

MicroPython (Command Line) Getting started guide

日期	作者	内容	备注
2019.4.29	xdd	Finsh	

		Document	
2019.4.30	xdd	Improve common commands	

What is MicroPython?

MicroPython, a complete software implementation of the Python 3 programming language, written in C, is optimized to run on a microcontroller. MicroPython is a complete Python compiler and runtime system that runs on top of the microcontroller hardware. An interactive prompt (REPL) is provided to the user to immediately execute the supported commands. In addition to including the selected core Python libraries, MicroPython includes modules that give programmers access to low-level hardware. MicroPython strives to be as compatible as possible with plain Python (called CPython), so if you know Python you already know MicroPython. On the other hand, the more you know about MicroPython, the better your performance in Python. In addition to implementing a set of core Python libraries, MicroPython also includes modules such as "machines" for accessing low-level hardware.

The advantages of MicroPython

1. Compiler Environment

C language, assembly language Before writing the program, you need to download the compilation environment. However, MicroPython does not need any compilation environment. Just insert the U port of the development board into the computer, and a drive letter can appear on the computer, just like inserting a USB flash drive. Then open the U disk, directly notepad main.py, it is comfortable that MicroPython does not need any tools and environment, any text tool + 1 development board can be developed and compiled, you can give it a nickname - pocket programming computer.



2.Operation difficulty

The basic operation of assembly language is simple, but it is relatively difficult to achieve complex project goals, the code is tedious, debugging debugging is difficult; C language has statements and a large number of library functions, which is relatively simple compared to assembly; MicroPython has more than C The library function, the code format has not only become concise, many functions and methods without the support of the library do not have to write it yourself, directly import xx can, the class library is imported ok. Previously, a single lamp required five or six lines of code. Now only one line of code is needed to

```
1 # main.py -- put your code here!
2 import pyb
3 pyb.LED(1).on()
```

illuminate the LED, So Easy.

3. Program structure

The assembly language generally uses the jump structure. Whether it is a subroutine call or a loop or a scatter theory, the jump mode is used. Interrupt operations or forced out of the block operations require precise control of the push stack. Very strict; C language generally uses a loop structure or a sequence structure, no longer need to jump around to execute the program, the interrupt operation will automatically push the stack without human intervention, embed the assembly to improve efficiency

when the timing requirements are high; MicroPython can use C
The program structure or the use of thread structure, the thread
structure requires strict and reasonable allocation of thread
working time can not conflict, the resources should have a good
control, can not waste the thread waste resources, in theory,
multi-thread can be more efficient Running, although a single core
can only run a single thread at the same time, but it is synchronous
from a macroscopic point of view, it can effectively reduce the
difficulty of programming in the case of less stringent time
requirements.

4. Writing method

In the case of not using macros, the assembly is usually a
direct operation unit, which needs to memorize a large number of
data unit uses, and because the number of program lines is large
(may be one or two thousand lines), it is necessary to look back
and forth the program code. The process is cumbersome. And
because the 8-bit unit complex data operation needs to be nested
in C language to complete; C language use of any variable needs
to be defined first, compared to not specifically remember the
variable name, the overall structure is clear, most programming
software has a jump Subroutine function, it is very convenient to

find the program block. There are a large number of finished header files containing various commonly used functions, relatively few complicated program code writing; MicroPython inherits the advantages of C writing, and does not need to be predefined, does not require any end symbols, just need to change lines to continue writing. But pay attention to the use of the Tab key in some special cases.

