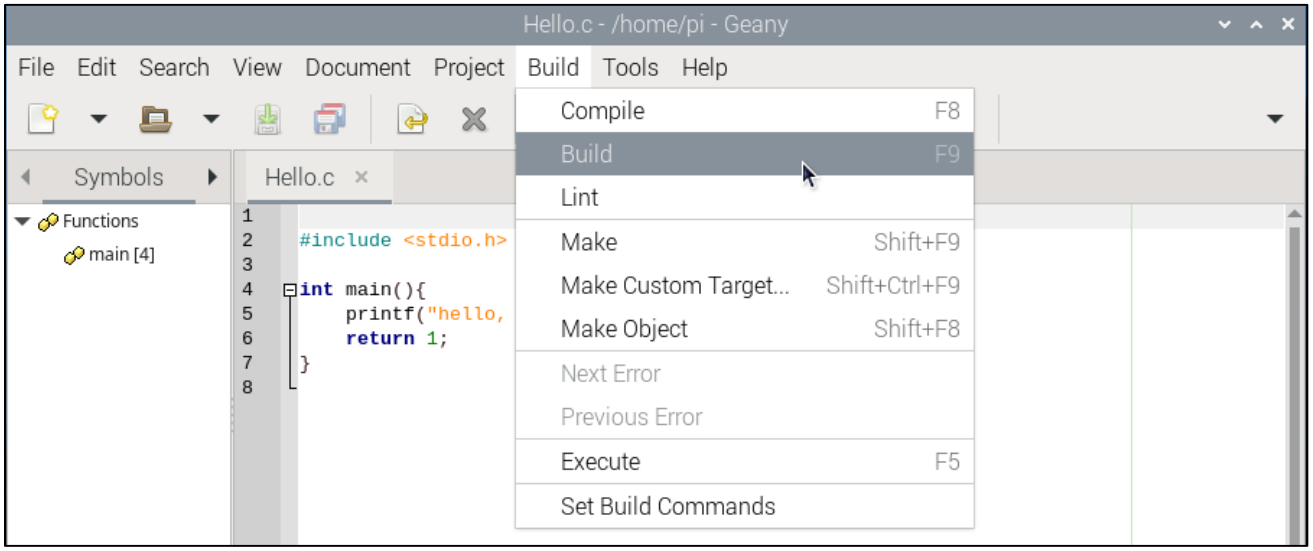
Or find and open Geany directly in the desktop main menu, and then click **File→Open** to open the "Hello.c", Or drag "Hello.c" to Geany directly.



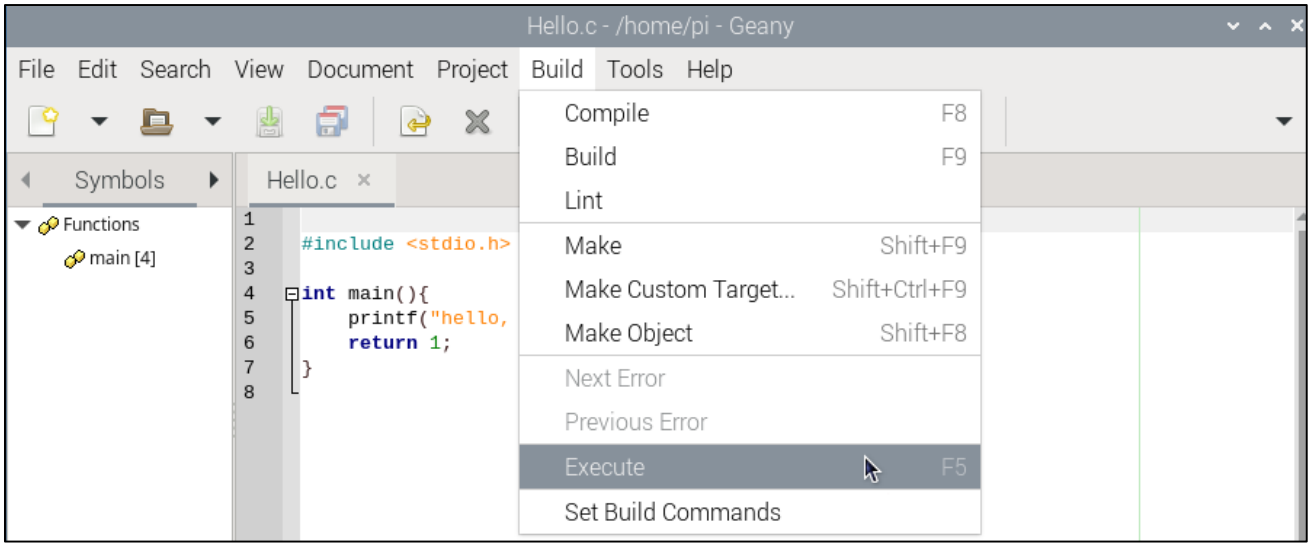




If you want to create a new code, click File→New→File→Save as (name.c or name.py). Then write the code.

Generate an executable file by clicking menu bar Build->Build.

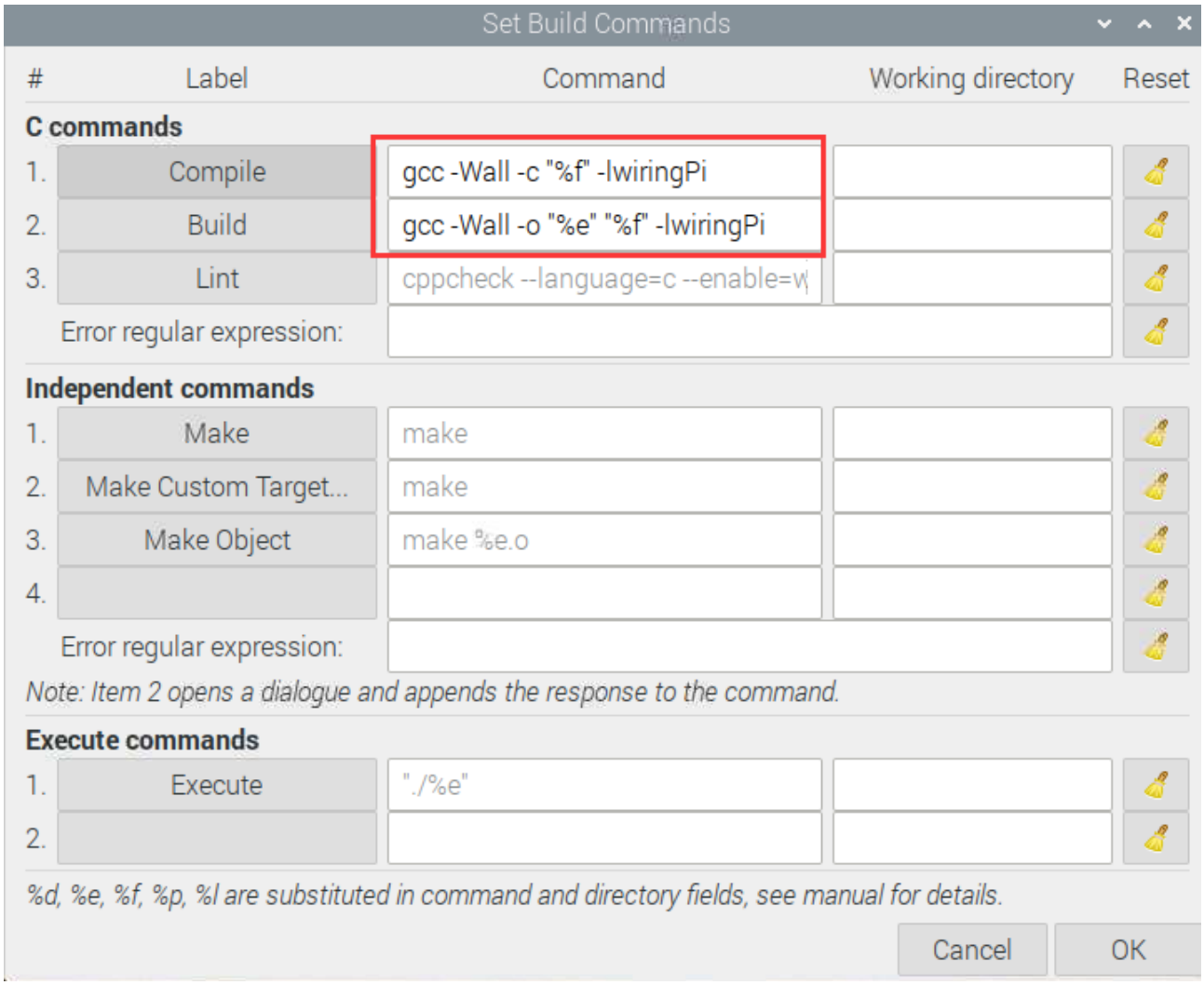



Then execute the generated file by clicking menu bar Build->Execute.




After the execution, a new terminal window will output the characters “Hello, World!”, as shown below:

You can click Build->Set Build Commands to set compiler commands. In later projects, we will use various compiler command options. If you choose to use Geany, you will need change the compiler command here. As is shown below:




Here we have identified three code editors: vi, nano and Geany. There are also many other good code editors available to you, and you can choose whichever you prefer to use.

In later projects, we will only use terminal to execute the project code. This way will not modify the code by mistake.

# Freenove Car, Robot and other products for Raspberry Pi

We also have car and robot kits for Raspberry Pi. You can visit our website for details.

