

# How to Install Python and Pip on Debian 12

Learn how to install Python and Pip on Debian 12 with our step-by-step guide. Get your Python development environment up and running quickly and easily.

---

# Contents

01	Introduction	3
02	Prerequisites	3
03	Install Python on Debian 12	3
04	Install Pip on Debian 12	5
05	Test the Python Installation	6
06	Install Multiple Python Versions	8
07	Create a Python Virtual Environment	11
08	Conclusion	13

# Introduction

Python is a versatile, high-level programming language widely used for web development, automation, data analysis, and more. Pip is Python's package installer, allowing you to easily install and manage additional libraries and dependencies not included in the standard library. Installing Python and Pip on Debian 12 sets up a solid foundation for a wide range of development and scripting tasks.

This article explains how to install Python and Pip on Debian 12. You will install multiple Python versions and create a virtual environment to develop applications on your server. If you're using Ubuntu instead, check out [how to install Python and Pip on Ubuntu 24.04](#).

## Prerequisites

Before you begin:

- [Deploy a Debian 12 instance on Vultr](#).
- Access the instance [using SSH](#).
- Create a [non-root user with sudo privileges and switch to the user](#).

## Install Python on Debian 12

Debian 12 comes with a specific version of Python pre-installed by default. While Python is also available in the official package repositories, it may not be the latest release. For those interested in installing Python on Debian with the most up-to-date features, the recommended approach is to compile it from source. Follow the steps below to install the latest Python version on your server.

1. View the default Python version on your server.

CONSOLE

```
$ python3 --version
```

Output:

```
Python 3.11.2
```

2. Install all Python dependency packages.

CONSOLE

```
$ sudo apt install -y make build-essential libssl-dev zlib1g-dev libbz2-dev libreadline-dev libsqlite3-dev wget curl llvmlibncursesw5-dev xz-utils tk-dev libxml2-dev libxmlsec1-dev libffi-dev liblzma-dev
```

3. Visit the [Python downloads page](#) and verify the latest version. For example, version `3.13.0`. Then, download the archive file.

CONSOLE

```
$ wget https://www.python.org/ftp/python/3.13.0/Python-3.13.0.tgz
```

4. Extract all files from the archive.

CONSOLE

```
$ tar xzf Python-3.13.0.tgz
```

5. Switch to the new Python directory, such as `Python-3.13.0`.

CONSOLE

```
$ cd Python-3.13.0
```

6. Run the `configure` script to prepare the Python source code before compiling.

```
CONSOLE
```

```
$ ./configure
```

7. Compile the Python source code into executable binaries using the `make` command.

```
CONSOLE
```

```
$ make
```

8. Install the new Python version as an alternative without replacing the default version on your server.

```
CONSOLE
```

```
$ sudo make altinstall
```

9. View the new Python version to verify the installation is successful.

```
CONSOLE
```

```
$ python3.13 --version
```

Output:

```
Python 3.13.0
```

## Install Pip on Debian 12

Pip is a Python package manager that installs and manages dependency packages in a development environment. Pip uses Python Package Index (PyPI) third-party libraries and other repositories to install dependencies in a Python

environment. Python automatically installs Pip on the server. Follow the steps below to verify the Pip version depending on the Python version on your server.

1. View the default Pip version on your server.

```
CONSOLE
```

```
$ pip --version
```

Output:

```
pip 23.0.1 from /usr/lib/python3/dist-packages/pip (python 3.11)
```

2. View the new Pip version that matches the Python `3.13` version you installed earlier.

```
CONSOLE
```

```
$ pip3.13 --version
```

Output:

```
pip 24.2 from /usr/local/lib/python3.13/site-packages/pip (python 3.13)
```

To set up Python and streamline package management in Rocky Linux 9, follow this guide to [Install Pip on Rocky Linux 9](#).

## Test the Python Installation

Follow the steps below to create a basic web server that listens for HTTP connections on port `5000` and test the Python installation on your server.

1. Create a new `hello.py` Python file using a text editor like `nano`.

```
CONSOLE
```

```
$ nano hello.py
```

2. Add the following contents to the `hello.py` file.

```
PYTHON

from http.server import HTTPServer, BaseHTTPRequestHandler

class Serv(BaseHTTPRequestHandler):
    def do_GET(self):
        html = '<h1 style="padding:50px">Greetings from
Vultr</h1>'
        self.send_response(200)
        self.end_headers()
        self.wfile.write(bytes(html, 'utf-8'))

httpd = HTTPServer(('0.0.0.0', 5000), Serv)
print("Web server is listening on port 5000\nPress Ctrl+C to
stop...")
httpd.serve_forever()
```

Save and close the file.

