Python Virtual Environments

Contents

[Python Virtual Environments 4](#_Toc107865517)

[Create It 4](#_Toc107865518)

[Activate It 4](#_Toc107865519)

[Install Packages Into It 4](#_Toc107865520)

[View Packages 4](#_Toc107865521)

[Deactivate It 4](#_Toc107865522)

[What Is a Python Virtual Environment? 4](#_Toc107865523)

[Viewing the Python folder structure 4](#_Toc107865524)

[pyvenv.cfg 5](#_Toc107865525)

[The paths that your Python session has access to in sys.path determine which locations Python can import modules from. 5](#_Toc107865526)

[Venv Modifies Your PYTHONPATH 6](#_Toc107865527)

[How Can You Customize a Virtual Environment? 7](#_Toc107865528)

[Specifying Your Python Version 8](#_Toc107865529)

[Virtual Environments and pyenv 8](#_Toc107865530)

[Why Do You Need Virtual Environments? 9](#_Toc107865531)

[Site Package 9](#_Toc107865532)

[Check the site-packages paths using sysconfig: 9](#_Toc107865533)

[PIP – Package installation Python 9](#_Toc107865534)

[sys.path 9](#_Toc107865535)

[PYTHONPATH 10](#_Toc107865536)

[Roaming folder in Windows and Python 10](#_Toc107865537)

[venv 11](#_Toc107865538)

[Virtual Environment Structures 11](#_Toc107865539)

[Prefix-Finding Process 11](#_Toc107865540)

[Activation: $ source venv/bin/activate 12](#_Toc107865541)

[Change the Command Prompt Name 12](#_Toc107865542)

[Overwrite Existing Environments 13](#_Toc107865543)

[Working With Multiple Environments 13](#_Toc107865544)

[Activating Multiple Versions Simultaneously 13](#_Toc107865545)

[Include the Global System Site-Packages 13](#_Toc107865546)

[Creating a virtual environment with pyenv 15](#_Toc107865547)

[Install pyenv from VS Code 15](#_Toc107865548)

[Installing Python on pyenv 16](#_Toc107865549)

[References 17](#_Toc107865550)

# Python Virtual Environments

## [Create It](https://realpython.com/python-virtual-environments-a-primer/#create-it)

|  |  |
| --- | --- |
| **Windows** | **Linux** |
| **PS>** python -m venv venv | **$** python3 -m venv venv |

## [Activate It](https://realpython.com/python-virtual-environments-a-primer/#activate-it)

|  |  |
| --- | --- |
| **Windows** | **Linux** |
| **PS>** venv\Scripts\Activate.ps1 | **$** source venv/bin/activate |

## [Install Packages Into It](https://realpython.com/python-virtual-environments-a-primer/#install-packages-into-it)

|  |  |
| --- | --- |
| **Windows** | **Linux** |
| **(venv) PS>** python -m pip install <package-name> | **(venv) $** python -m pip install <package-name> |

## [View Packages](https://realpython.com/python-virtual-environments-a-primer/#install-packages-into-it)

|  |  |
| --- | --- |
| **Windows** | **Linux** |
| **(venv) PS>** python -m pip list | **(venv) $** python -m pip list |

## [Deactivate It](https://realpython.com/python-virtual-environments-a-primer/#deactivate-it)

|  |  |
| --- | --- |
| **Windows** | **Linux** |
| **(venv) PS>** deactivate  **PS>** | **$ (venv) $** deactivate  **$** |

# What Is a Python Virtual Environment?

|  |  |
| --- | --- |
| * A Python virtual environment is a folder structure that gives you everything you need to run a lightweight yet isolated Python environment. | **Self-contained folder structure**   * When you create a new virtual environment using the venv module, Python creates a self-contained folder structure and copies or symlinks the Python executable files into that folder structure. |
|  |  |