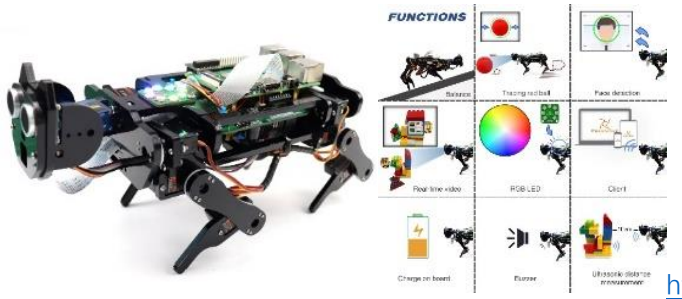
## https://www.amazon.com/freenove

**FNK0043** **Freenove 4WD Smart Car Kit for Raspberry Pi**



https://www.youtube.com/watch?v=4Zv0GZUQjZc

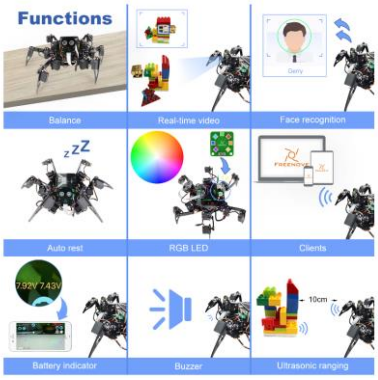
**FNK0050** **Freenove Robot Dog Kit for Raspberry Pi**




https://www.youtube.com/watch?v=7BmlZ8_R9d4

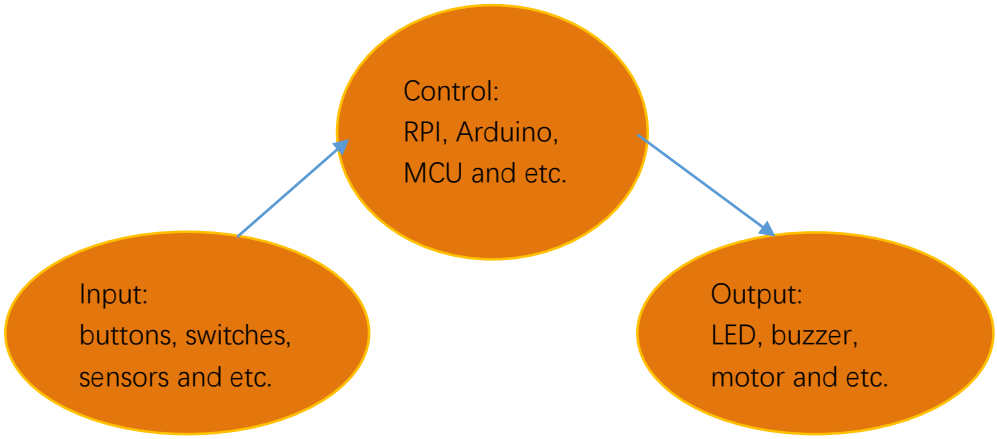
**FNK0052** **Freenove_Big_Hexapod_Robot_Kit_for_Raspberry_Pi**

https://youtu.be/LvghnJ2DNZ0

# Chapter 2 Buttons & LEDs

Usually, there are three essential parts in a complete automatic control device: INPUT, OUTPUT, and CONTROL. In last section, the LED module was the output part and RPI was the control part. In practical applications, we not only make LEDs flash, but also make a device sense the surrounding environment, receive instructions and then take the appropriate action such as turn on LEDs, make a buzzer beep and so on.




Next, we will build a simple control system to control an LED through a push button switch.

## Project 2.1 Push Button Switch & LED

In the project, we will control the LED state through a Push Button Switch. When the button is pressed, our LED will turn ON, and when it is released, the LED will turn OFF. This describes a Momentary Switch.

### Component List

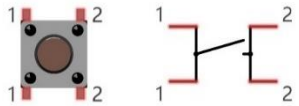
| | | | | |
|---|---|---|---|---|
| Raspberry Pi (with 40 GPIO) x1<br>GPIO Extension Board & Wire x1<br>Breadboard x1 | LED x1  | Resistor 220Ω x1  | Resistor 10kΩ x2  | Push Button Switch x1  |
| Jumper Wire  | | | | |

Please Note: In the code “button” represents switch action.

# Component knowledge

## Push Button Switch

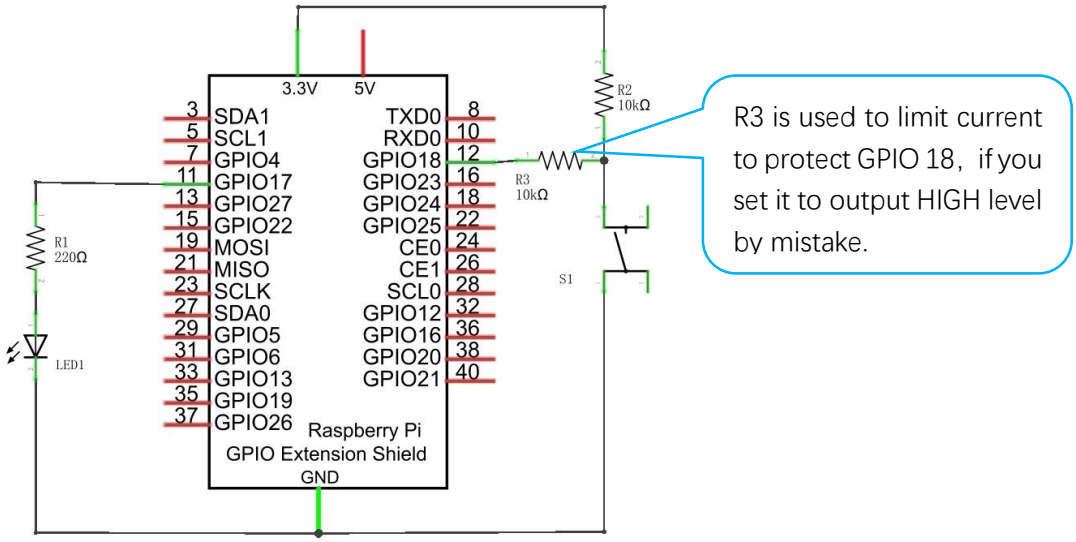
This type of Push Button Switch has 4 pins (2 Pole Switch). Two pins on the left are connected, and both left and right sides are the same per the illustration:



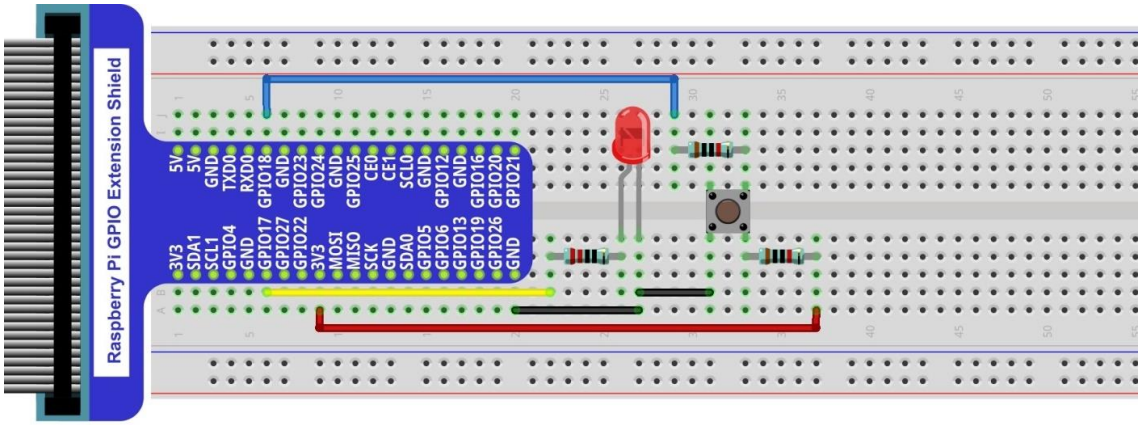

When the button on the switch is pressed, the circuit is completed (your project is Powered ON).

# Circuit

Schematic diagram




Hardware connection. If you need any support, please feel free to contact us via:support@freenove.com



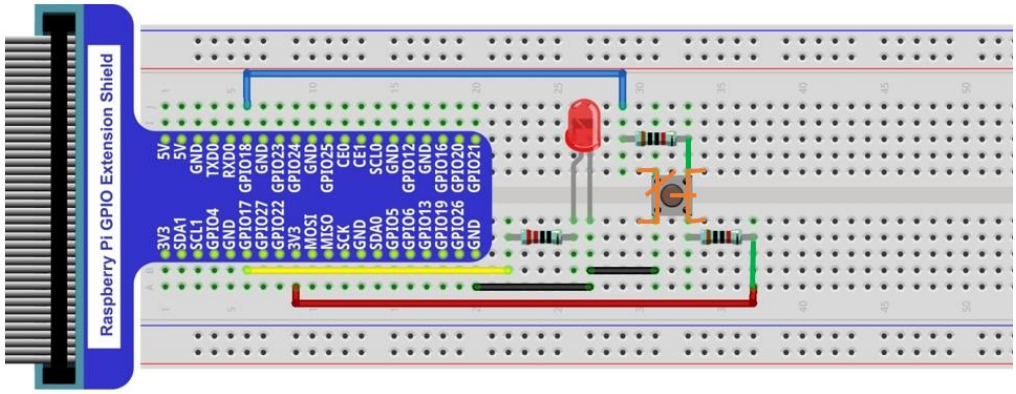

There are two kinds of push button switch in this kit.

