CHAPTER

Our First Python Forensics App

3

CHAPTER CONTENTS

INTRODUCTION

In 1998, I authored a paper entitled "Using SmartCards and Digital Signatures to Preserve Electronic Evidence" (Hosmer, 1998). The purpose of the paper was to advance the early work of Gene Kim, creator of the original Tripwire technology (Kim, 1993) as a graduate student at Purdue. I was interested in advancing the model of using one-way hashing technologies to protect digital evidence, and specifically I was applying the use of digital signatures bound to a SmartCard that provided two-factor authenticity of the signing (Figure 3.1).

Years later I added trusted timestamps to the equation adding provenance, or proof of the exact "when" of the signing.

Two-factor authentication combines a secure physical device such as a SmartCard with a password that unlocks the capability of the card's. This yields "something held" and "something known." In order to perform applications like signing, you must be in possession of the SmartCard and you must know the pin or password that unlocks the cards function.

Thus, my interest in applying one-way hashing methods, digital signature algorithms, and other cryptographic technologies to the field of forensics has been a 15-year journey . . . so far. The application of these technologies to evidence preservation, evidence identification, authentication, access control decisions and network protocols continues today. Thus I want to make sure that you have a firm understanding of the underlying technologies and the many applications for digital investigation, and of course the use of Python forensics.

FIGURE 3.1

Cryptographic SmartCard.

Before I dive right in and start writing code, as promised I want to set up some ground rules for using the Python programming language in forensic applications.

NAMING CONVENTIONS AND OTHER CONSIDERATIONS

During the development of Python forensics applications, I will define the rules and naming conventions that are being used throughout the cookbook chapters in the book. Part of this is to compensate for Python's lack of the enforcement of strongly typed variables and true constants. More importantly it is to define a style that will make the programs more readable, and easier to follow, understand, and modify or enhance.

Therefore, here are the naming conventions I will be using.

Constants

Rule: Uppercase with underscore separation

Example: HIGH_TEMPERATURE

Local variable name

Rule: Lowercase with bumpy caps (underscores are optional)

Example: currentTemperature

Global variable name

Rule: Prefix *gl* lowercase with bumpy caps (underscores are optional)

Note: Globals should be contained to a single module

Example: gl_maximumRecordedTemperature

Functions name

Rule: Uppercase with bumpy caps (underscores optional) with active voice

Example: ConvertFarenheitToCentigrade(. . .)

Object name

Rule: Prefix *ob_* lowercase with bumpy caps

Example: ob_myTempRecorder

Module

Rule: An underscore followed by lowercase with bumpy caps

Example: _tempRecorder

Class names

Rule: Prefix *class_* then bumpy caps and keep brief

Example: class_TempSystem

You will see many of these naming conventions in action during this chapter.

OUR FIRST APPLICATION "ONE-WAY FILE SYSTEM HASHING"

The objective for our first Python Forensic Application is as follows:

1. Build a useful application and tool for forensic investigators.
2. Develop several modules along the way that are *reusable* throughout the book and for future applications.
3. Develop a solid methodology for building Python forensic applications.
4. Begin to introduce more advanced features of the language.

Background

Before we can build an application that performs one-way file system hashing I need to better define one-way hashing. Many of you reading this are probably saying, "I already know what a one-way hashing is, let's move on." However, this is such an important underpinning for computer forensics it is worthy of a good definition, possibly even a better one that you currently have.

One-way hashing algorithms' basic characteristics

1. The one-way hashing algorithm takes a stream of binary data as input; this could be a password, a file, an image of a hard drive, an image of a solid state drive, a network packet, 1's and 0's from a digital recording, or basically any continuous digital input.
2. The algorithm produces a message digest which is a compact representation of the binary data that was received as input.
3. It is infeasible to determine the binary input that generated the digest with only the digest. In other words, it is not possible to reverse the process using the digest to recover the stream of binary data that created it.
4. It is infeasible to create a new binary input that will generate a given message digest.
5. Changing a single bit of the binary input data will generate a unique message digest.
6. Finally, it is infeasible to find two unique arbitrary streams of binary data that produce the same digest.

Table 3.1 Popular One-Way Hashing Algorithms

Algorithm	Creator	Length (Bits)	Related standard
MD5	Ronald Rivest	128	RFC 1321
SHA-1	NSA and published by NIST	160	FIPS Pub 180
SHA-2	NSA and published by NIST	224 256 384 512	FIPS Pub 180-2 FIPS Pub 180-3 FIPS PUB 180-4
RIPEMD-160	Hans Dobbertin	160	Open Academic Community
SHA-3	Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche	224, 256, 384, 512	FIPS-180-5

Popular cryptographic hash algorithms?

There are a number of algorithms that produce message digests. Table 3.1 provides background on some of the most popular algorithms.

What are the tradeoffs between one-way hashing algorithms?

The MD5 algorithm is still in use today, and for many applications the speed, convenience, and interoperability have made it the algorithm of choice. Due to attacks on the MD5 algorithm and the increased likelihood of collisions, many organizations are moving to SHA-2 (256 and 512 bits are the most popular sizes). Many organizations have opted to skip SHA-1 as it suffers from some of the same weaknesses as MD5.

Considerations for moving to SHA-3 are still in the future, and it will be a couple of years before broader adoption is in play. SHA-3 is completely different and was designed to be easier to implement in hardware to improve performance (speed and power consumption) for use in embedded or handheld devices. We will see how quickly the handheld devices' manufacturers adopt this newly established standard.

What are the best-use cases for one-way hashing algorithms in forensics?

Evidence preservation: When digital data are collected (for example, when imaging a mechanical or solid state drive), the entire contents—in other words every bit collected—are combined to create a unique one-way hashing value. Once completed the recalculation of the one-way hashing can be accomplished. If the new calculation matches the original, this can prove that the evidence has not been modified. This assumes of course that the original calculated hash value has been safeguarded against tampering since there is no held secret and the algorithms are available. Anyone could recalculate a hash, therefore the chain of custody of digital evidence, including the generated hash, must be maintained.

Search: One-way hashing values have been traditionally utilized to perform searches of known file objects. For example, if law enforcement has a collection of confirmed child-pornography files, the hashes could be calculated for each file. Then any suspect system could be scanned for the presence of this contraband by calculating the hash values of each file and comparing the resulting hashes to the known list of contraband hash values (those resulting from the child-pornography collection). If matches are found, then the files on the suspect system matching the hash values would be examined further.

Black Listing: Like the search example, it is possible to create a list of known bad hash files. These could represent contraband as with CP example, they could match known malicious code or cyber weapon files or the hashes of classified or proprietary documents. The discovery of hashes matching any of these Black Listed items would provide investigators with key evidence.

White Listing: By creating a list of known good or benign hashes (operating system or application executables, vendor supplied dynamic link libraries or known trustworthy application download files), investigators can use the lists to filter out files that they do not have to examine, because they were previously determined as a good file. Using this methodology you can dramatically reduce the number of files that require examination and then focus your attention on files that are not in the known good hash list.

Change detection: One popular defense against malicious changes to websites, routers, firewall configuration, and even operating system installations is to hash a "known good" installation or configuration. Then periodically you can re-scan the installation or configuration to ensure no files have changed. In addition, you must of course make sure no files have been added or deleted from the "known good" set.

Fundamental requirements

Now that we have a better understanding of one-way hashing and its uses, what are the fundamental requirements of our one-way file system hash application?

When defining requirements for any program or application I want to define them as succinctly as possible, and with little jargon, so anyone familiar with the domain could understand them—even if they are not software developers. Also, each requirement should have an identifier such that could be traced from definition, through design, development, and validation. I like to give the designers and developers room to innovate, thus I try to focus on WHAT not HOW during requirements definition (Table 3.2).

Table 3.2 Basic Requirements

Requirement number	Requirement name	Short description
000	Overview	The basic capability we are looking for is a forensic application that walks the file system at a defined starting point (for example, c:\ or /etc) and then generates a one-way hashing value for every file encountered
001	Portability	The application shall support Windows and Linux operating systems. As a general guideline, validation will be performed on Windows 7, Windows 8, and Ubuntu 12.04 LTS environments
002	Key functions	In addition to the one-way hashing generation, the application shall collect system metadata associated with each file that is hashed. For example, file attributes, file name, and file path at a minimum
003	Key results	The application shall provide the results in a standard output file format that offers flexibility
004	Algorithm selection	The application shall provide a wide diversity when specifying the one-way hashing algorithm(s) to be used
005	Error handling	The application must support error handling and logging of all operations performed. This will include a textual description and a timestamp

Design considerations

Now that I have defined the basic requirements for the application I need to factor in the design considerations. First, I would like to leverage or utilize as many of the built-in functions of the Python Standard Library as possible. Taking stock of the core capabilities, I like to map the requirements definition to Modules and Functions that I intend to use. This will then expose any new modules either from third party modules or new modules that need to be developed (Table 3.3).

One of the important steps as a designer or at least one of the fun parts is to name the program. I have decided to name this first program p-fish short for *P*ython-*fi*le *s*ystem *h*ashing.

