Python Basics

How to create a virtual environment

Version: 1.6

Bodo Schönfeld

January 24, 2024

Contents

1 Introduction

What does a virtual development environment do? It isolates different Python versions and their package installations from each other. Once set up for a Python project, this means that the packages used are only available to this project. Updating one of these packages does not affect other packages installed on the system. In addition, different Python versions can be installed and used for different projects.

Not installing packages globally has another advantage: it protects the integrity of the operating system. Package conflicts that can occur during a system-wide installation are also avoided.

2 virtualenv and venv

A virtual environment in Python can be created in different ways. In order to use a virtual development environment, it was previously necessary to install the **virtualenv** package. Since version 3.4, this is no longer necessary because parts of this package have been implemented in Python as a module called **venv**. Using `venv` has been the recommended way to set up a virtual environment since version 3.5. Therefore, the setup is explained below using this module.[1]

[1] The venv documentation is available online: https://docs.python.org/3/library/venv.html

3 Set up a virtual environment

After a new project has been set up, the virtual environment can already be added. Let's look at this with the following example.

Assuming the project has the name „my-project", the project structure could look like this:[2]

```
my-project
├── README.md
├── pytest.ini
├── requirements.txt
├── setup.cfg
├── setup.py
├── docs
├── my_project
│   ├── __init__.py
│   └── main.py
└── tests
    └── __init__.py
```

To create a virtual environment, first change to the project directory using the `cd` command:

```
$ cd my-project   # Linux, macOS
PS> cd my-project # Windows
```

The virtual environment is then created with the following instruction:

```
$ python3 -m venv venv  # Linux, macOS
PS> python -m venv venv # Windows
```

Note for Linux users: If the python3.x-venv package is not installed, the creation of the virtual environment may fail. This may be happen on **Debian**-based systems if

[2] https://niftycode.eu/the-structure-of-a-python-project/

an older Python version is used. On Ubuntu, the installation of the python3.x-venv package can be done with the following instructions (for Python 3.10):

```
$ sudo apt install python3.10-venv
```

On **Windows** you can use `py` instead of `python`.

```
PS> py -m venv venv
```

Note: If the path to the Python installation has not been added to the PATH variable on Windows, the full path must be used to create a virtual environment.

Executing this instruction causes a new directory to be created. In this example, it has the name **venv**. The name of the directory is arbitrary, but it is common practice among Python developers to use **venv** (or **.venv**). Especially on Linux systems and on macOS, a dot is often placed in front of the directory name, i.e. „.venv“ is used.

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

What is the point before „venv“ all about? By using „.venv“, this folder becomes a hidden folder that is no longer displayed in the Linux or macOS file manager.

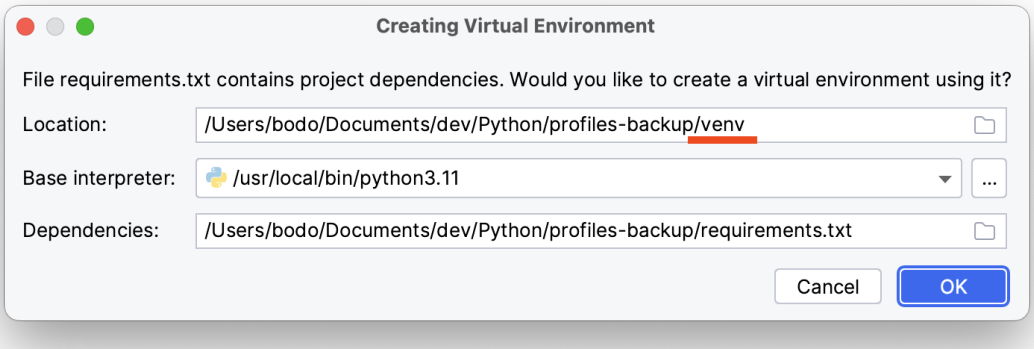
Note: On Linux or macOS, hidden directories (or files) are displayed with the `ls -a` command.