5.Readability

The assembly itself is difficult to read because the mark is small and the unit is directly manipulated. The visual structure is not only two columns of code but also needs to keep jumping. Therefore, it is difficult to read by itself; the simple operation code of C language is clear, but complicated. Finding methods in the method, a lot of complicated code is not a problem for me to find, it is very headache; MicroPython has better readability than C, and because of the elimination of many special symbols, the overall feeling is clearer, many methods are It is written separately as a separate library, which can be called directly. Each function has a separate class library, which is extremely convenient and easy to see at a glance. Through the comparison of these dimensions, you should know very easily why I said that MicroPython is

comfortable to use. At present, AI is the mainstream of current social development. The emergence of MicroPython is an opportunity for many beginners like me to do smart cars and learn hardware. It is easier to learn than C language and is more suitable for education. You can learn programming and write programs. It can be seen that MicroPython's future market prospects are very large, and it is a development board that is more powerful and easier to program than Arduino.

What can we do with MicroPython?

Of course it is lighting LED!!!

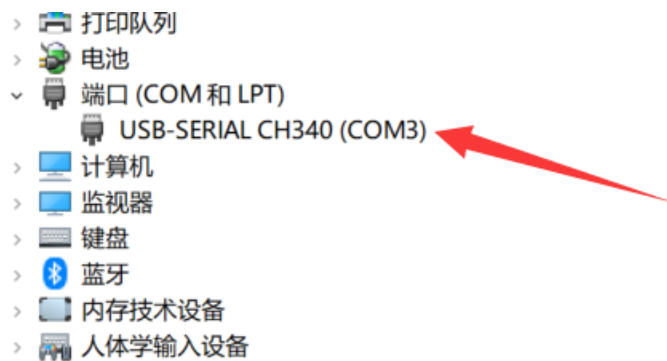
Not much nonsense, let's start lighting LED together!

The first step: connect the development board via USB

Unlike the STM32 version of the pyboard, the SWM320VET7 on the PineconePi ONE itself does not have a USB and can only access the SWM320VET7 via a TTL serial port. Let's first introduce the connection through the serial port.

On the PineconePi ONE development board, with a Micro USB interface, and a USB to serial port chip CH330N, it can easily