**The smaller push button switches are contained in a plastic bag.**

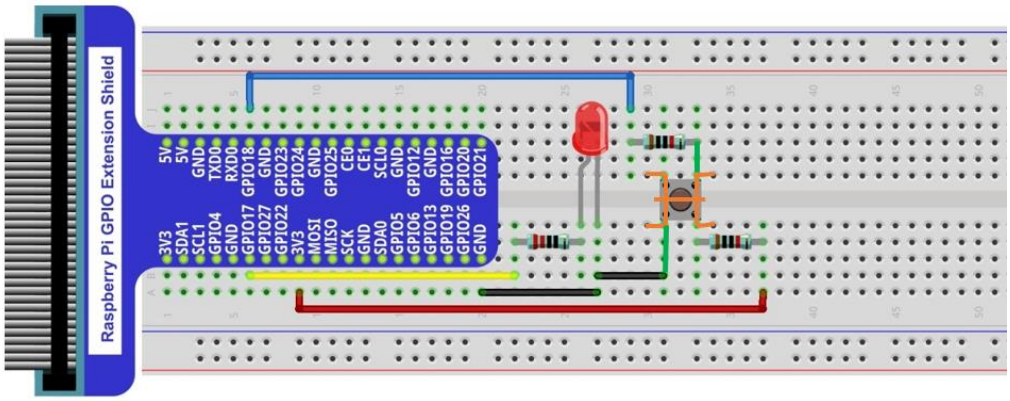
**Youtube video**: https://youtu.be/_5ge1d6f1nM

**This is how it works.**

**When button switch is released:**




**When button switch is pressed:**




# Code

This project is designed for learning how to use Push Button Switch to control an LED. We first need to read the state of switch, and then determine whether to turn the LED ON in accordance to the state of the switch.

## C Code 2.1.1 ButtonLED

First, observe the project result, then learn about the code in detail.

**If you have any concerns, please contact us via: support@freenove.com**

1. Use cd command to enter 02.1.1_ButtonLED directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/02.1.1_ButtonLED
```

2. Use the following command to compile the code “ButtonLED.c” and generate executable file “ButtonLED”

```
gcc ButtonLED.c -o ButtonLED -lwiringPi
```

3. Then run the generated file “ButtonLED”.

```
sudo ./ButtonLED
```

Later, the terminal window continues to print out the characters “led off…”. Press the button, then LED is turned on and then terminal window prints out the "led on…". Release the button, then LED is turned off and then terminal window prints out the "led off…". You can press "Ctrl+C" to terminate the program.

The following is the program code:

```
1  #include <wiringPi.h>
```

```
#include <stdio.h>

#define ledPin    0   //define the ledPin
#define buttonPin 1       //define the buttonPin

void  main(void)
{
    printf("Program is starting ... \n");

    wiringPiSetup();  //Initialize wiringPi.

    pinMode(ledPin, OUTPUT); //Set ledPin to output
    pinMode(buttonPin, INPUT);//Set buttonPin to input

    pullUpDnControl(buttonPin, PUD_UP);  //pull up to HIGH level
    while(1){
        if(digitalRead(buttonPin) == LOW){ //button is pressed
            digitalWrite(ledPin, HIGH);  //Make GPIO output HIGH level
            printf("Button is pressed, led turned on >>>\n");        //Output information on terminal
        }
        else {                              //button is released
            digitalWrite(ledPin, LOW);  //Make GPIO output LOW level
            printf("Button is released, led turned off <<<\n");      //Output information on terminal
        }
    }
}
```

In the circuit connection, LED and Button are connected with GPIO17 and GPIO18 respectively, which correspond to 0 and 1 respectively in wiringPI. So define ledPin and buttonPin as 0 and 1 respectively.

```
#define ledPin    0     //define the ledPin
#define buttonPin 1     //define the buttonPin
```

In the while loop of main function, use digitalRead(buttonPin) to determine the state of Button. When the button is pressed, the function returns low level, the result of "if" is true, and then turn on LED. Or, turn off LED.

```
if(digitalRead(buttonPin) == LOW){ //button has pressed down
            digitalWrite(ledPin, HIGH);   //led on
            printf("led on...\n");
        }
        else {              //button has released
            digitalWrite(ledPin, LOW);   //led off
            printf("...led off\n");
        }
```

Reference:

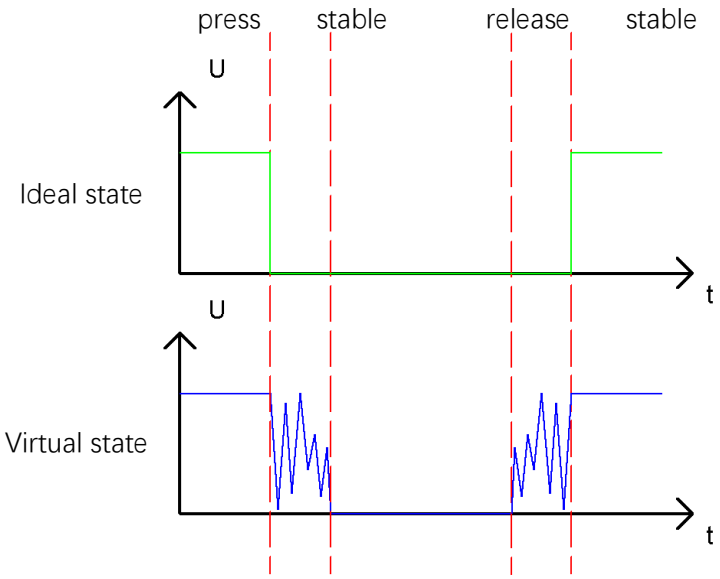
| `int digitalRead (int pin);` |
| --- |
| This function returns the value read at the given pin. It will be "**HIGH**" or "**LOW**"(1 or 0) depending on the logic level at the pin. |

# Project 2.2 MINI Table Lamp

We will also use a Push Button Switch, LED and RPi to make a MINI Table Lamp but this will function differently: Press the button, the LED will turn ON, and pressing the button again, the LED turns OFF. The ON switch action is no longer momentary (like a door bell) but remains ON without needing to continually press on the Button Switch.
First, let us learn something about the push button switch.

## Debounce a Push Button Switch

When a Momentary Push Button Switch is pressed, it will not change from one state to another state immediately. Due to tiny mechanical vibrations, there will be a short period of continuous buffeting before it stabilizes in a new state too fast for Humans to detect but not for computer microcontrollers. The same is true when the push button switch is released. This unwanted phenomenon is known as “bounce”.




Therefore, if we can directly detect the state of the Push Button Switch, there are multiple pressing and releasing actions in one pressing cycle. This buffeting will mislead the high-speed operation of the microcontroller to cause many false decisions. Therefore, we need to eliminate the impact of buffeting. Our solution: to judge the state of the button multiple times. Only when the button state is stable (consistent) over a period of time, can it indicate that the button is actually in the ON state (being pressed).
This project needs the same components and circuits as we used in the previous section.

# Code

In this project, we still detect the state of Push Button Switch to control an LED. Here we need to define a variable to define the state of LED. When the button switch is pressed once, the state of LED will be changed once. This will allow the circuit to act as a virtual table lamp.

## C Code 2.2.1 Tablelamp

First, observe the project result, and then learn about the code in detail.

**If you have any concerns, please contact us via: support@freenove.com**

1. Use cd command to enter 02.2.1_Tablelamp directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/02.2.1_Tablelamp
```

2. Use the following command to compile "Tablelamp.c" and generate executable file "Tablelamp".

```
gcc Tablelamp.c -o Tablelamp -lwiringPi
```

3. Tablelamp: Then run the generated file "Tablelamp".

```
sudo ./Tablelamp
```

When the program is executed, press the Button Switch once, the LED turns ON. Pressing the Button Switch again turns the LED OFF.

```
#include <wiringPi.h>
#include <stdio.h>

#define ledPin    0   //define the ledPin
#define buttonPin 1       //define the buttonPin
int ledState=LOW;     //store the State of led
int buttonState=HIGH; //store the State of button
int lastbuttonState=HIGH;//store the lastState of button
long lastChangeTime;  //store the change time of button state
long captureTime=50;  //set the stable time for button state
int reading;
int main(void)
{
    printf("Program is starting...\n");

    wiringPiSetup(); //Initialize wiringPi.

    pinMode(ledPin, OUTPUT);   //Set ledPin to output
    pinMode(buttonPin, INPUT); //Set buttonPin to input

    pullUpDnControl(buttonPin, PUD_UP);  //pull up to high level
    while(1){
        reading = digitalRead(buttonPin); //read the current state of button
        if( reading != lastbuttonState){  //if the button state has changed, record the time point
            lastChangeTime = millis();
        }
```

```
27          //if changing-state of the button last beyond the time we set, we consider that
28          //the current button state is an effective change rather than a buffeting
29          if(millis() - lastChangeTime > captureTime){
30              //if button state is changed, update the data.
31              if(reading != buttonState){
32                  buttonState = reading;
33                  //if the state is low, it means the action is pressing
34                  if(buttonState == LOW){
35                      printf("Button is pressed!\n");
36                      ledState = !ledState; //Reverse the LED state
37                      if(ledState){
38                          printf("turn on LED ...\n");
39                      }
40                      else {
41                          printf("turn off LED ...\n");
42                      }
43                  }
44                  //if the state is high, it means the action is releasing
45                  else {
46                      printf("Button is released!\n");
47                  }
48              }
49          }
50          digitalWrite(ledPin,ledState);
51          lastbuttonState = reading;
52      }
53
54      return 0;
55  }
```

This code focuses on eliminating the buffeting (bounce) of the button switch. We define several variables to define the state of LED and button switch. Then read the button switch state constantly in while () to determine whether the state has changed. If it has, then this time point is recorded.

```
reading = digitalRead(buttonPin); //read the current state of button
        if( reading != lastbuttonState){
            lastChangeTime = millis();
        }
```

**millis()**

This returns a number representing the number of milliseconds since your program called one of the wiringPiSetup functions. It returns to an unsigned 32-bit number value after 49 days because it “wraps” around and restarts to value 0.

