
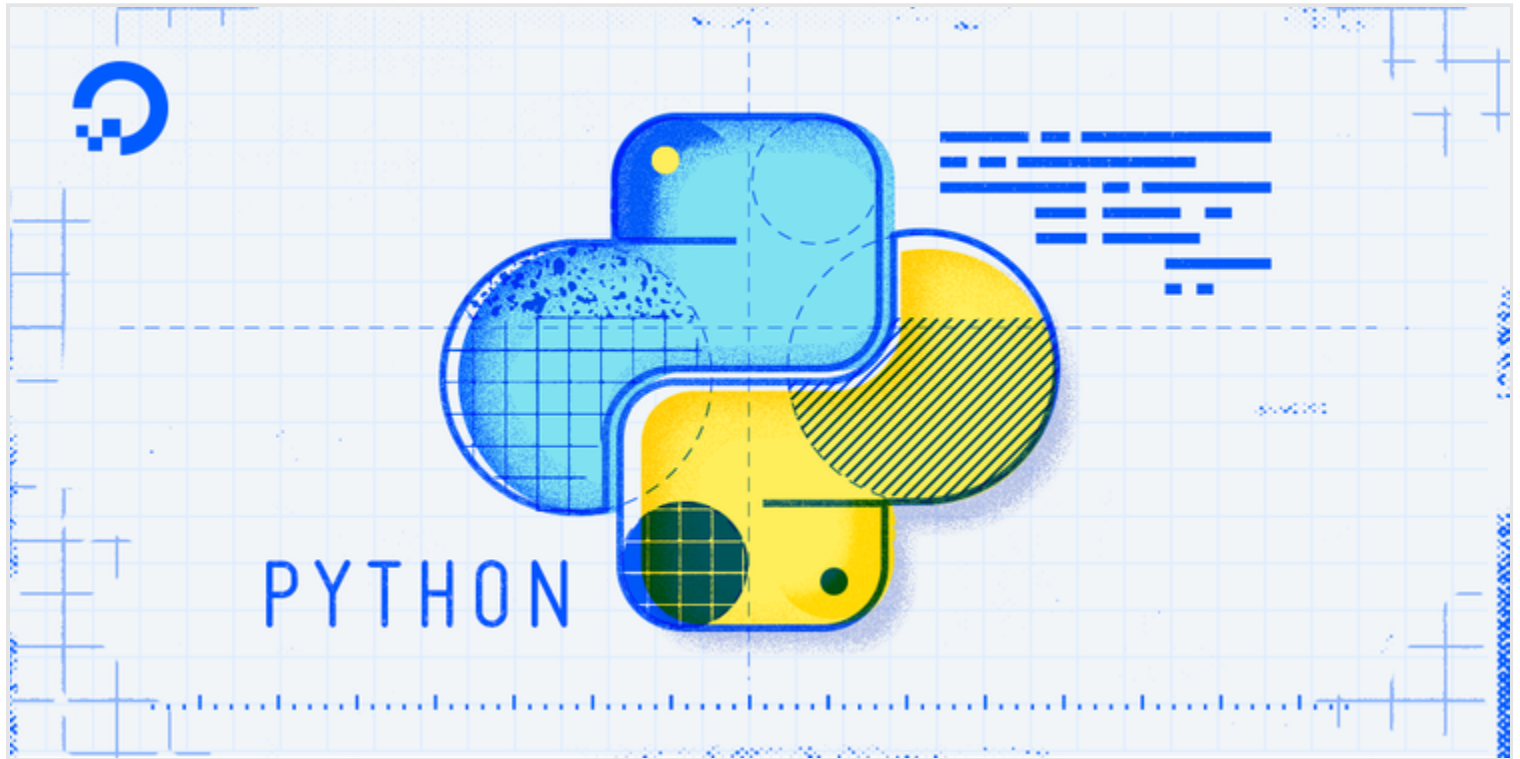



 Subscribe

 Share

 Contents ▾


How To Install Python 3 and Set Up a Programming Environment on Debian 9


4

Posted September 4, 2018  6.7k

PYTHON

DEVELOPMENT

DEBIAN 9

By: Lisa Tagliaferri

Not using **Debian 9**? Choose a different version:

Ubuntu 18.04



Ubuntu 16.04



Automated: Docker

request

Introduction

A flexible and versatile programming language, Python is effective for many use cases, including scripting, automation, data analysis, machine learning, and back-end development. First published in 1991 with a name inspired by the British comedy group Monty Python, the development team wanted to make Python a language that was fun to use. Quick to set up, and written in a relatively straightforward style with immediate feedback on errors, Python is a great choice for beginners and experienced developers alike. Python 3 is the most current version of the language and is considered to be the future of Python.