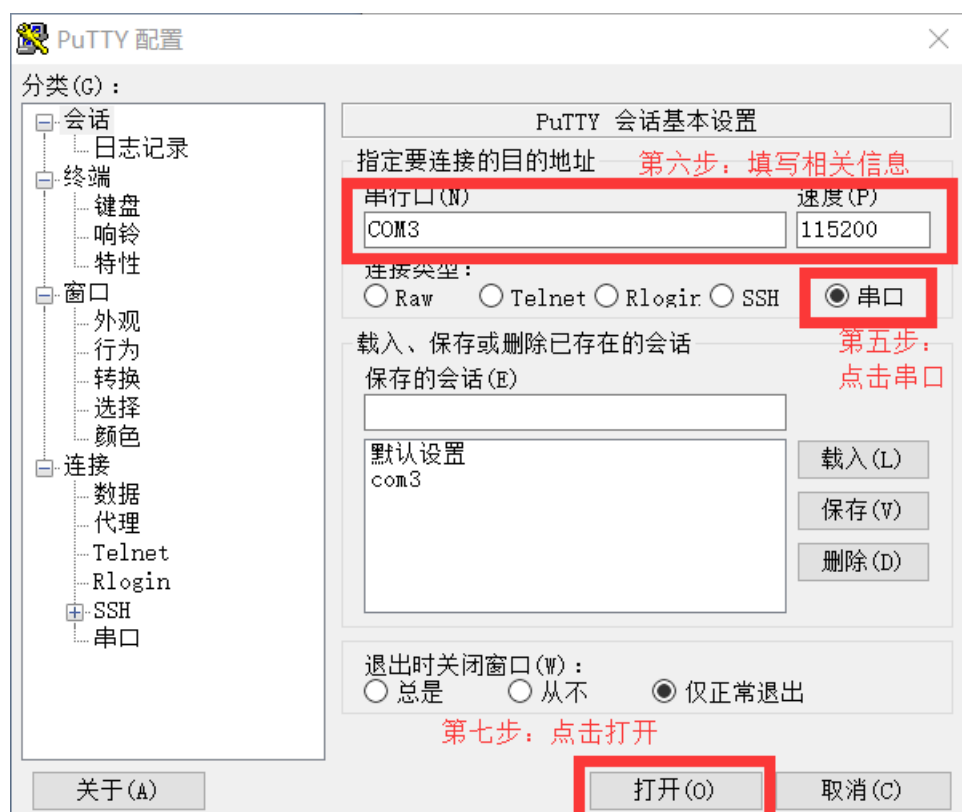
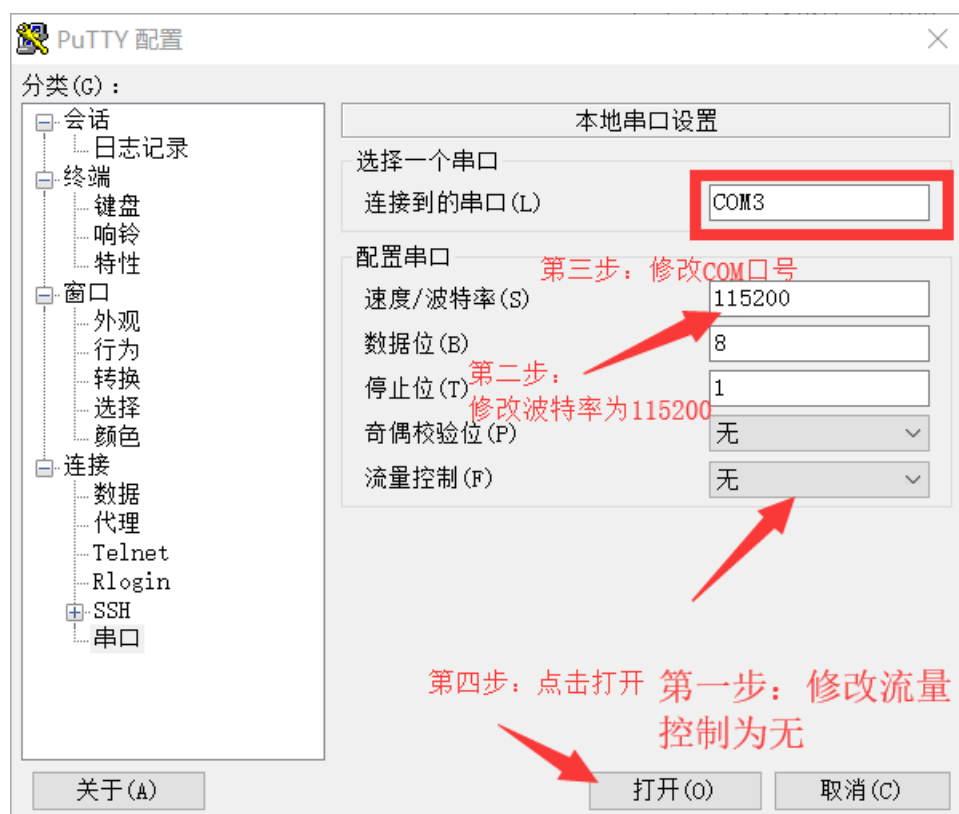
connect the computer to the PineconePi ONE. Before use, you need to install the CH330 driver (common to the CH340 driver) so that when the PineconePi ONE is connected to the computer, a serial device will appear. The following is the virtual serial port displayed on Windows. Under Linux, it is usually /dev/ttyUSB0.