## Viewing the Python folder structure

|  |  |
| --- | --- |
| **Windows** | **Linux** |
| **PS>** tree venv /F  **PS>** | **$** tree venv  **$** |
| The tree command displays the content of your venv directory in a very long tree structure. | **You may need to first install tree, for example with sudo apt install tree.** |
| * **Include\** is an initially empty folder that Python uses to include C header files for packages you might install that depend on C extensions. * **Lib\** contains the site-packages\ folder, which is one of the main reasons for creating your virtual environment. This folder is where you’ll install external packages that you want to use within your virtual environment. By default, your virtual environment comes preinstalled with two dependencies, pip and setuptools. You’ll learn more about them in a bit. * **Scripts\** contains the executable files of your virtual environment. Most notable are the Python interpreter (python.exe), the pip executable (pip.exe), and the activation script for your virtual environment, which comes in a couple of different flavors to allow you to work with different shells. In this tutorial, you’ve used Activate.ps1, which handles the activation of your virtual environment for Windows PowerShell. * **pyvenv.cfg** is a crucial file for your virtual environment. It contains only a couple of key-value pairs that Python uses to set variables in the sys module that determine which Python interpreter and which site-packages directory the current Python session will use. You’ll learn more about the settings in this file when you read about how a virtual environment works. | * **bin/** contains the executable files of your virtual environment. Most notable are the Python interpreter (python) and the pip executable (pip), as well as their respective symlinks (python3, python3.10, pip3, pip3.10). The folder also contains activation scripts for your virtual environment. Your specific activation script depends on what shell you use. For example, in this tutorial, you ran activate, which works for the Bash and Zsh shells. * **include/** is an initially empty folder that Python uses to include C header files for packages you might install that depend on C extensions. * **lib/** contains the site-packages/ directory nested in a folder that designates the Python version (python3.10/). site-packages/ is one of the main reasons for creating your virtual environment. This folder is where you’ll install external packages that you want to use within your virtual environment. By default, your virtual environment comes preinstalled with two dependencies, pip and setuptools. You’ll learn more about them in a bit. * **lib64/** in many Linux systems comes as a symlink to lib/ for compatibility reasons. Some Linux systems may use the distinction between lib/ and lib64/ to install different versions of libraries depending on their architecture. * **pyvenv.cfg** is a crucial file for your virtual environment. It contains only a couple of key-value pairs that Python uses to set variables in the sys module that determine which Python interpreter and which site-packages directory the current Python session will use. You’ll learn more about the settings in this file when you read about how a virtual environment works. |

## pyvenv.cfg

|  |  |
| --- | --- |
| **Windows** | **Linux** |
| **home = C:\Users\Name\AppData\Local\Programs\Python\Python310**  **include-system-site-packages = false**  **version = 3.10.3** | **home = /usr/local/bin**  **include-system-site-packages = false**  **version = 3.10.3** |
| **PS>** ls C:\Users\Name\AppData\Local\Programs\Python\Python310  **PS>** ls C:\Users\Name\AppData\Local\Programs\Python\Python310\Lib | **$** ls /usr/local/bin  **$** ls /usr/local/lib/python3.10 |
|  |  |

## The paths that your Python session has access to in sys.path determine which locations Python can import modules from.

If you activate your virtual environment and enter a Python interpreter, then you can confirm that the path to the standard library folder of your base Python installation is available:

|  |  |
| --- | --- |
| **Windows** | **Linux** |
| **PS>** python  >>> import sys  >>> from pprint import pp  >>> pp(sys.path)  ['',  'C:\\Users\\Name\\AppData\\Local\\Programs\\Python\\Python310\\python310.zip',  'C:\\Users\\Name\\AppData\\Local\\Programs\\Python\\Python310\\DLLs',  'C:\\Users\\Name\\AppData\\Local\\Programs\\Python\\Python310\\lib',  'C:\\Users\\Name\\AppData\\Local\\Programs\\Python\\Python310',  'C:\\Users\\Name\\path\\to\\venv',  'C:\\Users\\Name\\path\\to\\venv\\lib\\site-packages'] | **$** python  >>> import sys  >>> from pprint import pp  >>> pp(sys.path)  ['',  '/usr/local/lib/python310.zip',  '/usr/local/lib/python3.10',  '/usr/local/lib/python3.10/lib-dynload',  '/home/name/path/to/venv/lib/python3.10/site-packages'] |

## Venv Modifies Your PYTHONPATH

To assure that the scripts you want to run use the Python interpreter within your virtual environment, venv modifies the [PYTHONPATH](https://docs.python.org/3/using/cmdline.html#envvar-PYTHONPATH) environment variable that you can access using [sys.path](https://docs.python.org/3/library/sys.html#sys.path).

