Lab 15: Case Study: Building A Bill of Materials App in Python

ITE315: Scripting Languages and System Administration

Contents

1 Overview

Let's step through the process of building an application in Python. For this lab, we will write a program that creates a Bill of Materials for an installer application.

2 Background

Software installers need to keep track what files to install onto a system. For security reasons, the installer needs to make certain that the files it's being asked to install from a disk or over the network are correct. One way to do this is to include an encrypted "bill of materials" file in the software package. This file needs to include a list of the files in the install package and some unique identifer for each of the files.

2.1 Signatures and Hashing

In cryptography, we have the ability to create a "cryptographic hash" of a block of data. This is a cryptographically secure unique identifer generated by a hash function. There's lot of gory detail about such things that we'll gloss over for this problem. Just think of a cryptographic hash as being some really unique string of bits that identifies a block of data. We need some method of generating these values.

The tendency is to attempt to write your function. Don't give in to those base tendencies: use pre-existing tools to do this for you!

2.2 Problem Statement

So... here's our problem statement: Write a program in Python that, for a folder specified on the command line, write a text file contains the bill of materials (BoM) for the files in that folder. The BoM will have one line per file, with each line having two entries: the basename of the file and a crypto. hash for that file, separated by commas. If the user should fail to provide a folder name on the command-line, then generate the BoM for the current folder.

A few questions to consider:

- Where does the output go for this program? Into a file? To standard output?
- Is there special processing? What if the folder contains subfolders? Do you have to recur through the folder tree?
- Do you build or buy? Use some third party package for crypto or roll your own hash function?

3 Instructions

Let's work through the problem in a step-by-step manner.

3.1 First, We Make Some Hash

No silly students, neither the culinary kind nor the recreational pharmaceutical sort, but cryptographic hashing! Let's use some Google-Fu and search for "python cryptographic hashing". We get a couple of hits worth investigating: `hashlib` and `pycrypto`. The `hashlib` library comes with the Python

distribution while `pycrypto` is a Python front-end to the Crypto++ C++ library. Let's use `pycrypto`.

Use `pip` to install `pycrypto`:

```
pip install pcrypto
```

Looking at the documentation for `pcrypto`, this works on a string. Put the following into a `Python` file and have your code hash a string you get from standard input:

```
#!/usr/bin/python
from Crypto.Hash import SHA512
import sys
def main(argv):
    # Step 1: Confirm the correct number of parameters in argv
    # Step 2: Copy the string from the command line to a local
      variable
    #         and use a hasher to hash it.

if __name__ == __main__:
    main(sys.argv)
```

Hint: To print the hash of a string:

```
hasher = SHA512.new()
hasher.update(b"abcd")
print hasher.hexdigest()
```

3.2 Let's File It

So... the discovery is that the hashing functions work on a strings. That means we need to read the contents of a file into a string and then hash it.

Consider the following code:

```
def main(argv):
    if len(argv) < 2:
        print("Usage: hashfile.py fname")
        print("       fname -> name of file to be hashed")
    else:
        fname = argv[1]
        hasher = SHA512.new()
        with open(fname, 'rb') as aFile:
```

```
9               buf = aFile.read()
                hasher.update(buf)
11          print(hasher.hexdigest())
```

Enter this code into a `Python` file and execute it. Does this code do what we need?

Well... it's a leading trick question. We need something that will work for very large files. Given that strings have a 2^{16} size limit in Python, we need to read the data in $65,536$ byte block. Remembering that the Python `read()` function takes a single argument with the number of bytes to read, modify this code so that it will read the entire file in blocks. HINT: If you keep updating the same hashing, then it concatenates the hash code for the new value onto the old value.

3.3 Now For All Files

Just about there... we need to find a way to where we get hashes of all of the files in a folder. For now, we will limit ourselves to just the files in a folder.

In Perl, we had the ability to do pipelines. Python takes a more traditional approach.

Take a look at the Python documentation: `https://docs.python.org/2/library/os.html`.

Getting there, aren't we? In particular, we have the function `os.listdir()` in the `os` module. This takes a string containing a folder name and returns back a list of sub-folders and files in that folder.

Continuing our documentation dive helps us find the answer to the question: "Is this a file?". In the `os` module documentation, we see a mention of the `os.path` module. Look at the documentation for the that module. Note that we have a function `os.path.isfile()` that returns true if we have a regular file.

Now we can start to do coding. Start by copying your program from the previous section into a new program called `hashfiles.py`. Change the name of the current `main()` function to be called `hashAFile()`. Modify the body of `hashAFile()` so that it uses the parameter as the name of a file to hash and returns a string containing the hash index. Be careful with the indenting of your code... Don't forget that Python is a FIXED-FORMAT language... you'll end up putting the `return()` in your loop if you aren't careful!

Add a new function named `getFileList()`. This function should have one parameter: the name of the folder to be processed. Use the `os.listdir()` and `os.path.isfile()` functions to build a list of files that is returned to the caller. Return the processed list to the caller.

Now it's time to write our new `main()` function. As we did before, we need to check to see if the number of arguments as expected. If it's not, then use the `os.getcwd()` function to set our target folder to the current directory. Call the `getFileList()` function to get a list of the files in the target folder. Then, traverse that list to generate a hash code and print the file name and hash value to standard output.

4 Submission instructions

Attach a copy of your Python src to this assignment in Blackboard. Also attach a transcript of the execution of your script. NOTE: this is NOT screenshots! Use file redirection to get the output from program into a file.