Next, based on this review of Standard Library functions I must define what modules will be used in our first application:

1. `argparse` for user input
2. `os` for file system manipulation
3. `hashlib` for one-way hashing
4. `csv` for result output (other optional outputs could be added later)
5. `logging` for event and error logging
6. Along with useful miscellaneous modules like `time`, `sys`, and `stat`

Table 3.3 Standard Library Mapping

Requirement	Design considerations	Library selection
User input (000, 003, 004)	Each of these requirements needs input from the user to accomplish the task. For example, 000 requires the user to specify the starting directory path. 003 requires that the user specify a suitable output format. 004 requires us to allow the user to specify the hash algorithm. Details of the exception handling or default settings need to be defined (if allowed)	For this first program I have decided to use the command line parameters to obtain input from the user. Based on this design decision I can leverage the `argparse` Python Standard Library module
Walk the file system (000, 001)	This capability requires the program to traverse the directory structure starting at a specific starting point. Also, this must work on both Windows and Linux platforms	The `OS` Module from the Standard Library provides key methods that provide the ability to walk the file system, `OS` also provides abstraction which will provide cross platform compatibility. Finally, this module contains cross platform capabilities that provide access to metadata associated with files
Meta data collection (003)	This requires us to collect the directory path, filename, owner, modified/access/created times, permissions, and attributes such as read only, hidden, system or archive	
File hashing (000)	I must provide flexibility in the Hashing algorithms that the users could select. I have decided to support the most popular algorithms such as MD5 and several variations of SHA	The Standard Library module `hashlib` provides the ability to generate one-way hashing values. The library supports common hash algorithms such as “md5,” “sha1,” “sha224,” “sha256,” “sha384,” “sha512.” This should provide a sufficient set of selection for the user
Result output (003)	To meet this requirement I must be able to structure the program output to support a format that provides flexibility	The Standard Library offers multiple options that I could leverage. For example, the `csv` module provides the ability to create comma separated value output, whereas the `json` module (Java Object Notation) provides encoder and decoders for `JSON` objects and finally the `XML` module could be leveraged to create XML output

Continued

Table 3.3 Standard Library Mapping—cont'd		
Requirement	**Design considerations**	**Library selection**
Logging and error handling	I must expect that errors will occur during our walk of the file system, for example I might not have access to certain files, or certain files may be orphaned, or certain files maybe locked by the operating system or other applications. I need to handle these error conditions and log any notable events. For example, I should log information about the investigator, location, date and time, and information that pertains to the system that are walked	The Python Standard Library includes a `logging` facility which I can leverage to report any events or errors that occur during processing

Program structure

Next, I need to define the structure of our program, in other words how I intend to put the pieces together. This is critical, especially if our goal is to reuse components of this program in future applications. One way to compose the components is with a couple simple diagrams as shown in Figures 3.2 and 3.3.

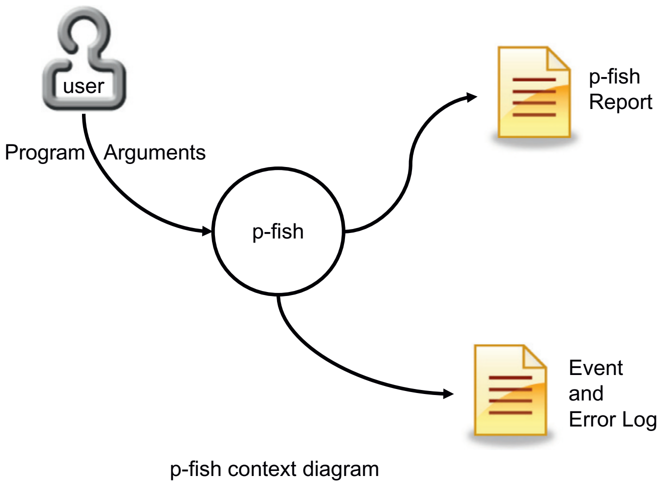


FIGURE 3.2

Context diagram: Python-file system hashing (p-fish).

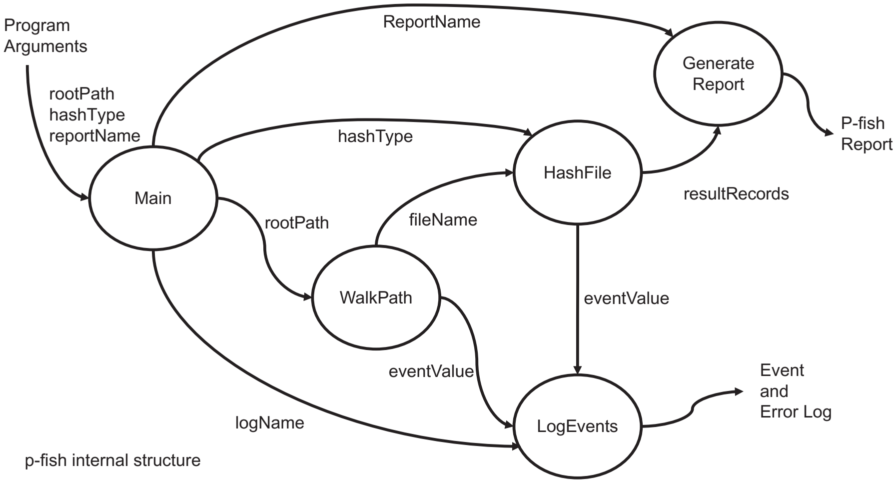


FIGURE 3.3

p-fish internal structure.

The context diagram is very straightforward and simply depicts the major inputs and outputs of the proposed program. A user specifies the program arguments, p-fish takes those inputs and processes (hashes, extracts metadata, etc.) the file system produces a report and any notable events or errors to the "p-fish report" and the "p-fish event and error log" files respectively.

Turning to the internal structure I have broken the program down into five major components. The `Main` program, `ParseCommandLine` function, `WalkPath` function, `HashFile` functions, `CSVWriter` class and logger (note logger is actually the Python logger module), that is utilized by the major functions of pfish. I briefly describe the operation of each below and during the code walk through section a more detailed line by line explanation of how each function operates is provided.

Main function

The purpose of the Main function is to control the overall flow of this program. For example, within Main I set up the Python logger, I display startup and completion messages, and keep track of the time. In addition, Main invokes the command line parser and then launches the `WalkPath` function. Once `WalkPath` completes Main will log the completion and display termination messages to the user and the log.

ParseCommandLine

In order to provide smooth operation of p-fish, I leverage parseCommandLine to not only parse but also validate the user input. Once completed, information that is germane to program functions such as `WalkPath`, `HashFile`, and `CSVWrite` is available from parser-generated values. For example, since the `hashType` is specified by the user, this value must be available to `HashFile`. Likewise the `CSVWriter` needs the path where the resulting pfish report will be written, and `WalkPath` requires the starting or `rootPath` to start the walk.

WalkPath function

The `WalkPath` function must start at the root of the directory tree or path and traverse every directory and file. For each valid file encountered it will call the `HashFile` function to perform the one-way hashing operations. Once all the files have been processed `WalkPath` will return control back to Main with the number of files successfully processed.

HashFile function

The `HashFile` function will open, read, hash, and obtain metadata regarding the file in question. For each file, a row of data will be sent to the `CSVWriter` to be included in the p-fish report. Once the file has been processed, `HashFile` will return control back to `WalkPath` in order to fetch the next file.

CSVWriter (class)

In order to provide an introduction to class and object usage I decided to create `CSVWriter` as a class instead of a simple function. You will see more of this in upcoming cookbook chapters but `CSVWriter` sets up nicely for a class/object demonstration. The csv module within the Python Standard Library requires that the "writer" be initialized. For example, I want the resulting csv file to have a header row made up of a static set of columns. Then subsequent calls to writer will contain data that fills in each row. Finally, once the program has processed all the files the resulting csv report must be closed. Note that as I walk through the program code you may wonder why I did not leverage classes and objects more for this program. I certainly could have, but felt for the first application I would create a more function-oriented example.

Logger

The built-in Standard Library logger provides us with the ability to write messages to a log file associated with p-fish. The program can write information messages, warning messages, and error messages. Since this is intended to be a forensic application, logging operations of the program is vital. You can expand the program to log additional events in the code, they can be added to any of the _pfish functions.

Writing the code

I decided to create two files, mainly to show you how to create your own Python module and also to give you some background on how to separate capabilities. For this first simple application, I created (1) pfish.py and (2) _pfish.py. As you may recall, all modules that are created begin with an underscore and since _pfish.py contains all the support functions for pfish I simply named it _pfish.py. If you would like to split out the modules to better separate the functions you could create separate modules for the `HashFile` function, the `WalkPath` function, etc. This is a decision that is typically based on how tightly or loosely coupled the functions are, or better stated, whether you wish to reuse individual functions later that need to standalone. If that is the case, then you should separate them out.

In Figure 3.4 you can see my IDE setup for the project pfish. You notice the project section to the far upper right that specifies the files associated with the project. I also have both files open—you can see the two tabs far left about half way down

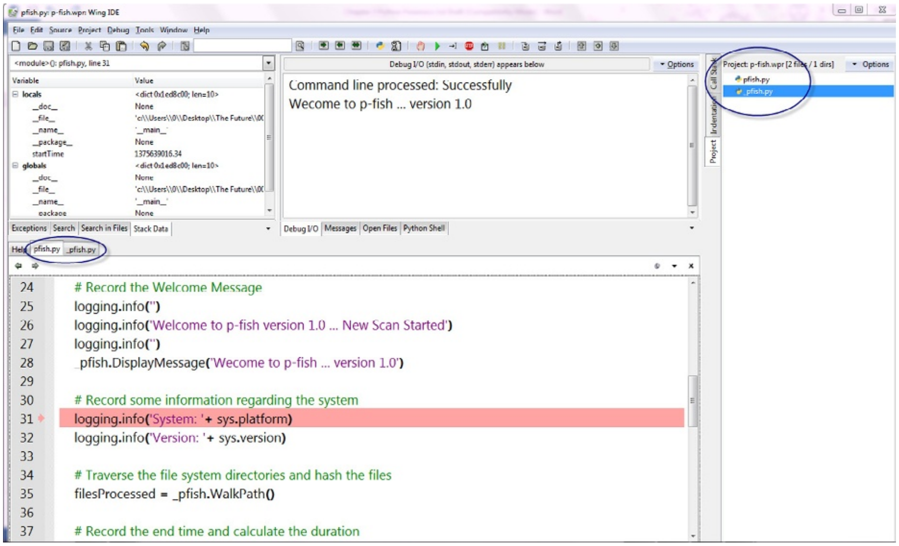


FIGURE 3.4

p-fish WingIDE setup.

where I can view the source code in each of the files. As you would expect in the upper left quadrant, you can see the program is running and the variables are available for inspection. Finally, in the upper center portion of the screen you can see the current display messages from the program reporting that the command line was processed successfully and the welcome message for pfish.

CODE WALK-THROUGH

I will be inserting dialog as I discuss each code section. The code walk-through will give you an in-depth look at all the code associated with the program. I will be first walking through each of the key functions and then will provide you with a complete listing of both files.

