pyimagesearch | gurus

Chttps://gurus.pyimagesearch.com

Setting up your Python + OpenCV development environment

🏠 (HTTPS://GURUS.PYIMAGESEARCH.COM) › SETTING UP YOUR PYTHON + OPENCV DEVELOPMENT ENVIRONMENT

This tutorial will guide you through setting up Python, OpenCV, and other necessary libraries and packages we will need inside the PyImageSearch Gurus course on both ***OSX*** and ***Ubuntu***. The Windows OS is not supported for this course and if you intend on using Windows, ***I highly recommend*** that you download and use the PyImageSearch Gurus virtual machine (https://gurus.pyimagesearch.com/pyimagesearch-virtual-machine/) instead.

- **Click here to setup your OSX development environment.**
- **Click here to setup your Ubuntu development environment.**

These instructions were **updated on December 3, 2018.** Please send @Adrian and @drhoffma a message in the Community (https://community.pyimagesearch.com/) if you encounter any problems.

OSX

This part of the tutorial will guide you through setting up Python, and OpenCV, and on OSX. The steps for this tutorial were gathered on a OSX 10.13.6 (High Sierra) machine, but these steps should work for all flavors of OSX 10.8 and above *(except Mojave.* ***Mojave is not recommended.*** *If you are using Mojave it is suggested that you use the PyImageSearch Gurus VM (https://gurus.pyimagesearch.com/pyimagesearch-virtual-machine/)).*

Step 1:

Before we get started installing our libraries, we'll first need to install XCode. I would suggest registering as an Apple Developer (https://developer.apple.com/register/index.action) (it's free) first to make the process easier.

From there, open up the *App Store* application and search for the XCode application. From there just click *Get* and *Install App* (and when prompted enter your Apple ID username and password):

(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/osx_install_xcode.jpg)

FIGURE 1: DOWNLOADING AND INSTALLING XCODE.

From here, XCode will automatically download and install — this process normally takes 15-20 minutes.

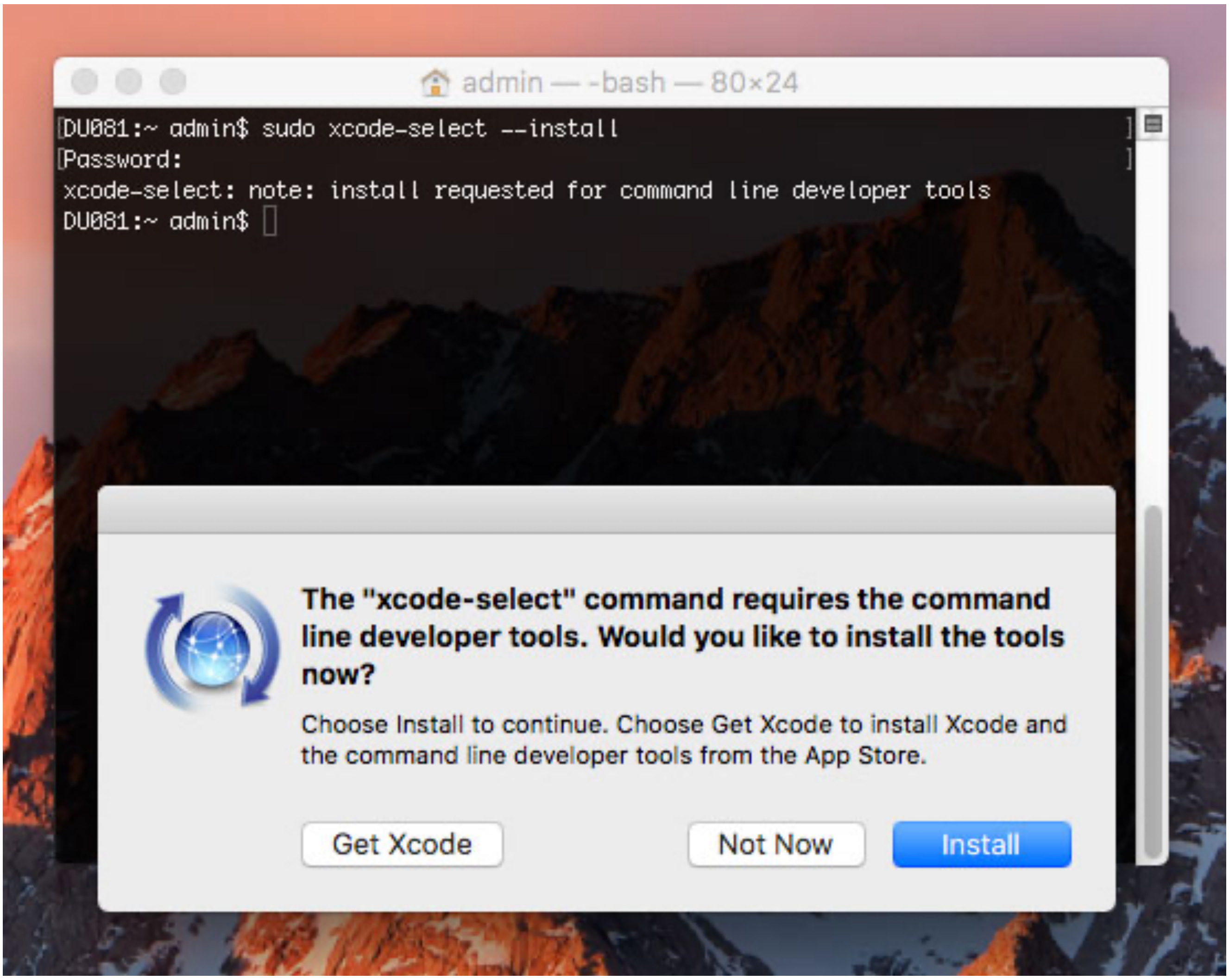
Go ahead and accept the Xcode license:

```
Setting up your OSX development environment                    Python
$ sudo xcodebuild -license
```

