How to Add Python to PATH

by Ian Currie

basics best-practices devops

Mark as Completed

Share

Table of Contents

You may need to add Python to PATH if you've installed Python, but typing `python` on the command line doesn't seem to work. You may be getting a message saying that the term `python` isn't recognized, or you may end up with the wrong version of Python running.

A common fix for these problems is adding Python to the PATH environment variable. In this tutorial, you'll learn how to add Python to PATH. You'll also learn about what PATH is and why PATH is vital for programs like the command line to be able to find your Python installation.

> **Note:** A path is the address of a file or folder on your hard drive. The PATH environment variable, also referred to as just PATH or *Path*, is a list of paths to directories that your operating system keeps and uses to find executable scripts and programs.

The steps that you'll need to take to add something to PATH will depend significantly on your operating system (OS), so be sure to skip to the relevant section if you're only interested in this procedure for one OS.

Note that you can use the following steps to add any program to PATH, not just Python.

> **Supplemental Code: Click here to download free supplemental code** that'll walk you through changing PATH across operating systems.

How to Add Python to PATH on Windows

The first step is to locate the directory in which your target Python executable lives. The path

The first step is to locate the directory in which your target Python executable lives. The path to the directory is what you'll be adding to the `PATH` environment variable.

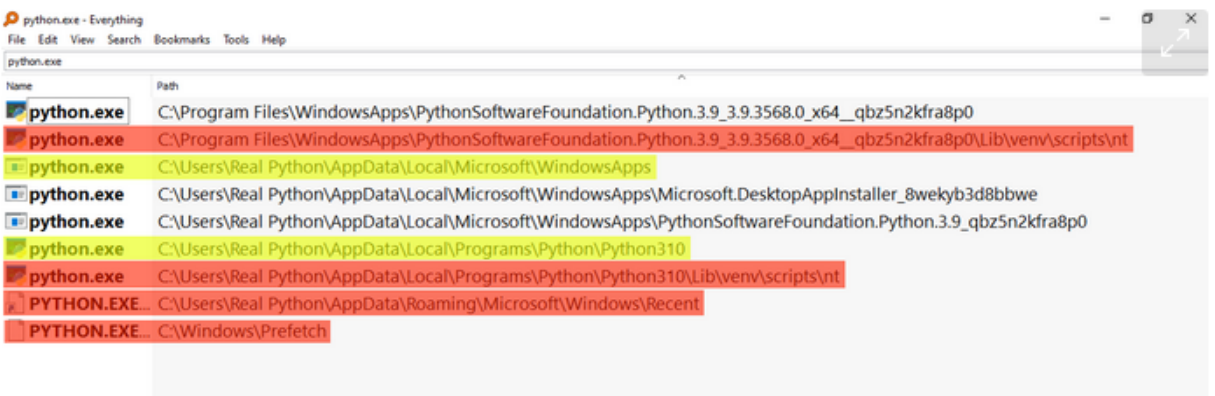
To find the Python executable, you'll need to look for a file called `python.exe`. The Python executable could be in a directory in `C:\Python\` or in your `AppData\` folder, for instance. If the executable were in `AppData\`, then the path would typically look something like this:

```
Shell
C:\Users\<USER>\AppData\Local\Programs\Python
```

In your case, the `<USER>` part would be replaced by your currently logged-in user name.

Once you've found the executable, make sure it works by double-clicking it and verifying that it starts up a Python REPL in a new window.