A virtual environment can also be created in a development environment like PyCharm or Visual Studio Code. A directory with the name „venv“ (or „.venv“) is then also created. Visual Studio Code creates the directory „.venv“, while PyCharm suggests „venv“.[3]

By the way, Microsoft's recommendation in the documentation for the creation of virtual environments is „.venv“.[4]

[3] However, Pycharm also has no problems with the .venv designation.
[4] See https://code.visualstudio.com/docs/python/environments

Figure 1: Creating a virtual environment in PyCharm

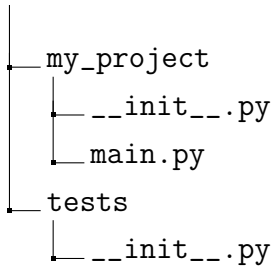


It is not uncommon to have multiple versions of Python on your system. In this case, it is possible to specify the Python version to be used. Version 3.12 will be used in the following example:

```
$ python3.12 -m venv .venv
```

Once a virtual environment has been created, the directory structure looks as follows, with „.venv“ selected as the folder name:

```
my-project
├── .venv
│   ├── bin
│   ├── include
│   ├── lib
│   └── pyvenv.cfg
├── README.md
├── pytest.ini
├── requirements.txt
├── setup.cfg
├── setup.py
└── docs
```


my_project
__init__.py
main.py
tests
__init__.py

4 Activate or deactivate the virtual environment

Before packages can be installed, the virtual environment must be activated. The packages installed after activation are then only available to this project (and *not* globally).

On **Linux** or **macOS**, use the following instruction:

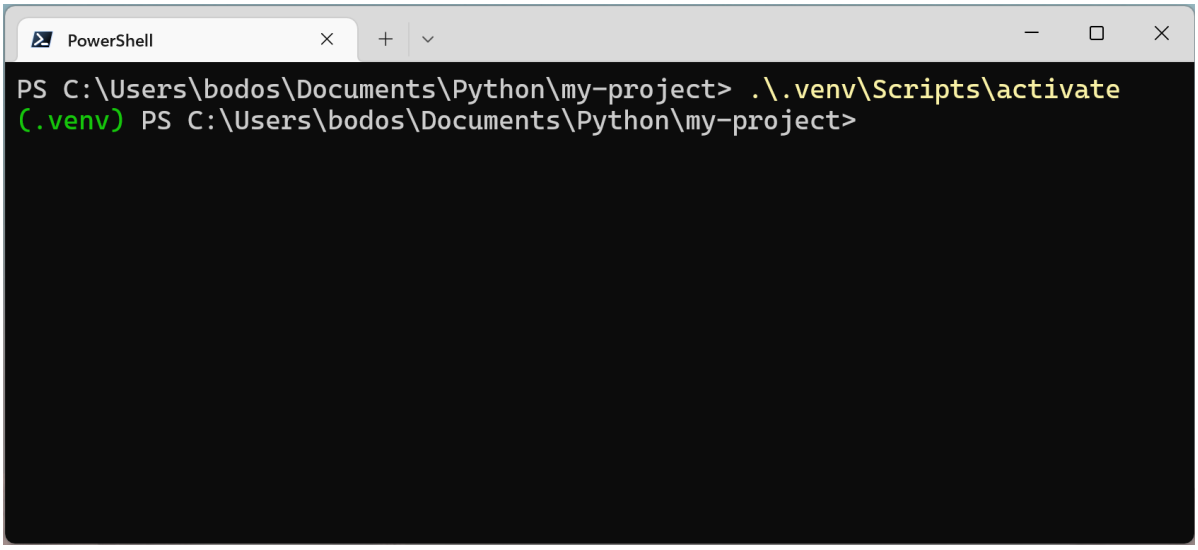
```
$ source .venv/bin/activate
```

And on **Windows**, use the following instruction:

```
PS> .venv\Scripts\activate
```

After activation, `.venv` is added to the prompt at the beginning of the line.