Then according to the recorded time point, evaluate the duration of the button switch state change. If the duration exceeds captureTime (buffeting time) we have set, it indicates that the state of the button switch has changed. During that time, the while () is still detecting the state of the button switch, so if there is a change, the time point of change will be updated. Then the duration will be evaluated again until the duration is determined to be a stable state because it exceeds the time value we set.

```
        if(millis() - lastChangeTime > captureTime){
            //if button state is changed,update the data.
            if(reading != buttonState){
                buttonState = reading;
```

Finally, we need to judge the state of Button Switch. If it is low level, the changing state indicates that the button Switch has been pressed, if the state is high level, then the button has been released. Here, we change the status of the LED variable, and then update the state of the LED.

```
                if(buttonState == LOW){
                    printf("Button is pressed!\n");
                    ledState = !ledState; //Reverse the LED state
                    if(ledState){
                        printf("turn on LED ...\n");
                    }
                    else {
                        printf("turn off LED ...\n");
                    }
                }
                //if the state is high, it means the action is releasing
                else {
                    printf("Button is released!\n");
                }
```

# Chapter 3 LED Bar Graph

We have learned how to control one LED to blink. Next, we will learn how to control a number of LEDs.

## Project 3.1 Flowing Water Light

In this project, we use a number of LEDs to make a flowing water light.

### Component List

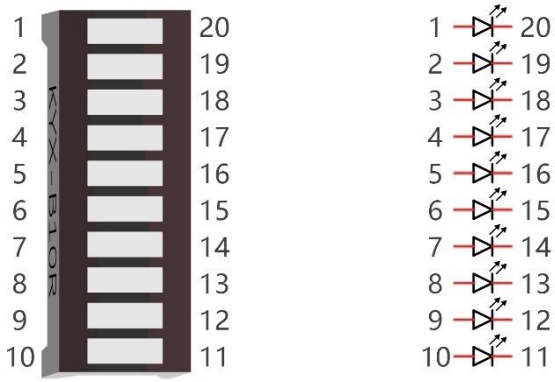
| | | |
|---|---|---|
| Raspberry Pi (with 40 GPIO) x1<br>GPIO Extension Board & Ribbon Cable x1<br>Breadboard x1 | Bar Graph LED x1  | Resistor 220Ω x10  |
| Jumper Wire x 1  | | |

### Component knowledge

Let us learn about the basic features of these components to use and understand them better.

#### Bar Graph LED

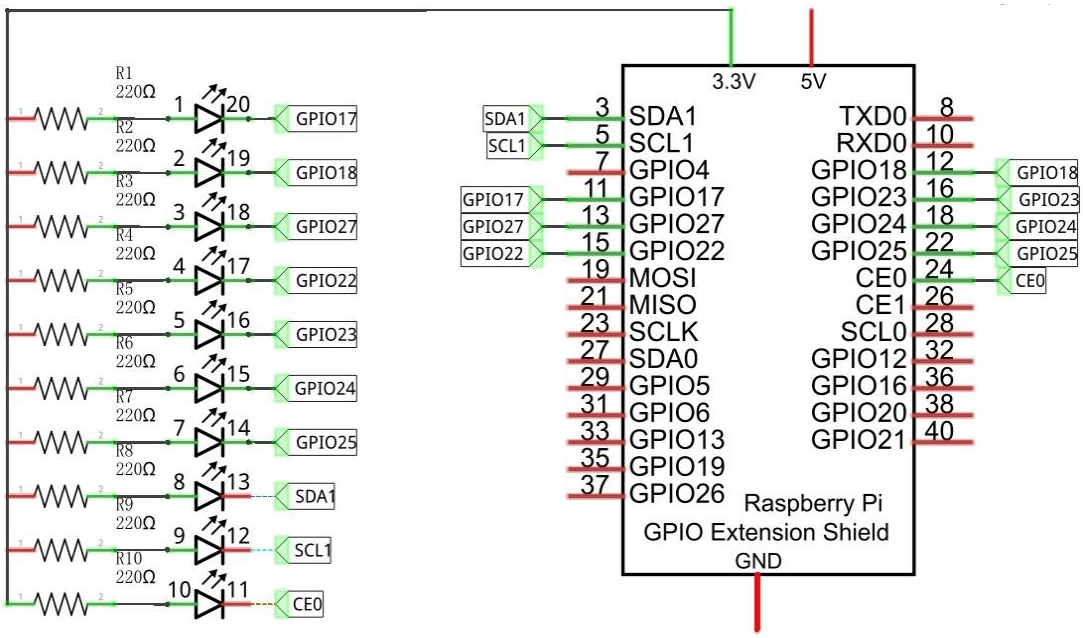
A Bar Graph LED has 10 LEDs integrated into one compact component. The two rows of pins at its bottom are paired to identify each LED like the single LED used earlier.



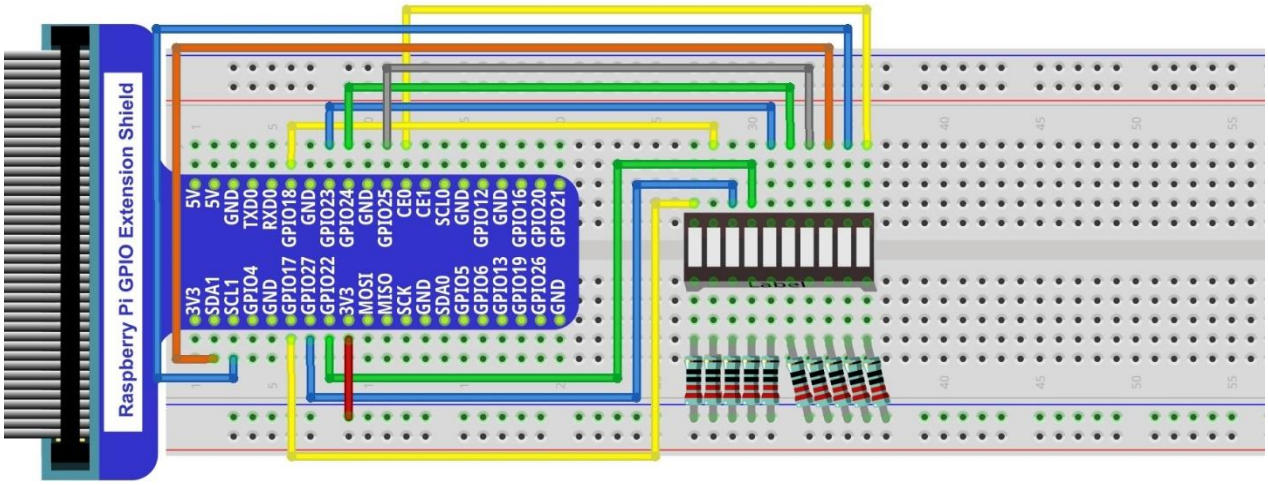
# Circuit

A reference system of labels is used in the circuit diagram below. Pins with the same network label are connected together.

Schematic diagram




Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com




**If LEDbar doesn't work, rotate LEDbar 180° to try. The label is random.**

**Youtube video: https://youtu.be/3rh-b05VoiU**

In this circuit, the cathodes of the LEDs are connected to the GPIO, which is different from the previous circuit. The LEDs turn ON when the GPIO output is low level in the program.

# Code

This project is designed to make a flowing water lamp, which are these actions: First turn LED #1 ON, then

turn it OFF. Then turn LED #2 ON, and then turn it OFF... and repeat the same to all 10 LEDs until the last LED is turns OFF. This process is repeated to achieve the "movements" of flowing water.

## C Code 3.1.1 LightWater

First, observe the project result, and then learn about the code in detail.

**If you have any concerns, please contact us via: support@freenove.com**

1. Use cd command to enter 03.1.1_LightWater directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/03.1.1_LightWater
```

2. Use the following command to compile "LightWater.c" and generate executable file "LightWater".

```
gcc LightWater.c -o LightWater -lwiringPi
```

3. Then run the generated file "LightWater".

```
sudo ./LightWater
```

After the program is executed, you will see that Bar Graph LED starts with the flowing water pattern flashing from left to right and then back from right to left.

The following is the program code:

```
#include <wiringPi.h>
#include <stdio.h>

#define ledCounts 10
int pins[ledCounts] = {0,1,2,3,4,5,6,8,9,10};

void main(void)
{
    int i;
    printf("Program is starting ... \n");

    wiringPiSetup(); //Initialize wiringPi.

    for(i=0;i<ledCounts;i++){       //Set pinMode for all led pins to output
        pinMode(pins[i], OUTPUT);
    }
    while(1){
        for(i=0;i<ledCounts;i++){   // move led(on) from left to right
            digitalWrite(pins[i],LOW);
            delay(100);
            digitalWrite(pins[i],HIGH);
        }
        for(i=ledCounts-1;i>-1;i--){   // move led(on) from right to left
            digitalWrite(pins[i],LOW);
            delay(100);
            digitalWrite(pins[i],HIGH);
        }
    }
}
```

In the program, configure the GPIO0-GPIO9 to output mode. Then, in the endless “while” loop of main function, use two “for” loop to realize flowing water light from left to right and from right to left.