To accept the license, simply scroll down and accept it.

Once you've accepted the license agreement, let's install Apple Command Line Tools. **This is required,** so that you'll have `make` , `gcc` , `clang` , etc. You can install the tools via:

```
Setting up your OSX development environment                  Shell
$ sudo xcode-select --install
```



```
[DU081:~ admin$ sudo xcode-select --install
[Password:
xcode-select: note: install requested for command line developer tools
DU081:~ admin$
```

The "xcode-select" command requires the command line developer tools. Would you like to install the tools now?

Choose Install to continue. Choose Get Xcode to install Xcode and the command line developer tools from the App Store.

FIGURE 2: INSTALLING APPLE COMMAND LINE TOOLS ON MACOS.

Click the "*Install*" button and wait about 5 minutes for the installation to complete.

Step 2:

Now that XCode is installed, we need to install Homebrew (http://brew.sh/), which labeled as *"The missing package manager for OSX"* (and they are not joking). Think of Homebrew as an (almost) equivalent to `apt-get` for Ubuntu.

To install Homebrew, simply head to the Homebrew (http://brew.sh/) website and copy and paste the command underneath the "Install Homebrew" section:

Setting up your OSX development environment Shell

```
$ ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/in
```

Now that you have Homebrew installed, you need to update it and grab the latest package (i.e. "formula") definitions. These formula are simply instructions on how to install a given library or package.

To update Homebrew, simply do:

Setting up your OSX development environment Shell

```
$ brew update
```

Step 3:

Before we proceed we need to update our PATH in our `~/.bash_profile` file to indicate that we want to use Homebrew packages *before* any system libraries or packages. This is an *absolutely critical step, so be sure not to skip it!*

Open up your `~/.bash_profile` file in your favorite editor (if the file does not exist, create it), and append the following lines:

~/.bash_profile Shell

```
# Homebrew
```

```
export PATH=/usr/local/bin:$PATH
```

From there, update your `~/.bash_profile` file to ensure the changes have been made:

Setting up your OSX development environment Shell

```
$ source ~/.bash_profile
```

PyImageSearch Gurus is fully Python 3 supported. While you may see some Python 2.7 compatible code in the course, you really should use Python 3 these days.

It is *extremely* important to use at a max of Python 3.6 (3.5 is fine as well). By default High Sierra and Mojave are coming with Python 3.7 now. It sounds good, ***but*** Python 3.7 is unsupported by Keras/TensorFlow (both are used often on this blog) and thus are not a good choice for OpenCV either.

These commands will install Python 3.6.5_1:

Setting up your OSX development environment Shell

```
$ brew install https://raw.githubusercontent.com/Homebrew/homebrew-co
$ brew switch python 3.6.5_1
```

Be sure to copy the entire command + URL.