Examining main—code walk-through

The embedded commentary is represented in italics, while the code itself is represented in a fixed-sized font.

```
# p-fish : Python File System Hash Program
# Author: C. Hosmer
# July 2013
# Version 1.0
#
```

The main program code is quite straightforward. At the top as you would expect, you see the import statements that make the Python Standard Library modules available for our use. What you have not seen before is the import statement referencing our own module in this case _pfish. Since I will be calling functions from Main that exist in our module, the module must import our own module _pfish.

```
import logging    # Python Library logging functions
import time       # Python Library time manipulation functions
import sys        # Python Library system specific parameters
import _pfish     # _pfish Support Function Module

if __name__ == '__main__':
    PFISH_VERSION = '1.0'
    # Turn on Logging
```

Next, you can see my initialization of the Python logging system. In this example I have hard-wired the log to be stored in the file aptly named pFishLog.log. I set the logging level to DEGUG and specified that I wanted the Time and Date recorded for each log event. By setting the level to DEBUG (which is the lowest level) this will ensure all messages sent to the logger will be visible.

```
    logging.basicConfig(filename='pFishLog.log',level=logging.
    DEBUG,format='%(asctime)s %(message)s')
```

Next, I pass control to process the command line arguments by calling the _pfish. `ParseCommandLine`*() function. I must prefix the function with _pfish, since the function exists in the _pfish module. If the parse is successful, the function will return here, if not, it will post a message to the user and exit the program. I will take a deeper look at the operation of* `ParseCommandLine`*() in the next section.*

```
    # Process the Command Line Arguments
    _pfish.ParseCommandLine()
```

I need to record the current starting time of the application in order to calculate elapsed time for processing. I use the Standard Library function `time.time`*() to acquire the time elapsed in seconds since the epoch. Note, forensically this is the time of the system we are running on, therefore if the time is a critical element in your investigation you should sync your system clock accordingly.*

```
    # Record the Starting Time
    startTime = time.time()
```

Next the program posts a message to the log reporting the start of the scan and display this on the user screen only if the verbose option was selected on the command line (more about this when I examine the `ParseCommandLine` *function). Notice that I used a CONSTANT to hold the version number instead of just embedding a magic number. Now we can just modify the CONSTANT in the future. Then anywhere* `PFISH_VERSION` *is used it will display the proper version number. I also logged the system platform and version in case there is a question in the future* *about the system that was used to process these data. This would be a great place to add information about the organization, investigator name, case number, and other information that is relevant to the case.*

```
# Post the Start Scan Message to the Log
logging.info('Welcome to p-fish version 1.0 ... New Scan Started')
_pfish.DisplayMessage('Welcome to p-fish ... version 1.0')
```

Note, since I created a constant PFISH_VERSION, we could use that to make the source code easier to maintain. That would look something like:

_pfish.DisplayMessage('Welcome to p-fish ... '+ PFISH_VERSION)

```
# Record some information regarding the system
logging.info('System: '+ sys.platform)
logging.info('Version: '+ sys.version)
```

Now the main program launches the `WalkPath` *function within the _pfish module that will traverse the directory structure starting at the predefined root path. This function returns the number of files that were successfully processed by* `WalkPath` *and* `HashFile`. *As you can see I use this value along with the ending time to finalize the log entries. By subtracting the* `startTime` *from the* `endTime` *I can determine the number of seconds it took to perform the file system hashing operations. You could convert the seconds into days, hours, minutes, and seconds of course.*

```
# Traverse the file system directories and hash the files
filesProcessed = _pfish.WalkPath()

# Record the end time and calculate the duration
endTime = time.time()
duration = endTime - startTime

logging.info('Files Processed: ' + str(filesProcessed) )
logging.info('Elapsed Time: ' + str(duration) + ' seconds')

logging.info('Program Terminated Normally')

_pfish.DisplayMessage("Program End)"
```

ParseCommandLine()

In the design section I made a couple of decisions that drove the development:

1. I decided that this first application would be a command line program.
2. I decided to provide several options to the user to manipulate the behavior of the program. This has driven the design and implementation of the command line options.

Based on this I provided the following command line options to the program.

Option	Description	Notes
-v	Verbose, if this option is specified then any calls to the `DisplayMessage`() function will be displayed to the standard output device, otherwise the program will run silently	
--MD5 --SHA256 --SHA512	Hash type selection, the user must specify the one-way hashing algorithm that would be utilized	The selection is mutually exclusive and at least one must be selected or the program will abort
-d	`rootPath`, this allows the user to specify the starting or root path for the walk	The directory must *exist* and must be *readable* or the program will abort
-r	`reportPath`, this allows the user to specify the directory where the resulting .csv file will be written	The directory must *exist* and must be *writable* or the program will abort

Even though at first some of these requirements might seem difficult, the `argparse` Standard Library provides great flexibility in handling them. This allows us to catch any possible user errors prior to program execution and also provides us with a way to report problems to the user to handle the exceptions.

```
def ParseCommandLine():
```

The majority of the process of using `argparse` *is knowing how to setup the parser. If you set the parser up correctly it will do all the hard work for you. I start by creating a new parser named "parser" and simply give it a description. Next, I add a new argument in this case –v or verbose. The option is –v and the resulting variable that is used is* `verbose`*. The help message associated with the argument is used by the help system to inform the user on how to use pfish. The –h option is built-in and requires no definition.*

```
parser = argparse.ArgumentParser('Python file system hashing ..
p-fish')

parser.add_argument('-v', '--verbose' help='allows progress messages to
be displayed', action='store_true')
```

*The next section defines a mutually exclusive group of arguments for selecting the specific hash type the user would like to generate. If you wanted to add in another option, for example sha384, you would simply add another argument to the group and follow the same format. Since I specified under the add_mutually_exclusive_group the option **required=True,*** `argparse` *will make sure that the user has only specified one argument and at least one.*

```
# setup a group where the selection is mutually exclusive ## and required.

group = parser.add_mutually_exclusive_group(required=True)

group.add_argument('--md5',help = 'specifies MD5 algorithm',
action='store_true')
```

```
group.add_argument('--sha256',   help = 'specifies SHA256
algorithm', action='store_true')

group.add_argument('--sha512',   help = 'specifies SHA512
algorithm', action='store_true')
```

Next I need to specify the starting point of our walk, and where the report should be created. This works the same as the previous setup, except I have added the ***type*** *option. This requires* `argparse` *to validate the type I have specified. In the case of the –d option, I want to make sure that the* `rootPath` *exists and is readable. For the* `reportPath`*, it must exist and be writable. Since* `argparse` *does not have built-in functions to validate a directory, I created the functions* `ValidateDirectory()` *and* `ValidateDirectoryWritable()`*. They are almost identical and they use Standard Library operating system functions to validate the directories as defined.*

```
parser.add_argument('-d', '--rootPath', type=
ValidateDirectory, required=True, help="specify the root
path for hashing)"

parser.add_argument('-r', '--reportPath', type=
ValidateDirectoryWritable, required=True, help="specify the
path for reports and logs will be written)"

# create a global object to hold the validated arguments,
# # these will be available then to all the Functions
within # the _pfish.py module

global gl_args
global gl_hashType
```

Now the parser can be invoked. I want to store the resulting arguments (once validated) in a global variable so they can be accessible by the functions within the _pfish module. This would be a great opportunity to create a class to handle this which would avoid the use of the global variables. This is done in Chapter 4.

```
gl_args = parser.parse_args()
```

If the parser was successful (in other words `argparse` *validated the command line parameters), I want to determine which hashing algorithm the user selected. I do that by examining each value associated with the hash types. If the user selected sha256 for example, the* `gl_args.sha256` *would be True and md5 and sha512 would be false. Therefore, by using a simple if/elif language routine I can determine which was selected.*

```
if gl_args.md5:
    gl_hashType = 'MD5'
elif gl_args.sha256:
    gl_hashType = 'SHA256'
elif gl_args.sha512:
    gl_hashType = 'SHA512'
else:
    gl_hashType = "Unknown"
```

```
        logging.error('Unknown Hash Type Specified')

    DisplayMessage("Command line processed: Successfully)"
  return
```

ValiditingDirectoryWritable

As mentioned above, I needed to create functions to validate the directories provided by the users for both the report and starting or root path of the Walk. I accomplish this by leveraging the Python Standard Library module `os`. I leverage both the `os.path.isdir` and `os.access` methods associated with this module.

```
def ValidateDirectoryWritable(theDir):
```

I first check to see if in fact the directory string that the user provided exists. If the test fails then I raise an error within `argparse` *and provide the message "Directory does not exist." This message would be provided to the user if the test fails.*

```
 # Validate the path is a directory
    if not os.path.isdir(theDir):
            raise argparse.ArgumentTypeError('Directory does not
            exist')
```

Next I validate that write privilege is authorized to the directory and once again if the test fails I raise an exception and provide a message.

```
 # Validate the path is writable
    if os.access(theDir, os.W_OK):
          return theDir
    else:
          raise argparse.ArgumentTypeError('Directory is not writable')
```

Now that I have completed the implementation of the `ParseCommandLine` function, let us examine a few examples of how the function rejects improper command line arguments. In Figure 3.5, I created four improperly formed command lines:

(1) I mistyped the root directory as TEST_DIR instead of simply TESTDIR
(2) I mistyped the –sha512 parameter as –sha521
(3) I specified two hash types –sha512 and –md5
(4) Finally, I did not specify any hash type

As you can see in each case, ParseCommandLine rejected the command.

In order to get the user back on track they simply have to utilize the –h or help option as shown in Figure 3.6 to obtain the proper command line argument instructions.

WalkPath

Now let us walk through the `WalkPath` function that will traverse the directory structure, and for each file will call the `HashFile` function. I think you will be pleasantly surprised how simple this is.