```
    while(1){
        for(i=0;i<ledCounts;i++){   // move led(on) from left to right
            digitalWrite(pins[i],LOW);
            delay(100);
            digitalWrite(pins[i],HIGH);
        }
        for(i=ledCounts-1;i>-1;i--){   // move led(on) from right to left
            digitalWrite(pins[i],LOW);
            delay(100);
            digitalWrite(pins[i],HIGH);
        }
    }
}
```

# Chapter 4 Analog & PWM

In previous chapters, we learned that a Push Button Switch has two states: Pressed (ON) and Released (OFF), and an LED has a Light ON and OFF state. Is there a middle or intermediated state? We will next learn how to create an intermediate output state to achieve a partially bright (dim) LED.
First, let us learn how to control the brightness of an LED.

## Project 4.1 Breathing LED

We describe this project as a Breathing Light. This means that an LED that is OFF will then turn ON gradually and then gradually turn OFF like "breathing". Okay, so how do we control the brightness of an LED to create a Breathing Light? We will use PWM to achieve this goal.

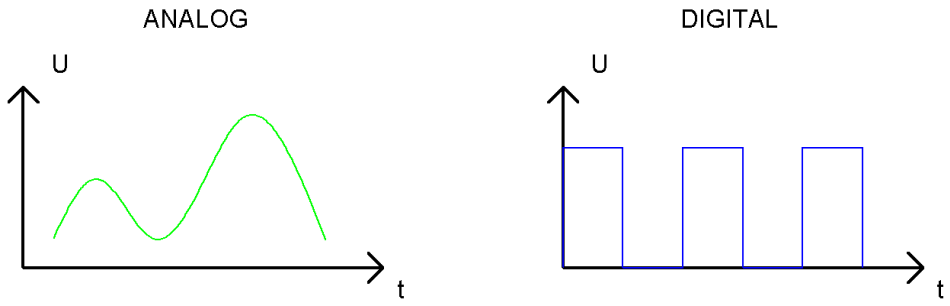
### Component List

| | | |
|---|---|---|
| Raspberry Pi (with 40 GPIO) x1<br>GPIO Extension Board & Ribbon Cable x1<br>Breadboard x1 | LED x1  | Resistor 220Ω x1  |
| Jumper Wire  | | |

### Component Knowledge

#### Analog & Digital

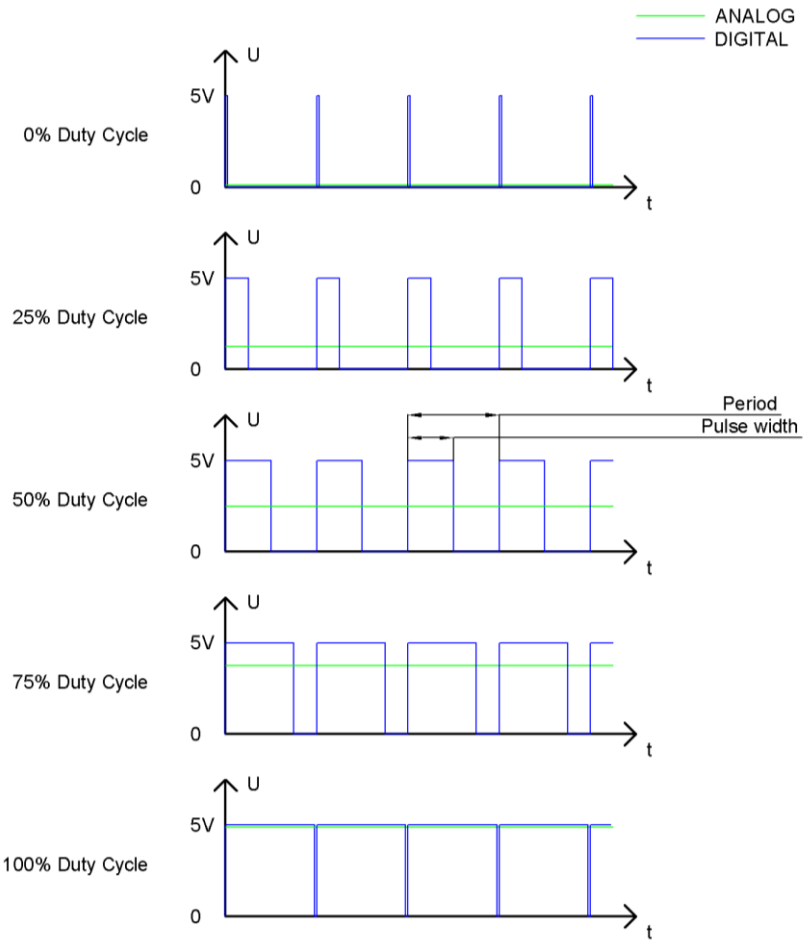
An Analog Signal is a continuous signal in both time and value. On the contrary, a Digital Signal **or discrete-time signal is a time series consisting of a sequence of quantities**. Most signals in life are analog signals. A familiar example of an Analog Signal would be how the temperature throughout the day is continuously changing and could not suddenly change instantaneously from 0℃ to 10℃. However, Digital Signals can instantaneously change in value. This change is expressed in numbers as 1 and 0 (the basis of binary code). Their differences can more easily be seen when compared when graphed as below.




Note that the Analog signals are curved waves and the Digital signals are “Square Waves”.
In practical applications, we often use binary as the digital signal, that is a series of 0’s and 1’s. Since a binary signal only has two values (0 or 1) it has great stability and reliability. Lastly, both analog and digital signals can be converted into the other.

## PWM

PWM, Pulse-Width Modulation, is a very effective method for using digital signals to control analog circuits. Digital processors cannot directly output analog signals. PWM technology makes it very convenient to achieve this conversion (translation of digital to analog signals).
PWM technology uses digital pins to send certain frequencies of square waves, that is, the output of high levels and low levels, which alternately last for a while. The total time for each set of high levels and low levels is generally fixed, which is called the period (Note: the reciprocal of the period is frequency). The time of high level outputs are generally called “pulse width”, and the duty cycle is the percentage of the ratio of pulse duration, or pulse width (PW) to the total period (T) of the waveform. The longer the output of high levels last, the longer the duty cycle and the higher the corresponding voltage in the analog signal will be. The following figures show how the analog signal voltages vary between 0V-5V (high level is 5V) corresponding to the pulse width 0%-100%:




The longer the PWM duty cycle is, the higher the output power will be. Now that we understand this relationship, we can use PWM to control the brightness of an LED or the speed of DC motor and so on.

It is evident, from the above, that PWM is not actually analog but the effective value of voltage is equivalent to the corresponding analog value. Therefore, by using PWM, we can control the output power of to an LED and control other devices and modules to achieve multiple effects and actions.

In RPi, GPIO18 pin has the ability to output to hardware via PWM with a 10-bit accuracy. This means that 100% of the pulse width can be divided into $2^{10}$=1024 equal parts.

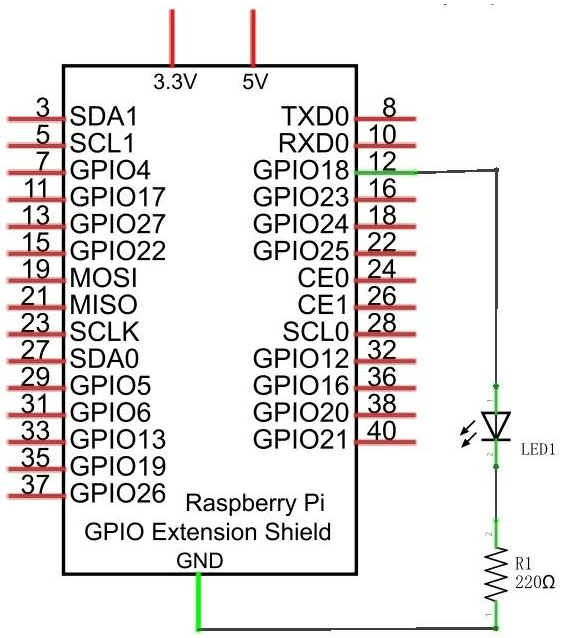
The wiringPi library of C provides both a hardware PWM and a software PWM method.

The hardware PWM only needs to be configured, does not require CPU resources and is more precise in time control. The software PWM requires the CPU to work continuously by using code to output high level and low level. This part of the code is carried out by multi-threading, and the accuracy is relatively not high enough.

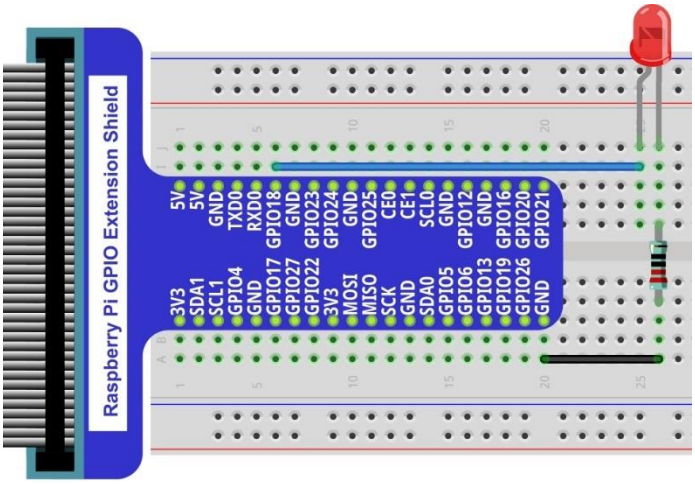
In order to keep the results running consistently, we will use PWM.

## Circuit

Schematic diagram




Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com




**Youtube video: https://youtu.be/rYxykuVgYtA**

## Code

This project uses the PWM output from the GPIO18 pin to make the pulse width gradually increase from 0% to 100% and then gradually decrease from 100% to 0% to make the LED glow brighter then dimmer.

### C Code 4.1.1 BreathingLED

First, observe the project result, and then learn about the code in detail.

**If you have any concerns, please contact us via: support@freenove.com**

1. Use cd command to enter 04.1.1_BreathingLED directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/04.1.1_BreathingLED
```

2. Use following command to compile "BreathingLED.c" and generate executable file "BreathingLED".

```
gcc BreathingLED.c -o BreathingLED -lwiringPi
```

3. Then run the generated file "BreathingLED"

```
sudo ./BreathingLED
```

After the program is executed, you'll see that LED is turned from on to off and then from off to on gradually like breathing.