If you inspect that variable *without* an active virtual environment, you’ll see the default path locations for your default Python installation:

|  |  |
| --- | --- |
| **Windows** | **Linux** |
| **PS>** python  >>> import sys  >>> from pprint import pp  >>> pp(sys.path)  ['',  'C:\\Users\\Name\\AppData\\Local\\Programs\\Python\\Python310\\python310.zip',  'C:\\Users\\Name\\AppData\\Local\\Programs\\Python\\Python310\\DLLs', >>> import sys  >>> from pprint import pp  >>> pp(sys.path)  ['',  '/usr/local/lib/python310.zip',  '/usr/local/lib/python3.10',  '/usr/local/lib/python3.10/lib-dynload',  '/usr/local/lib/python3.10/site-packages']  'C:\\Users\\Name\\AppData\\Local\\Programs\\Python\\Python310\\lib',  'C:\\Users\\Name\\AppData\\Local\\Programs\\Python\\Python310',  'C:\\Users\\Name\\AppData\\Roaming\\Python\\Python310\\site-packages',  'C:\\Users\\Name\\AppData\\Local\\Programs\\Python\\Python310\\lib\\site-packages'] | **$** python  >>> import sys  >>> from pprint import pp  >>> pp(sys.path)  ['',  '/usr/local/lib/python310.zip',  '/usr/local/lib/python3.10',  '/usr/local/lib/python3.10/lib-dynload',  '/home/name/path/to/venv/lib/python3.10/site-packages'] >>> import sys  >>> from pprint import pp  >>> pp(sys.path)  ['',  '/usr/local/lib/python310.zip',  '/usr/local/lib/python3.10',  '/usr/local/lib/python3.10/lib-dynload',  '/usr/local/lib/python3.10/site-packages'] |

Inspect your path

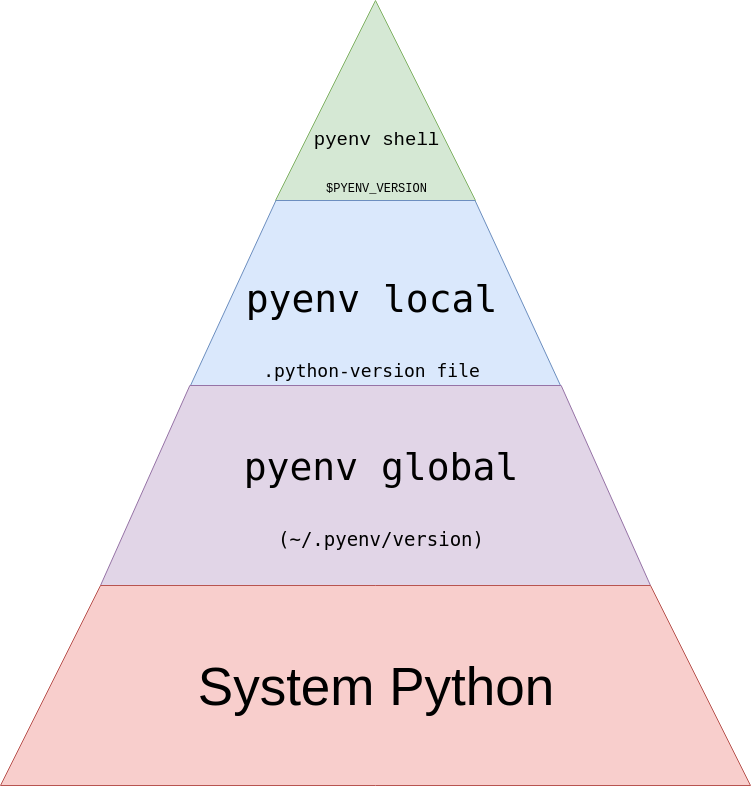
|  |  |
| --- | --- |
| **Windows** | |
| **PS>** $Env:Path  C:\Users\Name\path\to\venv\Scripts;C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Users\Name\AppData\Local\Programs\Python\Python310\Scripts\;C:\Users\Name\AppData\Local\Programs\Python\Python310\;c:\users\name\.local\bin;c:\users\name\appdata\roaming\python\python310\scripts  **PS>** | |
| **Linux** |
| **$** echo $PATH  /home/name/path/to/venv/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/home/name/.local/bin |

