Setting up Python Virtual Environments (venv)

A Python **virtual environment (venv)** is an isolated environment that allows you to install and manage dependencies without affecting the global Python installation.

This is crucial for maintaining project-specific dependencies and avoiding conflicts between packages.

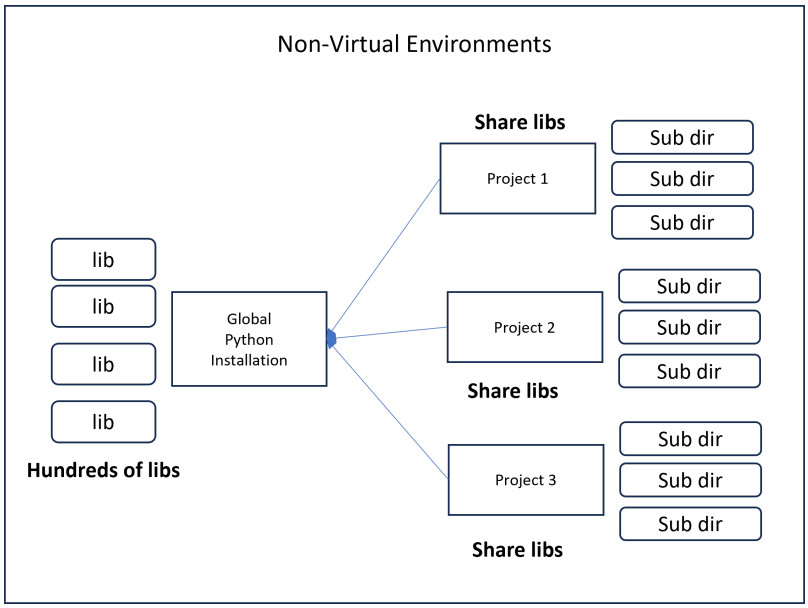
The standard name for a virtual environment is `venv`, and it is widely recommended for consistency. In this guide, we'll cover how to create, manage, and verify a virtual environment.

What is a python Virtual Environment?

Let's start with what is a python virtual environment.

Python Global Environment first

Let me first explain what is a python enviornment with a diagram.

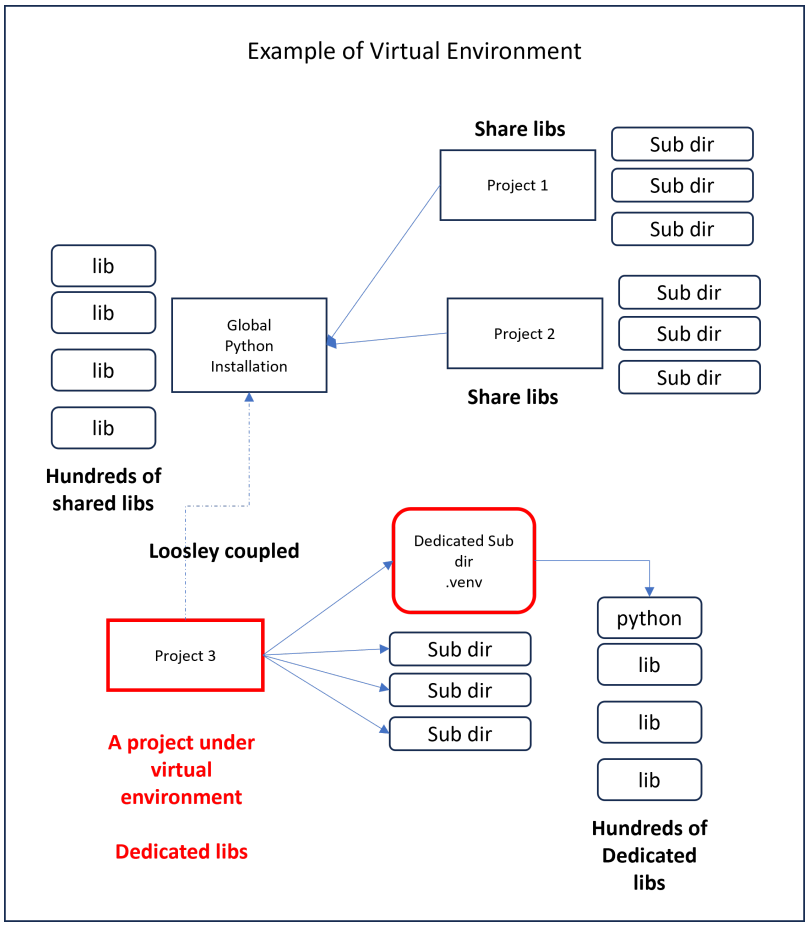


Few take aways

1. Python is installed in its installation directory (however one chooses)
2. All subsequent library installs for any project will sit under the common installation choice
3. ALl projects have access to all the libraries sharing them (with potential overlaps)
4. Each project contains its own sub directories and files.

Python Virtual environment

Let's contrast this with a virtual environment setup



Key takeaways, that will explain why the "Projet 3" is setup to use a virtual environment.

1. A sub directory called ".venv" is created for just that project
2. Python is then instructed to put all installs to use that as the target destination
3. The dependent libraries must be inside this sub directory to be used by project 3
4. Global libs and global python is ignored in preference to this .venv
5. In other words the project 3 ignores the global installation

Why is this useful?