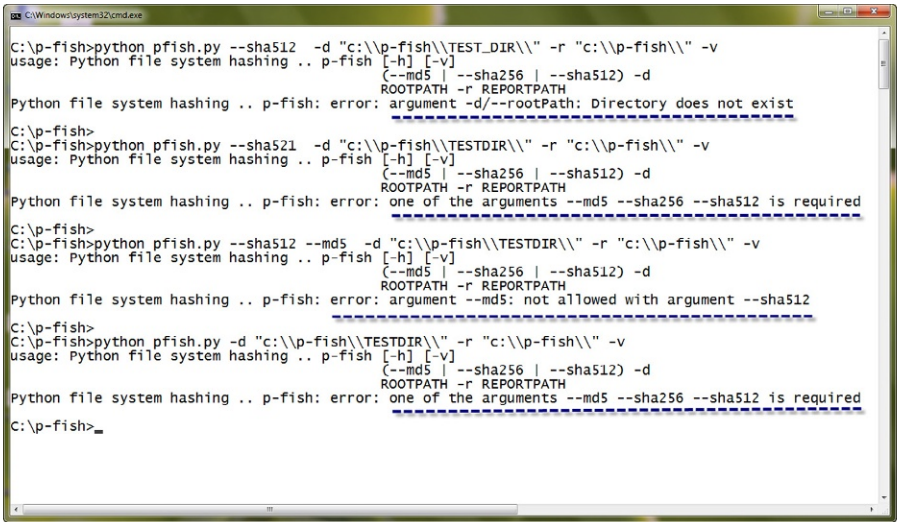


FIGURE 3.5

Demonstration of ParseCommandLine.

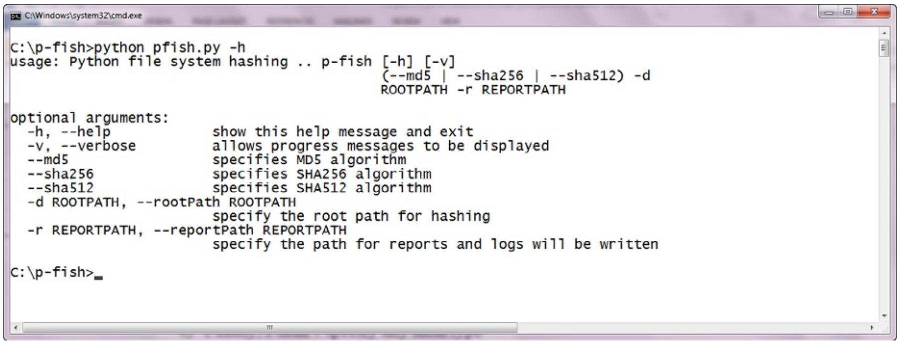


FIGURE 3.6

pfish –h command.

```
def WalkPath():
```

I first initialize the variable `processCount` *in order to count the number of successfully processed files and I post a message to the log file to document the root path value.*

```
processCount = 0
errorCount = 0

log.info('Root Path: ' + gl_args.rootPath)
```

Next I initialize the `CSVWriter` *with the* `reportPath` *provided on the command line by the user. I also provide the* `hashType` *selected by the user so it can be included in the Header line of the CSV file. I will cover the* `CSVWriter` *class later in this chapter.*

```
oCVS = _CSVWriter(gl_args.reportPath+'fileSystemReport.csv',
gl_hashType)

 # Create a loop that process all the files starting
 # at the rootPath, all sub-directories will also be
 # processed
```

Next I create a loop using the `os.walk` *method and the* `rootpath` *specified by the user. This will create a list of file names that is processed in the next loop. This is done for each directory found within the path.*

```
 for root, dirs, files in os.walk(gl_args.rootPath):

          # for each file obtain the filename and call the
          # HashFile Function
```

The next loop processes each file in the list of files and calls the function `HashFile` *with the file name joined with the path, along with the simple file name for use by* `HashFile`. *The call also passes* `HashFile` *with access to the CVS writer so that the results of the hashing operations can be written to the CVS file.*

```
 for file in files:
     fname = os.path.join(root, file)
     result = HashFile(fname, file, oCVS)

     # if successful then increment ProcessCount
```

The process and error counts are incremented accordingly

```
      if result is True:
          processCount += 1
      # if not successful, the increment the ErrorCount
      else:
           errorCount += 1
```

Once all the directories and files have been processed the `CVSWriter` *is closed and the function returns to the main program with the number of successfully processed files.*

```
oCVS.writerClose()
return(processCount)
```

HashFile

Below is the code for the `HashFile` function, it is clearly the longest for this program, but also quite simple and straightforward. Let us walk through the process.

```
def HashFile(theFile, simpleName, o_result):
```

For each file several items require validation before we attempt to hash the file.
(1) Does the path exist
(2) Is the path a link instead of an actual file
(3) Is the file real (making sure it is not orphaned)
For each of these tests there is a corresponding log error that is posted to the log file if failure occurs. If the file is bypassed the program will simply return to `WalkFile` *and process the next file.*

```
# Verify that the path is valid
if os.path.exists(theFile):
    #Verify that the path is not a symbolic link
    if not os.path.islink(theFile):
        #Verify that the file is real
        if os.path.isfile(theFile):
```

The next part is a little tricky. Even through our best efforts to determine the existence of the file, there may be cases where the file cannot be opened or read. This could be caused by permission issues, the file is locked or possibly corrupted. Therefore, I utilize the try methods while attempting to open and then read from the files. Note that I'm careful to open the file as read-only the "`rb`*" option. Once again if an error occurs a report is generated and logged and the program moves on to the next file.*

```
try:
    #Attempt to open the file
    f = open(theFile, 'rb')
except IOError:
    #if open fails report the error
    log.warning('Open Failed: ' + theFile)
    return
else:
    try:
        # Attempt to read the file
        rd = f.read()
    except IOError:
        # if read fails, then close the file and
        # report error
        f.close()
        log.warning('Read Failed: ' + theFile)
        return
    else:
        #success the file is open and we can
        #read from it
        #lets query the file stats
```

Once the file has been successfully opened and verified that reading from the file is allowed, I extract the attributes associated with the file. These include owner, group, size, MAC times, and mode. I will include these in the record that is posted to the CSV file.

```
theFileStats = os.stat(theFile)
     (mode, ino, dev, nlink, uid, gid, size,
     atime, mtime, ctime) = os.stat(theFile)

# Display progress to the user

DisplayMessage("Processing File: " + theFile)

# convert the file size to a string
fileSize = str(size)

# convert the MAC Times to strings

modifiedTime = time.ctime(mtime)
accessTime = time.ctime(atime)
createdTime = time.ctime(ctime)

# convert the owner, group and file mode

ownerID = str(uid)
groupID = str(gid)
fileMode = bin(mode)
```

Now that the file attributes have been collected the actual hashing of the file occurs. I need to hash the file as specified by the user (i.e., which one-way hashing algorithm should be utilized). I'm using the Python Standard Library module `hashlib` *as we experimented with in Chapter 2.*

```
#process the file hashes

if gl_args.md5:
    #Calcuation the MD5
    hash = hashlib.md5()
    hash.update(rd)
    hexMD5 = hash.hexdigest()
    hashValue = hexMD5.upper()
elif gl_args.sha256:
    #Calculate the SHA256
    hash=hashlib.sha256()
    hash.update(rd)
    hexSHA256 = hash.hexdigest()
    hashValue = hexSHA256.upper()
elif gl_args.sha512:
    #Calculate the SHA512
    hash=hashlib.sha512()
    hash.update(rd)
    hexSHA512 = hash.hexdigest()
    hashValue = hexSHA512.upper()
else:
    log.error('Hash not Selected')
#File processing completed
#Close the Active File
```

Now that processing of the file is complete the file must be closed. Next I use the CSV class to write out the record to the report file and return successfully to the caller in this case `WalkPath`.

```
                f.close()

                    # write one row to the output file

                    o_result.writeCSVRow(simpleName,
                    theFile, fileSize, modifiedTime,
                    accessTime, createdTime, hashValue,
                    ownerID, groupID, mode)
                    return True
```

This section posts the warning messages to the log file relating to problems encountered processing the file.

```
        else:
            log.warning('[' + repr(simpleName) + ', Skipped NOT a File' + ']')
            return False
        else:
            log.warning('[' + repr(simpleName) + ', Skipped Link
            NOT a File' + ']')
            return False
    else:
            log.warning('[' + repr(simpleName) + ', Path does NOT exist' + ']')
        return False
```

CSVWriter

The final code walk-through section I will cover in this chapter is the `CSVWriter`. As I mentioned earlier, I created this code as a class instead of a function to make this more useful and to introduce you to the concept of classes in Python. The class only has three methods, the constructor or `init`, `writeCSVRow`, and `writerClose`. Let us examine each one.

```
class _CSVWriter:
```

The constructor or init method accomplishes three basic initializations:
(1) Opens the output csvFile
(2) Initializes the csv.writer
(3) Writes the header row with the names of each column
If any failure occurs during the initialization an exception is thrown and a log entry is generated.

```
def __init__(self, fileName, hashType):
    try:
        # create a writer object and write the header row
        self.csvFile = open(fileName, 'wb')
```

```
        self.writer = csv.writer(self.csvFile,
          delimiter=',', quoting=csv.QUOTE_ALL)
        self.writer.writerow( ('File', 'Path', 'Size',
          'Modified Time', 'Access Time', 'Created Time',
       hashType, 'Owner', 'Group', 'Mode') )
    except:
          log.error('CSV File Failure')
```

The second method `writeCSVRow` *receives a record from* `HashFile` *upon successful completion of each file hash. The method then uses the csv writer to actually place the record in the report file.*

```
def writeCSVRow(self, fileName, filePath, fileSize, mTime,
       aTime, cTime, hashVal, own, grp, mod):

              self.writer.writerow( (fileName, filePath,
                  fileSize, mTime, aTime, cTime, hashVal, own,
                  grp, mod))
```

Finally, the writeClose method, as you expect, simply closes the csvFile.

```
def writerClose(self):
             self.csvFile.close()
```

