INSTALLING PYTHON PACKAGES

APT

Some Python packages can be found in the Raspbian archives and can be installed using APT. For example:

sudo apt-get update

sudo apt-get install python3-picamera

This is the preferred method of installing software, as it means that the modules you install can be kept up to date easily with the usual sudo apt-get updateand sudo apt-get upgrade commands.

Python packages in Raspbian which are compatible with Python 2.x will always have a python- prefix. So, the picamera package for Python 2.x is named python-picamera (as shown in the example above). Python 3 packages always have a python3- prefix. So, to install picamera for Python 3 you would use:

sudo apt-get install python3-picamera

Uninstalling packages installed via APT can be accomplished as follows:

sudo apt-get remove python3-picamera

They can be completely removed with purge:

sudo apt-get purge python3-picamera

pip

Not all Python packages are available in the Raspbian archives, and those that are can sometimes be out-of-date. If you can't find a suitable version in the Raspbian archives, you can install packages from the [Python Package Index](http://pypi.python.org/) (PyPI). To do so, use the pip tool.

pip is installed by default in Raspbian Jessie (but not Raspbian Wheezy or Jessie Lite). You can install it with apt:

sudo apt-get install python3-pip

To get the Python 2 version:

sudo apt-get install python-pip

pip3 installs modules for Python 3, and pip installs modules for Python 2.

For example, the following command installs the Unicorn HAT library for Python 3:

pip3 install unicornhat

The following command installs the Unicorn HAT library for Python 2:

pip install unicornhat

**Note**: In Raspbian Wheezy, the command for managing Python 3 packages was pip-3.2, not pip3.

Uninstall Python modules with pip3 uninstall or pip uninstall.

Upload your own Python modules to pip with the [guide at PyPI](https://wiki.python.org/moin/CheeseShopTutorial#Submitting_Packages_to_the_Package_Index).

## Installing with get-pip.py

To install pip, securely download [get-pip.py](https://bootstrap.pypa.io/get-pip.py). [[1]](https://pip.pypa.io/en/stable/installing/#id7):

curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py

Inspect get-pip.py for any malevolence. Then run the following:

python get-pip.py

**Warning**

Be cautious if you are using a Python install that is managed by your operating system or another package manager. get-pip.py does not coordinate with those tools, and may leave your system in an inconsistent state.

get-pip.py also installs [setuptools](https://packaging.python.org/key_projects/#setuptools) [[2]](https://pip.pypa.io/en/stable/installing/#id8) and [wheel](https://packaging.python.org/key_projects/#wheel) if they are not already. [setuptools](https://packaging.python.org/key_projects/#setuptools) is required to install [source distributions](https://packaging.python.org/glossary/#term-source-distribution-or-sdist). Both are required in order to build a [Wheel Cache](https://pip.pypa.io/en/stable/reference/pip_install/#wheel-cache) (which improves installation speed), although neither are required to install pre-built [wheels](https://packaging.python.org/glossary/#term-wheel).

**Note**

The get-pip.py script is supported on the same python version as pip. For the now unsupported Python 2.6, alternate script is available [here](https://bootstrap.pypa.io/2.6/get-pip.py).

### get-pip.py options

**--no-setuptools**

If set, do not attempt to install [setuptools](https://packaging.python.org/key_projects/#setuptools)

**--no-wheel**

If set, do not attempt to install [wheel](https://packaging.python.org/key_projects/#wheel)

upgrade pip

python -m pip install --upgrade pip setuptools wheel

And in Python 3 this works:

python3 -m pip install --upgrade pip

安装scipy：（--user安装在了'/home/pi/.local/bin'中，需要将'/home/pi/.local/bin'加到PATH变量目录上）

python -m pip install --user numpy scipy matplotlib ipython jupyter pandas sympy nose

scipy安装需要wheel，报错Failed building wheel for scipy时：

1. 加—use-wheel
2. 若加—use-wheel也不管用时，安装下面软件

sudo apt-get install libatlas-base-dev gfortran

echo $PATH 查看PATH变量的值

**在PATH变量后面加一个路径：**

**export PATH=$PATH:/place/with/the/file**

# tar解压、压缩命令

### 1. Creating an archive using tar command

#### Creating an uncompressed tar archive using option cvf

This is the basic command to create a tar archive.

$ tar cvf archive\_name.tar dirname/

In the above command:

* c – create a new archive
* v – verbosely list files which are processed.
* f – following is the archive file name

#### Creating a tar gzipped archive using option cvzf

The above tar cvf option, does not provide any compression. To use a gzip compression on the tar archive, use the z option as shown below.

$ tar cvzf archive\_name.tar.gz dirname/

* z – filter the archive through gzip

**Note:** .tgz is same as .tar.gz

**Note:** I like to keep the ‘cvf’ (or tvf, or xvf) option unchanged for all archive creation (or view, or extract) and add additional option at the end, which is easier to remember. i.e cvf for archive creation, cvfz for compressed gzip archive creation, cvfj for compressed bzip2 archive creation etc., For this method to work properly, don’t give – in front of the options.

#### Creating a bzipped tar archive using option cvjf

Create a bzip2 tar archive as shown below:

$ tar cvfj archive\_name.tar.bz2 dirname/

* j – filter the archive through bzip2

**gzip vs bzip2**: bzip2 takes more time to compress and decompress than gzip. bzip2 archival size is less than gzip.

**Note:** .tbz and .tb2 is same as .tar.bz2

### 2. Extracting (untar) an archive using tar command

#### Extract a \*.tar file using option xvf

Extract a tar file using option x as shown below:

$ tar xvf archive\_name.tar

* x – extract files from archive

#### Extract a gzipped tar archive ( \*.tar.gz ) using option xvzf

Use the option z for uncompressing a gzip tar archive.

$ tar xvfz archive\_name.tar.gz

#### Extracting a bzipped tar archive ( \*.tar.bz2 ) using option xvjf

Use the option j for uncompressing a bzip2 tar archive.

$ tar xvfj archive\_name.tar.bz2

**Note:** In all the above commands v is optional, which lists the file being processed.

# WRITING A PYTHON PROGRAM

To demonstrate creating and executing a Python program, we’ll make a simple “hello world” program. To begin, open the Nano text editor and create a new file named hello-world.py by entering this at the command prompt:

sudo nano hello-world.py

Enter this code into Nano, then press Ctrl-X and Y to exit and save the file:

#!/usr/bin/python # coment

print "Hello, World!";

All Python program files will need to be saved with a “.py” extension. You can write the program in any text editor such as Notepad or Notepad++, just be sure to save the file with a “.py” extension.

## RUNNING A PYTHON PROGRAM

To run the program without making it executable, navigate to the location where you saved your file, and enter this at the command prompt:

python hello-world.py.

# MQTT Python

Use [curl](http://curl.haxx.se/) from the Terminal to post a message to the [HTTP](https://docs.shiftr.io/interfaces/http) interface.

curl -X POST "http://miniroc\_shiftr:abcd1234@broker.shiftr.io/try" -d "Kevin Hello world"

Here is a very simple example that subscribes to the broker $SYS topic tree and prints out the resulting messages:

import paho.mqtt.client as mqtt

# The callback for when the client receives a CONNACK response from the server.

def on\_connect(client, userdata, flags, rc):

print("Connected with result code "+str(rc))

# Subscribing in on\_connect() means that if we lose the connection and reconnect then subscriptions will be renewed.

client.subscribe("$SYS/#")

# The callback for when a PUBLISH message is received from the server.

def on\_message(client, userdata, msg):

print(msg.topic+" "+str(msg.payload))

client = mqtt.Client()

client.on\_connect = on\_connect

client.on\_message = on\_message

client.connect("iot.eclipse.org", 1883, 60)

# Blocking call that processes network traffic, dispatches callbacks and

# handles reconnecting.

# Other loop\*() functions are available that give a threaded interface and a

# manual interface.

client.loop\_forever()

### Client

You can use the client class as an instance, within a class or by subclassing. The general usage flow is as follows:

* Create a client instance
* Connect to a broker using one of the connect\*() functions
* Call one of the loop\*() functions to maintain network traffic flow with the broker
* Use subscribe() to subscribe to a topic and receive messages
* Use publish() to publish messages to the broker
* Use disconnect() to disconnect from the broker

Callbacks will be called to allow the application to process events as necessary. These callbacks are described below.

