

Python for Engineers

Python Tip: Always Use a Virtual Environment



Shantnu Tiwari

Aug 21, 2023 • 2 min read

I have been using Python so long that using a virtual environment for each project has become second nature. But I recently had the chance to work with beginners and had to explain why a venv is needed.

The actual steps of creating an environment is easy— 1 or 2 lines of code. The hard part is understanding why you would want to and what problem it solves.

The danger of messing with system Python

Most *nix systems, but especially Linux and Mac come with Python installed as part of the system. This is because Python is used for many system libraries that are installed on these systems. If your system Python goes corrupt, many of these utilities will start behaving unexpectedly.

And so when using Python and installing libraries, you *install* globally – because then you are changing the li

Signup for More!

system python.

Python's libraries are usually well written, but most are written by volunteers and can cause issues. Not by themselves, but due to clashes with other libraries. If library *ABC* depends on another library *XYZ* version 2.1, and you update *XYZ* to 3.3, that might cause *ABC* to break, especially if *XYZ* isn't backward compatible.

I've seen this happen even with popular and well tested libraries like PyTorch, Keras and Numpy— all are used in machine learning and you think would be well tested together.

Multiple Projects

This can even happen if you share libraries between multiple projects. Project 1 and 2 are sharing Python libraries, and one day you update the dependencies for Project 2, but that causes Project 1 to start failing.

Solution: Each Python uses its own virtual environment. That means no libraries will clash with each other. If you want you can even have multiple environments with different Python versions.

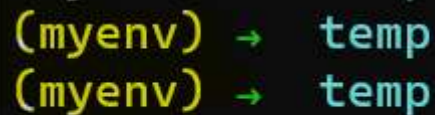
The step to create a venv is simple.

```
python -m venv myenv
```

where *myenv* is the name of the folder. You can then activate the environment (on Linux and Mac):

```
source ./myenv/bin/activate
```

You will know the virtual env has activated because you will see a *(myenv)* on the command line:



```
(myenv) → temp  
(myenv) → temp
```

You can also use the Linux *which* command to confirm you are using the local python:

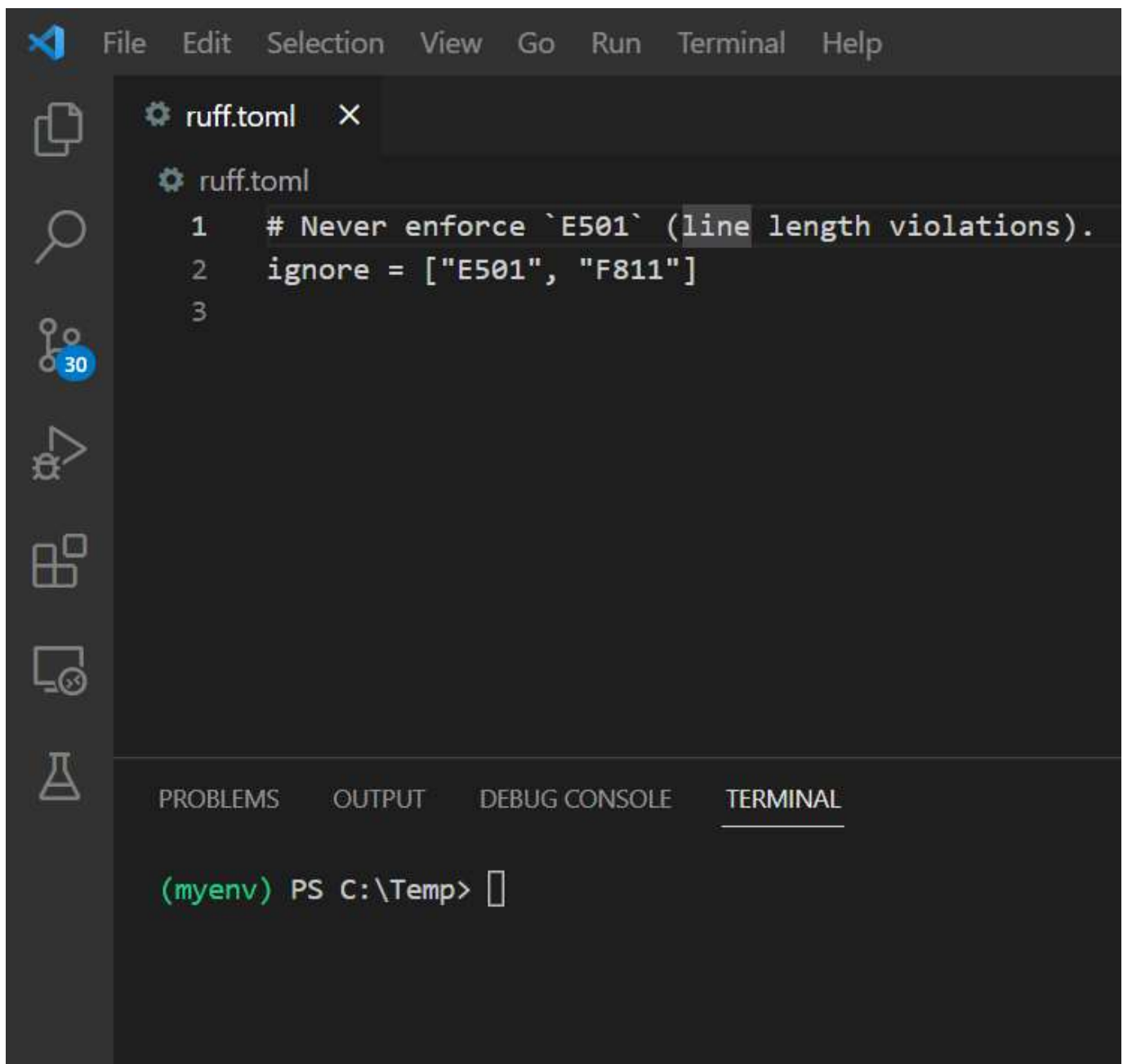


```
(myenv) → temp  
(myenv) → temp which python  
/home/st/temp/myenv/bin/python  
(myenv) → temp which pip  
/home/st/temp/myenv/bin/pip  
(myenv) → temp
```

You can now install any python library you want, and it will only be installed in this folder and in this environment. When you are done, you can delete the *myenv* folder and get rid of all your local python libraries.

Automatically activating the environment

There are shell tools that will automatically activate the virtual environment, but I found the best way is to use VS Code. Once you activate a venv it, Code remembers and will automatically load it next time.



tldr;

- Never install Python libraries globally
- Each project you have must have its own virtual environment

Advanced further reading: A bit advanced but 2 excellent reads that look at alternatives to pip and why you shouldn't use them:

<https://www.bitecode.dev/p/relieving-your-python-packaging-pain>

<https://www.bitecode.dev/p/why-not-tell-people-to-simply-use>

**Sign up to know when the next post is out,
and get exclusive content**

Enter your email

Subscribe

Awesome Python Library: Tenacity

Link: <https://tenacity.readthedocs.io/en/latest/>
When writing code or tests in Python, one issue I had was when the code would fail du...



Shantnu Tiwari
Apr 10, 2024 • 2 min read

So Google's Gemini Doesn't Like Python Programming and Sanskrit?

I have been playing around with Google's Gemini Pro. Recently, I wanted to write a blog on Python's decorators and wanted to get...



Shantnu Tiwari
Feb 25, 2024 • 4 min read

Python for Engineers © 2024

Powered by Ghost