## How Can You Customize a Virtual Environment?

<https://realpython.com/python-virtual-environments-a-primer/#how-can-you-customize-a-virtual-environment>

# 

# Specifying Your Python Version

There are 3 ways to modify which version of python you’re using and how commands get resolved and how they interact with each other.

|  |
| --- |
| **$ pyenv versions**  \* system (set by /home/realpython/.pyenv/version)  2.7.15  3.6.8  3.8-dev |
| **$ pyenv global 3.6.8**  **$ pyenv versions**  system  2.7.15  \* 3.6.8 (set by /home/realpython/.pyenv/version)  3.8-dev  **$ cat ~/.pyenv/version**  3.6.8 |
| **$ pyenv local 2.7.15**  **$ pyenv versions**  system  \* 2.7.15 (set by /home/realpython/.python-version)  3.6.8  3.8-dev  **$ ls -a**  **$ pyenv local 2.7.15**  **$ pyenv versions**  system  \* 2.7.15 (set by /home/realpython/.python-version)  3.6.8  3.8-dev  **$ ls -a**  . .. .python-version  **$ cat .python-version**  2.7.15 |

# Virtual Environments and pyenv

As long as you don’t close your terminal, every Python package that you’ll install will end up in this isolated environment instead of your global Python site-packages. That means you can now work on your Python project without worrying about dependency conflicts.

<https://realpython.com/python-virtual-environments-a-primer/#create-it>

# Why Do You Need Virtual Environments?

|  |  |
| --- | --- |
| Python community suggests that you use virtual environments for all your projects.  Python isn’t great at dependency management. | If you’re not specific, then pip will place all the external packages that you install in a folder called site-packages/ in your base Python installation. |
| Technically, Python comes with two site-packages folders:   * purelib/ should contain only modules written in pure Python code. * platlib/ should contain binaries that aren’t written in pure Python, for example .dll, .so, or .pydist files. |  |

## Site Package

|  |  |
| --- | --- |
| * Site-packages is the target directory of manually built Python packages. * When you build and install Python packages from source (using distutils, probably by executing python setup.py install), you will find the installed modules in site-packages by default. | **standard locations:**   * prefix/lib/pythonX.Y/site-packages * ~/. local/lib/pythonX.Y/site-packages   Ubuntu:   * usr/local/lib/python3.6/site-packages |
| **Installing/add Python packages**   * '$PIP install <package-name>'   **With a setup.py**   * open a command or terminal window * go to root directory where setup.py is located Enter: $python setup.py install | **Uninstalling/removing Python packages**  '$PIP uninstall <package-name>' |

## Check the site-packages paths using sysconfig:

>>> import sysconfig

>>> sysconfig.get\_path("purelib")

'/home/name/path/to/**venv**/lib/python3.10/site-packages'

>>> sysconfig.get\_path("platlib")

'/home/name/path/to/**venv**/lib/python3.10/site-packages'

## PIP – Package installation Python

|  |  |
| --- | --- |
|  |  |
| * pip --version |  |

## sys.path

|  |  |
| --- | --- |
| * Python is set up to find these modules by adding the relevant path to sys.path. * During initialization, Python automatically imports the site module, which sets the defaults for this argument. | * The paths that your Python session has access to in sys.path determine which locations Python can import modules from. |
| * If you activate your virtual environment and enter a Python interpreter, then you can confirm that the path to the standard library folder of your base Python installation is available:   >>> import sys  >>> from pprint import pp  >>> pp(sys.path) | * Because the path to the directory that contains your standard library modules is available in sys.path, you’ll be able to import any of them when you work with Python from within your virtual environment. |