## [Python: Subscribing to MQTT topic](https://techtutorialsx.com/2017/04/23/python-subscribing-to-mqtt-topic/)

Posted on [April 23, 2017](https://techtutorialsx.com/2017/04/23/python-subscribing-to-mqtt-topic/)by [antepher](https://techtutorialsx.com/author/antepher/)

The objective of this post is to explain how to connect to a MQTT broker and subscribe to a topic, using Python.

## ****Introduction****

The objective of this post is to explain how to connect to a MQTT broker and subscribe to a topic, using Python. For this example, we will be using [paho-mqtt](https://pypi.python.org/pypi/paho-mqtt/#installation), which is a MQTT Python client library. The easiest way to install it is via [pip](https://pypi.python.org/pypi/pip), with the following command:

|  |  |
| --- | --- |
| 1 | pip install paho-mqtt |

As [MQTT broker](http://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment), we will use [CloudMQTT](https://www.cloudmqtt.com/), which offers a free plan, amongst other options. You can check [here](https://www.cloudmqtt.com/docs.html) how to create an account and how to create a broker instance.

Since the basics on how to connect to the CloudMQTT broker were already covered in [this](https://techtutorialsx.com/2017/04/14/python-publishing-messages-to-mqtt-topic/) previous post, those sections of the code will not be described in detail. Please read that post first for the basics of paho-mqtt.

## ****The code****

First, we need to import the [client](https://github.com/eclipse/paho.mqtt.python/blob/master/src/paho/mqtt/client.py) class from the MQTT library, to have access to all the functionality needed to connect to the broker and subscribe to topics. Additionally, we will also import Python’s [time](https://docs.python.org/2/library/time.html) module, so we can have access to the [sleep](https://docs.python.org/2/library/time.html#time.sleep) function.

We will also declare some auxiliary global variables. The first one will be used to control the initial state of connection to the broker, and the other ones will hold the information needed to connect to the broker. All the credentials needed for the connection should be obtained from the CloudMQTT instance information page.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | import paho.mqtt.client as mqttClient  import time    Connected = False #global variable for the state of the connection    broker\_address= "m11.cloudmqtt.com"  port = 12948  user = "yourUSer"  password = "yourPass" |

Now, we will create a new client instance. We will pass as argument of the constructor a unique client identifier in format of a string. Next, we will call the [username\_pw\_set](https://github.com/eclipse/paho.mqtt.python#username_pw_set)method, which allows to specify a username and, optionally, a password. For the broker we are trying to connect to, the password is needed.

We also need to specify a **on\_connect** callback function, which is called when the broker responds to the connection request. We will just assign the function for now and define the actual code latter.

Since we are going to subscribe to a topic, we will also specify a **on\_message** callback function, which is called for every message received on a subscribed topic [1]. Again, we will define the actual function code latter.

|  |  |
| --- | --- |
| 1  2  3  4 | client = mqttClient.Client("Python")               #create new instance  client.username\_pw\_set(user, password=password)    #set username and password  client.on\_connect= on\_connect                      #attach function to callback  client.on\_message= on\_message                      #attach function to callback |

Finally, we will call the [connect](https://github.com/eclipse/paho.mqtt.python#connect-reconnect-disconnect) method, for establishing the connection to the broker. This is a blocking method. Then, we will call a method called [loop\_start](https://github.com/eclipse/paho.mqtt.python#loop_start--loop_stop), which will run a thread in background to handle the network connection and sending/receiving data.

Since establishing the connection may take a while, we will do a loop until the previously declared **Connected** variable is set to true. This will be done in our callback function, which we still need to specify.

Once the connection is established, we need to call the [subscribe](https://github.com/eclipse/paho.mqtt.python#subscribe-1) method, passing as input the topic that we want to subscribe to. In this case, we will subscribe to the “python/test” topic.

Note that this method can be called with a different set of arguments, and we are using it in its simple form. Also remember that when a message is received, it will be handled on a function we called **on\_message**, yet to be defined.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | client.connect(broker\_address, port=port)  #connect to broker  client.loop\_start()                        #start the loop    while Connected != True:    #Wait for connection      time.sleep(0.1)    client.subscribe("python/test") |

After subscribing to the topic, we will do an infinite loop with a small delay in each iteration, since the messages will be handled by a callback function. So, we don’t need to worry about polling anything.

We will run this loop on a try-except block, where the except block will catch a keyboard interrupt. This way, we can end the loop by sending a ctrl+C command on the Python shell.

Since the program will finish in the except block, we call the [disconnect](https://github.com/eclipse/paho.mqtt.python#disconnect) method, to disconnect from the broker. After that, we also need to call the [loop\_stop](https://github.com/eclipse/paho.mqtt.python#loop_start--loop_stop) method, to end the previously mentioned background thread.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | try:      while True:          time.sleep(1)    except KeyboardInterrupt:      print "exiting"      client.disconnect()      client.loop\_stop() |

Now, we still need to declare the **on\_message** callback function. As can be seen [here](https://github.com/eclipse/paho.mqtt.python#on_message7), this function receives 3 arguments. We are only going to use the argument named in the previous link as **message**, which is an instance of the [MQTTMessage](https://github.com/eclipse/paho.mqtt.python/blob/e77d54f4a1ce523740ce4a8c5a759e7a3ac649d0/src/paho/mqtt/client.py#L345) class[1].

We will access a [member](https://www.tutorialspoint.com/python/python_classes_objects.htm) of this class called **payload**, which we will use to get the actual message received from the subscribed topic. So, on the **on\_message** function, we will print the message payload, as can be seen bellow.

|  |  |
| --- | --- |
| 1  2 | def on\_message(client, userdata, message):      print "Message received: "  + message.payload |

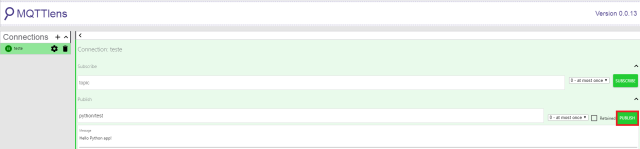
You can check the full source code bellow, which already includes the **on\_connect**function declaration.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48 | import paho.mqtt.client as mqttClient  import time    def on\_connect(client, userdata, flags, rc):        if rc == 0:            print("Connected to broker")            global Connected                #Use global variable          Connected = True                #Signal connection        else:            print("Connection failed")    def on\_message(client, userdata, message):      print "Message received: "  + message.payload    Connected = False   #global variable for the state of the connection    broker\_address= "m11.cloudmqtt.com"  #Broker address  port = 12948                         #Broker port  user = "yourUser"                    #Connection username  password = "yourPassword"            #Connection password    client = mqttClient.Client("Python")               #create new instance  client.username\_pw\_set(user, password=password)    #set username and password  client.on\_connect= on\_connect                      #attach function to callback  client.on\_message= on\_message                      #attach function to callback    client.connect(broker\_address, port=port)          #connect to broker    client.loop\_start()        #start the loop    while Connected != True:    #Wait for connection      time.sleep(0.1)    client.subscribe("python/test")    try:      while True:          time.sleep(1)    except KeyboardInterrupt:      print "exiting"      client.disconnect()      client.loop\_stop() |

## ****Testing the code****

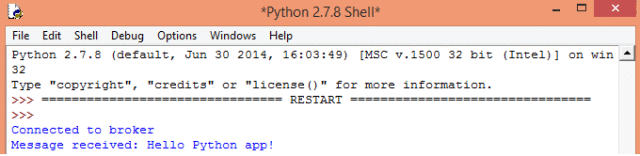
For the testing part, we can use IDLE, the Python IDE. So, after finishing the code, run it. As we defined, a message should be printed after the connection to the broker is established.