Full code listing pfish.py

```
#
# p-fish : Python File System Hash Program
# Author: C. Hosmer
# July 2013
# Version 1.0
#
import logging          # Python Standard Library Logger
import time             # Python Standard Library time functions
import sys              # Python Library system specific parameters
import _pfish           # _pfish Support Function Module

if __name__ == '__main__':

    PFISH_VERSION = '1.0'

    # Turn on Logging
    logging.basicConfig(filename='pFishLog.log',level=logging.DEBUG,
    format='%(asctime)s %(message)s')

    # Process the Command Line Arguments
    _pfish.ParseCommandLine()

    # Record the Starting Time
```

```
startTime = time.time()

# Record the Welcome Message
logging.info('')
logging.info('Welcome to p-fish version '+ PFISH_VERSION +' ... New Scan
Started')
logging.info('')
_pfish.DisplayMessage('Welcome to p-fish ... version '+
PFISH_VERSION)

# Record some information regarding the system
logging.info('System: '+ sys.platform)
logging.info('Version: '+ sys.version)

# Traverse the file system directories and hash the files
filesProcessed = _pfish.WalkPath()

# Record the end time and calculate the duration
endTime = time.time()
duration = endTime - startTime
logging.info('Files Processed: ' + str(filesProcessed) )
logging.info('Elapsed Time: ' + str(duration) + ' seconds')
logging.info('')
logging.info('Program Terminated Normally')
logging.info('')

_pfish.DisplayMessage("Program End")
```

Full code listing _pfish.py

```
#
# pfish support functions, where all the real work gets done
#

# Display Message()       ParseCommandLine()       WalkPath()
# HashFile()              class _CVSWriter
# ValidateDirectory()     ValidateDirectoryWritable()
#

import os                 #Python Standard Library - Miscellaneous
                          operating system interfaces
import stat               #Python Standard Library - functions for
                          interpreting os results
import time               #Python Standard Library - Time access and
                          conversions functions
import hashlib            #Python Standard Library - Secure hashes and
                          message digests
import argparse           #Python Standard Library - Parser for command-
                          line options, arguments
import csv                #Python Standard Library - reader and writer for
                          csv files
```

```
import logging           #Python Standard Library – logging facility

log = logging.getLogger('main._pfish')

#
# Name: ParseCommand() Function
#
# Desc: Process and Validate the command line arguments
#           use Python Standard Library module argparse
#
# Input: none
#
# Actions:
#            Uses the standard library argparse to process the
             command line
#            establishes a global variable gl_args where any of the
             functions can
#            obtain argument information
#
def ParseCommandLine():

    parser = argparse.ArgumentParser('Python file system hashing ..
    p-fish')

    parser.add_argument('-v', '—verbose', help='allows progress messages
    to be displayed', action='store_true')

    # setup a group where the selection is mutually exclusive and
    required.
    group = parser.add_mutually_exclusive_group(required=True)
    group.add_argument('--md5',   help = 'specifies MD5 algorithm',
    action='store_true')
    group.add_argument('--sha256',   help = 'specifies SHA256
    algorithm', action='store_true')
    group.add_argument('--sha512',   help = 'specifies SHA512
    algorithm', action='store_true')

    parser.add_argument('-d', '--rootPath', type=
    ValidateDirectory, required=True, help="specify the root
    path for hashing")
    parser.add_argument('-r', '--reportPath', type=
    ValidateDirectoryWritable, required=True, help="specify the
    path for reports and logs will be written")

    # create a global object to hold the validated arguments, these will be
    available then
    # to all the Functions within the _pfish.py module

    global gl_args
    global gl_hashType

    gl_args = parser.parse_args()
```

```
    if gl_args.md5:
        gl_hashType = 'MD5'
    elif gl_args.sha256:
        gl_hashType = 'SHA256'
    elif gl_args.sha512:
        gl_hashType = 'SHA512'
    else:
        gl_hashType = "Unknown"
        logging.error('Unknown Hash Type Specified')

    DisplayMessage("Command line processed: Successfully")

    return

# End ParseCommandLine====================================

#
# Name: WalkPath() Function
#
# Desc: Walk the path specified on the command line
#          use Python Standard Library module os and sys
#
# Input: none, uses command line arguments
#
# Actions:
#             Uses the standard library modules os and sys
#             to traverse the directory structure starting a root
#             path specified by the user. For each file discovered,
              WalkPath
#             will call the Function HashFile() to perform the file
              hashing
#

def WalkPath():

    processCount = 0
    errorCount = 0

    oCVS = _CSVWriter(gl_args.reportPath+'fileSystemReport.csv',
    gl_hashType)

    # Create a loop that process all the files starting
    # at the rootPath, all sub-directories will also be
    # processed

    log.info('Root Path: ' + gl_args.rootPath)

    for root, dirs, files in os.walk(gl_args.rootPath):

        # for each file obtain the filename and call the HashFile Function
        for file in files:
            fname = os.path.join(root, file)
```

```
            result = HashFile(fname, file, oCVS)

            # if hashing was successful then increment the ProcessCount
            if result is True:
                processCount += 1
            # if not successful, the increment the ErrorCount
            else:
                ErrorCount += 1

        oCVS.writerClose()

        return(processCount)

#End WalkPath==========================================

#
# Name: HashFile Function
#
# Desc: Processes a single file which includes performing a hash of the
        file
#           and the extraction of metadata regarding the file processed
#           use Python Standard Library modules hashlib, os, and sys
#
# Input: theFile = the full path of the file
#           simpleName = just the filename itself
#
# Actions:
#           Attempts to hash the file and extract metadata
#           Call GenerateReport for successful hashed files
#
def HashFile(theFile, simpleName, o_result):

    # Verify that the path is valid
    if os.path.exists(theFile):

        #Verify that the path is not a symbolic link
        if not os.path.islink(theFile):

            #Verify that the file is real
            if os.path.isfile(theFile):

                try:
                    #Attempt to open the file
                    f = open(theFile, 'rb')
                except IOError:
                    #if open fails report the error
                    log.warning('Open Failed: '+ theFile)
                    return
                else:
                    try:
                        # Attempt to read the file
```

```
            rd = f.read()
        except IOError:
            # if read fails, then close the file and
            report error
            f.close()
            log.warning('Read Failed: ' + theFile)
            return
    else:
            #success the file is open and we can read from it
            #lets query the file stats

            theFileStats = os.stat(theFile)
            (mode, ino, dev, nlink, uid, gid, size, atime,
            mtime, ctime) = os.stat(theFile)

            #Print the simple file name
            DisplayMessage("Processing File: " + theFile)

            # print the size of the file in Bytes
            fileSize = str(size)

            #print MAC Times
            modifiedTime = time.ctime(mtime)
            accessTime = time.ctime(atime)
            createdTime = time.ctime(ctime)

            ownerID = str(uid)
            groupID = str(gid)
            fileMode = bin(mode)

            #process the file hashes

            if gl_args.md5:
                #Calcuation and Print the MD5
                hash = hashlib.md5()
                hash.update(rd)
                hexMD5 = hash.hexdigest()
                hashValue = hexMD5.upper()
            elif gl_args.sha256:
                hash=hashlib.sha256()
                hash.update(rd)
                hexSHA256 = hash.hexdigest()
                hashValue = hexSHA256.upper()
            elif gl_args.sha512:
                #Calculate and Print the SHA512
                hash=hashlib.sha512()
                hash.update(rd)
                hexSHA512 = hash.hexdigest()
                hashValue = hexSHA512.upper()
            else:
                log.error('Hash not Selected')
```

```
                    #File processing completed
                    #Close the Active File
                    print "============================="
                    f.close()

                    # write one row to the output file

                    o_result.writeCSVRow(simpleName, theFile,
                    fileSize, modifiedTime, accessTime, createdTime,
                    hashValue, ownerID, groupID, mode)
                    return True
            else:
                log.warning('[' + repr(simpleName) + ', Skipped NOT a
                File' + ']')
                return False
        else:
                log.warning('[' + repr(simpleName) + ', Skipped Link NOT a
                File' + ']')
          return False
else:
                log.warning('[' + repr(simpleName) + ', Path does NOT
                exist' + ']')
return False

# End HashFile Function ===================================

#
# Name: ValidateDirectory Function
#
# Desc: Function that will validate a directory path as
#         existing and readable. Used for argument validation only
#
# Input: a directory path string
#
# Actions:
#            if valid will return the Directory String
#
#            if invalid it will raise an ArgumentTypeError within
             argparse
#            which will in turn be reported by argparse to the user
#
def ValidateDirectory(theDir):

    # Validate the path is a directory
    if not os.path.isdir(theDir):
        raise argparse.ArgumentTypeError('Directory does not exist')

    # Validate the path is readable
    if os.access(theDir, os.R_OK):
        return theDir
   else:
```

```
        raise argparse.ArgumentTypeError('Directory is not readable')

#End ValidateDirectory ====================================

#
# Name: ValidateDirectoryWritable Function
#
# Desc: Function that will validate a directory path as
#           existing and writable. Used for argument validation only
#
# Input: a directory path string
#
# Actions:
#              if valid will return the Directory String
#
#              if invalid it will raise an ArgumentTypeError within
#              argparse
#              which will in turn be reported by argparse to the user
#

def ValidateDirectoryWritable(theDir):

    # Validate the path is a directory
    if not os.path.isdir(theDir):
        raise argparse.ArgumentTypeError('Directory does not exist')

    # Validate the path is writable
    if os.access(theDir, os.W_OK):
        return theDir
    else:
        raise argparse.ArgumentTypeError('Directory is not writable')

#End ValidateDirectoryWritable ==============================

#==================================================
#
# Name: DisplayMessage() Function
#
# Desc: Displays the message if the verbose command line option is present
#
# Input: message type string
#
# Actions:
#              Uses the standard library print function to display the
#              message
#
def DisplayMessage(msg):

    if gl_args.verbose:
        print(msg)
```

```
    return

#End DisplayMessage=========================================

#
# Class: _CSVWriter
#
# Desc: Handles all methods related to comma separated value operations
#
# Methods constructor:     Initializes the CSV File
#              writeCVSRow:   Writes a single row to the csv file
#              writerClose:     Closes the CSV File

class _CSVWriter:

    def __init__(self, fileName, hashType):
        try:
            # create a writer object and then write the header row
            self.csvFile = open(fileName, 'wb')
            self.writer = csv.writer(self.csvFile, delimiter=',',
            quoting=csv.QUOTE_ALL)
            self.writer.writerow( ('File', 'Path', 'Size', 'Modified Time',
            'Access Time', 'Created Time', hashType, 'Owner', 'Group', 'Mode') )
        except:
            log.error('CSV File Failure')

    def writeCSVRow(self, fileName, filePath, fileSize, mTime, aTime,
    cTime, hashVal, own, grp, mod):

        self.writer.writerow( (fileName, filePath, fileSize, mTime,
        aTime, cTime, hashVal, own, grp, mod))

    def writerClose(self):
        self.csvFile.close()
```

RESULTS PRESENTATION

Now that the walk-through has been completed and I have gone through a deep dive into the code, let us take a look at the results. In Figure 3.7, I executed the program with the following options:

C:\p-fish>Python pfish.py --md5 -d "c:\\p-fish\\TESTDIR\\" -r "c:\\p-fish\\" –v

The –v or verbose option was selected and the program displayed information regarding every file processed was selected as expected.