Let's verify:

Setting up your OSX development environment Python

```
$ python3
Python 3.6.5 (default, Jun 17 2018, 12:13:06)
[GCC 4.2.1 Compatible Apple LLVM 9.1.0 (clang-902.0.39.2)] on darwin
Type "help", "copyright", "credits" or "license" for more informatio
>>>
```

Great! I can see that we have Python 3.6.5 installed now.

Let's verify one more thing:

Setting up your OSX development environment Shell

```
$ which python3
/usr/local/bin/python3
```

If you see `/usr/local/bin/python3` you are using the *Homebrew* Python (which is what we desire). If you see `/usr/bin/python3` then you are using the *system* Python and you likely need to fix your bash profile and/or source it.

Take the time now to verify you are using the *Homebrew* version of Python and *not* the system version.

Before we go any further, let's install Python, wget, and cmake and other libraries via brew:

Setting up your OSX development environment Shell

```
$ brew install wget cmake
$ brew install jpeg libpng libtiff openexr
$ brew install eigen tbb
```

Step 4:

Alright, time to get `virtualenv` and `virtualenvwrapper` installed and configured correctly. These packages allow us to create separate Python environments for each project we are working on. This is especially useful if you have projects that require different (or conflicting) versions of a given library:

Setting up your OSX development environment Python

```
$ sudo pip3 install virtualenv virtualenvwrapper
```

Again, we need to update our `~/.bash_profile` file by appending the following lines:

~/.bash_profile Shell

```
# virtualenv and virtualenvwrapper
export WORKON_HOME=$HOME/.virtualenvs
export VIRTUALENVWRAPPER_PYTHON=/usr/local/bin/python3
source /usr/local/bin/virtualenvwrapper.sh
```

And finally we'll reload our `~/.bash_profile` file:

Setting up your OSX development environment Shell

```
$ source ~/.bash_profile
```

Now we are ready to make our `gurus` virtual environment which contains Python 3:

Setting up your OSX development environment Shell

```
$ mkvirtualenv gurus -p python3
```

***For users that already have a previous gurus virtual environment**, you may do one of two things.*

1. Remove the environment (`rmvirtualenv gurus`) and then make a new one as below.
2. Create an additional environment called `gurus_new` and use `gurus_new` for the remainder of these instructions.

Step 5:

Finally! Let's start installing some Python packages. We need to install NumPy since the OpenCV Python bindings represent images as multi-dimensional NumPy arrays:

Setting up your OSX development environment Shell

```
$ workon gurus
$ pip install numpy
```

Step 6:

Compiling OpenCV from source is needed to use the latest OpenCV code or and to activate certain features (such as patented algorithms which we will cover in this course). Using pip, brew, or apt to install OpenCV is ***not*** recommended.

First, let's download the source code:

Setting up your OSX development environment Shell

```
$ cd ~
$ wget -O opencv.zip https://github.com/opencv/opencv/archive/3.4.4.z
$ wget -O opencv_contrib.zip https://github.com/opencv/opencv_contrib
```

Then unpack the archives:

Setting up your OSX development environment Shell

```
$ unzip opencv.zip
$ unzip opencv_contrib.zip
```

I also like to rename the directories and I highly recommend you do so:

Setting up your OSX development environment Python

```
$ mv opencv-3.4.4 opencv
$ mv opencv_contrib-3.4.4 opencv_contrib
```

If you skip renaming the directories, don't forget to update the CMake paths next.

Followed by configuring the build with CMake (it is very important that you copy the CMake command ***exactly*** as it appears here, taking care to copy and past the ***entire*** command; I would suggest clicking the "<=>" button in the toolbar below to expand the entire command):

Setting up your OSX development environment Shell

```
$ cd ~/opencv
$ mkdir build
$ cd build
$ workon gurus
$ cmake -D CMAKE_BUILD_TYPE=RELEASE \
    -D CMAKE_INSTALL_PREFIX=/usr/local \
    -D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib/modules \
    -D PYTHON3_LIBRARY=`python -c 'import subprocess ; import sys ;
    -D PYTHON3_INCLUDE_DIR=`python -c 'import distutils.sysconfig as
    -D PYTHON3_EXECUTABLE=$VIRTUAL_ENV/bin/python \
    -D BUILD_opencv_python2=OFF \
    -D BUILD_opencv_python3=ON \
    -D INSTALL_PYTHON_EXAMPLES=ON \
    -D INSTALL_C_EXAMPLES=OFF \
    -D OPENCV_ENABLE_NONFREE=ON \
    -D BUILD_EXAMPLES=ON ..
```