To test the subscription to the topic, we will be using an application that can publish to the topic where the Python program subscribed. This application will be [MQTTlens](https://chrome.google.com/webstore/detail/mqttlens/hemojaaeigabkbcookmlgmdigohjobjm). So, open it and connect to the broker. Then, publish a message to the “python/test” topic, as shown in figure 1.



**Figure 1** – Publishing message to MQTT topic.

You should get an output similar to figure 2 on IDLE, where the Python application is running.



**Figure 2** – Receiving message from subscribed MQTT topic.

6次循环

for x in range(6):  
  print(x)

fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
  if x == "banana":  
    break  
  print(x)

## Comments

Python has commenting capability for the purpose of in-code documentation.

Comments start with a #, and Python will render the rest of the line as a comment:

### Example

Comments in Python:

#This is a comment.  
print("Hello, World!")

### Docstrings

Python also has extended documentation capability, called docstrings.

Docstrings can be one line, or multiline.

Python uses triple quotes at the beginning and end of the docstring:

### Example

Docstrings are also comments:

"""This is a   
multiline docstring."""  
print("Hello, World!")

## Complex

Complex numbers are written with a "j" as the imaginary part:

### Example

Complex:

x = 3+5j  
y = 5j  
z = -5j  
  
print(type(x))  
print(type(y))  
print(type(z))

## Specify a Variable Type

There may be times when you want to specify a type on to a variable. This can be done with casting. Python is an object-orientated language, and as such it uses classes to define data types, including its primitive types.

Casting in python is therefore done using constructor functions:

* int() - constructs an integer number from an integer literal, a float literal (by rounding down to the previous whole number) literal, or a string literal (providing the string represents a whole number)
* float() - constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)
* str() - constructs a string from a wide variety of data types, including strings, integer literals and float literals

### Example

Integers:

x = int(1)   # x will be 1  
y = int(2.8) # y will be 2  
z = int("3") # z will be 3

## Python Collections (Arrays)

There are four collection data types in the Python programming language:

* **List** is an collection which is ordered and changeable. Allows duplicate members.
* **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
* **Set** is a collection which is unordered and unindexed. No duplicate members.
* **Dictionary** is a collection which is unordered, changeable and indexed. No duplicate members.

When choosing a collection type, it is useful to understand the properties of that type. Choosing the right type for a particular data set could mean retention of meaning, and, it could mean an increase in efficiency or security.

## List

A list is a collection which is ordered and changeable. In Python lists are written with square brackets.

### Example

Create a List:

thislist = ["apple", "banana", "cherry"]  
print(thislist)

## Tuple

A tuple is a collection which is ordered and unchangeable. In Python tuples are written with round brackets.

### Example

Create a Tuple:

thistuple = ("apple", "banana", "cherry")  
print(thistuple)

## Set

A set is a collection which is unordered and unindexed. In Python sets are written with curly brackets.

### Example

Create a Set::

thisset = {"apple", "banana"}  
print(thisset)

## The set() Constructor

It is also possible to use the set() constructor to make a set. You can use the add() object method to add an item, and the remove() object method to remove an item from the set. The len() function returns the size of the set.

### Example

Using the set() constructor to make a set:

thisset = set(("apple", "banana", "cherry")) # note the double round-brackets  
print(thisset)

## Dictionary

A dictionary is a collection which is unordered, changeable and indexed. In Python dictionaries are written with curly brackets, and they have keys and values.

### Example

Using the len() method to return the number of items:

thisdict = {  
  "apple": "green",  
  "banana": "yellow",  
  "cherry": "red"  
}  
print(thisdict)

## File

It is good practice to use the [**with**](https://docs.python.org/2/reference/compound_stmts.html#with) keyword when dealing with file objects. This has the advantage that the file is properly closed after its suite finishes, even if an exception is raised on the way. It is also much shorter than writing equivalent [**try**](https://docs.python.org/2/reference/compound_stmts.html#try)-[**finally**](https://docs.python.org/2/reference/compound_stmts.html#finally) blocks:

**>>> with** open('workfile', 'r') **as** f:

**...**  read\_data = f.read()

**>>>** f.closed

True

File objects have some additional methods, such as [**isatty()**](https://docs.python.org/2/library/stdtypes.html#file.isatty) and [**truncate()**](https://docs.python.org/2/library/stdtypes.html#file.truncate) which are less frequently used; consult the Library Reference for a complete guide to file objects.

The [**with**](https://docs.python.org/2/reference/compound_stmts.html#with) statement is used to wrap the execution of a block with methods defined by a context manager (see section [With Statement Context Managers](https://docs.python.org/2/reference/datamodel.html#context-managers)). This allows common [**try**](https://docs.python.org/2/reference/compound_stmts.html#try)…[**except**](https://docs.python.org/2/reference/compound_stmts.html#except)…[**finally**](https://docs.python.org/2/reference/compound_stmts.html#finally) usage patterns to be encapsulated for convenient reuse.

## MQTT publish Image

### 用Base64 encode避免internet传送时出现意外字符

### 转化为Json来做internet传输

## ML in Python

### 安装scipy, Quandl, sklearn

sudo python –m pip install scipy, Quandl, sklearn

python -m lets you run modules as scripts. If your module is just one .py file it'll be executed (which usually means code under if \_\_name\_\_ == '\_\_main\_\_'). If your module is a directory, Python will look for \_\_main\_\_.py (next to \_\_init\_\_.py) and will run it.

One of Python's mottoes is "batteries included", and this goes for python -m as well. Here are some (all?) of the gems hidden in the standard library. Sadly not all of them have help, but I poked around in the source code to see the usage.

## ~/.profile

On UNIX like systems, everything runs off a shell. A shell/terminal/command line is what runs evey program on \*NIX systems. When the user first logs in, there is a shell that is first launched then the login manager is executed inside that shell. This shell that is invoked at the time of login is called **login shell**.

Login shell is the parent process. After you login, everything that you see, the graphics the UI, and all the progams that you start will naturally be child processes of the login manager which in turn is a child process of a login shell.

When a Bourne compatible login shell starts, it first sources (parse & execute) the systemwide files like /etc/profile. Then it sources the user specific initialization file. Every user specific file resides in the particular user’s home directory which in most cases is /home/<user>/. So the file that login shell will source for user specific initialization is ~/.profile (~ is a shorthand for home directory).

The original bourne shell was just called sh.Now, there are many Bourne compatible shells like the most famous is bash, another awesome shell is zsh. So this applies to all sh compatible shells, they first source /etc/profile, then their own systemwide config (like /etc/bashrc or /etc/zshrc) then for user specific init they source ~/.profile then they source their own login configs (~/.bash\_profile or ~/.zprofile).

Here is a good article to read about different shell configs:

* [Bash - ArchWiki](https://wiki.archlinux.org/index.php/Bash#Configuration_files)
* [Zsh - ArchWiki](https://wiki.archlinux.org/index.php/Zsh#Startup.2FShutdown_files)

.profile file in Linux comes under the System startup files(defines user environment after reading the initialization files that you have set up when you log in to shell).

File like /etc/profile controls variables for profile of all users of the system whereas, .profile allows you to customize your own environment.

The .profile file is present in your home ($HOME) directory and lets you customize your individual working environment.

.profile file controls the following by default:

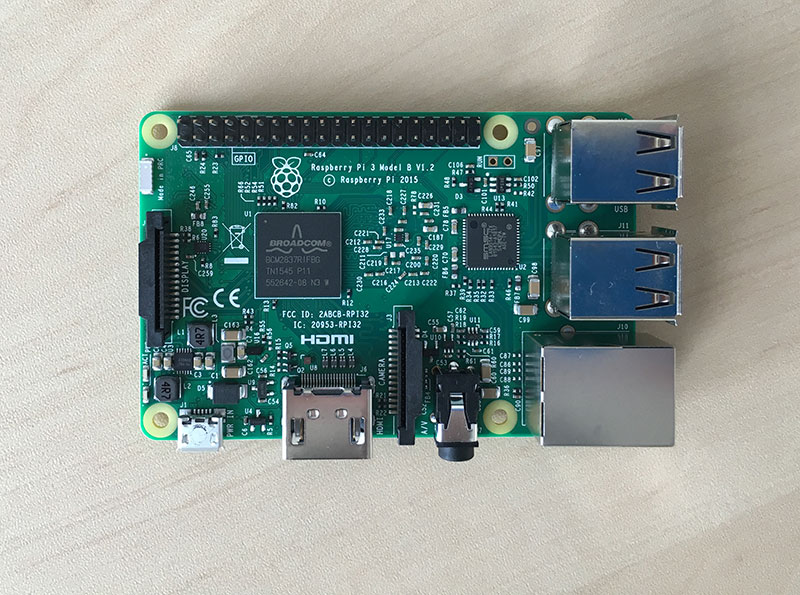
1. Shells to open
2. Prompt appearance
3. Keyboard Sound.
4. The .profile file contains your individual profile that overrides the variables set in the /etc/profile file. For more details refer to [IBM Knowledge Center](https://www.ibm.com/support/knowledgecenter/en/ssw_aix_71/com.ibm.aix.osdevice/prof_file.htm) and follow the contents and links there. Hope this helps.

# Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3

by [**Adrian Rosebrock**](https://www.pyimagesearch.com/author/adrian/) on April 18, 2016 in [**OpenCV 3**](https://www.pyimagesearch.com/category/opencv-3/), [**Raspberry Pi**](https://www.pyimagesearch.com/category/raspberry-pi/), [**Tutorials**](https://www.pyimagesearch.com/category/tutorials/)

[0](https://twitter.com/share?original_referer=https%3A%2F%2Fwww.pyimagesearch.com%2F&source=tweetbutton&text=Install+guide%3A+Raspberry+Pi+3+%2B+Raspbian+Jessie+%2B+OpenCV+3&url=https%3A%2F%2Fwww.pyimagesearch.com%2F2016%2F04%2F18%2Finstall-guide-raspberry-pi-3-raspbian-jessie-opencv-3%2F&via=pyimagesearch)

[0](https://www.linkedin.com/cws/share?url=https://www.pyimagesearch.com/2016/04/18/install-guide-raspberry-pi-3-raspbian-jessie-opencv-3/)



Can you believe it’s been over ***four years*** since the [original Raspberry Pi model B](http://www.amazon.com/gp/product/B009SQQF9C/ref=as_li_tl?ie=UTF8&camp=1789&creative=390957&creativeASIN=B009SQQF9C&linkCode=as2&tag=trndingcom-20&linkId=YZLP5GWTKJ5E7BGB) was released? Back then the Pi Model B shipped with only 256MB of RAM and a 700MHz single core processor.

Just over ***one year ago*** the [Raspberry Pi 2](http://www.amazon.com/gp/product/B008XVAVAW/ref=as_li_tl?ie=UTF8&camp=1789&creative=390957&creativeASIN=B008XVAVAW&linkCode=as2&tag=trndingcom-20&linkId=KGMCPNATKSKQHAZX) was unleashed on the world. And man, for something called a “Pi”, this beast made an impact on the computer world like an asteroid. This board sported 1GB of RAM and a 900MHz ***quad-core*** processor — quite the upgrade from the original single core, 700MHz system!

In my opinion, ***the Raspberry Pi 2 is what made computer vision possible on the Pi platform***(at least from a Python + OpenCV perspective). The original model B simply didn’t have the processing capacity (or the RAM) to be powerful enough to process images video streams for anything more than trivial operations — the Pi 2 changed all that.

In fact, the Raspberry Pi 2 had such a meaningful impact on the computer vision space, that I even took the time to make a all code examples in [*Practical Python and OpenCV*](https://www.pyimagesearch.com/practical-python-opencv/) compatible with the Pi.

And now we have the [Raspberry Pi 3](http://www.amazon.com/gp/product/B01C6FFNY4/ref=as_li_tl?ie=UTF8&camp=1789&creative=390957&creativeASIN=B01C6FFNY4&linkCode=as2&tag=trndingcom-20&linkId=M7C6OJOPQD3EOLIU):

* 1.2Ghz 64-bit quad-core processor.
* 1GB RAM.
* Integrated 802.11n wireless and bluetooth.

Personally, I was hoping for a bit more RAM (perhaps in the range of 1.5-2GB). But upgrading to a 64-bit processor with 33% increased performance is well worth it.

Just as I have done in previous blog posts, I’ll be demonstrating **how to install OpenCV 3 with Python bindings on Raspbian Jessie**.

If you are looking for previous installation instructions for different platforms, please consult this list:

* [How to install OpenCV 3.0 on ***Raspbian Jessie***.](https://www.pyimagesearch.com/2015/10/26/how-to-install-opencv-3-on-raspbian-jessie/)
* [Installing OpenCV on your Raspberry Pi Zero running ***Raspbian Jessie***.](https://www.pyimagesearch.com/2015/12/14/installing-opencv-on-your-raspberry-pi-zero/)
* [Installing OpenCV 3.0 for both Python 2.7 and Python 3+ on ***Raspbian Wheezy*.**](https://www.pyimagesearch.com/2015/07/27/installing-opencv-3-0-for-both-python-2-7-and-python-3-on-your-raspberry-pi-2/)
* [Install OpenCV 2.4 for Python 2.7 on ***Raspbian Wheezy***.](https://www.pyimagesearch.com/2015/02/23/install-opencv-and-python-on-your-raspberry-pi-2-and-b/)

Otherwise, let’s proceed with getting OpenCV 3 installed on your brand new Raspberry Pi 3!

## Assumptions

In this tutorial, I am going to assume that you already own a [**Raspberry Pi 3**](http://www.amazon.com/gp/product/B01C6FFNY4/ref=as_li_tl?ie=UTF8&camp=1789&creative=390957&creativeASIN=B01C6FFNY4&linkCode=as2&tag=trndingcom-20&linkId=KQ4HNH4UOHXJG44M) with [**Raspbian Jessie installed**](https://www.raspberrypi.org/downloads/raspbian/).

You should also have either:

* Physical access to your Raspberry Pi 3 so that you can open up a terminal and execute commands.
* Remote access via SSH.

I’ll be doing the majority of this tutorial via SSH, but as long as you have access to a terminal, you can easily follow along.

## Installing OpenCV 3 on a Raspberry Pi 3 running Raspbian Jessie

If you’ve ever installed OpenCV on a Raspberry Pi (or any other platform before), you know that the process can be quite time consuming with many dependencies and pre-requisites that have to be installed. **The goal of this tutorial is to thus guide you step-by-step through the compile and installation process.**

In order to make the installation process go more smoothly, I’ve included timings for each step so you know when to take a break, grab a cup of coffee, and checkup on email while the Pi compiles OpenCV. That said, the Pi 3 is substantially faster than the Pi 2, so the time it takes to compile OpenCV has decreased **dramatically.**

Anyway, let’s go ahead and get started installing OpenCV 3 on your brand new Raspberry Pi 3 running Raspbian Jessie.

### Step #1: Expand filesystem

Are you using a brand new install of Raspbian Jessie?

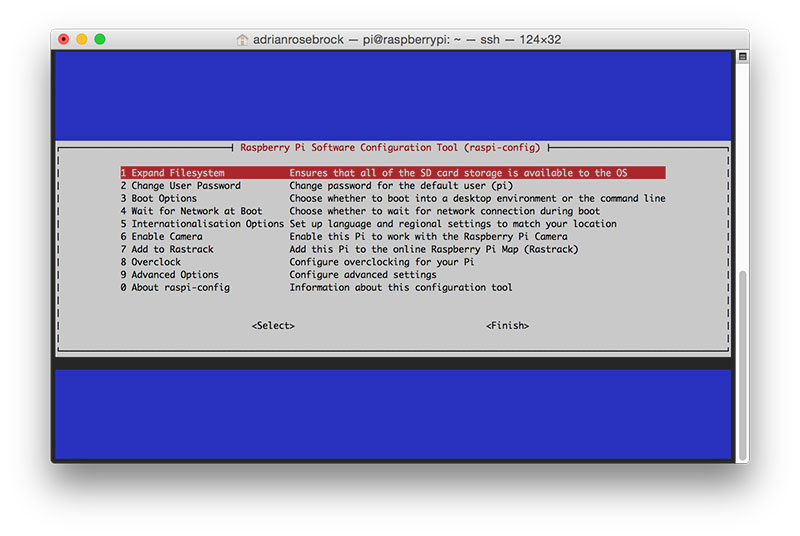
If so, the first thing you should do is expand your filesystem to include all available space on your micro-SD card:

Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3

Shell



|  |  |
| --- | --- |
| 1 | $ sudo raspi-config |



**Figure 1:** Expanding the filesystem on your Raspberry Pi 3.

Once prompted, you should select the first option, ***“1. Expand File System”***, ***hit Enter*** on your keyboard, arrow down to the ***“<Finish>”*** button, and then reboot your Pi:

Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3

Shell



|  |  |
| --- | --- |
| 1 | $ sudo reboot |

After rebooting, your file system should have been expanded to include all available space on your micro-SD card. You can verify that the disk has been expanded by executing df -h  and examining the output:

Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3

Shell



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | $ df -h  Filesystem      Size  Used Avail Use% Mounted on  /dev/root       7.2G  3.3G  3.6G  48% /  devtmpfs        459M     0  459M   0% /dev  tmpfs           463M     0  463M   0% /dev/shm  tmpfs           463M  6.4M  457M   2% /run  tmpfs           5.0M  4.0K  5.0M   1% /run/lock  tmpfs           463M     0  463M   0% /sys/fs/cgroup  /dev/mmcblk0p1   60M   20M   41M  34% /boot  tmpfs            93M     0   93M   0% /run/user/1000 |

As you can see, my Raspbian filesystem has been expanded to include all 8GB of the micro-SD card.

However, even with my filesystem expanded, I have already used 48% of my 8GB card!

OpenCV, along with all its dependencies, will need a few gigabytes during the compile, so you should delete the Wolfram engine to free up some space on your Pi:

Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3

Shell



|  |  |
| --- | --- |
| 1 | $ sudo apt-get purge wolfram-engine |

**After removing the Wolfram Engine, you can reclaim almost 700mb!**

### Step #2: Install dependencies

[This isn’t the first time I’ve discussed how to install OpenCV on the Raspberry Pi](https://www.pyimagesearch.com/opencv-tutorials-resources-guides/), so I’ll keep these instructions on the briefer side, allowing you to work through the installation process: I’ve also included the ***amount of time it takes to execute each command*** so you can plan your OpenCV + Raspberry Pi 3 install accordingly (OpenCV itself takes **1h 12m** to compile).

The first step is to update and upgrade any existing packages:

Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3

Shell



|  |  |
| --- | --- |
| 1  2 | $ sudo apt-get update  $ sudo apt-get upgrade |

**Timing: 1m 26s**

We then need to install some developer tools, including [CMake](https://cmake.org/), which helps us configure the OpenCV build process:

Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3

Shell



|  |  |
| --- | --- |
| 1 | $ sudo apt-get install build-essential cmake pkg-config |

**Timing: 40s**

Next, we need to install some image I/O packages that allow us to load various image file formats from disk. Examples of such file formats include JPEG, PNG, TIFF, etc.:

Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3

Python



|  |  |
| --- | --- |
| 1 | $ sudo apt-get install libjpeg-dev libtiff5-dev libjasper-dev libpng12-dev |

**Timing: 32s**

Just as we need image I/O packages, we also need video I/O packages. These libraries allow us to read various video file formats from disk as well as work directly with video streams:

Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3

Shell



|  |  |
| --- | --- |
| 1  2 | $ sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev  $ sudo apt-get install libxvidcore-dev libx264-dev |

**Timing: 34s**

The OpenCV library comes with a sub-module named highgui  which is used to display images to our screen and build basic GUIs. In order to compile the highgui  module, we need to install the GTK development library:

Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3

Shell



|  |  |
| --- | --- |
| 1 | $ sudo apt-get install libgtk2.0-dev |

**Timing: 3m 6s**

Many operations inside of OpenCV (namely matrix operations) can be optimized further by installing a few extra dependencies:

Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3

Shell



|  |  |
| --- | --- |
| 1 | $ sudo apt-get install libatlas-base-dev gfortran |

**Timing: 46s**

These optimization libraries are especially important for resource constrained devices such as the Raspberry Pi.

Lastly, let’s install both the Python 2.7 and Python 3 header files so we can compile OpenCV with Python bindings:

Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3

Shell



|  |  |
| --- | --- |
| 1 | $ sudo apt-get install python2.7-dev python3-dev |

**Timing: 45s**

If you skip this step, you may notice an error related to the Python.h  header file not being found when running make  to compile OpenCV.

### Step #3: Download the OpenCV source code

Now that we have our dependencies installed, let’s grab the 3.1.0  archive of OpenCV from the official [OpenCV repository](https://github.com/Itseez/opencv). (Note: As future versions of openCV are released, you can replace 3.1.0  with the latest version number):

Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3

Shell



|  |  |
| --- | --- |
| 1  2  3 | $ cd ~  $ wget -O opencv.zip https://github.com/Itseez/opencv/archive/3.1.0.zip  $ unzip opencv.zip |

**Timing: 1m 26s**

We’ll want the full install of OpenCV 3 ([to have access to features such as SIFT and SURF](https://www.pyimagesearch.com/2015/07/16/where-did-sift-and-surf-go-in-opencv-3/), for instance), so we also need to grab the [opencv\_contrib](https://github.com/itseez/opencv_contrib) repository as well:

Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3

Shell



|  |  |
| --- | --- |
| 1  2 | $ wget -O opencv\_contrib.zip https://github.com/Itseez/opencv\_contrib/archive/3.1.0.zip  $ unzip opencv\_contrib.zip |

**Timing: 43s**

You might need to expand the command above using the “<=>” button during your copy and paste. The .zip  in the 3.1.0.zip  may appear to be cutoff in some browsers. The full URL of the OpenCV 3.1.0 archive is:

<https://github.com/Itseez/opencv_contrib/archive/3.1.0.zip>

***Note:*** Make sure your *opencv*  and *opencv\_contrib*  versions are the same (in this case,*3.1.0* ). If the versions numbers do not match up, then you’ll likely run into either compile-time or runtime.

### Step #4: Python 2.7 or Python 3?

Before we can start compiling OpenCV on our Raspberry Pi 3, we first need to install pip , a Python package manager:

Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3

Shell



|  |  |
| --- | --- |
| 1  2 | $ wget https://bootstrap.pypa.io/get-pip.py  $ sudo python get-pip.py |

**Timing: 20s**

If you’re a longtime PyImageSearch reader, then you’ll know that I’m a huge fan of both [virtualenv](https://virtualenv.pypa.io/en/latest/) and [virtualenvwrapper](https://virtualenvwrapper.readthedocs.org/en/latest/). Installing these packages is not a requirement and you can absolutely get OpenCV installed without them, but that said, ***I highly recommend you install them*** as other PyImageSearch tutorials in the future will also leverage Python virtual environments. I’ll also be assuming that you have both virtualenv  and virtualenvwrapper  installed throughout the remainder of this guide.

**So, given that, what’s the point of using**virtualenv**and**virtualenvwrapper**?**

First, it’s important to understand that a virtual environment is a special tool used to keep the dependencies required by different projects in separate places by creating isolated, independent Python environments for each of them.

In short, it solves the “Project X depends on version 1.x, but Project Y needs 4.x” dilemma. It also keeps your global site-packages  neat, tidy, and free from clutter.

If you would like a full explanation on why Python virtual environments are good practice, absolutely [give this excellent blog post on RealPython a read](https://realpython.com/blog/python/python-virtual-environments-a-primer/).

It’s ***standard practice*** in the Python community to be using virtual environments of some sort, so I highly recommend that you do the same:

Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3

Shell



|  |  |
| --- | --- |
| 1  2 | $ sudo pip install virtualenv virtualenvwrapper  $ sudo rm -rf ~/.cache/pip |

**Timing: 9s**

Now that both virtualenv  and virtualenvwrapper  have been installed, we need to update our ~/.profile  file to include the following lines at the bottom of the file:

Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3

Shell



|  |  |
| --- | --- |
| 1  2  3 | # virtualenv and virtualenvwrapper  export WORKON\_HOME=$HOME/.virtualenvs  source /usr/local/bin/virtualenvwrapper.sh |

In previous tutorials, I’ve recommended using your favorite terminal-based text editor such asvim , emacs , or nano  to update the ~/.profile  file. If you’re comfortable with these editors, go ahead and update the file to reflect the changes mentioned above.