In Figure 3.8, I examine the c:\p-fish directory and discover that two files were created there, which are the two resulting files for the pfish.py.

1. fileSystemReport.csv
2. pFishLog.log

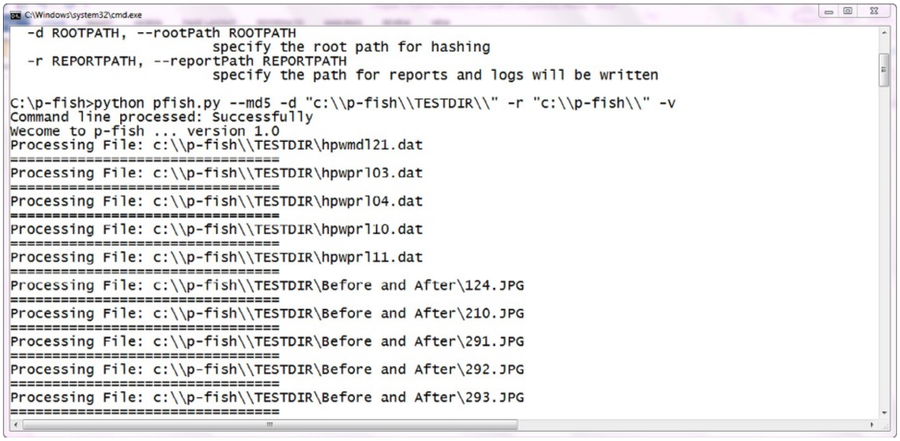


FIGURE 3.7

Test run of pfish.py.

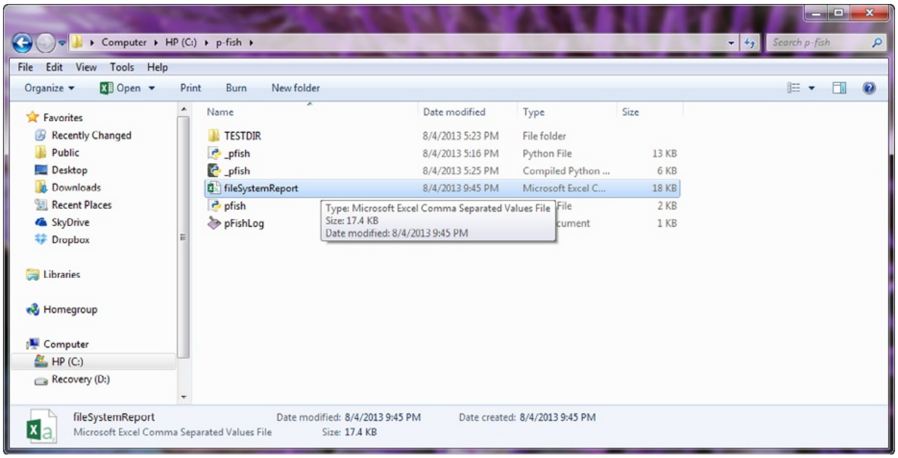
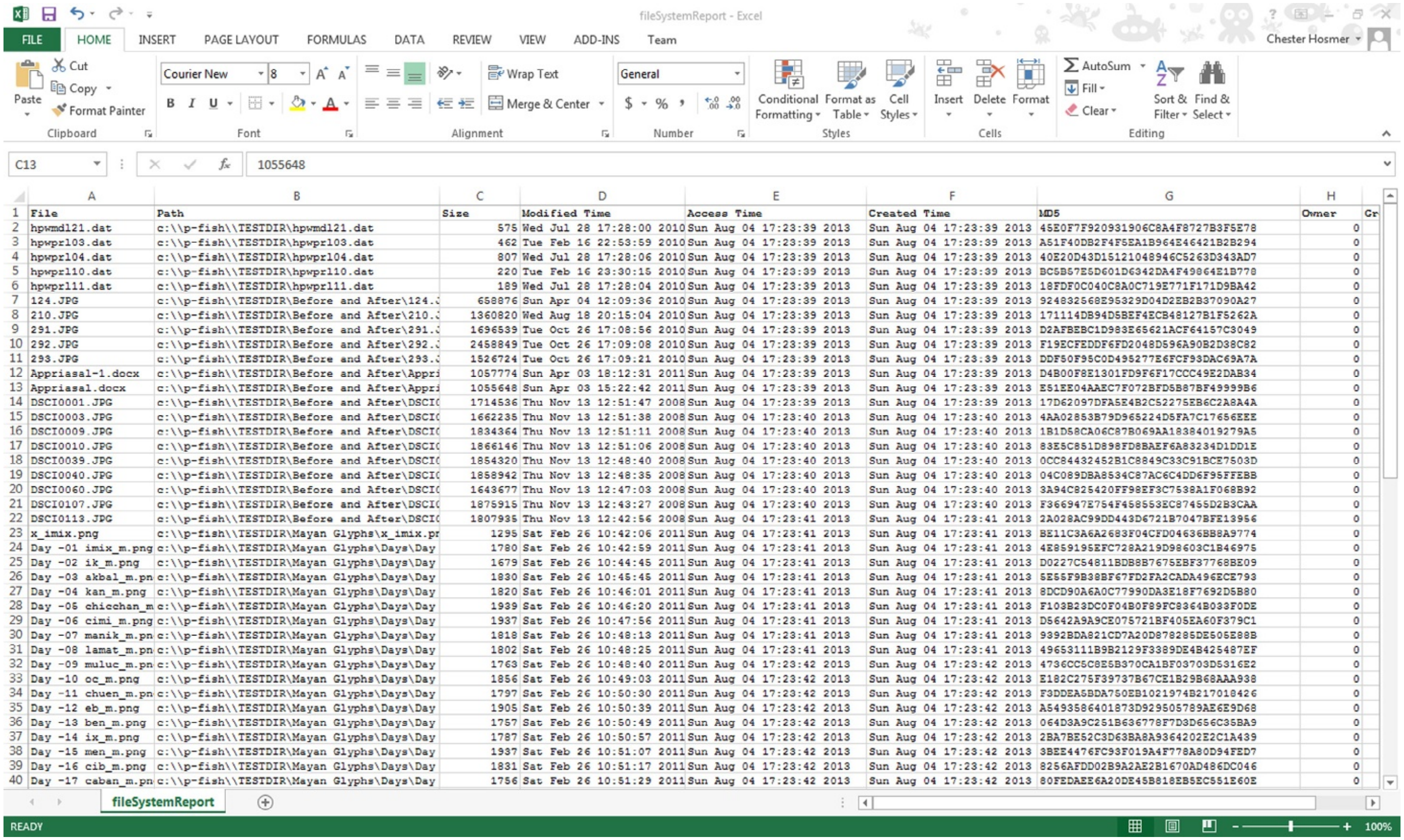


FIGURE 3.8

Result directory after pfish execution.

By choosing to leverage the Python `csv` module to create the report file Windows already recognizes it as a file that Microsoft Excel can view. Opening the file we see the results in Figure 3.9, a nicely formatted column report that can now manipulate with Excel (sort columns, search for specific values, arrange in date order, and examine each of the results). You notice that the hash value is in a column named MD5 that is labeled as such because I passed the appropriate heading value during the initialization of the `csv`.