## PYTHONPATH

* Python is set up to find these modules by adding the relevant path to sys.path. During initialization, Python automatically imports the site module, which sets the defaults for this argument.
* The paths that your Python session has access to in sys.path determine which locations Python can import modules from.
* If you activate your virtual environment and enter a Python interpreter, then you can confirm that the path to the standard library folder of your base Python installation is available:

|  |  |
| --- | --- |
| * To assure that the scripts you want to run use the Python interpreter within your virtual environment, venv modifies the PYTHONPATH environment variable that you can access using sys.path. | If you inspect that variable without an active virtual environment, you’ll see the default path locations for your default Python installation:  >>> import sys  >>> from pprint import pp  >>> pp(sys.path) |
| **No virtualization**  '/usr/local/lib/python310.zip',  '/usr/local/lib/python3.10',  '/usr/local/lib/python3.10/lib-dynload',  '/usr/local/lib/python3.10/site-packages'] | **With virtualization activation**  '/usr/local/lib/python310.zip',  '/usr/local/lib/python3.10',  '/usr/local/lib/python3.10/lib-dynload',  '/home/name/path/to/**venv**/lib/python3.10/site-packages'] |
| **Note:** if the path to your base Python’s site-packages directory isn’t in this list anymore, Python won’t load modules from there. | Python replaced the default site-packages directory path with the one that lives inside your virtual environment. This change means that Python will load any external packages installed in your virtual environment. |

## Roaming folder in Windows and Python

|  |  |
| --- | --- |
| Roaming folder on Windows contains an additional site-packages directory relevant for installations that use the --user flag with pip. This folder provides a small degree of virtualization, but it still collects all --user installed packages in one spot. | The Roaming folder is used to store data that will be synced across multiple Windows systems. This is often used for storing settings like bookmarks, saved passwords, and so on. |
| **ProgramData** folder contains global application data that is not user-specific and is available to all users on the computer. Any global data is put in here. | **AppData** folder contains user-specific preferences and profile configurations and is further divided into three subfolders:   * **Roaming folder** contains data that can move with the user profile from a computer to a computer * **Local folder** contains data that cannot move with your user profile. * **LocalLow folder** includes low-level access data, eg. temporary files of your browser when running in a protected mode. |

## venv

|  |  |
| --- | --- |
| When you create a virtual environment using venv, the module re-creates the file and folder structure of a standard Python installation on your operating system. Python also copies or symlinks into that folder structure the Python executable with which you’ve called venv: | If you locate your system-wide Python installation on your operating system and inspect the folder structure there, then you’ll see that your virtual environment resembles that structure. |
| On Windows, you may notice that python.exe in your base Python installation isn’t in Scripts\ but is one folder level up. In your virtual environment, the executable is intentionally located in the Scripts\ folder. | The Global environment standard folder structure is the same as in your virtual environment. venv creates this folder structure to assure that Python will work as expected in **isolation**, without the need to apply many additional changes. |

## Virtual Environment Structures

|  |  |
| --- | --- |
| On your command line, navigate to the folder that contains your virtual environment.  $ tree venv | You may need to first install tree, for example with  $ sudo apt install tree |
|  |  |

<https://realpython.com/python-virtual-environments-a-primer/#a-folder-structure>

## Prefix-Finding Process

|  |  |
| --- | --- |
| * With the standard folder structure in place, the Python interpreter in your virtual environment can understand where all relevant files are located. * prefix-finding process according to the [venv specification](https://www.python.org/dev/peps/pep-0405/#specification). | Instead of looking for the os module to determine the location of the standard library, the Python interpreter first looks for a pyvenv.cfg file. If the interpreter finds this file and it contains a home key, then the interpreter will use that key to set the value for two variables:   * **sys.base\_prefix** will hold the path to the Python executable used to create this virtual environment, which you can find at the path defined under the home key in pyvenv.cfg. * **sys.prefix** will point to the directory containing pyvenv.cfg. |
| **Explore Prefix Paths: venv activation**  >>> import sys  >>> sys.prefix  '/home/name/path/to/venv'  >>> sys.base\_prefix  '/usr/local' | **Explore Prefix Paths: global**  >>> import sys  >>> sys.prefix  '/usr/local'  >>> sys.base\_prefix  '/usr/local' |