Otherwise, you should simply use cat  and output redirection to handle updating~/.profile :

Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3

Shell



|  |  |
| --- | --- |
| 1  2  3 | $ echo -e "\n# virtualenv and virtualenvwrapper" >> ~/.profile  $ echo "export WORKON\_HOME=$HOME/.virtualenvs" >> ~/.profile  $ echo "source /usr/local/bin/virtualenvwrapper.sh" >> ~/.profile |

Now that we have our ~/.profile  updated, we need to reload it to make sure the changes take affect. You can force a reload of your ~/.profile  file by:

1. Logging out and then logging back in.
2. Closing a terminal instance and opening up a new one
3. Or my personal favorite, ***just use the****source****command:***

Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3

Shell



|  |  |
| --- | --- |
| 1 | $ source ~/.profile |

***Note:*** I recommend running the *source ~/.profile*  file **each time** you open up a new terminal to ensure your system variables have been setup correctly.

#### Creating your Python virtual environment

Next, let’s create the Python virtual environment that we’ll use for computer vision development:

Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3

Shell



|  |  |
| --- | --- |
| 1 | $ mkvirtualenv cv -p python2 |

This command will create a new Python virtual environment named cv  using ***Python 2.7***.

If you instead want to use ***Python 3***, you’ll want to use this command instead:

Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3

Shell



|  |  |
| --- | --- |
| 1 | $ mkvirtualenv cv -p python3 |

Again, ***I can’t stress this point enough:*** the cv  Python virtual environment is ***entirely independent and sequestered*** from the default Python version included in the download of Raspbian Jessie. Any Python packages in the global site-packages  directory will not be available to the cv  virtual environment. Similarly, any Python packages installed in site-packages  of cv  will not be available to the global install of Python. Keep this in mind when you’re working in your Python virtual environment and it will help avoid a lot of confusion and headaches.

#### How to check if you’re in the “cv” virtual environment

If you ever reboot your Raspberry Pi; log out and log back in; or open up a new terminal, you’ll need to use the workon  command to re-access the cv  virtual environment. In previous blog posts, I’ve seen readers use the mkvirtualenv  command — ***this is entirely unneeded!*** The mkvirtualenv  command is meant to be executed only once: to actually create the virtual environment.

After that, you can use workon  and you’ll be dropped down into your virtual environment:

Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3

Shell



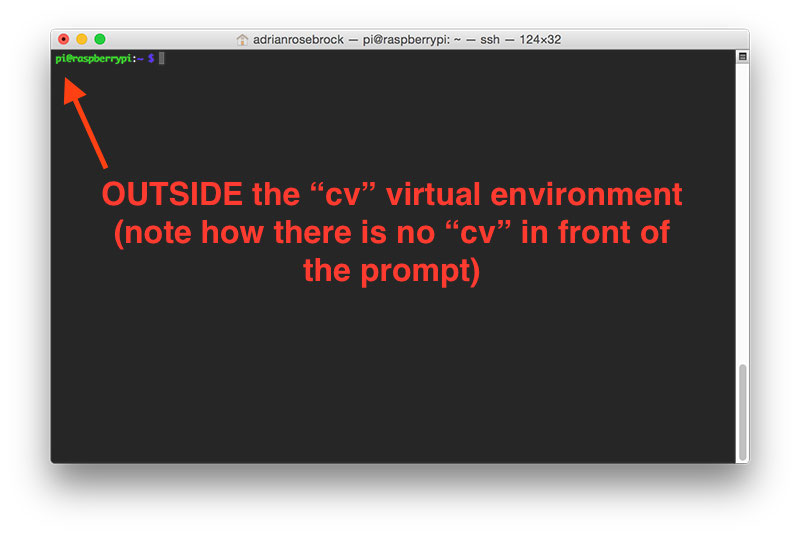
|  |  |
| --- | --- |
| 1  2 | $ source ~/.profile  $ workon cv |

To validate and ensure you are in the cv  virtual environment, examine your command line — if you see the text *(cv)*  preceding your prompt, then you ***are*** in the *cv*  virtual environment:



**Figure 2:** Make sure you see the “(cv)” text on your prompt, indicating that you ***are*** in the cv virtual environment.

Otherwise, if you ***do not*** see the (cv)  text, then you ***are not*** in the cv  virtual environment:



**Figure 3:** If you do not see the “(cv)” text on your prompt, then you ***are not*** in the cv virtual environment and need to run “source” and “workon” to resolve this issue.

To fix this, simply execute the source  and workon  commands mentioned above.

### Installing NumPy on your Raspberry Pi

Assuming you’ve made it this far, you should now be in the cv  virtual environment (which you should stay in for the rest of this tutorial). Our only Python dependency is [NumPy](http://www.numpy.org/), a Python package used for numerical processing:

Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3

Shell



|  |  |
| --- | --- |
| 1 | $ pip install numpy |

**Timing: 9m 39s**

Be sure to grab a cup of coffee or go for a nice walk, the NumPy installation can take a bit of time.

***Note:*** Another question I’ve often seen is ***“Help, my NumPy installation has hung and it’s not installing!”*** Actually, it is installing, it just takes time to pull down the sources and compile. Be patient. The Raspberry Pi isn’t as fast as your laptop/desktop.

### Step #5: Compile and Install OpenCV

We are now ready to compile and install OpenCV! Double-check that you are in the cv  virtual environment by examining your prompt (you should see the (cv)  text preceding it), and if not, simply execute workon :

Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3

Shell



|  |  |
| --- | --- |
| 1 | $ workon cv |

Once you have ensured you are in the cv  virtual environment, we can setup our build using CMake:

Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3

Shell



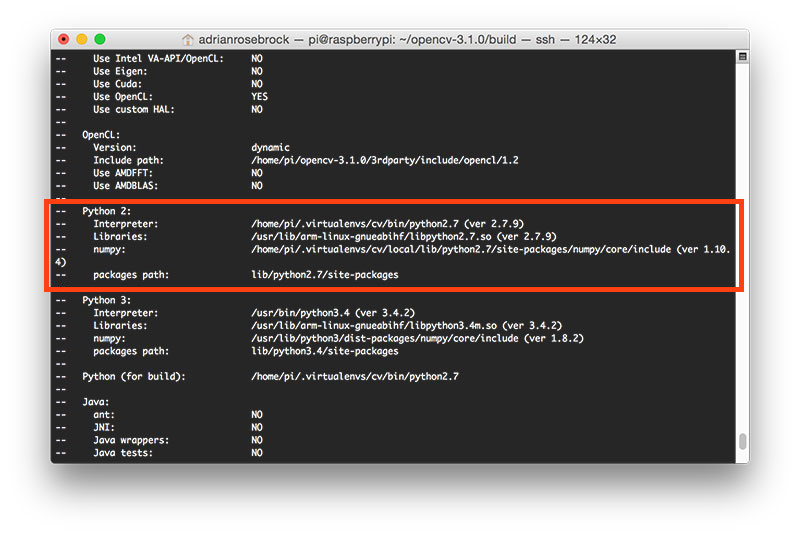
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | $ cd ~/opencv-3.1.0/  $ mkdir build  $ cd build  $ cmake -D CMAKE\_BUILD\_TYPE=RELEASE \      -D CMAKE\_INSTALL\_PREFIX=/usr/local \      -D INSTALL\_PYTHON\_EXAMPLES=ON \      -D OPENCV\_EXTRA\_MODULES\_PATH=~/opencv\_contrib-3.1.0/modules \      -D BUILD\_EXAMPLES=ON .. |

**Timing: 1m 57s**

Now, before we move on to the actual compilation step, ***make sure you examine the output of CMake!***

Start by scrolling down the section titled Python 2  and Python 3 .