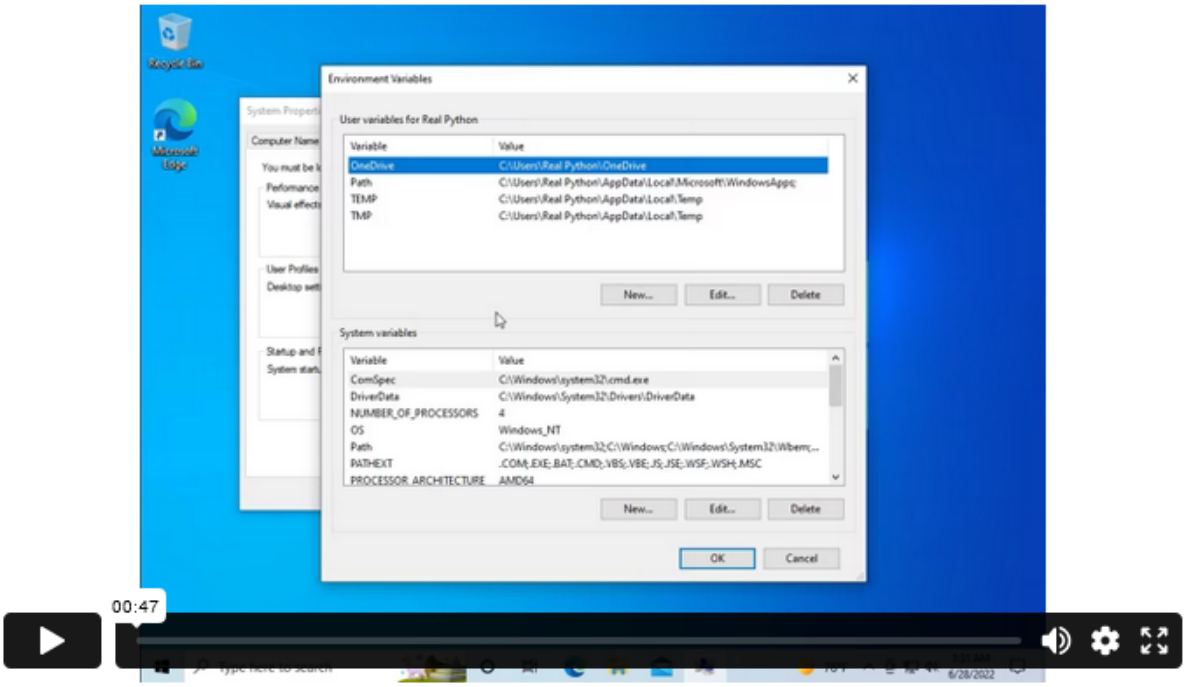
If you're struggling to find the right executable, you can use Windows Explorer's search feature. The issue with the built-in search is that it's painfully slow. To perform a super-fast full system search for any file, a great alternative is Everything:



Those paths highlighted in yellow, namely those at `\WindowsApps` and `\Python310`, would be ideal candidates to add to `PATH` because they look like executables at the root level of an installation. Those highlighted in red wouldn't be suitable because some are part of a virtual environment—you can see `venv` in the path—and some are shortcuts or internal Windows installations.

You may also encounter Python executables that are installed within the folder for a different program. This is due to the fact that many applications bundle their own version of Python within them. These bundled Python installations would also be unsuitable.

Once you've located your Python executable, open the Start menu and search for the *Edit the system environment variables* entry, which opens up a *System Properties* window. In the *Advanced* tab, click on the button *Environment Variables*. There you'll see *User* and *System* variables, which you'll be able to edit:



In the section entitled *User Variables*, double-click on the entry that says *Path*. Another window will pop up showing a list of paths. Click the *New* button and paste the path to your Python executable there. Once that's inserted, select your newly added path and click the *Move Up* button until it's at the top.

That's it! You may need to reboot your computer for the changes to take effect, but you should now be able to call `python` from the command line.

For setting the `PATH` environment variable from the command line, check out the section on Configuring Environment Variables in the Windows Python coding setup guide. You can also find instructions in the supplemental materials:

> **Supplemental Code: Click here to download free supplemental code** that'll walk you through changing PATH across operating systems.

You may also want to set up `PATH` on your Linux or macOS machine, or perhaps you're using Windows Subsystem for Linux (WSL). If so, read the next section for the procedure on UNIX-based systems.

How to Add Python to `PATH` on Linux and macOS

Since Python typically comes pre-installed on UNIX-based systems, the most common problem on Linux and macOS is for the wrong `python` to run, rather than not finding *any* `python`. That said, in this section, you'll be troubleshooting not being able to run `python` at all.

> **Note:** Depending on your particular system, you may have a `python` program for Python 2, and a `python3` for Python 3. In other instances, both `python` and `python3` will point to the same executable.