Step 2: Connect the development board with terminal software

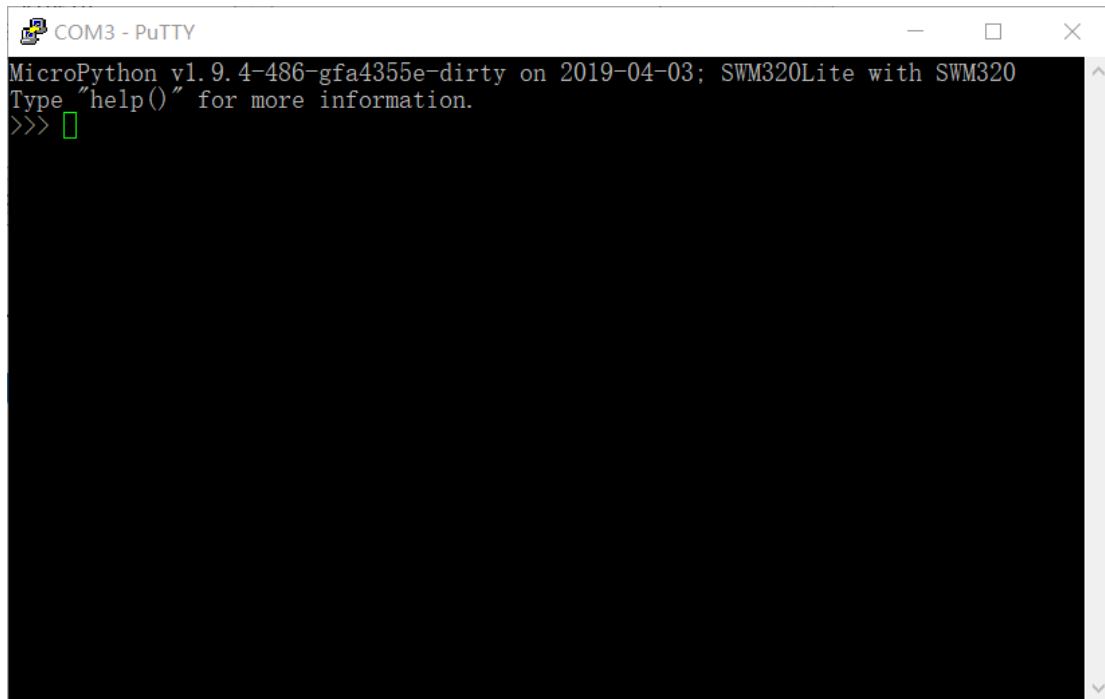
The uPyCraft IDE (V1.1) can be used to connect to the development board. Use the documentation and download link at: <http://docs.dfrobot.com.cn/upycraft/>

PS: Putty software can be found under the PineconePi ONE repository / MicroPython / putty /



Finally, we press the S1 button on the development board to

perform a hardware reset, and we can see the following screen on.PuTTY:



```
COM3 - PuTTY
MicroPython v1.9.4-486-gfa4355e-dirty on 2019-04-03; SWM320Lite with SWM320
Type "help()" for more information.
>>> █
```

- Let's introduce the general operation:
- Under the terminal, flexible use of shortcut keys can help us.
Commonly used shortcut keys are:
- CTRL-A – on a blank line, enter raw REPL mode (This shortcut is not for entering the program, generally do not use)
- CTRL-B – In the empty command line, return to the normal REPL interaction mode
- CTRL-C – Interrupt the running program
- CTRL-D – Soft reset
- CTRL-E – Paste mode (**Press the right mouse button to paste**)

- Up and down direction keys– Bring up the previous input command

When running the program, if there is a problem, you can use Ctrl-C to abort the operation at any time, or use Ctrl-D to soft reset under the empty command line. If you still can't solve the problem, just press the reset button to perform a hard reset.

In case of doubt, you can type `help()` to view the help, or even view the help of a function or library, such as `help(machine)`.

You can also use `dir()` to view the modules, functions, variables that have been loaded, or use `dir` to view the contents of a library, such as `dir(machine)`.

The following can finally light LED up! ! !