1. Avoid library conflicts
2. When the project is moved, you can reinstall in the new place precisely the same libraries. (Read further how to do this)

The key steps in doing this are

1. Create a virtual environment for a given project
2. Activate that virtual environment on the command line where python is executed
3. See what libraries are available in that virtual environment

Read the rest of the article to go into the details of these

Creating a Virtual Environment

You can create a virtual environment using the `venv` module, which is included in Python 3.3 and later:

```
python -m venv .venv
```

1. `venv` is the standard name for virtual environments.
2. This creates a directory `venv/` containing an isolated Python setup.

Note on the virtial enviornment path

1. Run that command from the root directory of your project
2. Because the last argument is a path
3. So it will treat it as a sub directory under the root, so it matters where you run this from
4. Or you have to explicitly specify the path
5. The convention is to use a local sub directory under root ".venv"

Note on the VScode ask If you run this from vscode it may ask you if you want to use this virtual environment for the entire workspace? I have not experimened with that to see what it means. I suspect it will apply to all projects in that workspace. My choice has been for each project.

Do your research if you have an occasion to use the work space option.

Activating the Virtual Environment

- **Windows (Command Prompt):**

```
.venv\Scripts\activate
```

- **Windows (PowerShell):**

```
.venv\Scripts\Activate.ps1
```

- **macOS/Linux (Bash/Zsh):**

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

Once activated, the terminal prompt may change to indicate the active environment, e.g., `(.venv) $`.

Note: On newer windows running a powershell script is more secure and so will not allow you to run the activate script. You can change the execution policy by running the following command in an elevated PowerShell (Run as Administrator):

```
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser
```

You may want to run this powershell outside of vscode so that you can run it as an admin. I am not sure how you do that inside vscode.

Installing Packages in a Virtual Environment

After activation, you can install packages using `pip`:

```
pip install package_name
```

For example:

```
pip install requests
```

To save installed packages to a `requirements.txt` file:

```
pip freeze > requirements.txt
```

This allows you to recreate the environment later.

To install dependencies from a requirements file:

```
pip install -r requirements.txt
```

How to Check If You Are Running Inside a Virtual Environment

Various ways of checking if you are running under a virtual environment.

Check the Terminal Prompt

When a virtual environment is activated, the terminal usually **shows the environment name** in parentheses at the beginning of the command line:

```
(.venv) user@computer:~/project$
```

If you see this, it means the virtual environment is active.

Use `which python` or `where python`

To check the Python path:

Windows (Command Prompt or PowerShell):

```
```powershell
where python
```
```

macOS/Linux:

```
```bash
which python
```
```

If the output shows a path inside your virtual environment directory (e.g., `venv/bin/python` or `venv\Scripts\python.exe`), then you are running inside `venv`.

Use `sys` in Python

You can verify by checking `sys.prefix` inside Python:

```

```

```
import sys
print(sys.prefix)
```

- If the output **points to the virtual environment directory** (`venv`), then you are inside the virtual environment.
- If it points to the **global Python installation**, then the virtual environment is **not active**.

Use the `VIRTUAL_ENV` Environment Variable

Check for the `VIRTUAL_ENV` environment variable:

macOS/Linux:

```
```bash
echo $VIRTUAL_ENV
```
```

Windows (Command Prompt):

```
```cmd
echo %VIRTUAL_ENV%
```
```

Windows (PowerShell):

```
```powershell
$env:VIRTUAL_ENV
```
```

If it **prints a path**, that's your virtual environment.

If it's **empty**, you are not inside a virtual environment.

Deactivating and Deleting a Virtual Environment

To exit the virtual environment:

```
deactivate
```

To delete a virtual environment, remove its directory:

```
rm -rf .venv  # macOS/Linux
rd /s /q .venv  # Windows (Command Prompt)
```

Opening a Python Command Line in VS Code

How to use vscode to get a python command line.

Using the Integrated Terminal

1. Open **VS Code**.
2. Open your project folder.
3. Open the integrated terminal:
   - Use the shortcut **Ctrl +** `` ` `` (backtick) or
   - Go to **View > Terminal** from the menu.
4. In the terminal, type:

   ```
   python
   ```

   or, if using a virtual environment:

   ```
   .venv\Scripts\python   # Windows
   source .venv/bin/python  # macOS/Linux
   ```

Using the Python Interactive Window (Jupyter-like)

1. Install the Python extension in VS Code.
2. Open the Command Palette (`Ctrl + Shift + P`).
3. Search for **"Python: Show Python Interactive Window"** and select it.
4. A new interactive window will open.

Most Common Names for Virtual Environments

The most commonly used names for Python virtual environments are:

1. `venv` – The standard and most widely used name.
2. `env` – Another popular choice, though it can sometimes conflict with system variables.
3. `.venv` – Preferred in some projects because the dot (`.`) hides it in file explorers and Git.

Note: If you don't specify a name when creating a virtual environment, Python will return an error because a name is required.

Can I switch to a virtual environment if I had already used the global environment for the project?

1. Yes, you can.
2. Go ahead and setup the venv for the project even though previously you had used global setup
3. When you setup the venv, you have to re-pip-install all the modules again as it won't recognize global anymore
4. It is recommended to do so

Add venv to .gitgnore

1. Clearly you don't want the .venv sub directory to be checked into your git server
2. So you have to place an entry in the .gitignore file in the root directory of your project

Here is that line in .gitignore

```
.venv/
```

indicating all files under .venv sub directory needs to be ignored for git.

Conclusion

Ensuring that you are working within a Python virtual environment helps avoid dependency conflicts and keeps your projects organized. By using these methods, you can efficiently create, manage, and verify your virtual environments.

Key References

1. Python Official venv Documentation
2. Virtual Environments and Packages – Python.org
3. Managing Python Dependencies – Real Python