The above Python application listens for connections on all addresses on the server and outputs the `Greetings from Vultr` message when you access it using an HTTP client such as Curl or web browser.

3. Allow network connections to port `5000` through the firewall.

```
CONSOLE

$ sudo ufw allow 5000
```

4. Run the application using your latest Python version as a background process.

```
CONSOLE

$ python3.13 hello.py &
```

Output:

```
Web server is listening on port 5000  
Press Ctrl+C to stop...
```

5. Access your server's public IP address using a web browser like Chrome on port `5000`.

```
http://SERVER-IP:5000
```

## Greetings from Vultr!

### Install Multiple Python Versions

When developing multiple Python projects, you may require different Python versions on your server. The default package repositories on Debian 12 do not include multiple Python versions. Follow the steps below to install and switch between multiple versions using the `asdf` tool on your server.

1. Install `asdf` dependency packages.

CONSOLE

```
$ sudo apt install -y curl git
```

2. Clone the `asdf` project repository. Visit the [releases page](#) to verify the latest version.

CONSOLE

```
$ git clone https://github.com/asdf-vm/asdf.git ~/.asdf --  
branch v0.14.1
```

3. Open the `.bashrc` file in your shell environment.

CONSOLE

```
$ nano ~/.bashrc
```

4. Add the following configurations at the end of the file.

BASH

```
. "$HOME/.asdf/asdf.sh"  
. "$HOME/.asdf/completions/asdf.bash"
```

Save and close the file.

5. Reload the `.bashrc` file to apply changes in your session.

CONSOLE

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

6. View the new `asdf` version.

CONSOLE

```
$ asdf --version
```

Output:

```
v0.14.1-f00f759
```

7. Install the `asdf` plugin for Python.

CONSOLE

```
$ asdf plugin-add python
```

- List all available Python versions available in the `asdf` sources.

```
CONSOLE
```

```
$ asdf list-all python
```

Output:

```
2.1.3
2.2.3
2.3.7
2.4.0
2.4.1
.....
stackless-3.4.2
stackless-3.4.7
stackless-3.5.4
stackless-3.7.5
```

- Install a specific Python version, such as `3.10.0`.

```
CONSOLE
```

```
$ asdf install python 3.10.0
```

- List all installed Python versions.

```
CONSOLE
```

```
$ asdf list
```

Output:

```
python
 3.10.0
```

11. Change the default Python version to the new version.

CONSOLE

```
$ asdf global python 3.10.0
```

12. View the active Python version to verify your changes.

CONSOLE

```
$ python3 --version
```

Output:

```
Python 3.10.0
```

13. Run the following command to switch back to the pre-installed Python version on your server.

CONSOLE

```
$ asdf global python system
```

## Create a Python Virtual Environment

A virtual environment creates an isolated environment to store Python project files with all necessary dependencies and libraries. You can use the native `venv` module to manage the Python virtual environments on your server. Follow the steps below to create a new Python virtual environment.

1. Install the `python3-venv` package.

CONSOLE

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

2. Create a new `myproject` directory.

CONSOLE

```
$ mkdir myproject
```

3. Switch to the `myproject` directory.

CONSOLE

```
$ cd myproject
```

4. Create a new `myenv` virtual environment.

CONSOLE

```
$ python3 -m venv .myenv
```

5. Activate the virtual environment.

CONSOLE

```
$ . .myenv/bin/activate
```

Output:

```
(.myenv) user@SERVER:~/myproject$
```

6. Install a specific Python module, such as Flask to use in the environment.

CONSOLE

```
$ pip install flask
```

7. View the Flask version.

CONSOLE

```
$ flask --version
```

Output:

```
Python 3.11.2
Flask 3.0.3
Werkzeug 3.0.4
```

8. Run the `deactivate` command to exit the virtual environment.

CONSOLE

```
$ deactivate
```

## Conclusion

You have installed Python and Pip on a Debian 12. In addition, you've installed and activated multiple Python versions using `asdf` to meet your project needs. You can integrate Python with other packages like database servers to develop and manage applications. For more information, please visit the [Python documentation](#).



VULTR