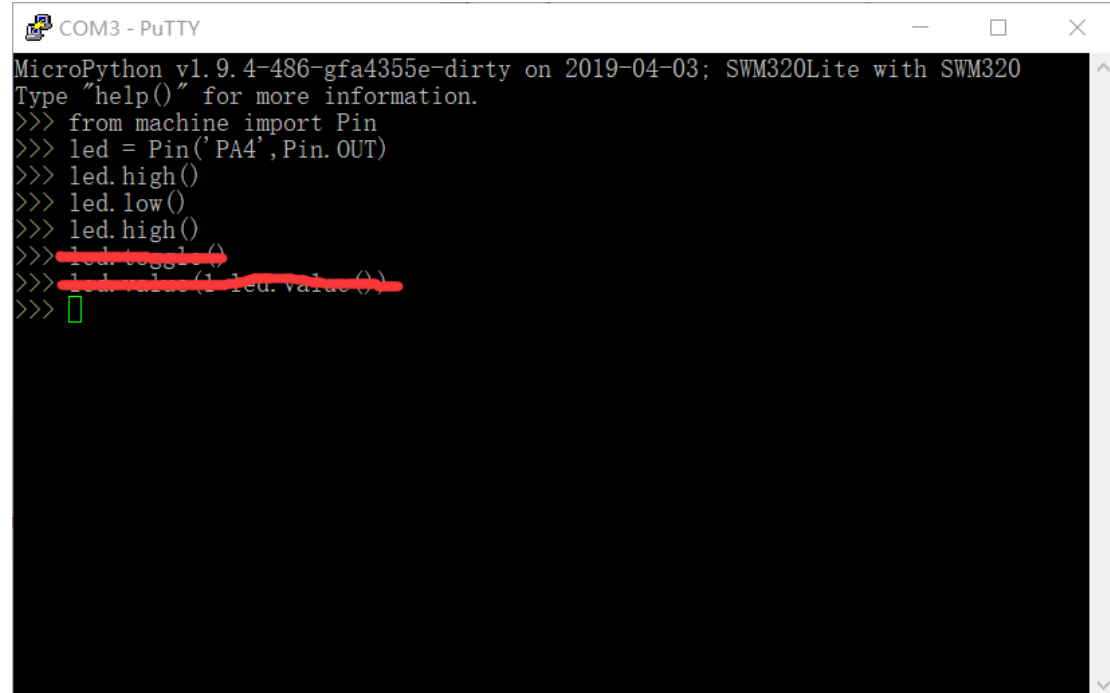
Let's enter the code (of course, you can CTRL+E into the paste mode, just click the right mouse button to paste)

```
from machine import Pin
```

```
led = Pin('PB12', Pin.OUT)
```

```
led.low()
```

```
led.high()
```



The screenshot shows a PuTTY window titled 'COM3 - PuTTY'. The terminal displays the following text: 'MicroPython v1.9.4-486-gfa4355e-dirty on 2019-04-03; SWM320Lite with SWM320'. Below this, the user has entered several commands: '>>> from machine import Pin', '>>> led = Pin('PA4', Pin.OUT)', '>>> led.high()', '>>> led.low()', and '>>> led.high()'. The next line, '>>> led.toggle()', is crossed out with a red line. The following line, '>>> print(led.value())', is also crossed out with a red line. The prompt '>>>' is followed by a green cursor. The terminal window has standard window controls (minimize, maximize, close) in the top right corner.

So we Light a LED!

Appendix

(Common command guide)

MicroPython port to SWM320 (Cortex-M4 MCU)

After the MCU is powered on, it automatically runs boot.py and main.py.