Note: *For the above CMake command, I spent considerable time creating, testing, and refactoring it. I'm confident that it will save you time and frustration if you use it exactly as it appears.* ***Make sure you click the "<=>" button in the toolbar of the code block above to expand the code block.*** *This will enable you to copy and paste the* ***entire*** *command.*

Inspect the CMake output to ensure that your Python 3 gurus virtual environment is being used for the Python 3 interpreter as well as NumPy.

Then we're ready to perform the compilation of OpenCV:

Setting up your OSX development environment Shell

```
$ make -j4
```

Note: *The number '4' above specifies that we have 4 cores/processors for compiling. If you have a different number of processors you can update the* `-j` *switch. For only one core/processor simply just use the* `make` *command (from the build directory enter* `make clean` *prior to retrying if your build failed or got stuck).*

From there you can install OpenCV:

Setting up your OSX development environment Shell

```
$ sudo make install
```

At this point, your Python 3 bindings for OpenCV should reside in the following folder:

Setting up your OSX development environment Shell

```
$ ls /usr/local/python/cv2/python-3.6
cv2.cpython-36m-darwin.so
```

Let's rename them to simply `cv2.so` :

Setting up your OSX development environment Shell

```
$ cd /usr/local/python/cv2/python-3.6
$ sudo mv cv2.cpython-36m-darwin.so cv2.so
```

Pro-tip: *If you are installing OpenCV 3 and OpenCV 4 alongside each other, instead of renaming the file to* `cv2.so` *, you might consider naming it* `cv2.opencv3.4.4.so` *and then in the next sub-step sym-link appropriately from that file to* `cv2.so` *as well.*

Our last sub-step is to sym-link our OpenCV `cv2.so` bindings into our `cv` virtual environment:

Setting up your OSX development environment Shell

```
$ cd ~/.virtualenvs/gurus/lib/python3.6/site-packages/
$ ln -s /usr/local/python/cv2/python-3.6/cv2.so cv2.so
$ cd ~
```

Note: *Replace python3.6 with your version of Python. To figure out your version you may enter* `python3 --version` *in the terminal.*

Step 7:

Finally, we can test out the install:

Setting up your OSX development environment Shell

```
$ python
>>> import cv2
>>> cv2.__version__
```

```
'3.4.4'
```

If `cv2` imports without error, then we are all set!

If your output properly shows the version of OpenCV that you installed, then you're ready to move on.

Step 8:

Let's go ahead and take a second to install some other Python libraries that we'll be using inside this course:

Setting up your OSX development environment Python

```
$ workon gurus
$ pip install scipy matplotlib
$ pip install scikit-learn
$ pip install scikit-image
$ pip install mahotas imutils Pillow json_minify
```

We'll be adding to this list of libraries as we go through the course, but this will be enough to get us started.

Step 9:

Finally, let's write a Python script to load an image from disk and display it on our screen.

We'll start by downloading the PyImageSearch Gurus logo image:

Setting up your OSX development environment Shell

```
$ cd ~
$ wget https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/pyi
```

This places the `pyimagesearch_gurus_logo.png` image in our home directory.

Now let's create a new file, name it `test_install.py`, and insert the following code:

test_install.py Python

```python
import cv2
image = cv2.imread("pyimagesearch_gurus_logo.png")
cv2.imshow("Test Image", image)
cv2.waitKey(0)
```

Save the file, exit, and then execute the following command:

test_install.py Shell

```shell
$ workon gurus
$ python test_install.py
```

And if all goes well you should see the PyImageSearch Gurus logo image displayed on your screen:

(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/osx_test_install.png)

FIGURE 2: TESTING OUT OUR PYTHON + OPENCV INSTALL ON UBUNTU BY DISPLAYING THE PYIMAGESEARCH GURUS LOGO IMAGE TO OUR SCREEN.

And there you have it! OpenCV + Python are now installed on your OSX system!

Next up, let's investigate how to setup our development environment on Ubuntu.

Linux/Ubuntu

Now that we have reviewed the installation instructions for OSX, let's move on to Ubuntu. Once you have completed the steps below, you will have a complete Ubuntu setup that is *identical* to the PyImageSearch Gurus virtual machine (https://gurus.pyimagesearch.com/pyimagesearch-virtual-machine/).