The following is the program code:

```
1  #include <wiringPi.h>
2  #include <stdio.h>
3  #include <softPwm.h>
4  #define ledPin    1
5  void main(void)
6  {
```

```
7       int i;
8
9       printf("Program is starting ... \n");
10
11      wiringPiSetup();  //Initialize wiringPi.
12
13      softPwmCreate(ledPin,  0, 100);//Creat SoftPWM pin
14
15      while(1){
16          for(i=0;i<100;i++){  //make the led brighter
17              softPwmWrite(ledPin, i);
18              delay(20);
19          }
20          delay(300);
21          for(i=100;i>=0;i--){  //make the led darker
22              softPwmWrite(ledPin, i);
23              delay(20);
24          }
25          delay(300);
26      }
27  }
```

First, create a software PWM pin.

```
    softPwmCreate(ledPin,  0, 100);//Creat SoftPWM pin
```

There are two “for” loops in the next endless “while” loop. The first loop outputs a power signal to the ledPin PWM from 0% to 100% and the second loop outputs a power signal to the ledPin PWM from 100% to 0%.

```
    while(1){
        for(i=0;i<100;i++){
            softPwmWrite(ledPin, i);
            delay(20);
        }
        delay(300);
        for(i=100;i>=0;i--){
            softPwmWrite(ledPin, i);
            delay(20);
        }
        delay(300);
    }
```

You can also adjust the rate of the state change of LED by changing the parameter of the delay() function in the “for” loop.

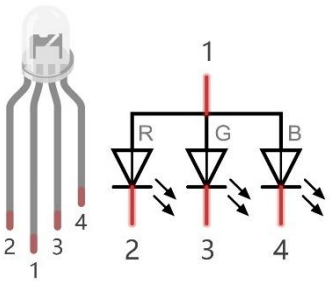
| **int softPwmCreate (int pin, int initialValue, int pwmRange) ;** |
| --- |
| This creates a software controlled PWM pin. |
| **void softPwmWrite (int pin, int value) ;** |
| This updates the PWM value on the given pin. |

For more details, please refer https://github.com/WiringPi/WiringPi

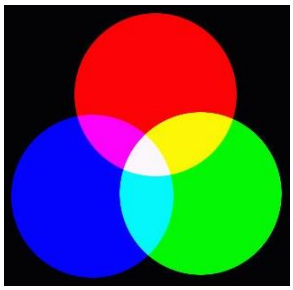
# Chapter 5 RGB LED

In this chapter, we will learn how to control a RGB LED.

An RGB LED has 3 LEDs integrated into one LED component. It can respectively emit Red, Green and Blue light. In order to do this, it requires 4 pins (this is also how you identify it). The long pin (1) is the common which is the Anode (+) or positive lead, the other 3 are the Cathodes (-) or negative leads. A rendering of a RGB LED and its electronic symbol are shown below. We can make RGB LED emit various colors of light and brightness by controlling the 3 Cathodes (2, 3 & 4) of the RGB LED




Red, Green, and Blue light are called 3 Primary Colors when discussing light (Note: for pigments such as paints, the 3 Primary Colors are Red, Blue and Yellow). When you combine these three Primary Colors of light with varied brightness, they can produce almost any color of visible light. Computer screens, single pixels of cell phone screens, neon lamps, etc. can all produce millions of colors due to phenomenon.



RGB

If we use a three 8 bit PWM to control the RGB LED, in theory, we can create $2^8*2^8*2^8$=16777216 (16 million) colors through different combinations of RGB light brightness.

Next, we will use RGB LED to make a multicolored LED.

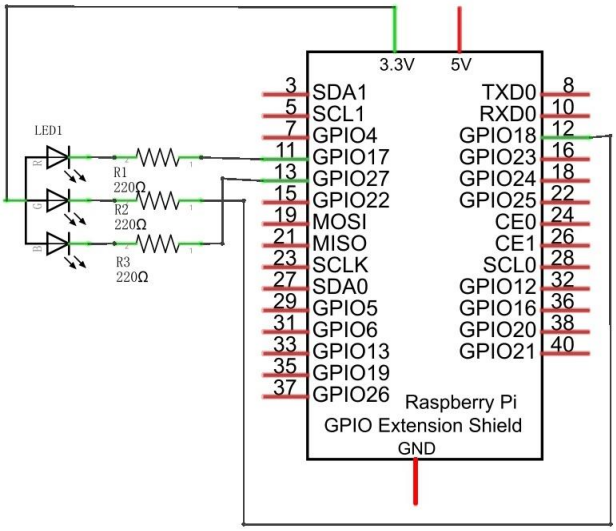
# Project 5.1 Multicolored LED

## Component List

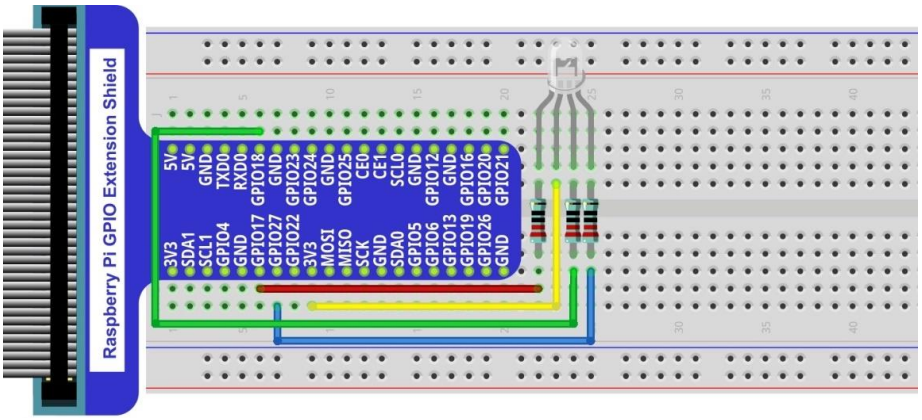
| | | |
|---|---|---|
| Raspberry Pi (with 40 GPIO) x1<br>GPIO Extension Board & Wire x1<br>Breadboard x1 | RGB LED x1  | Resistor 220Ω x3  |
| Jumper Wire  | | |

## Circuit

Schematic diagram




Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



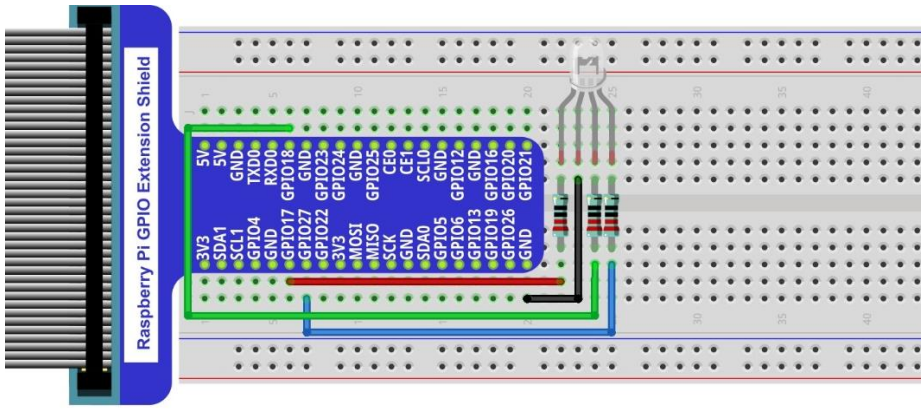

Video: https://youtu.be/tbnX2AsX2y4

In this kit, the RGB led is **Common anode**. The **voltage difference** between LED will make it work. There is

no visible GND. The GPIO ports can also receive current while in output mode.

**If circuit above doesn't work, the RGB LED may be common cathode. Please try following wiring.**

There is no need to modify code for random color.




# Code

We need to use the software to make the ordinary GPIO output PWM, since this project requires 3 PWM and in RPi only one GPIO has the hardware capability to output PWM,

## C Code 5.1.1 Colorful LED

First, observe the project result, and then learn about the code in detail.

**If you have any concerns, please contact us via: support@freenove.com**

1. Use cd command to enter 05.1.1_ColorfulLED directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/05.1.1_ColorfulLED
```

2. Use following command to compile "ColorfulLED.c" and generate executable file "ColorfulLED".
   Note: in this project, the software PWM uses a multi-threading mechanism. So "-lpthread" option need to be add to the compiler.

```
gcc ColorfulLED.c -o ColorfulLED -lwiringPi -lpthread
```

3. And then run the generated file "ColorfulLED".

```
sudo ./ColorfulLED
```

After the program is executed, you will see that the RGB LED shows lights of different colors randomly.

The following is the program code:

```
1  #include <wiringPi.h>
2  #include <softPwm.h>
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  #define ledPinRed    0
7  #define ledPinGreen  1
8  #define ledPinBlue   2
9
10 void setupLedPin(void)
11 {
```

```
12      softPwmCreate(ledPinRed,   0,  100); //Creat SoftPWM pin for red
13      softPwmCreate(ledPinGreen,0,  100);  //Creat SoftPWM pin for green
14      softPwmCreate(ledPinBlue,  0,  100);  //Creat SoftPWM pin for blue
15  }
16
17  void setLedColor(int r, int g, int b)
18  {
19      softPwmWrite(ledPinRed,    r);  //Set the duty cycle
20      softPwmWrite(ledPinGreen,  g);   //Set the duty cycle
21      softPwmWrite(ledPinBlue,   b);   //Set the duty cycle
22  }
23
24  int main(void)
25  {
26      int r,g,b;
27
28      printf("Program is starting ...\n");
29
30      wiringPiSetup();  //Initialize wiringPi.
31
32      setupLedPin();
33      while(1){
34          r=random()%100;  //get a random in (0,100)
35          g=random()%100;  //get a random in (0,100)
36          b=random()%100;  //get a random in (0,100)
37          setLedColor(r,g,b);//set random as the duty cycle value
38  // If you are using common anode RGBLED,it should be setLedColor(100-r,100-g,100-b)
39  // If you want show red,  it should be setLedColor(0,100,100)
40          printf("r=%d,   g=%d,   b=%d \n",r,g,b);
41          delay(1000);
42      }
43      return 0;
44  }
```

First, in subfunction of ledInit(), create the software PWM control pins used to control the R, G, B pin respectively.