This tutorial will get your Debian 9 server set up with a Python 3 programming environment. Programming on a server has many advantages and supports collaboration across development projects.

Prerequisites

In order to complete this tutorial, you should have a non-root user with `sudo` privileges on a Debian 9 server. To learn how to achieve this setup, follow our [Debian 9 initial server setup guide](#).

If you're not already familiar with a terminal environment, you may find the article "[An Introduction to the Linux Terminal](#)" useful for becoming better oriented with the terminal.

With your server and user set up, you are ready to begin.

Step 1 — Setting Up Python 3

Debian Linux ships with both Python 3 and Python 2 pre-installed. To make sure that our versions are up-to-date, let's update and upgrade the system with the `apt` command to work with the **A**dvanced **P**ackaging **T**ool:

```
$ sudo apt update
$ sudo apt -y upgrade
```

The `-y` flag will confirm that we are agreeing for all items to be installed.

Once the process is complete, we can check the version of Python 3 that is installed in the system by typing:

```
$ python3 -V
```

You'll receive output in the terminal window that will let you know the version number. While this number may vary, the output will be similar to this:

Output

Python **3.5.3**

To manage software packages for Python, let's install **pip**, a tool that will install and manage programming packages we may want to use in our development projects. You can learn more about modules or

packages that you can install with pip by reading [“How To Import Modules in Python 3.”](#)

```
$ sudo apt install -y python3-pip
```

Python packages can be installed by typing:

```
$ pip3 install package_name
```

Here, **package_name** can refer to any Python package or library, such as Django for web development or NumPy for scientific computing. So if you would like to install NumPy, you can do so with the command `pip3 install numpy`.

There are a few more packages and development tools to install to ensure that we have a robust set-up for our programming environment:

```
$ sudo apt install build-essential libssl-dev libffi-dev python3-dev
```

Once Python is set up, and pip and other tools are installed, we can set up a virtual environment for our development projects.

Step 2 — Setting Up a Virtual Environment

Virtual environments enable you to have an isolated space on your server for Python projects, ensuring that each of your projects can have its own set of dependencies that won't disrupt any of your other projects.

Setting up a programming environment provides us with greater control over our Python projects and over how different versions of packages are handled. This is especially important when working with third-party packages.

You can set up as many Python programming environments as you want. Each environment is basically a directory or folder on your server that has a few scripts in it to make it act as an environment.

While there are a few ways to achieve a programming environment in Python, we'll be using the **venv** module here, which is part of the standard Python 3 library. Let's install venv by typing:

```
$ sudo apt install -y python3-venv
```

With this installed, we are ready to create environments. Let's either choose which directory we would like to put our Python programming environments in, or create a new directory with `mkdir`, as in:

```
$ mkdir environments  
$ cd environments
```

Once you are in the directory where you would like the environments to live, you can create an environment by running the following command:

```
$ python3.5 -m venv my_env
```

Essentially, `pyvenv` sets up a new directory that contains a few items which we can view with the `ls` command:

```
$ ls my_env
```

Output

```
bin include lib lib64 pyvenv.cfg share
```

Together, these files work to make sure that your projects are isolated from the broader context of your local machine, so that system files and project files don't mix. This is good practice for version control and to ensure that each of your projects has access to the particular packages that it needs. Python Wheels, a built-package format for Python that can speed up your software production by reducing the number of times you need to compile, will be in the Ubuntu 18.04 `share` directory.