So what's the benefit of using your own custom install over the VM?

For one, you'll gain access to to your webcam and attached USB devices. Due to security reasons, VirtualBox does not allow for you to access your webcam/USB camera from within the virtual machine.

Secondly, you'll notice some substantial performance gains. While the Ubuntu virtual machine is more than fast enough to run the examples inside the PyImageSearch Gurus course, nothing beats the performance of running natively in a non-virtualized environment.

So with that said, is setting up your own development environment *required*?

Absolutely not — and certainly do not feel pressured to create your own development environment. The PyImageSearch Gurus virtual machine is *fully capable* of helping you get through this course.

But this guide is made available for you, just in case you want to (1) setup your own virtual machine or (2) install Python, OpenCV, and all other libraries natively on your system.

Anyway, let's go ahead and get started on setting up your own Ubuntu development environment. The steps for this tutorial were gathered **Ubuntu 16.04 LTS 64-bit**, but should work with the most recent versions of Ubuntu (18.04) as well.

Step 1:

Update the `apt-get` package manager and upgrade any pre-installed packages:

Setting up your Ubuntu development environment Shell

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

Step 2:

Install the required developer tools and packages:

Setting up your Ubuntu development environment Python

```
$ sudo apt-get install build-essential cmake unzip pkg-config
```

It is likely that `pkg-config` is already installed, but just in case it is not, be sure to include it in your `apt-get` command.

Step 3:

Install the necessary image I/O packages. These packages allow you to load various image file formats such as JPEG, PNG, TIFF, etc.

Setting up your Ubuntu development environment Shell

```
$ sudo apt-get install libjpeg-dev libtiff-dev libpng-dev
```

Now let's try to install `libjasper-dev` :

Setting up your Ubuntu development environment Shell

```
$ sudo apt-get install libjasper-dev
```

If you receive an error about `libjasper-dev` being missing then follow the following instructions:

Setting up your Ubuntu development environment Shell

```shell
sudo add-apt-repository "deb http://security.ubuntu.com/ubuntu xenial-security main"
sudo apt update
sudo apt install libjasper1 libjasper-dev
```

Otherwise (or once `libjasper-dev` is installed), keep going.

Step 4:

Install the GTK development library. This library is used to build Graphical User Interfaces (GUIs) and is required for the `highgui` library of OpenCV which allows you to view images on your screen:

Setting up your Ubuntu development environment — Python

```
$ sudo apt-get install libgtk-3-dev
```

Step 5:

Install the necessary video I/O packages. These packages are used to load video files using OpenCV:

Setting up your Ubuntu development environment — Shell

```
$ sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev
$ sudo apt-get install libxvidcore-dev libx264-dev
```

Step 6:

Install libraries that are used to optimize various operations within OpenCV:

Setting up your Ubuntu development environment — Shell

```
$ sudo apt-get install libatlas-base-dev gfortran
```

Step 7:

Now we can install the Python 3.6 development tools:

Setting up your Ubuntu development environment — Shell

```
$ sudo apt-get install python3.6-dev
```

Install `pip`, a Python package manager:

Setting up your Ubuntu development environment — Shell

```
$ wget https://bootstrap.pypa.io/get-pip.py
$ sudo python3 get-pip.py
```

Step 8:

Install `virtualenv` and `virtualenvwrapper` , which allow us to create separate Python environments for each project we are working on. This is especially useful if you have projects that require different (or conflicting) versions of a given library:

Setting up your Ubuntu development environment Shell

```shell
$ sudo pip install virtualenv virtualenvwrapper
$ sudo rm -rf ~/get-pip.py ~/.cache/pip
```

Then, append your `~/.bashrc` file to include the following lines:

Setting up your Ubuntu development environment Shell

```shell
# virtualenv and virtualenvwrapper
export WORKON_HOME=$HOME/.virtualenvs
export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3
source /usr/local/bin/virtualenvwrapper.sh
```

This will ensure that both `virtualenv` and `virtualenvwrapper` are loaded each time you login.

Reload the contents of your `~/.bashrc` file:

Setting up your Ubuntu development environment Shell

```shell
$ source ~/.bashrc
```

For users that already have a previous gurus virtual environment*, you may do one of two things.*

1. Remove the environment (`rmvirtualenv gurus`) and then make a new one as below.
2. Create an additional environment called `gurus_new` and use `gurus_new` for the remainder of these instructions.

