Linux PYTHON PATH Environment Variable

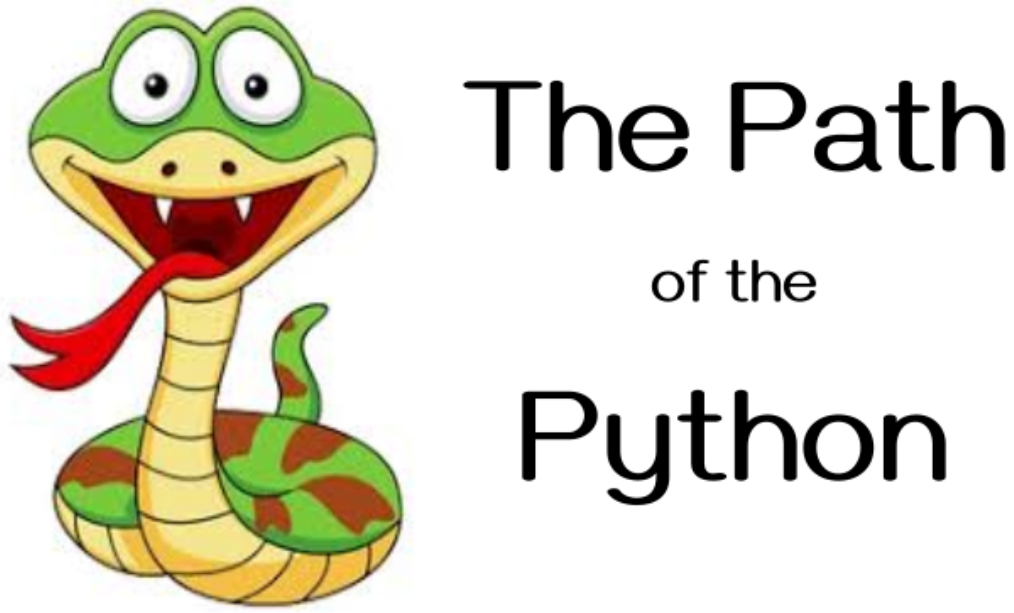

Figure 1: The Path of the Python

Why this separate document? I made many MANY strange mistakes with my PYTHONPATH environment variable in Linux. It's not a lot different from the one you'd make in Windows, but I got tripped multiple times. I wanted to help others from wasting the time that I did. I also wanted to document my findings for ***future Thom***, because ***future Thom*** my forget some things that are fresh in the mind of ***current Thom***.

Where To Locate Your PYTHONPATH Directories

Your PYTHONPATH is a directory or list of directories that contain your reuseable Python tools that you'd like to import into the many Python code files that you will write. Where should this directory reside on your file system? Where you want it to reside.

In case you read, "make sure you put an `__init__.py` file in each directory added to your PYTHONPATH environment variable", you do not need to do this anymore with Python3.

Even for subdirectories below a directory on your PYTHONPATH, you no longer need an `__init__.py` file in order to access those.

By adding subdirectories, and by adding additional directories to your PYTHONPATH, you can have any type of directory organization that you wish to have for these reuseable Python module files and reliably import them into your scripts.

PLEASE Use Git Version Control For Your Reuseable Python Modules

I also want to encourage you to make each directory structure that you add to your PYTHONPATH a git repo. This way you can maintain these Python tools using version control, push updates to them on your favorite git cloud services such as DagsHub, GitHub, GitLab, etc. The advantages are of course that you can then:

- clone these tools to any other machine where you want to also use those reuseable tools,

- push changes you make from one machine up to your central repo, and
- pull your updates from your central repo down to the other machines.

Make Sure Your Modules In Your PYTHONPATH Directories Can Be Found By Your Python Programs

FIRST, check to see if you your system has an existing PYTHONPATH environment variable set by using

`python3 -c "import sys; print(sys.path)`.

You most likely do have one already established.

If you do not have any existing PYTHONPATH, ***which will be highly unlikely in linux***, you'd do the following.

```
export PYTHONPATH="/home/user/my_first_cool_python_dir"
```

NOTE that `~` is Linux shorthand for `/home/user`, but I could not use that in place of `/home/user` in the above export statement and get it to work.

Thus, to add new directories, ***that contain your valuable reusable scripts***, add the following line for each such directory to your `~/.bashrc` file.

```
export PYTHONPATH="${PYTHONPATH}:/home/${USER}/your/path/to/your/reusable/python/modules"
```

OR, using some other ABSOLUTE PATH,

```
export PYTHONPATH="${PYTHONPATH}:/your/path/to/your/reusable/python/modules"
```

On some systems you may add these lines to a `~/.profile` file or some other similarly named file. The main thing is to find the file name for your OS that loads environment variables during startup.

It's likely best to add new `export PYTHONPATH="${PYTHONPATH}:/dir1/dir2"` lines below your other ones where `/dir1/dir2` is some absolute path to your reusable Python modules.

To make these additions to your `~/.bashrc` take effect immediately without rebooting, run `source ~/.bashrc` from a terminal. However, please note that if you wanted certain variables off of the PYTHONPATH, this will not help you. To be extra safe about how your PYTHONPATH environment variable is rebuilt, just reboot.

I also needed to restart my integrated development environment (IDE) for my imports to run without errors.

How To Import Your Functions, Classes, And Variables

A module in a directory on your PYTHONPATH is simply a Python file having a `.py` extension. From within a python script, you'd do the following to import a function named `my_function` from within the `my_module.py` file that is in one of the directories of your PYTHONPATH.

```
import my_module as mm

result = mm.my_function(arg1, arg2, ...)
```

OR something like

```
from my_module import my_function

result = my_function(arg1, arg2, ...)
```

If importing a class that you plan to instantiate, you'd do something like the following.

```
import my_class as My_C

my_c = My_C(arg1, arg2, ...)  # etc.
```

You can even store important variables, lists, and other objects for use in various ways.

```
import my_object as mo

...
result = some_function(mo, arg2, arg3, ...)
```

What if you want to access functions, classes, or variables from a subdirectory of one of you PYTHONPATH directories?

```
from subdirectory_of_PythonPath_directory.my_module import my_function

result = my_function(arg1, arg2, ...)
```

Whether or not you add sub directories under PYTHONPATH directories is up to you. Organize your Python modules the way that you want to.