The first step is locating your target Python executable. It should be a program that you can run by first navigating to the containing directory and then typing `./python` on the command line.

You need to prepend the call to the Python executable with its relative path in the current folder (`./`) because otherwise you'll invoke whichever Python is currently recorded on your PATH. As you learned earlier, this might not be the Python interpreter that you want to run.

Often the Python executable can be found in the `/bin/` folder. But if Python is already in the `/bin/` folder, then it's most likely already on PATH because `/bin/` is automatically added by the system. If this is the case, then you may want to skip to the section on the order of paths within PATH.

Since you're probably here because you've installed Python but it's still not being found when you type `python` on the command line, though, you'll want to search for it in another location.

> **Note:** A great search utility for quickly searching large folders is fzf. It works from the command line and will search all files and folders within your current working directory. So you might search for `python` from your home directory, for instance. fzf will then show you the paths that contain `python`.

That said, it might be that `/bin/` has been removed from PATH altogether, in which case you might skip forward to the section on mangaging PATH.

Once you've located your Python executable and are sure it's working, take note of the path for later. Now it's time to start the process of adding it to your PATH environment variable.

First, you'll want to navigate to your home folder to check out what configuration scripts you have available:

Shell

```
$ cd ~
$ ls -a
```

You should see a bunch of configuration files that begin with a period (`.`). These are colloquially known as dotfiles and are hidden from `ls` by default.

One or two dotfiles get executed whenever you log in to your system, another one or two run

One or two dotfiles get executed whenever you log in to your system, another one or two run whenever you start a new command-line session, and most others are used by other applications for configuration settings.

You're looking for the files that run when you start your system or a new command-line session. They'll probably have names similar to these:

- `.profile`
- `.bash_profile`
- `.bash_login`
- `.zprofile`
- `.zlogin`

The keywords to look for are *profile* and *login*. You should, in theory, only have one of these, but if you have more than one, you may need to read the comments in them to figure out which ones run on login. For example, `.profile` file on Ubuntu will typically have the following comment:

```
Shell
# This file is not read by bash(1), if ~/.bash_profile or ~/.bash_login
# exists.
```

So, if you have `.profile` but also `.bash_profile`, then you'll want to use `.bash_profile`.

You can also use a `.bashrc` or `.zshrc` file, which are scripts that run whenever you start a new command-line session. Run command (rc) files are common places to put `PATH` configurations.

> **Note:** Pedantically speaking, rc files are generally for settings that affect the look and feel of your command-line prompt, not for configuring environment variables like `PATH`. But you can use rc files for your `PATH` configuration if you prefer.

To add the Python path to the beginning of your `PATH` environment variable, you're going to be executing a single command on the command line.

Use the following line, replacing `<PATH_TO_PYTHON>` with your actual path to the Python executable, and replace `.profile` with the login script for your system:

```
Shell
$ echo export PATH="<PATH_TO_PYTHON>:$PATH" >> ~/.profile
```

This command adds `export PATH="<PATH_TO_PYTHON>:$PATH"` to the end of `.profile`. The command `export PATH="<PATH_TO_PYTHON>:$PATH"` prepends `<PATH_TO_PYTHON>` to the `PATH`

This command adds `export PATH="<PATH_TO_PYTHON>:$PATH"` to the end of `.profile`. The command `export PATH="<PATH_TO_PYTHON>:$PATH"` prepends `<PATH_TO_PYTHON>` to the PATH environment variable. It's similar to the following operation in Python:

Python

```
>>> PATH = "/home/realpython/apps:/bin"
>>> PATH = f"/home/realpython/python:{PATH}"
>>> PATH
'/home/realpython/python:/home/realpython/apps:/bin'
```

Since PATH is just a string separated by colons, prepending a value involves creating a string with the new path, a colon, then the old path. With this string, you set the new value of PATH.

To refresh your current command-line session, you can run the following command, replacing `.profile` with whichever login script you've chosen:

Shell

```
$ source ~/.profile
```

Now, you should be able to call `python` from the command line directly. The next time you log in, Python should automatically be added to PATH.

If you're thinking this process seems a bit opaque, you're not alone! Read on for more of a deep dive into what's going on.