Figure 2: Virtual environment in the PowerShell



After activation, you can install packages with `pip` or `pip3`

```
pip3 install <Package>  # Linux, macOS
pip install <Package>   # Windows
```

As already mentioned, these packages are only available in this project. This ensures that different versions can be installed for different projects.

The virtual environment is **deactivated** on all operating systems with the `deactivate` command:

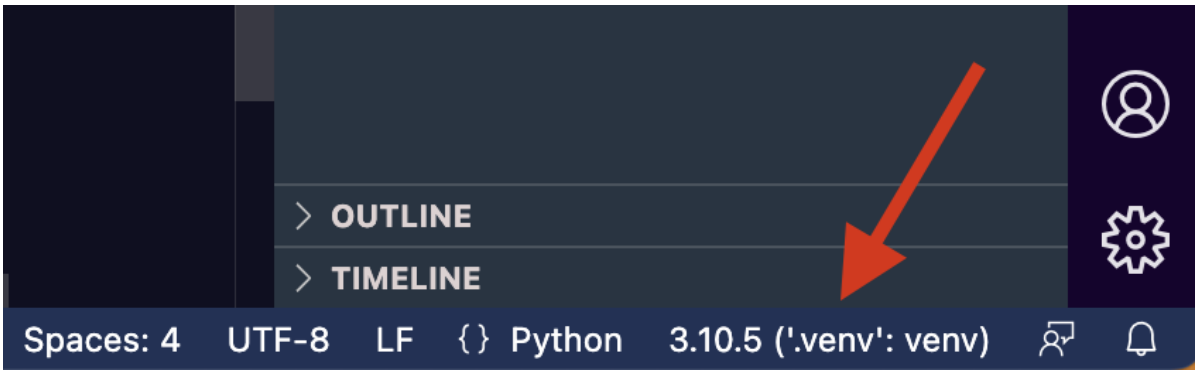
```
$ deactivate    # Linux, macOS
PS> deactivate  # Windows
```

5 Development Environments

A virtual environment created in the Terminal or PowerShell is generally recognized automatically by development environments or code editors. In general, this is always the case if "„.venv“ or „venv“ is used as the directory name. A recognized development environment therefore does not have to be activated manually. This should already have been done by the development environment.

This can be checked in the status bar of the respective application. You will see „(.venv)“ there, for example. In some programs, the Python interpreter used is also displayed.

Figure 3: Indication of an activated virtual environment in VS Code



If **Visual Studio Code** does not automatically recognize an existing virtual environment, it can be activated via the **Command Palette**. To do this, enter „env“ as the search term. This can also be used to create a new virtual environment, whereby Visual Studio Code selects „.venv“ as the directory name by default.

PyCharm users have the option of selecting a Python interpreter if a virtual environment is not available. Optionally, a virtual environment can also be created, whereby „venv“ is suggested as the directory name by default.

6 Version Control (Git)

It should also be mentioned that the „.venv“ (or „venv“) directory should not be included in version control (git). This would result in the packages installed for this project also being transferred to Github (or another provider).

Instead, it makes sense to add the „requirements.txt“ file to the project. All modules belonging to the project are listed in this file so that they can be installed with the following command if required:

```
$ pip3 install -r requirements.txt  # Linux, macOS
PS> pip install -r requirements.txt # Windows
```

Excluding the „.venv“ folder is achieved by making a corresponding entry in the „.gitignore“ file. Simply add the line „/.venv“ to this file. A typical „.gitignore“ file could have the following content:

```
/.idea
/.vscode
/.venv
/build
/dist
/docs
```

7 License

- Version: 1.6 - January 24th, 2024
- Author: @niftycode
- Homepage: https://bodo-schoenfeld.de
- Github: https://github.com/niftycode

Attribution 4.0 International (CC BY 4.0):

You are free to:

- **Share** — copy and redistribute the material in any medium or format
- **Adapt** — remix, transform, and build upon the material for any purpose, even commercially