Now we are ready to make our `gurus` virtual environment which contains Python 3:

Setting up your Ubuntu development environment Shell

```
$ mkvirtualenv gurus -p python3
```

Or you may have elected to keep your existing `gurus` environment and you want to create a new one:

Setting up your Ubuntu development environment Shell

```
$ mkvirtualenv gurus_new -p python3
```

For the remainder of these instructions we'll refer to `gurus` as our environment. If you named your new environment something else, such as `gurus_new` , then be sure to use it where needed.

Step 10:

We also need to install NumPy since the OpenCV Python bindings represent images as multi-dimensional NumPy arrays:

Setting up your Ubuntu development environment Shell

```
$ workon gurus
$ pip install numpy
```

Let's go ahead and take a second to install some other Python libraries that we'll be using inside this course:

Setting up your Ubuntu development environment Shell

```
$ pip install scipy matplotlib
$ pip install scikit-learn
$ pip install scikit-image
$ pip install mahotas imutils Pillow json_minify
```

We'll be adding to this list as we go through the course, but this will be good enough to get us started.

Step 11:

Compiling OpenCV from source is needed to use the latest OpenCV code or and to activate certain features (such as patented algorithms which we will cover in this course). Using pip, brew, or apt to install OpenCV is ***not*** recommended.

First, let's download the source code:

Setting up your Ubuntu development environment Shell

```
$ cd ~
$ wget -O opencv.zip https://github.com/opencv/opencv/archive/3.4.4.z
$ wget -O opencv_contrib.zip https://github.com/opencv/opencv_contrib
```

Then unpack the archives:

Setting up your Ubuntu development environment Shell

```
$ unzip opencv.zip
$ unzip opencv_contrib.zip
```

I also like to rename the directories and I highly recommend you do so:

Setting up your Ubuntu development environment Python

```
$ mv opencv-3.4.4 opencv
$ mv opencv_contrib-3.4.4 opencv_contrib
```

If you skip renaming the directories, don't forget to update the CMake paths next.

Setup the build:

Setting up your Ubuntu development environment Python

```
$ cd opencv
$ mkdir build
$ cd build
$ workon gurus
$ cmake -D CMAKE_BUILD_TYPE=RELEASE \
    -D CMAKE_INSTALL_PREFIX=/usr/local \
    -D WITH_CUDA=OFF \
    -D INSTALL_PYTHON_EXAMPLES=ON \
    -D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib/modules \
    -D OPENCV_ENABLE_NONFREE=ON \
    -D BUILD_EXAMPLES=ON ..
```

Inspect the CMake output to ensure that your Python 3 gurus virtual environment is being used for the Python 3 interpreter as well as NumPy.

Now we can finally compile OpenCV:

Setting up your Ubuntu development environment Shell

```
$ make -j4
```

Where the `4` can be replaced with the number of cores on your system which can speedup the compilation process.

And assuming that OpenCV compiled without error, you can now install it on your Ubuntu system:

Setting up your Ubuntu development environment Shell

```
$ sudo make install
$ sudo ldconfig
```

Step 12:

At this point, your Python 3 bindings for OpenCV should reside in the following folder:

Setting up your Ubuntu development environment Shell

```
$ ls /usr/local/python/cv2/python-3.6
cv2.cpython-36m-x86_64-linux-gnu.so
```

Let’s rename them to simply `cv2.so` :

Setting up your Ubuntu development environment Shell

```
$ cd /usr/local/python/cv2/python-3.6
$ sudo mv cv2.cpython-36m-x86_64-linux-gnu.so cv2.so
```

Pro-tip: *If you are installing OpenCV 3 and OpenCV 4 alongside each other, instead of renaming the file to* `cv2.so` *, you might consider naming it* `cv2.opencv4.0.0.so` *and then in the next sub-step sym-link appropriately from that file to* `cv2.so` *as well.*

Our last sub-step is to sym-link our OpenCV `cv2.so` bindings into our `cv` virtual environment:

Setting up your Ubuntu development environment Shell

```
$ cd ~/.virtualenvs/gurus/lib/python3.6/site-packages/
$ ln -s /usr/local/python/cv2/python-3.6/cv2.so cv2.so
$ cd ~
```

Step 13:

Finally, we can give our OpenCV and Python installation a test drive:

Setting up your Ubuntu development environment Python

```
$ workon gurus
$ python
>>> import cv2
>>> cv2.__version__
'3.4.4'
```

If `cv2` imports without error, then we are all set!