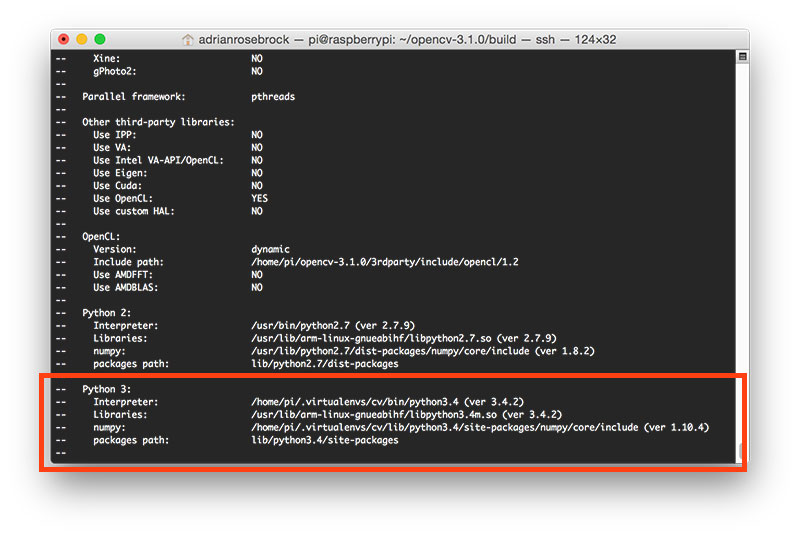
***If you are compiling OpenCV 3 for Python 2.7***, then make sure your Python 2  section includes valid paths to the Interpreter , Libraries , numpy  and packages path , similar to my screenshot below:



**Figure 4:** Ensuring that Python 2.7 will be used when compiling OpenCV 3 for Raspbian Jessie on the Raspberry Pi 3.

Notice how the Interpreter  points to our python2.7  binary located in the cv  virtual environment. The numpy  variable also points to the NumPy installation in the cv  environment.

Similarly, ***if you’re compiling OpenCV for Python 3***, make sure the Python 3  section looks like the figure below:



**Figure 5:** Checking that Python 3 will be used when compiling OpenCV 3 for Raspbian Jessie on the Raspberry Pi 3.

Again, the Interpreter  points to our python3.4  binary located in the cv  virtual environment while numpy  points to our NumPy install.

In either case, if you **do not** see the cv  virtual environment in these variables paths, ***it’s almost certainly because you are NOT in the****cv****virtual environment prior to running CMake!***

If this is the case, access the cv  virtual environment using workon cv  and re-run thecmake  command outlined above.

Finally, we are now ready to compile OpenCV:

Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3

Shell



|  |  |
| --- | --- |
| 1 | $ make -j4 |

**Timing: 1h 12m**

***Note:*** Compiling OpenCV in 72 minutes on the Raspberry Pi 3 is a ***24%*** improvement over the previous 95 minutes for the Raspberry Pi 2. That extra 300MHz makes a big difference!

The -j4  command controls the number of cores to leverage when compiling OpenCV 3. The Raspberry Pi 3 has four cores, thus we supply a value of 4  to allow OpenCV to compile faster.

However, due to race conditions, there are times when make  errors out when using multiple cores. If this happens to you, I suggest starting the compilation over again and using only onecore:

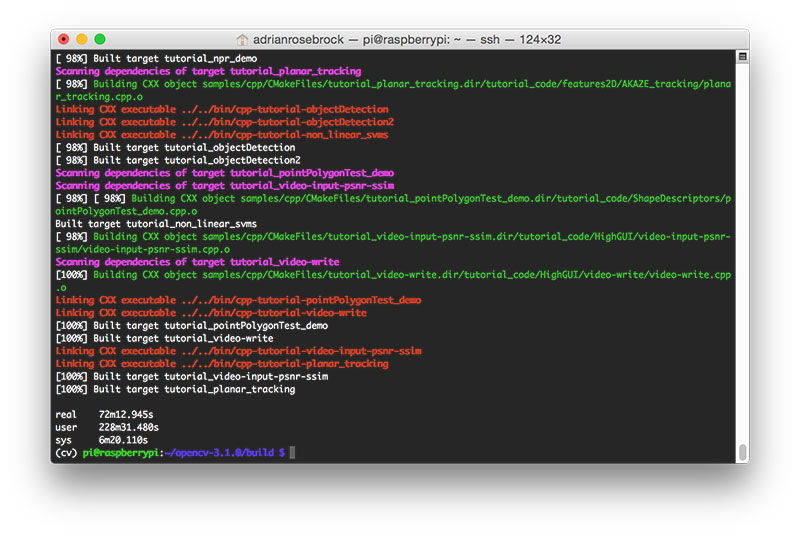
Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3

Shell



|  |  |
| --- | --- |
| 1  2 | $ make clean  $ make |

Once OpenCV 3 has finished compiling, your output should look similar to mine below:



**Figure 5:** Our OpenCV 3 compile on Raspbian Jessie has completed successfully.

From there, all you need to do is install OpenCV 3 on your Raspberry Pi 3:

Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3

Shell



|  |  |
| --- | --- |
| 1  2 | $ sudo make install  $ sudo ldconfig |

**Timing: 52s**

### Step #6: Finish installing OpenCV on your Pi

We’re almost done — just a few more steps to go and you’ll be ready to use your Raspberry Pi 3 with OpenCV 3.

#### For Python 2.7:

Provided your **Step #5** finished without error, OpenCV should now be installed in/usr/local/lib/python2.7/site-pacakges . You can verify this using the ls  command:

Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3

Shell



|  |  |
| --- | --- |
| 1  2  3 | $ ls -l /usr/local/lib/python2.7/site-packages/  total 1852  -rw-r--r-- 1 root staff 1895772 Mar 20 20:00 cv2.so |

***Note:*** In some cases, OpenCV can be installed in */usr/local/lib/python2.7/dist-packages*  (note the *dist-packages*  rather than *site-packages* . If you do not find the *cv2.so*  bindings in *site-packages* , we be sure to check *dist-packages* .

Our final step is to [sym-link](https://en.wikipedia.org/wiki/Symbolic_link) the OpenCV bindings into our cv  virtual environment for Python 2.7:

Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3

Shell



|  |  |
| --- | --- |
| 1  2 | $ cd ~/.virtualenvs/cv/lib/python2.7/site-packages/  $ ln -s /usr/local/lib/python2.7/site-packages/cv2.so cv2.so |

#### For Python 3:

After running make install , your OpenCV + Python bindings should be installed in/usr/local/lib/python3.4/site-packages . Again, you can verify this with the ls  command:

Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3

Shell



|  |  |
| --- | --- |
| 1  2  3 | $ ls -l /usr/local/lib/python3.4/site-packages/  total 1852  -rw-r--r-- 1 root staff 1895932 Mar 20 21:51 cv2.cpython-34m.so |

I honestly don’t know why, perhaps it’s a bug in the CMake script, but when compiling OpenCV 3 bindings for Python 3+, the output .so  file is named cv2.cpython-34m.so  (or some variant of) rather than simply cv2.so  (like in the Python 2.7 bindings).

Again, I’m not sure exactly why this happens, but it’s an easy fix. All we need to do is rename the file:

Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3

Shell



|  |  |
| --- | --- |
| 1  2 | $ cd /usr/local/lib/python3.4/site-packages/  $ sudo mv cv2.cpython-34m.so cv2.so |

After renaming to cv2.so , we can sym-link our OpenCV bindings into the cv  virtual environment for Python 3.4:

Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3

Shell



|  |  |
| --- | --- |
| 1  2 | $ cd ~/.virtualenvs/cv/lib/python3.4/site-packages/  $ ln -s /usr/local/lib/python3.4/site-packages/cv2.so cv2.so |

### Step #7: Testing your OpenCV 3 install

**Congratulations, you now have OpenCV 3 installed on your Raspberry Pi 3 running Raspbian Jessie!**

But before we pop the champagne and get drunk on our victory, let’s first verify that your OpenCV installation is working properly.