System

CPU Freq

```
import machine

machine.freq() # Get current CPU frequency (120Mhz)
```

file

```
import os

os.listdir() # View system current file list

os.listdir('lib') # View the directory lib

os.mkdir("PineconePi") # Create folder PineconePi
```

```
os.chdir("PineconePi") # Go to the sub-category PineconePi
```

```
os.getcwd() # Get current directory
```

```
os.rmdir("hx") # Delete folder (can only be an empty folder)
```

```
os.rename("hm.txt","11") # Rename file
```

```
os.remove("11") # Delete Files
```

```
f = open('main.py', 'r') # Open the main.py to read
```

```
f.read() #read main.py
```

```
f.close() #Close main.py
```

```
f = open('main.py', 'W') #Open the main.py to write
```

```
f.write("#PineconePi ONE\n") # Write content "#PineconePi ONE "to  
main.py and wrap
```

```
f.close() #Close main.py
```

Pin

OutPut

```
from machine import Pin
led = Pin('PA5', Pin.OUT)
led.low()
led.high()
led.toggle()
led.value(1-led.value())
```

Putin

```
key = Pin('PA4', Pin.IN)
key = Pin('PA4', Pin.IN, mode=Pin.PULL_UP) # Open internal
pull-up and pull-down
key.value()
```

Interrupt

```
led = Pin('PA5', Pin.OUT)
```



```
key = Pin('PA4', Pin.IN, irq=Pin.FALL_EDGE, callback=lambda key:  
led.toggle())
```

Timer

Timer

```
from machine import Pin, Timer  
  
led = Pin('PA5', Pin.OUT)  
  
tmr = Timer(0, 10000, callback=lambda tmr: led.toggle())  
tmr.start()  
  
# The second parameter '10000' is the timing period duration in  
units of 0.1ms.
```

counter

```
from machine import Pin, Timer

led = Pin('PA5', Pin.OUT)

ctr = Timer(2, 3, mode=Timer.MODE_COUNTER, pin='PA4',
callback=lambda ctr: led.toggle())

ctr.start()
```

Timer4 and Timer5 do not have a Counter function.

UART

```
from machine import UART

ser = UART(1, 57600, tx='PA9', rx='PA10') # Equivalent to the
following line

ser = UART(1, 57600, bits=UART.DATA_8BIT,
parity=UART.PARITY_None, stop=UART.STOP_1BIT, tx='PA9',
rx='PA10')

ser.write('Hi from Synwit\n')

if ser.any(): s = ser.read()
```

UART0 is the micropython command interface, do not use in the

user program.

SPI

```
from machine import Pin, SPI

spi = SPI(0, 100000, mosi='PC4', miso='PC5', sclk='PC7')

# Equivalent to the following line
spi = SPI(0, 100000, polarity=0, phase=0, bits=8, mosi='PC4',
miso='PC5', sclk='PC7')

spi_cs = Pin('PC2', Pin.OUT)

spi_cs.high()

spi_cs.low()

spi.write(b'\x33\x78\xAF')

spi_cs.high()
```

I2C

```
from machine import I2C

i2c = I2C(0, 100000, scl='PA10', sda='PA9')
```

```
i2c.writeto(0x6C, '\x33\x78\xAF')
```

#The pins on ports A and C have internal pull-ups, and the external pull-up resistors are not connected. The pins on port B must be connected with pull-up resistors.

ADC

```
from machine import ADC

adc = ADC(0, chns=ADC.ADC_CH4)

for i in range(5): print(adc.read())

adc.chn_select(ADC.ADC_CH4|ADC.ADC_CH5|ADC.ADC_CH6|ADC.
ADC_CH7)

for i in range(5): print(adc.read())
```

PWM

Independent output

```
import time
```

```
from machine import PWM

pwm = PWM(0, 15000, 5000, pin='PA10')

pwm.start()

time.sleep_ms(50)

pwm.period(30000)

pwm.duty(15000)

#The counting clock of PWM is 15MHz, and the counting period is
15000. The square wave frequency is 1KHz.
```

互补输出

```
import time

from machine import PWM

pwm = PWM(1, 1500, 500, mode=PWM.MODE_COMPL, pinA='PA9',
pinB='PA11', deadzone=50)

pwm.start()

time.sleep_ms(50)

pwm.period(3000)

pwm.duty(1500)

# Dead zone maximum value 63
```

notice

1、The PA12 pin has no peripheral function mapping function and can only be used as a pin.