```
    void setupLedPin(void)
    {
        softPwmCreate(ledPinRed,   0,  100); //Creat SoftPWM pin for red
        softPwmCreate(ledPinGreen,0,  100);  //Creat SoftPWM pin for green
        softPwmCreate(ledPinBlue,  0,  100);  //Creat SoftPWM pin for blue
    }
```

Then create subfunction, and set the PWM of three pins.

```
    void setLedColor(int r, int g, int b)
```

```
{
    softPwmWrite(ledPinRed,   r);  //Set the duty cycle
    softPwmWrite(ledPinGreen, g);   //Set the duty cycle
    softPwmWrite(ledPinBlue,  b);   //Set the duty cycle
}
```

Finally, in the “while” loop of main function, get three random numbers and specify them as the PWM duty cycle, which will be assigned to the corresponding pins. So RGB LED can switch the color randomly all the time.

```
    while(1){
        r=random()%100;  //get a random in (0,100)
        g=random()%100;  //get a random in (0,100)
        b=random()%100;  //get a random in (0,100)
        setLedColor(r,g,b);//set random as the duty cycle value
        printf("r=%d,   g=%d,   b=%d \n",r,g,b);
        delay(1000);
    }
```

The related function of PWM Software can be described as follows:

**long random();**

This function will return a random number.

For more details about Software PWM, please refer to: https://github.com/WiringPi/WiringPi

# Chapter 6 Buzzer

In this chapter, we will learn about buzzers and the sounds they make. And in our next project, we will use an active buzzer to make a doorbell and a passive buzzer to make an alarm.

## Project 6.1 Doorbell

We will make a doorbell with this functionality: when the Push Button Switch is pressed the buzzer sounds and when the button is released, the buzzer stops. This is a momentary switch function.

### Component List

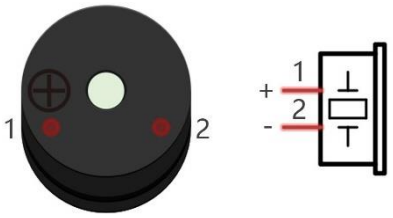
<table>
<tr><td colspan="2">Raspberry Pi (with 40 GPIO) x1<br>GPIO Extension Board & Ribbon Cable x1<br>Breadboard x1</td><td colspan="3">Jumper Wire<br>  </td></tr>
<tr><td>NPN transistorx1<br>(S8050)<br>  </td><td>Active buzzer x1<br>  </td><td>Push Button Switch x1<br>  </td><td>Resistor 1kΩ x1<br>  </td><td>Resistor 10kΩ x2<br>  </td></tr>
</table>

# Component knowledge

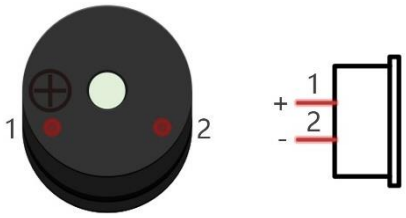
## Buzzer

A buzzer is an audio component. They are widely used in electronic devices such as calculators, electronic alarm clocks, automobile fault indicators, etc. There are both active and passive types of buzzers. Active buzzers have oscillator inside, these will sound as long as power is supplied. Passive buzzers require an external oscillator signal (generally using PWM with different frequencies) to make a sound.

Active buzzer




Passive buzzer




Active buzzers are easier to use. Generally, they only make a specific sound frequency. Passive buzzers require an external circuit to make sounds, but passive buzzers can be controlled to make sounds of various frequencies. The resonant frequency of the passive buzzer in this Kit is 2kHz, which means the passive buzzer is the loudest when its resonant frequency is 2kHz.

### How to identify active and passive buzzer?

1. As a rule, there is a label on an active buzzer covering the hole where sound is emitted, but there are exceptions to this rule.
2. Active buzzers are more complex than passive buzzers in their manufacture. There are many circuits and crystal oscillator elements inside active buzzers; all of this is usually protected with a waterproof coating (and a housing) exposing only its pins from the underside. On the other hand, passive buzzers do not have protective coatings on their underside. From the pin holes, view of a passive buzzer, you can see the circuit board, coils, and a permanent magnet (all or any combination of these components depending on the model.

Active buzzer bottom

Passive buzzer bottom

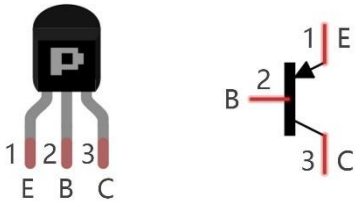
## Transistors

A transistor is required in this project due to the buzzer's current being so great that GPIO of RPi's output capability cannot meet the power requirement necessary for operation. A NPN transistor is needed here to

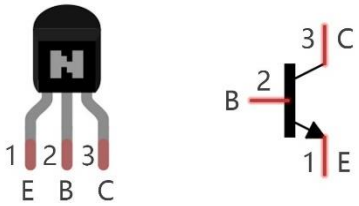
amplify the current.

Transistors, full name: semiconductor transistor, is a semiconductor device that controls current (think of a transistor as an electronic “amplifying or switching device”. Transistors can be used to amplify weak signals, or to work as a switch. Transistors have three electrodes (PINs): base (b), collector (c) and emitter (e). When there is current passing between "be" then "ce" will have a several-fold current increase (transistor magnification), in this configuration the transistor acts as an amplifier. When current produced by "be" exceeds a certain value, "ce" will limit the current output. at this point the transistor is working in its saturation region and acts like a switch. Transistors are available as two types as shown below: PNP and NPN,

PNP transistor




NPN transistor



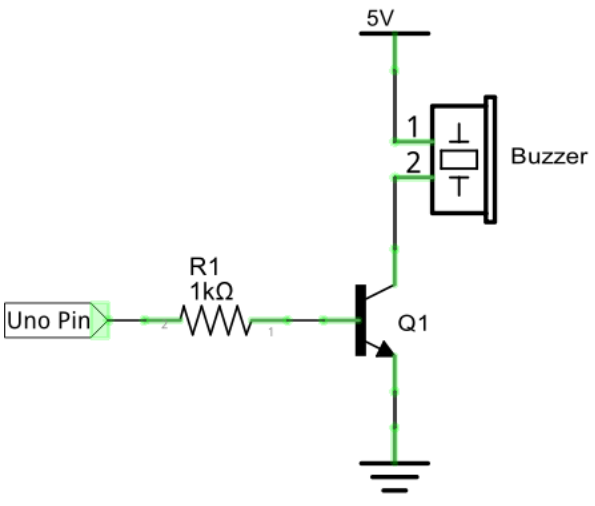

In our kit, the PNP transistor is marked with 8550, and the NPN transistor is marked with 8050.

Thanks to the transistor's characteristics, they are often used as switches in digital circuits. As micro-controllers output current capacity is very weak, we will use a transistor to amplify its current in order to drive components requiring higher current.

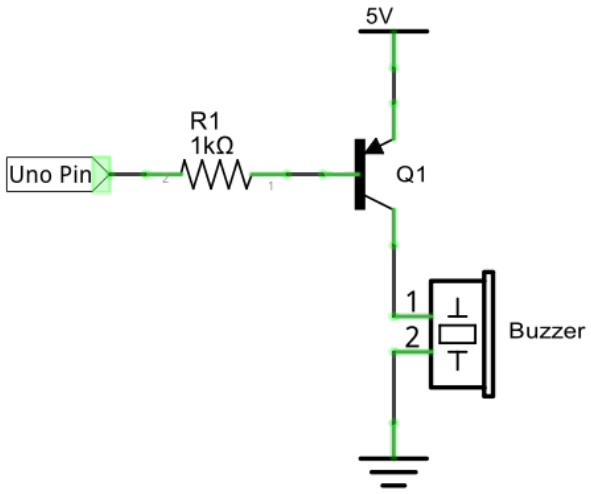
When we use a NPN transistor to drive a buzzer, we often use the following method. If GPIO outputs high level, current will flow through R1 (Resistor 1), the transistor conducts current and the buzzer will make sounds. If GPIO outputs low level, no current will flow through R1, the transistor will not conduct currentand buzzer will remain silent (no sounds).

When we use a PNP transistor to drive a buzzer, we often use the following method. If GPIO outputs low level, current will flow through R1. The transistor conducts current and the buzzer will make sounds. If GPIO outputs high level, no current flows through R1, the transistor will not conduct current and buzzer will remain silent (no sounds). Below are the circuit schematics for both a NPN and PNP transistor to power a buzzer.

NPN transistor to drive buzzer

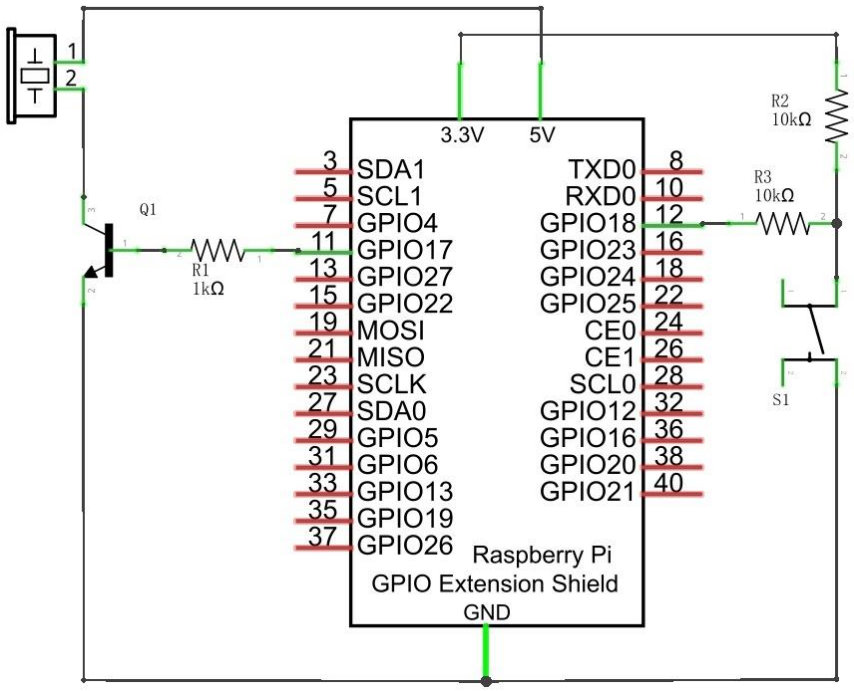



PNP transistor to drive buzzer

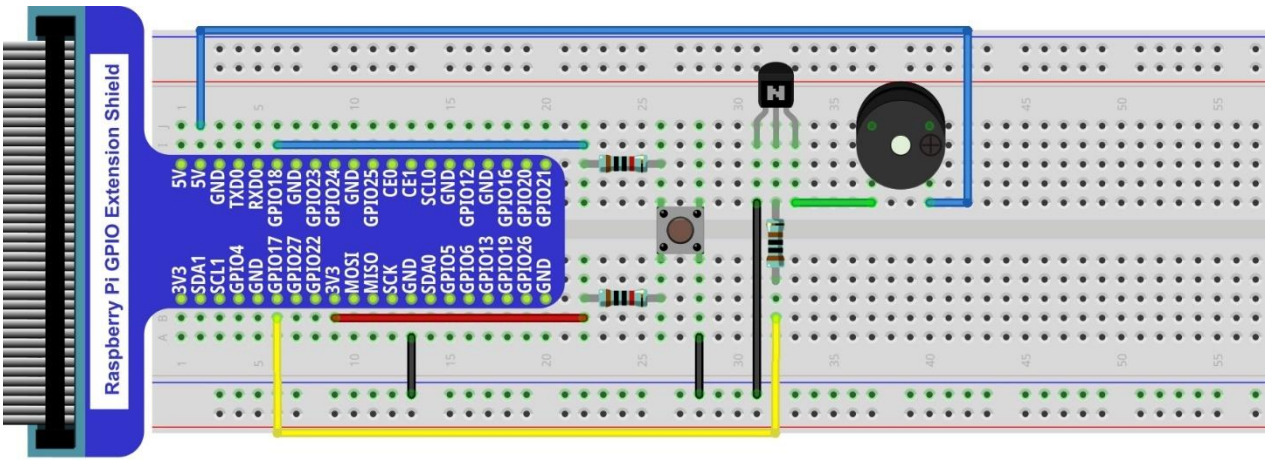


## Circuit

Schematic diagram with RPi GPIO Extension Shield




Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com




Video: https://youtu.be/R_dmi3YwY-U

Note: in this circuit, the power supply for the buzzer is 5V, and pull-up resistor of the push button switch is connected to the 3.3V power feed. Actually, the buzzer can work when connected to the 3.3V power feed but this will produce a weak sound from the buzzer (not very loud).

## Code

In this project, a buzzer will be controlled by a push button switch. When the button switch is pressed, the buzzer sounds and when the button is released, the buzzer stops. It is analogous to our earlier project that controlled an LED ON and OFF.

## C Code 6.1.1 Doorbell

First, observe the project result, and then learn about the code in detail.

**If you have any concerns, please contact us via: support@freenove.com**

1. Use cd command to enter 06.1.1_Doorbell directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/06.1.1_Doorbell
```