Open up a new terminal, execute the source  and workon  commands, and then finally attempt to import the Python + OpenCV bindings:

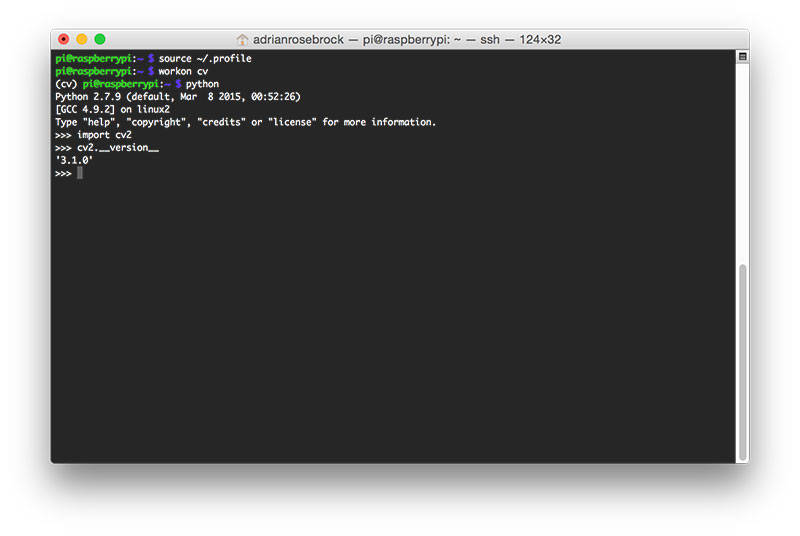
Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3

Shell



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | $ source ~/.profile  $ workon cv  $ python  >>> import cv2  >>> cv2.\_\_version\_\_  '3.1.0'  >>> |

As you can see from the screenshot of my own terminal, **OpenCV 3 has been successfully installed on my Raspberry Pi 3 + Python 2.7 environment:**



**Figure 5:** Confirming OpenCV 3 has been successfully installed on my Raspberry Pi 3 running Raspbian Jessie.

Once OpenCV has been installed, you can remove both the opencv-3.1.0  andopencv\_contrib-3.1.0  directories to free up a bunch of space on your disk:

Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3

Shell



|  |  |
| --- | --- |
| 1 | $ rm -rf opencv-3.1.0 opencv\_contrib-3.1.0 |

However, be cautious with this command! Make sure OpenCV has been properly installed on your system before blowing away these directories. A mistake here could cost you ***hours*** in compile time.

## Troubleshooting and FAQ

***Q.*** When I try to execute mkvirtualenv  and workon , I get a “command not found error”.

***A.*** There are three reasons why this could be happening, all of them related to **Step #4**:

1. Make certain that you have installed virtualenv  and virtualenvwrapper  via pip . You can check this by running pip freeze  and then examining the output, ensuring you see occurrences of both virtualenv  and virtualenvwrapper .
2. You might not have updated your ~/.profile  correctly. Use a text editor such as nano  to view your ~/.profile  file and ensure that the proper export  and source  commands are present (again, check **Step #4** for the contents that should be appended to ~/.profile .
3. You did not source  your ~/.profile  after editing it, rebooting, opening a new terminal, etc. Any time you open a new terminal and want to use a virtual environment, make sure you execute source ~/.profile  to load the contents — this will give you access to themkvirtualenv  and workon  commands.

***Q.*** After I open a new terminal, logout, or reboot my Pi, I cannot execute mkvirtualenv  orworkon .

***A.*** See **reason #3** from the previous question.

***Q.*** When I (1) open up a Python shell that imports OpenCV or (2) execute a Python script that calls OpenCV, I get an error: ImportError: No module named cv2 .

***A.*** Unfortunately, this error is extremely hard to diagnose, mainly because there are multiple issues that could be causing the problem. To start, make sure you are in the cv  virtual environment by using workon cv . If the workon  command fails, then see the first question in this FAQ. If you’re still getting an error, investigate the contents of the site-packages  directory for your cv  virtual environment. You can find the site-packages  directory in~/.virtualenvs/cv/lib/python2.7/site-packages/  or~/.virtualenvs/cv/lib/python3.4/site-packages/  (depending on which Python version you used for the install). Make sure that your sym-link to the cv2.so  file is valid and points to an existing file.

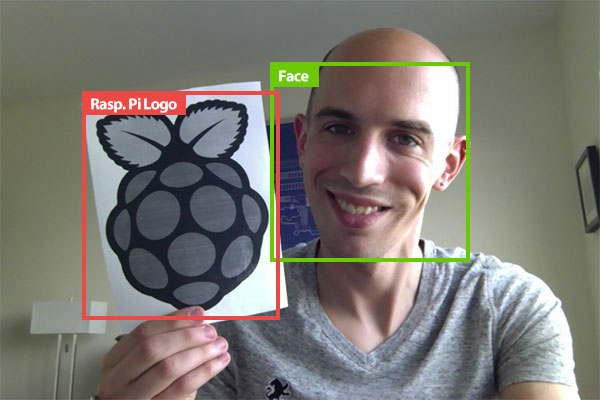
## So, what’s next?

Congrats! You have a brand new, fresh install of OpenCV on your Raspberry Pi — and I’m sure you’re just itching to leverage your Raspberry Pi to build some awesome computer vision apps.

But I’m also willing to bet that *you’re just getting started learning computer vision and OpenCV*, and you’re probably feeling a bit confused and overwhelmed on where exactly to start.

Personally, I’m a big fan of **learning by example**, so a good first step would be to read [this blog post](https://www.pyimagesearch.com/2015/03/30/accessing-the-raspberry-pi-camera-with-opencv-and-python/) on accessing your Raspberry Pi Camera with the picamera module. This tutorial details the *exact steps* you need to take to (1) capture photos from the camera module and (2) access the raw video stream.

And if you’re *really interested* in leveling-up your computer vision skills, you should definitely check out my book, [*Practical Python and OpenCV + Case Studies*](https://www.pyimagesearch.com/practical-python-opencv/?src=pi-opencv-install). My book not only *covers the basics of computer vision and image processing*, but also teaches you how to solve real world computer vision problems including ***face detection in images and video streams***, ***object tracking in video***, and ***handwriting recognition.***

[](https://www.pyimagesearch.com/practical-python-opencv/?src=pi-opencv-install)

**All code examples covered in the book are guaranteed to run on the Raspberry Pi 2 and Pi 3 as well!** Most programs will also run on the B+ and Zero models, but might be a bit slow due to the limited computing power of the B+ and Zero.

So let’s put your fresh install of OpenCV on your Raspberry Pi to good use — [***just click here***](https://www.pyimagesearch.com/practical-python-opencv/?src=pi-opencv-install)***to learn more about the real-world projects you can solve using your Raspberry Pi + Practical Python and OpenCV .***

## Summary

In this blog post, we learned how to install ***OpenCV 3*** with either Python 2.7 or Python 3 bindings on your ***Raspberry Pi 3*** running ***Raspbian Jessie***.

If you are running a different version of Raspbian (such as Raspbian Wheezy) or want to install a different version of OpenCV (such as OpenCV 2.4), please consult the following tutorials:

* [How to install OpenCV 3.0 on ***Raspbian Jessie***.](https://www.pyimagesearch.com/2015/10/26/how-to-install-opencv-3-on-raspbian-jessie/)
* [Installing OpenCV on your Raspberry Pi Zero running ***Raspbian Jessie***.](https://www.pyimagesearch.com/2015/12/14/installing-opencv-on-your-raspberry-pi-zero/)
* [Installing OpenCV 3.0 for both Python 2.7 and Python 3+ on ***Raspbian Wheezy*.**](https://www.pyimagesearch.com/2015/07/27/installing-opencv-3-0-for-both-python-2-7-and-python-3-on-your-raspberry-pi-2/)
* [Install OpenCV 2.4 for Python 2.7 on ***Raspbian Wheezy***.](https://www.pyimagesearch.com/2015/02/23/install-opencv-and-python-on-your-raspberry-pi-2-and-b/)

## But before you go…

I tend to utilize the Raspberry Pi quite a bit on this blog, so if you’re interested in learning more about the Raspberry Pi + computer vision, ***enter your email address in the form below to be notified when these posts go live!***