However, let's not stop there. Let's write a Python script to load an image from disk and display it on our screen.

We'll start by downloading the PyImageSearch Gurus logo image:

Setting up your Ubuntu development environment Python

```
$ cd ~
$ wget https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/pyi
```

This places the `pyimagesearch_gurus_logo.png` image in our home directory.

Now let's create a new file, name it `test_install.py` , and insert the following code:

test_install.py Python

```python
import cv2
image = cv2.imread("pyimagesearch_gurus_logo.png")
cv2.imshow("Test Image", image)
cv2.waitKey(0)
```

Save the file, exit, and then execute the following command:

test_install.py Shell

```shell
$ python test_install.py
```

And if all goes well you should see the PyImageSearch Gurus logo image displayed on your screen:

(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/vm_test_ubuntu_install.png)

FIGURE 3: TESTING OUT OUR PYTHON + OPENCV INSTALL ON UBUNTU BY DISPLAYING THE PYIMAGESEARCH GURUS LOGO IMAGE TO OUR SCREEN.

And there you have it! OpenCV + Python (along with the other necessary libraries/packages for the PyImageSearch Gurus course) is now successfully installed on your system!

Step 14 (Optional):

If you are creating your own Ubuntu VirtualBox virtual machine you'll want to install the *Guest Additions* so you can update the screen resolution of your VM. The default screen resolution is only *640 x 480*, making the VM almost unusable. Once you install the Guest Additions you'll be able to increase your screen resolution.

To install the Guest Additions, select *Devices* followed by *Insert Guest Additions CD image…*:

(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/vm_guest_additions.jpg)

FIGURE 4: INSTALLING GUEST ADDITIONS ON YOUR UBUNTU VIRTUALBOX VIRTUAL MACHINE.

From there, the VM will automatically install the Guest Additions.

And after a quick reboot, you'll be able to increase your screen resolution size:

(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/vm_after_guest_additions.jpg)

FIGURE 5: INCREASING SCREEN RESOLUTION AFTER GUEST ADDITIONS ARE INSTALLED.

Note: Once you have Guest Additions installed, you ***do not*** want to upgrade the Ubuntu version from within the VM. This ***almost always*** breaks the Guest Additions and you'll have to re-install them. Unfortunately, the problem with this is that the Guest Additions don't always re-install properly. So again, if you choose to install Guest Additions, ***do not*** apply any Ubuntu updates.

Summary

This tutorial detailed setting up your Python and OpenCV development environment on both OSX and Ubuntu for the PyImageSearch Gurus course. It is *by no means required* that you setup your own environment — but if you feel so included, the steps provided in this article will help you do so.

However, if you are using the Windows OS, I ***highly recommend*** that you either use the PyImageSearch Gurus VM or setup a new system running Ubuntu. We will not be covering troubleshooting Windows based problems inside this course.

Ready to continue the course?

Click the button below to **continue your journey to computer vision guru**.

I'm ready, let's go! (/pyimagesearch-gurus-course/)

Resources & Links

- PyImageSearch Gurus Community (https://community.pyimagesearch.com/)
- PyImageSearch Virtual Machine (https://gurus.pyimagesearch.com/pyimagesearch-virtual-machine/)
- Setting up your own Python + OpenCV environment (https://gurus.pyimagesearch.com/setting-up-your-python-opencv-development-environment/)
- Course Syllabus & Content Release Schedule (https://gurus.pyimagesearch.com/course-

syllabus-content-release-schedule/)

- Member Perks & Discounts (https://gurus.pyimagesearch.com/pyimagesearch-gurus-discounts-perks/)
- Your Achievements (https://gurus.pyimagesearch.com/achievements/)
- Official OpenCV documentation (http://docs.opencv.org/index.html)

Your Account

- Account Info (https://gurus.pyimagesearch.com/account/)
- Support (https://gurus.pyimagesearch.com/contact/)
- Logout (https://gurus.pyimagesearch.com/wp-login.php?action=logout&redirect_to=https%3A%2F%2Fgurus.pyimagesearch.com%2F&_wpnonce=5736b21c

Search