Saif Uddin Mahmud Follow
Tech Enthusiast | College Junior
Jul 31 · 4 min read

Make Linux CLI tools

using Python!

There is no going back once you get comfortable with the Ubuntu Terminal—you get used to doing things *much faster* (with the added benefit of looking like a pro 😎).

Even with the massive arsenal of CLI tools available at your fingertips, you may want a customized one to automate some of your day-to-day wizardry. Bored out of my mind on a slow summer day, I found myself wanting to generate random passwords and hashes from the CLI. Below, I'll show you the two ways of making a CLI tool in Python so you can save the hours I wasted searching for a way!

*To follow along, you'll need a **Linux** machine/VM with **Python 3.x installed along with pip3**. Alternatively, you can use a free, online instance on Amazon EC2 / Koding. All the code used in this tutorial is available here and here.*

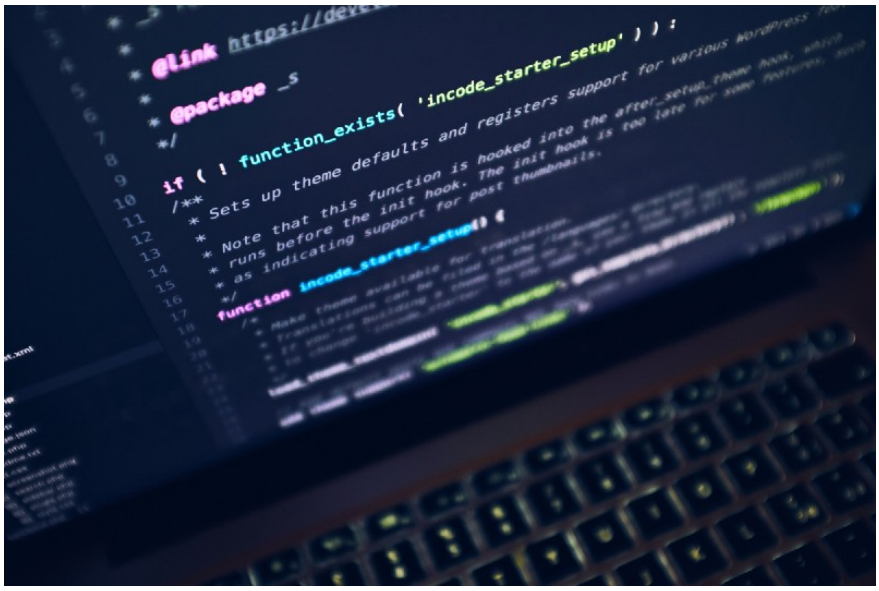

Photo by Luca Bravo on Unsplash

#1 Using the command line

Of course, no CLI tool is complete without the parameters, arguments, and options. Although you may be inclined to reinvent the seemingly simple wheel using *args and *kwargs, please refrain from doing so. We'll be using a third-party library `click` —based on `optparse`. Why click? I'll let the creators of click answer that.

First, you'll need to go to your home directory and create a folder named `bin`. We're going to store our python program in this folder, and add this folder to our `PATH`. We'll do this in our *.profile* so it persists across sessions.

```
$ gedit ~/.profile
```

```
//Add the next line to end of the file
export PATH=$PATH":$HOME/bin"
```

```
//Restart terminal or run the following
$ source ~/.profile
```

Now copy the following code snippet into a file called `genChars` inside bin (you can change the name to whatever you want to use in the CLI). The reason we do not have to include `.py` is that the shebang on line 1 lets the OS know what to run it with.

```
#!/usr/bin/env python3
import sys
import random
import click

random.seed()

def genRandInt():
    return chr(random.randint(48, 57))

def genRandLowerCase():
    return chr(random.randint(97, 122))

def genRandUpperCase():
    return chr(random.randint(65, 90))

@click.command()
@click.option('--length', '-l', default=12, help='Def
def main(length):
    '''
    CLI tool to help you generate a random string.
    '''
```

Make sure you have `click` installed. You can do this by running the command `pip3 install click`. Now run `chmod +x genChars` to make the file executable. Done!

You can now open up a terminal anywhere in the system and type `$ genChars -l 15` to generate a random 15-char string. `-l` is optional, and the default length is 12.

To get a better sense `click` and `function decorators`, refer to the links.

#2 Using PIP

Python makes it super easy to build a `package` and subsequently install it with `pip`. After that, you'll be able to use your program directly from the command line!

Let's use pwHash to illustrate this second method. Set up the following folder structure anywhere in your system:

```
- pwHash
    -- pwHash
        --- __init__.py
        --- __main__.py
    -- install.sh
    -- setup.py
```

Let's observe the files individually.

__init__.py
The sole purpose of this file is to let `pip` know that this is a package. To that end, an empty file will do for now.

__main__.py
This file holds all the code. There can be multiple files, but since our tool is pretty simple and straightforward, we can dump it all inside a single file. Let's look at the code.

```
1   #!/usr/bin/env python3
2   import hashlib
3   import click
4   import sys
5
6   @click.command()
7   @click.argument('stringtohash', nargs=1)
8   @click.argument('algorithm', nargs=1)
9   def hashThis(stringtohash, algorithm):
10      """
11      CLI tool used to generate the the hash of STRINGT
12      """
13
14      #if algorithm is not available, show err
15      if algorithm not in hashlib.algorithms_guaranteed
16          sys.exit("Invalid algorithm. Valid list: " +
```

The code should be self-explanatory but do drop any questions you have in the comments below.

```
1   from setuptools import setup
2
3   setup(
4       name= 'pwHash',
5       version = '0.1.0',
6       packages = ['pwHash'],
7       entry_points = {
8           'console_scripts': [
9               'pwHash = pwHash.__main__:hashThis'
10          ]
11      },
12      install_requires = ['click']
```

setup.py
This file lets pip know the details of the package such as name, version, and entry_points. Another important aspect it tackles is the dependency list indicated on line 12; `pip` will install all the packages in the list if they don't exist on the user's system.

install.sh
This is a one-liner (`pip3 install -e .`) you could type into the Terminal manually too. Make sure you run `chmod +x install.sh` first to give it executable rights. `-e` lets `pip` know this is in development mode (so we do not have to keep installing and uninstalling after every change). `.` asks `pip` to install all the packages in the folder (which, if you recall, is only 1 anyway in our case)

Once you run install.sh, our job here is done! You can now open up the CLI and type in `pwHash [stringToHash] [algorithmToUse]` to use the tool. Don't worry if you don't know what algorithm to use right now, using a wrong algorithm will flash out all the valid-and-available ones on the screen for you.

. . .

You can use either of the methods to build your own awesome CLI tools. Do let us know when you do so, we'll be sure to check your masterpiece out! **Good luck!**