Understanding What PATH Is

PATH is an environment variable that contains a list of paths to folders. Each path in PATH is separated by a colon or a semicolon—a colon for UNIX-based systems and a semicolon for Windows. It's like a Python variable with a long string as its value. The difference is that PATH is a variable accessible by almost all programs.

Programs like the command line use the PATH environment variable to find executables. For example, whenever you type the name of a program into the command line, the command line will search various places for the program. One of the places that the command line searches is PATH.

All the paths in PATH need to be directories—they shouldn't be files or executables directly. Programs that use PATH take each directory in turn and search all the files within it. Subdirectories within directories in PATH don't get searched, though. So it's no good just adding your root path to PATH!

It's also important to note that programs that use PATH typically don't search for anything

It's also important to note that programs that use PATH typically don't search for anything except executables. So, you can't use PATH as a way to define shortcuts to commonly used files.

Understanding the Importance of Order Within PATH

If you type `python` into the command line, the command line will look in each folder in the PATH environment variable for a `python` executable. Once it finds one, it'll *stop searching*. This is why you *prepend* the path to your Python executable to PATH. Having the newly added path *first* ensures that your system will find this Python executable.

A common issue is having a failed Python installation on your PATH. If the corrupted executable is the first one that the command line comes across, then the command line will try and run that and then abort any further searching. The quick fix for this is just adding your new Python directory *before* the old Python directory, though you'd probably want to clean your system of the bad Python installation too.

Reordering PATH on Windows is relatively straightforward. You open the GUI control panel and adjust the order using the *Move Up* and *Move Down* buttons. If you're on a UNIX-based operating system, however, the process is more involved. Read on to learn more.

Managing Your PATH on UNIX-based Systems

Usually, your first task when managing your PATH is to see what's in there. To see the value of any environment variable in Linux or macOS, you can use the `echo` command:

Shell

```
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/home/realpython/badpython:/usr/bin:/sbin:/b
```

Note that the `$` symbol is used to tell the command line that the following identifier is a variable. The issue with this command is that it just dumps all the paths on one line, separated by colons. So you might want to take advantage of the `tr` command to translate colons into newlines:

Shell

```
$ echo $PATH | tr ":" "\n"
/usr/local/sbin
/usr/local/bin
/usr/sbin
/home/realpython/badpython
/usr/bin
/sbin
```

```
/home/realpython/badpython
/usr/bin
/sbin
/bin
/usr/games
/usr/local/games
```

In this example, you can see that `badpython` is present in PATH. The ideal course of action would be to perform some PATH archaeology and figure out where it gets added to PATH, but for now, you just want to remove it by adding something to your login script .

Since PATH is a shell string, you don't have access to convenient methods to remove parts of it, like you would if it were a Python list. That said, you can pipe together a few shell commands to achieve something similar:

Shell

```
export PATH=`echo $PATH | tr ":" "\n" | grep -v 'badpython' | tr "\n" ":"`
```

This command takes the list from the previous command and feeds it into `grep`, which, together with the `-v` switch, will filter out any lines containing the substring `badpython`. Then you can translate the newlines back to colons, and you have a new and valid PATH string that you use right away to replace your old PATH string.

Though this can be a handy command, the ideal solution would be to figure out where that bad path gets added. You could try looking at other login scripts or examine specific files in `/etc/`. In Ubuntu, for instance, there's a file called `environment`, which typically defines a starting path for the system. In macOS, that might be `/etc/paths`. There can also be `profile` files and folders in `/etc/` that might contain startup scripts.

The main difference between configurations in `/etc/` and in your home folder is that what's in `/etc/` is system-wide, while whatever's in your home folder will be scoped to your user.

It can often involve a bit of archeology to track down where something gets added to your PATH, though. So, you may want to add a line in your login or rc script that filters out certain entries from PATH as a quick fix.

Conclusion

In this tutorial, you've learned how to add Python, or any other program, to your PATH environment variable on Windows, Linux, and macOS. You also learned a bit more about what PATH is and why its internal order is vital to consider. Finally, you also discovered how you might manage your PATH on a UNIX-based system, seeing as it's more complex than managing your PATH on Windows.