## Activation: $ source venv/bin/activate

(venv) $

|  |  |
| --- | --- |
| **Two critical actions happen in the activation script:**   1. **Path:** It sets the VIRTUAL\_ENV variable to the root folder path of your virtual environment and prepends the relative location of its Python executable to your PATH. 2. **Command prompt:** It changes the command prompt to the name that you passed when creating the virtual environment. It takes that name and puts it into parentheses, for example (venv). | **These changes put the convenience of virtual environments into effect within your shell:**   1. **Path:** Because the path to all the executables in your virtual environment now lives at the front of your PATH, your shell will invoke the internal versions of pip or Python when you just type pip or python. 2. **Command prompt:** Because the script changed your command prompt, you’ll quickly know whether or not your virtual environment is activated. |
| You can inspect your PATH variable before and after activation of your virtual environment. If you’ve activated your virtual environment, then you’ll see the path to the folder containing your internal executables at the beginning of PATH: | **Linus:**  $ echo $PATH  /home/name/path/to/venv/bin:  /usr/local/sbin:/usr/local/bin:  /usr/sbin:  /usr/bin:  /sbin:/bin:  /home/name/.local/bin |
| **Windows:**  PS> $Env:Path  C:\Users\Name\path\to\venv\Scripts;  C:\Windows\system32;  C:\Windows;  C:\Windows\System32\Wbem;  C:\Users\Name\AppData\Local\  Programs\Python\Python310\Scripts\;  C:\Users\Name\AppData\Local\  Programs\Python\Python310\;  c:\users\name\.local\bin;c:\users\name\appdata\roaming\python\python310\scripts | **Running From Anywhere With Absolute Paths:**  **$ /home/name/path/to/venv/bin/python**  Absolute paths can be helpful for running a scheduled script on your remote server or [in a Docker container](https://pythonspeed.com/articles/activate-virtualenv-dockerfile/). Specifically, you’ll want to use absolute paths if the script requires external dependencies that you want to isolate from the rest of your server in a Python virtual environment. |

## Change the Command Prompt Name

|  |  |
| --- | --- |
| $ python3 -m venv your-fancy-name  $ source your-fancy-name/bin/activate  (your-fancy-name) $ | PS> python -m venv your-fancy-name  PS> your-fancy-name\Scripts\Activate.ps1  (your-fancy-name) PS> |
| If you give your virtual environment folder an alternate name, you’ll also need to consider that name when you want to run your activation script, as shown in the code example above. | If you want the convenience of seeing a different command prompt, but you want to keep the folder name descriptive so that you’ll know it contains a virtual environment, then you can pass your desired command prompt name to --prompt:  PS> python -m venv venv --prompt="dev-env"  PS> venv\Scripts\Activate.ps1  (dev-env) PS> |

## Overwrite Existing Environments

|  |  |
| --- | --- |
| $ python3 -m venv venv  $ venv/bin/pip install requests  $ venv/bin/pip list  Package Version  ------------------ ---------  certifi 2021.10.8  charset-normalizer 2.0.12  idna 3.3  pip 22.0.4  requests 2.27.1  setuptools 58.1.0  urllib3 1.26.9  $ python3 -m venv venv --clear  $ venv/bin/pip list  Package Version  ---------- -------  pip 22.0.4  setuptools 58.1.0 | PS> python -m venv venv  PS> venv\Scripts\pip.exe install requests  PS> venv\Scripts\pip.exe list  Package Version  ------------------ ---------  certifi 2021.10.8  charset-normalizer 2.0.12  idna 3.3  pip 22.0.4  requests 2.27.1  setuptools 58.1.0  urllib3 1.26.9  PS> python -m venv venv --clear  PS> venv\Scripts\pip.exe list  Package Version  ---------- -------  pip 22.0.4  setuptools 58.1.0 |

## Working With Multiple Environments

Two environments set up: venv and venv-copy. If absolute path does not exist venv will create one.