	File	Path	Size	Modified Time	Access Time	Created Time	MD5	Owner	Gr
2	hpwmdl21.dat	c:\\p-fish\\TESTDIR\hpwmdl21.dat	575	Wed Jul 28 17:28:00 2010	Sun Aug 04 17:23:39 2013	Sun Aug 04 17:23:39 2013	45E0F7F920931906C8A4F8727B3F5E78	0	
3	hpwprl03.dat	c:\\p-fish\\TESTDIR\hpwprl03.dat	462	Tue Feb 16 22:53:59 2010	Sun Aug 04 17:23:39 2013	Sun Aug 04 17:23:39 2013	A51F40DB2F4F5EA1B964E46421B2B294	0	
4	hpwprl04.dat	c:\\p-fish\\TESTDIR\hpwprl04.dat	807	Wed Jul 28 17:28:06 2010	Sun Aug 04 17:23:39 2013	Sun Aug 04 17:23:39 2013	40E20D43D15121048946C5263D343AD7	0	
5	hpwprl10.dat	c:\\p-fish\\TESTDIR\hpwprl10.dat	220	Tue Feb 16 23:30:15 2010	Sun Aug 04 17:23:39 2013	Sun Aug 04 17:23:39 2013	BC5B57E5D601D6342DA4F49864E1B778	0	
6	hpwprl11.dat	c:\\p-fish\\TESTDIR\hpwprl11.dat	189	Wed Jul 28 17:28:04 2010	Sun Aug 04 17:23:39 2013	Sun Aug 04 17:23:39 2013	18FDF0C040C8A0C719E771F171D9BA42	0	
7	124.JPG	c:\\p-fish\\TESTDIR\Before and After\124.J	658876	Sun Apr 04 12:09:36 2010	Sun Aug 04 17:23:39 2013	Sun Aug 04 17:23:39 2013	924832568E95329D04D2EB2B37090A27	0	
8	210.JPG	c:\\p-fish\\TESTDIR\Before and After\210.J	1360820	Wed Aug 18 20:15:04 2010	Sun Aug 04 17:23:39 2013	Sun Aug 04 17:23:39 2013	171114DB94D5BEF4ECB48127B1F5262A	0	
9	291.JPG	c:\\p-fish\\TESTDIR\Before and After\291.J	1696539	Tue Oct 26 17:08:56 2010	Sun Aug 04 17:23:39 2013	Sun Aug 04 17:23:39 2013	D2AFBEBC1D983E65621ACF64157C3049	0	
10	292.JPG	c:\\p-fish\\TESTDIR\Before and After\292.J	2458849	Tue Oct 26 17:09:08 2010	Sun Aug 04 17:23:39 2013	Sun Aug 04 17:23:39 2013	F19ECFEDDF6FD2048D596A90B2D38C82	0	
11	293.JPG	c:\\p-fish\\TESTDIR\Before and After\293.J	1526724	Tue Oct 26 17:09:21 2010	Sun Aug 04 17:23:39 2013	Sun Aug 04 17:23:39 2013	DDF50F95C0D495277E6FCF93DAC69A7A	0	
12	Appriasal-1.docx	c:\\p-fish\\TESTDIR\Before and After\Appri	1057774	Sun Apr 03 18:12:31 2011	Sun Aug 04 17:23:39 2013	Sun Aug 04 17:23:39 2013	D4B00F8E1301FD9F6F17CCC49E2DAB34	0	
13	Appriasal.docx	c:\\p-fish\\TESTDIR\Before and After\Appri	1055648	Sun Apr 03 15:22:42 2011	Sun Aug 04 17:23:39 2013	Sun Aug 04 17:23:39 2013	E51EE04AAEC7F072BFD5B87BF49999B6	0	
14	DSCI0001.JPG	c:\\p-fish\\TESTDIR\Before and After\DSCI(	1714536	Thu Nov 13 12:51:47 2008	Sun Aug 04 17:23:39 2013	Sun Aug 04 17:23:39 2013	17D62097DFA5E4B2C52275EB6C2A8A4A	0	
15	DSCI0003.JPG	c:\\p-fish\\TESTDIR\Before and After\DSCI(	1662235	Thu Nov 13 12:51:38 2008	Sun Aug 04 17:23:40 2013	Sun Aug 04 17:23:40 2013	4AA02853B79D965224D5FA7C17656EEE	0	
16	DSCI0009.JPG	c:\\p-fish\\TESTDIR\Before and After\DSCI(	1834364	Thu Nov 13 12:51:11 2008	Sun Aug 04 17:23:40 2013	Sun Aug 04 17:23:40 2013	1B1D58CA06C87B069AA18384019279A5	0	
17	DSCI0010.JPG	c:\\p-fish\\TESTDIR\Before and After\DSCI(	1866146	Thu Nov 13 12:51:06 2008	Sun Aug 04 17:23:40 2013	Sun Aug 04 17:23:40 2013	83E5C851D898FD8BAEF6A83234D1DD1E	0	
18	DSCI0039.JPG	c:\\p-fish\\TESTDIR\Before and After\DSCI(	1854320	Thu Nov 13 12:48:40 2008	Sun Aug 04 17:23:40 2013	Sun Aug 04 17:23:40 2013	0CC84432452B1C8849C33C91BCE7503D	0	
19	DSCI0040.JPG	c:\\p-fish\\TESTDIR\Before and After\DSCI(	1858942	Thu Nov 13 12:48:35 2008	Sun Aug 04 17:23:40 2013	Sun Aug 04 17:23:40 2013	04C089DBA8534C87AC6C4DD6F95FFEBB	0	
20	DSCI0060.JPG	c:\\p-fish\\TESTDIR\Before and After\DSCI(	1643677	Thu Nov 13 12:47:03 2008	Sun Aug 04 17:23:40 2013	Sun Aug 04 17:23:40 2013	3A94C825420FF98EF3C7538A1F068B92	0	
21	DSCI0107.JPG	c:\\p-fish\\TESTDIR\Before and After\DSCI(	1875915	Thu Nov 13 12:43:27 2008	Sun Aug 04 17:23:40 2013	Sun Aug 04 17:23:40 2013	F366947E754F458553EC87455D2B3CAA	0	
22	DSCI0113.JPG	c:\\p-fish\\TESTDIR\Before and After\DSCI(	1807935	Thu Nov 13 12:42:56 2008	Sun Aug 04 17:23:41 2013	Sun Aug 04 17:23:41 2013	2A028AC99DD443D6721B7047BFE13956	0	
23	x_imix.png	c:\\p-fish\\TESTDIR\Mayan Glyphs\x_imix.pr	1295	Sat Feb 26 10:42:06 2011	Sun Aug 04 17:23:41 2013	Sun Aug 04 17:23:41 2013	BE11C3A6A2683F04CFD04636BB8A9774	0	
24	Day -01 imix_m.png	c:\\p-fish\\TESTDIR\Mayan Glyphs\Days\Day	1780	Sat Feb 26 10:42:59 2011	Sun Aug 04 17:23:41 2013	Sun Aug 04 17:23:41 2013	4E859195EFC728A219D98603C1B46975	0	
25	Day -02 ik_m.png	c:\\p-fish\\TESTDIR\Mayan Glyphs\Days\Day	1679	Sat Feb 26 10:44:45 2011	Sun Aug 04 17:23:41 2013	Sun Aug 04 17:23:41 2013	D0227C54811BDB8B7675EBF37768BE09	0	
26	Day -03 akbal_m.pn	c:\\p-fish\\TESTDIR\Mayan Glyphs\Days\Day	1830	Sat Feb 26 10:45:45 2011	Sun Aug 04 17:23:41 2013	Sun Aug 04 17:23:41 2013	5E55F9B38BF67FD2FA2CADA496ECE793	0	
27	Day -04 kan_m.png	c:\\p-fish\\TESTDIR\Mayan Glyphs\Days\Day	1820	Sat Feb 26 10:46:01 2011	Sun Aug 04 17:23:41 2013	Sun Aug 04 17:23:41 2013	8DCD90A6A0C77990DA3E18F7692D5B80	0	
28	Day -05 chicchan_m	c:\\p-fish\\TESTDIR\Mayan Glyphs\Days\Day	1939	Sat Feb 26 10:46:20 2011	Sun Aug 04 17:23:41 2013	Sun Aug 04 17:23:41 2013	F103B23DC0F04B0F89FC8364B033F0DE	0	
29	Day -06 cimi_m.png	c:\\p-fish\\TESTDIR\Mayan Glyphs\Days\Day	1937	Sat Feb 26 10:47:56 2011	Sun Aug 04 17:23:41 2013	Sun Aug 04 17:23:41 2013	D5642A9A9CE075721BF405EA60F379C1	0	
30	Day -07 manik_m.pn	c:\\p-fish\\TESTDIR\Mayan Glyphs\Days\Day	1818	Sat Feb 26 10:48:13 2011	Sun Aug 04 17:23:41 2013	Sun Aug 04 17:23:41 2013	9392BDA821CD7A20D878285DE505E88B	0	
31	Day -08 lamat_m.pn	c:\\p-fish\\TESTDIR\Mayan Glyphs\Days\Day	1802	Sat Feb 26 10:48:25 2011	Sun Aug 04 17:23:41 2013	Sun Aug 04 17:23:41 2013	49653111B9B2129F3389DE4B425487EF	0	
32	Day -09 muluc_m.pn	c:\\p-fish\\TESTDIR\Mayan Glyphs\Days\Day	1763	Sat Feb 26 10:48:40 2011	Sun Aug 04 17:23:42 2013	Sun Aug 04 17:23:42 2013	4736CC5C8E5B370CA1BF03703D5316E2	0	
33	Day -10 oc_m.png	c:\\p-fish\\TESTDIR\Mayan Glyphs\Days\Day	1856	Sat Feb 26 10:49:03 2011	Sun Aug 04 17:23:42 2013	Sun Aug 04 17:23:42 2013	E182C275F39737B67CE1B29B68AAA938	0	
34	Day -11 chuen_m.pn	c:\\p-fish\\TESTDIR\Mayan Glyphs\Days\Day	1797	Sat Feb 26 10:50:30 2011	Sun Aug 04 17:23:42 2013	Sun Aug 04 17:23:42 2013	F3DDEA5BDA750EB1021974B217018426	0	
35	Day -12 eb_m.png	c:\\p-fish\\TESTDIR\Mayan Glyphs\Days\Day	1905	Sat Feb 26 10:50:39 2011	Sun Aug 04 17:23:42 2013	Sun Aug 04 17:23:42 2013	A5493586401873D929505789AE6E9D68	0	
36	Day -13 ben_m.png	c:\\p-fish\\TESTDIR\Mayan Glyphs\Days\Day	1757	Sat Feb 26 10:50:49 2011	Sun Aug 04 17:23:42 2013	Sun Aug 04 17:23:42 2013	064D3A9C251B636778F7D3D656C35BA9	0	
37	Day -14 ix_m.png	c:\\p-fish\\TESTDIR\Mayan Glyphs\Days\Day	1787	Sat Feb 26 10:50:57 2011	Sun Aug 04 17:23:42 2013	Sun Aug 04 17:23:42 2013	2BA7BE52C3D63BA8A9364202E2C1A439	0	
38	Day -15 men_m.png	c:\\p-fish\\TESTDIR\Mayan Glyphs\Days\Day	1937	Sat Feb 26 10:51:07 2011	Sun Aug 04 17:23:42 2013	Sun Aug 04 17:23:42 2013	3BEE4476FC93F019A4F778A80D94FED7	0	
39	Day -16 cib_m.png	c:\\p-fish\\TESTDIR\Mayan Glyphs\Days\Day	1831	Sat Feb 26 10:51:17 2011	Sun Aug 04 17:23:42 2013	Sun Aug 04 17:23:42 2013	8256AFDD02B9A2AE2B1670AD486DC046	0	
40	Day -17 caban_m.pn	c:\\p-fish\\TESTDIR\Mayan Glyphs\Days\Day	1756	Sat Feb 26 10:51:29 2011	Sun Aug 04 17:23:42 2013	Sun Aug 04 17:23:42 2013	80FEDAEE6A20DE45B818EB5EC551E60E	0	

FIGURE 3.9

Examining the Result File with Microsoft Excel.