To use this environment, you need to activate it, which you can achieve by typing the following command that calls the **activate** script:

```
$ source my_env/bin/activate
```

Your command prompt will now be prefixed with the name of your environment, in this case it is called `my_env`. Depending on what version of Debian Linux you are running, your prefix may appear somewhat differently, but the name of your environment in parentheses should be the first thing you see on your line:

```
(my_env) sammy@ubuntu:~/environments$
```

This prefix lets us know that the environment `my_env` is currently active, meaning that when we create programs here they will use only this particular environment's settings and packages.

Note: Within the virtual environment, you can use the command `python` instead of `python3`, and `pip` instead of `pip3` if you would prefer. If you use Python 3 on your machine outside of an environment, you will need to use the `python3` and `pip3` commands exclusively.

After following these steps, your virtual environment is ready to use.

Step 3 — Creating a “Hello, World” Program

Now that we have our virtual environment set up, let's create a traditional "Hello, World!" program. This will let us test our environment and provides us with the opportunity to become more familiar with Python if we aren't already.

To do this, we'll open up a command-line text editor such as nano and create a new file:

```
(my_env) sammy@ubuntu:~/environments$ nano hello.py
```

Once the text file opens up in the terminal window we'll type out our program:

```
print("Hello, World!")
```

Exit nano by typing the `CTRL` and `X` keys, and when prompted to save the file press `y`.

Once you exit out of nano and return to your shell, let's run the program:

```
(my_env) sammy@ubuntu:~/environments$ python hello.py
```

The `hello.py` program that you just created should cause your terminal to produce the following output:

Output

```
Hello, World!
```

To leave the environment, simply type the command `deactivate` and you will return to your original directory.

Conclusion

Congratulations! At this point you have a Python 3 programming environment set up on your Debian 9 Linux server and you can now begin a coding project!

If you are using a local machine rather than a server, refer to the tutorial that is relevant to your operating system in our ["How To Install and Set Up a Local Programming Environment for Python 3" series](#).

With your server ready for software development, you can continue to learn more about coding in Python by reading our free [*How To Code in Python 3* eBook](#), or consulting our [Programming Project tutorials](#).

Download our free Python eBook!

[*How To Code in Python* eBook in EPUB format](#)

[*How To Code in Python* eBook in PDF format](#)

By: Lisa Tagliaferri

♡ Upvote (4)

📄 Subscribe

🔗 Share



We just made it easier for you to deploy faster.

[TRY FREE](#)

Related Tutorials

Bias-Variance for Deep Reinforcement Learning: How To Build a Bot for Atari with OpenAI Gym

How To Ensure Code Quality with SonarQube on Ubuntu 18.04

How to Manually Set Up a Prisma Server on Ubuntu 18.04

How To Display Data from the DigitalOcean API with React

How To Set Up Jupyter Notebook with Python 3 on Ubuntu 18.04

3 Comments

Leave a comment...

^ alephian November 21, 2018



o Hi Lisa!

Greetings from Argentina!

Awesome Python Installation guide -with excellent background understanding in / linux / debian /.

Lisa. Please, Do you know about Python v.3.7.0 in Ubuntu 18.04 arch64? It is because GNOME Tools whole background is not possible / remove / purge / Python 3.6 -default in Ubuntu 18.04.

So, it is not easy deploy an ' environment variable ' to work safely.

Please, If you know about the right way to Python v.3.7.0 in Ubuntu 18.04 Client, I will appreciate to much your kindly answer -also a QuickStart Guide redirection.

fernando

^ ltagliaferri MOD November 21, 2018



o Hi Fernando!

I would recommend using this tutorial as a template for that: <https://www.admintome.com/blog/install-python-3-7-0-on-ubuntu-18-04/>

Just make sure you are using **sudo** commands as appropriate and are in the right working directory when implementing the commands.

^ alephian November 22, 2018



o Dear One!

Thanks for your kindly soon answer!

Have a wonderful day!

fernando





Copyright © 2019 DigitalOcean™ Inc.

[Community](#) [Tutorials](#) [Questions](#) [Projects](#) [Tags](#) [Newsletter](#) [RSS](#) 

[Distros & One-Click Apps](#) [Terms, Privacy, & Copyright](#) [Security](#) [Report a Bug](#) [Write for DOnations](#) [Shop](#)