PS> python -m venv venv C:\Users\Name\Documents\virtualenvs\venv-copy

## Activating Multiple Versions Simultaneously

## Include the Global System Site-Packages

|  |  |
| --- | --- |
| * In some situations, you might want to keep access to your base Python’s site-packages directory instead of severing that tie. * For example, you might have already set up a package that’s compiled during installation, such as Bokeh, in your global Python environment. | * You can access all modules you’ve installed to your base Python’s site-packages directory by adding the --system-site-packages flag when creating your virtual environment:   PS> python -m venv venv --system-site-packages  $ source venv/bin/activate  (venv) PS> |
| * Once again, you’ve created a new virtual environment named venv, but this time you passed the --system-site-packages argument. * Adding this optional argument resulted in a different setting in your pyvenv.cfg file:   home = /usr/local/bin  include-system-site-packages = true  version = 3.10.3 | * This change means that you’ll see an additional entry to sys.path, which allows the Python interpreter in your virtual environment to also access the system site-packages directory. * Make sure your virtual environment is active, then start the Python interpreter to check the path variables |
| **Linux:**  >>> import sys  >>> from pprint import pp  >>> pp(sys.path)  ['',  '/usr/local/lib/python310.zip',  '/usr/local/lib/python3.10',  '/usr/local/lib/python3.10/lib-dynload',  '/home/name/path/to/venv/lib/python3.10/site-packages',  **'/home/name/.local/lib/python3.10/site-packages',**  **'/usr/local/lib/python3.10/site-packages']** |  |

# Creating a virtual environment with pyenv

## Install pyenv from VS Code

<https://realpython.com/intro-to-pyenv/>

1. bash
2. sudo apt-get update
3. sudo apt-get install -y make build-essential libssl-dev zlib1g-dev \

libbz2-dev libreadline-dev libsqlite3-dev wget curl llvm libncurses5-dev \

libncursesw5-dev xz-utils tk-dev libffi-dev liblzma-dev python-openssl

1. sudo apt install python3-venv
2. sudo apt install pyenv
3. curl https://pyenv.run | bash

This will install pyenv along with a few plugins that are useful:

* pyenv: The actual pyenv application
* pyenv-virtualenv: Plugin for pyenv and virtual environments
* pyenv-update: Plugin for updating pyenv
* pyenv-doctor: Plugin to verify that pyenv and build dependencies are installed
* pyenv-which-ext: Plugin to automatically lookup system commands

**Note:** The above command is the same as downloading the [pyenv-installer script](https://github.com/pyenv/pyenv-installer/blob/master/bin/pyenv-installer) and running it locally. So if you’d like to see exactly what you’re running, you can view the file yourself. Alternatively, if you really don’t want to run a script, you can checkout the [manual installation instructions.](https://github.com/pyenv/pyenv#basic-github-checkout)

At the end of the run, you should see something like this:

WARNING: seems you still have not added 'pyenv' to the load path.

# Load pyenv automatically by appending

# the following to

# ~/.bash\_profile if it exists, otherwise ~/.profile (for login shells)

# and ~/.bashrc (for interactive shells) :

**export PYENV\_ROOT="$HOME/.pyenv"**

**command -v pyenv >/dev/null || export PATH="$PYENV\_ROOT/bin:$PATH"**

**eval "$(pyenv init -)"**

# Restart your shell for the changes to take effect.

# Load pyenv-virtualenv automatically by adding

# the following to ~/.bashrc:

**eval "$(pyenv virtualenv-init -)"**

The output will be based on your shell. But you should follow the instructions to add pyenv to your path and to initialize pyenv/pyenv-virtualenv auto completion. Once you’ve done this, you need to reload your shell:

1. $ exec "$SHELL" # Or just restart your terminal
2. pyenv exec python –m venv .venv

# Installing Python on pyenv

frankt@IT-017969-FL:~/.pyenv/versions$ pyenv install 3.9.12

Downloading Python-3.9.12.tar.xz...