2. Use following command to compile "Doorbell.c" and generate executable file "Doorbell.c".

```
gcc Doorbell.c -o Doorbell -lwiringPi
```

3. Then run the generated file "Doorbell".

```
sudo ./Doorbell
```

After the program is executed, press the push button switch and the will buzzer sound. Release the push button switch and the buzzer will stop.

The following is the program code:

```
#include <wiringPi.h>
#include <stdio.h>

#define buzzerPin 0   //define the buzzerPin
#define buttonPin 1       //define the buttonPin

void main(void)
{
    printf("Program is starting ... \n");

    wiringPiSetup();

    pinMode(buzzerPin, OUTPUT);
    pinMode(buttonPin, INPUT);

    pullUpDnControl(buttonPin, PUD_UP);  //pull up to HIGH level
    while(1){

        if(digitalRead(buttonPin) == LOW){ //button is pressed
            digitalWrite(buzzerPin, HIGH);   //Turn on buzzer
            printf("buzzer turned on >>> \n");
        }
        else {                  //button is released
            digitalWrite(buzzerPin, LOW);   //Turn off buzzer
            printf("buzzer turned off <<< \n");
        }
    }
}
```

The code is exactly the same as when we used a push button switch to control an LED. You can also try using the PNP transistor to achieve the same results.

# Project 6.2 Alertor

Next, we will use a passive buzzer to make an alarm.
The list of components and the circuit is similar to the doorbell project. We only need to take the Doorbell circuit and replace the active buzzer with a passive buzzer.

## Code

In this project, our buzzer alarm is controlled by the push button switch. Press the push button switch and the buzzer will sound. Release the push button switch and the buzzer will stop.
As stated before, it is analogous to our earlier project that controlled an LED ON and OFF.
To control a passive buzzer requires PWM of certain sound frequency.

### C Code 6.2.1 Alertor

First, observe the project result, and then learn about the code in detail.
**If you have any concerns, please contact us via: support@freenove.com**

1. Use cd command to enter 06.2.1_Alertor directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/06.2.1_Alertor
```

2. Use following command to compile "Alertor.c" and generate executable file "Alertor". "-lm" and "-lpthread" compiler options need to added here.

```
gcc Alertor.c -o Alertor -lwiringPi -lm -lpthread
```

3. Then run the generated file "Alertor".

```
sudo ./Alertor
```

After the program is executed, press the push button switch and the buzzer will sound. Release the push button switch and the buzzer will stop.
The following is the program code:

```
#include <wiringPi.h>
#include <stdio.h>
#include <softTone.h>
#include <math.h>

#define buzzerPin    0     //define the buzzerPin
#define buttonPin      1       //define the buttonPin

void alertor(int pin){
    int x;
    double sinVal, toneVal;
    for(x=0;x<360;x++){ // frequency of the alertor is consistent with the sine wave
        sinVal = sin(x * (M_PI / 180));     //Calculate the sine value
        toneVal = 2000 + sinVal * 500; //Add the resonant frequency and weighted sine value
        softToneWrite(pin,toneVal);             //output corresponding PWM
        delay(1);
```

```
17      }
18  }
19  void stopAlertor(int pin){
20      softToneWrite(pin,0);
21  }
22  int main(void)
23  {
24      printf("Program is starting ... \n");
25
26      wiringPiSetup();
27
28      pinMode(buzzerPin, OUTPUT);
29      pinMode(buttonPin, INPUT);
30      softToneCreate(buzzerPin); //set buzzerPin
31      pullUpDnControl(buttonPin, PUD_UP);  //pull up to HIGH level
32      while(1){
33          if(digitalRead(buttonPin) == LOW){ //button is pressed
34              alertor(buzzerPin);   // turn on buzzer
35              printf("alertor turned on >>> \n");
36          }
37          else {                  //button is released
38              stopAlertor(buzzerPin);   // turn off buzzer
39              printf("alertor turned off <<< \n");
40          }
41      }
42      return 0;
43  }
```

The code is the same to the active buzzer but the method is different. A passive buzzer requires PWM of a certain frequency, so you need to create a software PWM pin though softToneCreate (buzzeRPin). Here softTone is designed to generate square waves with variable frequency and a duty cycle fixed to 50%, which is a better choice for controlling the buzzer.

```
    softToneCreate(buzzeRPin);
```

In the while loop of the main function, when the push button switch is pressed the subfunction alertor() will be called and the alarm will issue a warning sound. The frequency curve of the alarm is based on a sine curve. We need to calculate the sine value from 0 to 360 degrees and multiplied by a certain value (here this value is 500) plus the resonant frequency of buzzer. We can set the PWM frequency through softToneWrite (pin, toneVal).

```
    void alertor(int pin){
        int x;
        double sinVal, toneVal;
        for(x=0;x<360;x++){ //The frequency is based on the sine curve.
            sinVal = sin(x * (M_PI / 180));
            toneVal = 2000 + sinVal * 500;
```

```
            softToneWrite(pin,toneVal);
            delay(1);
        }
    }
```

If you want to stop the buzzer, just set PWM frequency of the buzzer pin to 0.

```
void stopAlertor(int pin){
    softToneWrite(pin,0);
}
```

The related functions of softTone are described as follows:

| int softToneCreate (int pin) ; |
| --- |
| This creates a software controlled tone pin. |
| void softToneWrite (int pin, int freq) ; |
| This updates the tone frequency value on the given pin. |

For more details about softTone, please refer to : https://github.com/WiringPi/WiringPi

# What's Next?

THANK YOU for participating in this learning experience! If you have completed all of the projects successfully you can consider yourself a Raspberry Pi Master.

We have reached the end of this Tutorial. If you find errors, omissions or you have suggestions and/or questions about the Tutorial or component contents of this Kit, please feel free to contact us: **support@freenove.com**
We will make every effort to make changes and correct errors as soon as feasibly possible and publish a revised version.

If you are interesting in processing, you can study the Processing.pdf in the unzipped folder.

If you want to learn more about Arduino, Raspberry Pi, Smart Cars, Robotics and other interesting products in science and technology, please continue to visit our website. We will continue to launch fun, cost-effective, innovative and exciting products.
http://www.freenove.com/

Thank you again for choosing Freenove products.