-> https://www.python.org/ftp/python/3.9.12/Python-3.9.12.tar.xz

Installing Python-3.9.12...

Installed Python-3.9.12 to /home/frankt/.pyenv/versions/3.9.12

frankt@IT-017969-FL:~/.pyenv/versions$ pwd

/home/frankt/.pyenv/versions

frankt@IT-017969-FL:~/.pyenv/versions$ ls -a -n

total 0

drwxr-xr-x 1 1000 1000 512 May 16 18:38 .

drwxr-xr-x 1 1000 1000 512 May 15 11:05 ..

drwxr-xr-x 1 1000 1000 512 May 16 18:40 3.9.12

frankt@IT-017969-FL:~/.pyenv/versions$ python -V

pyenv: python: command not found

The `python' command exists in these Python versions:

3.9.12

Note: See 'pyenv help global' for tips on allowing both

python2 and python3 to be found.

frankt@IT-017969-FL:~/.pyenv/versions$ pyenv help global

Usage: pyenv global <version> <version2> <..>

Sets the global Python version(s). You can override the global version at

any time by setting a directory-specific version with `pyenv local'

or by setting the `PYENV\_VERSION' environment variable.

<version> can be specified multiple times and should be a version

tag known to pyenv. The special version string `system' will use

your default system Python. Run `pyenv versions' for a list of

available Python versions.

Example: To enable the python2.7 and python3.7 shims to find their

respective executables you could set both versions with:

'pyenv global 3.7.0 2.7.15'

frankt@IT-017969-FL:~/.pyenv/versions$ pyenv global 3.9.12

frankt@IT-017969-FL:~/.pyenv/versions$ python -V

Python 3.9.12

[frankt@IT-017969-FL:~/.pyenv/versions$](mailto:frankt@IT-017969-FL:~/.pyenv/versions$)

# References

* [Python Virtual Environments: A Primer](https://realpython.com/python-virtual-environments-a-primer/)
* <https://realpython.com/intro-to-pyenv/>
* [Managing Multiple Python Versions With pyenv](https://realpython.com/intro-to-pyenv/#exploring-pyenv-commands)
* [[SOLVED] pyenv: python :command not found](https://www.linuxfixes.com/2022/03/solved-pyenv-python-command-not-found.html)
* [How to install Python the smart way](https://www.infoworld.com/article/3530140/how-to-install-python-the-smart-way.html#:~:text=For%20the%20sake%20of%20compatibility,in%20this%20case%2C%20Python%203.7.)
* [Socket Programming in Python (Guide)](https://realpython.com/python-sockets/)
* [How Does a Virtual Environment Work?](https://realpython.com/python-virtual-environments-a-primer/#how-does-a-virtual-environment-work)
* PEP 405 –<https://peps.python.org/pep-0405/#specification> Python Virtual Environments
* [Local, LocalLow and Roaming folders in AppData on Windows 11/10 explained](https://www.thewindowsclub.com/local-localnow-roaming-folders-windows-10)
* [Managing Application Dependencies](https://packaging.python.org/en/latest/tutorials/managing-dependencies/)
* [An Overview of Packaging for Python](https://packaging.python.org/en/latest/overview/)
* [Python Packaging User Guide](https://packaging.python.org/en/latest/overview/)
* [What Are Python Wheels and Why Should You Care?](https://realpython.com/python-wheels/)
* [Getting Started with Python in VS Code](https://code.visualstudio.com/docs/python/python-tutorial)
* [Using Python environments in VS Code](https://code.visualstudio.com/docs/python/environments#_global-virtual-and-conda-environments)
* [Check the Version of the Python Interpreter](https://www.geeksforgeeks.org/check-the-version-of-the-python-interpreter/)
* [Virtualenv and venv: Python virtual environments explained](https://www.infoworld.com/article/3239675/virtualenv-and-venv-python-virtual-environments-explained.html)
* [Environment Variable](https://www.techopedia.com/definition/15664/environment-variable)