FIGURE 3.10

Contents of the pFishLog file.

The generated pFishLog.log file results are depicted in Figure 3.10. As expected we find the welcome message, the details regarding the Windows environment, the root path that was specified by the user, the number of files processed, and the elapsed time of just under 1 s. In this example no errors were encountered and the program terminated normally.

Moving to a Linux platform for execution only requires us to copy two Python Files.

1. pfish.py
2. _pfish.py

Execution under Linux (Ubuntu version 12.04 LTS in this example) works ***without*** changing any Python code and produces the following results shown in Figures 3.11–3.13.

You notice that the pFishLog file under Linux has a number of warnings; this is due to lack of read access to many of the files within the /etc directory at the user privilege level I was running and due to some of the files being locked because they were in use.

CHAPTER REVIEW

In this chapter I created our first useable Python forensics application. The pfish.py program executes on both Windows and Linux platforms and through some ingenuity I only used Python Standard Library modules to accomplish this along with our own code. I also scratched the surface with `argparse` allowing us to not only parse the command line but also validate command line parameters before they were used by the application.

I also enabled the Python logger and reported events and errors to the logging system to provide a forensic record of our actions. I provided the user with the capability of selecting among the most popular one-way hashing algorithms and the program extracted key attributes of each file that was processed. I also leveraged the `cvs` module to create a nicely formatted output file that can be opened and processed by standard applications on both Windows and Linux systems. Finally, I implemented our first class in Python with many more to come.

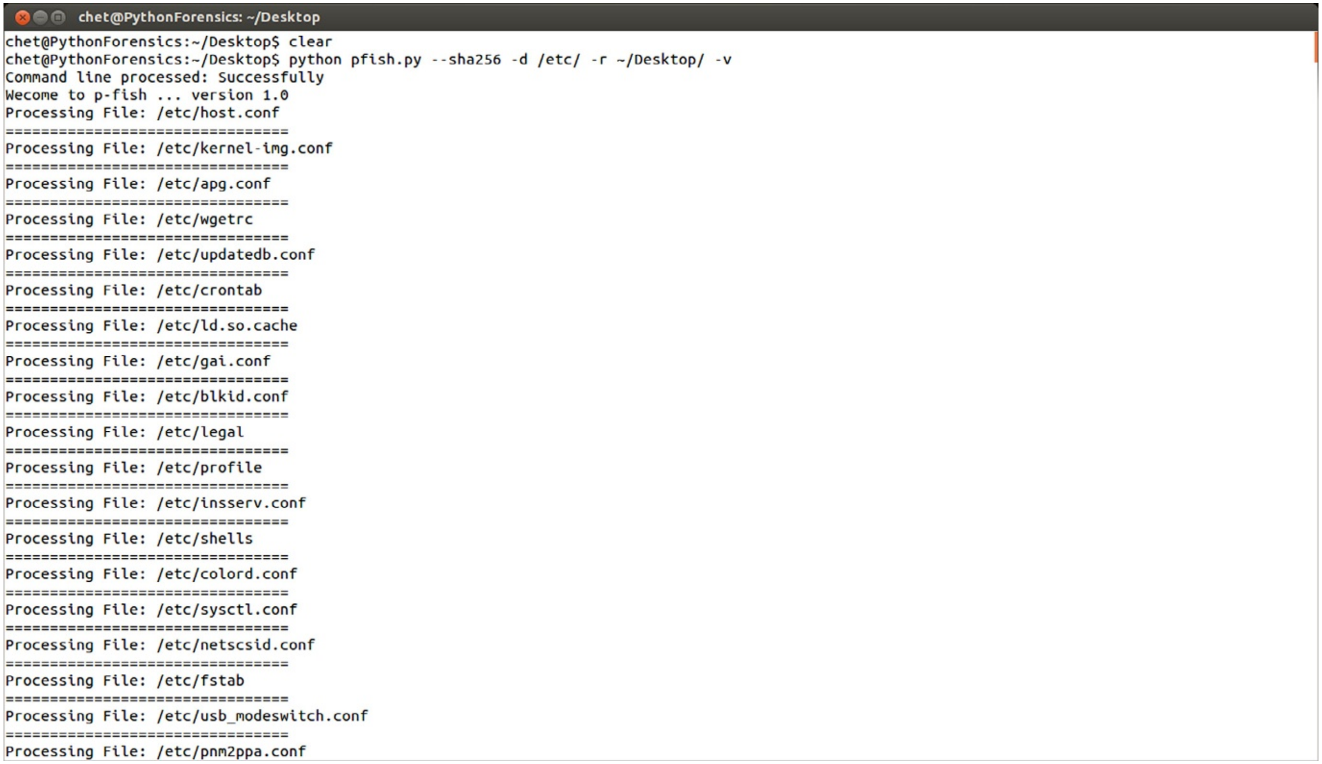
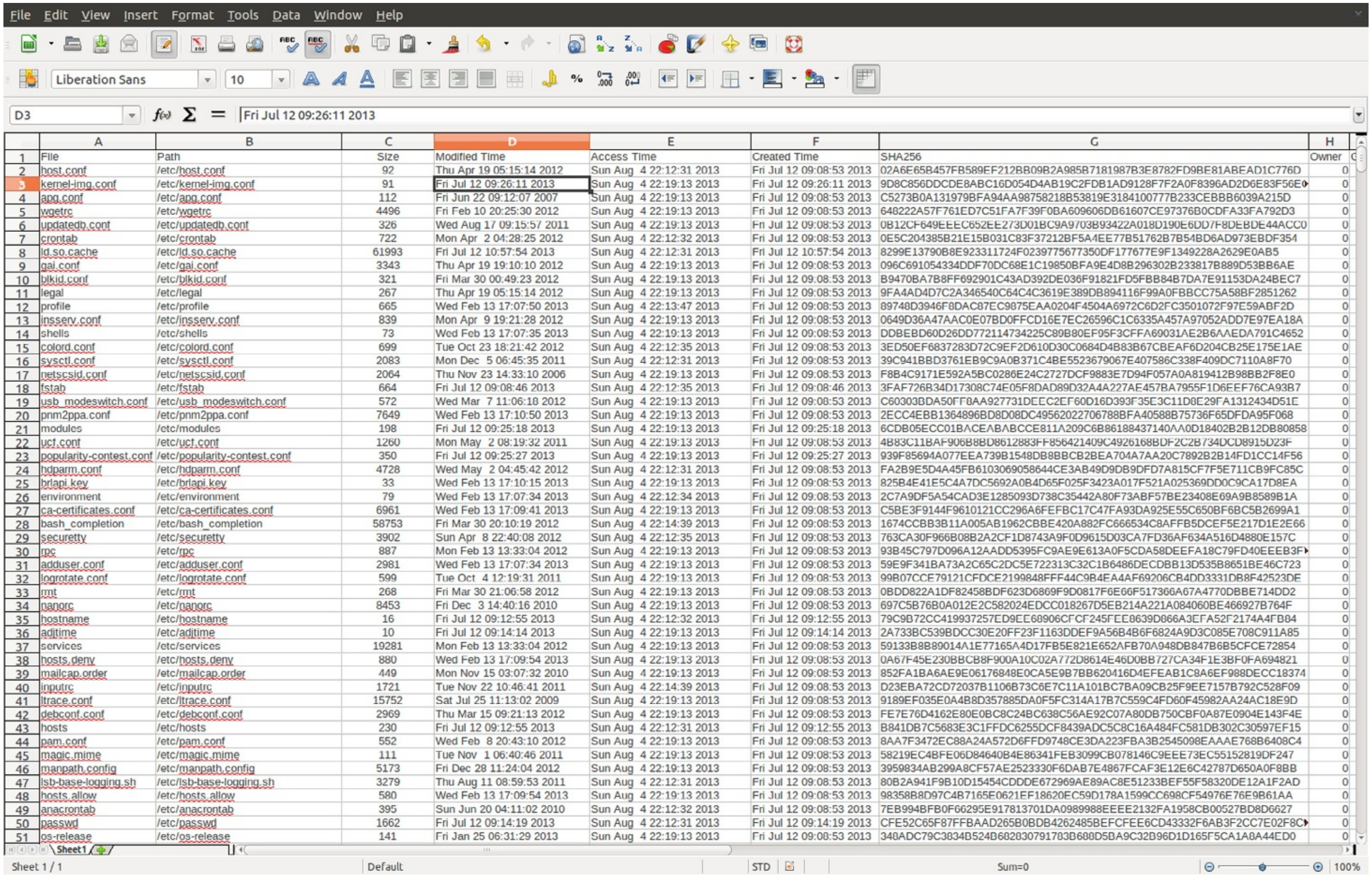


FIGURE 3.11

Linux Command Line Execution.



	A	B	C	D	E	F	G	H
1	File	Path	Size	Modified Time	Access Time	Created Time	SHA256	Owner
2	host.conf	/etc/host.conf	92	Thu Apr 19 05:15:14 2012	Sun Aug 4 22:12:31 2013	Fri Jul 12 09:08:53 2013	02A6E65B457FB589EF212BB09B2A985B7181987B3E8782FD9BE81ABEAD1C776D	0
3	kernel-img.conf	/etc/kernel-img.conf	91	Fri Jul 12 09:26:11 2013	Sun Aug 4 22:19:13 2013	Fri Jul 12 09:26:11 2013	9D8C856DDCDE8ABC16D054D4AB19C2FDB1AD9128F7F2A0F8396AD2D6E83F56E0	0
4	apg.conf	/etc/apg.conf	112	Fri Jun 22 09:12:07 2007	Sun Aug 4 22:19:13 2013	Fri Jul 12 09:08:53 2013	C5273B0A131979BFA94AA98758218B53819E3184100777B233CEBBB6039A215D	0
5	wgetrc	/etc/wgetrc	4496	Fri Feb 10 20:25:30 2012	Sun Aug 4 22:19:13 2013	Fri Jul 12 09:08:53 2013	648222A57F761ED7C51FA7F39F0BA609606DB61607CE97376B0CDFA33FA792D3	0
6	updatedb.conf	/etc/updatedb.conf	326	Wed Aug 17 09:15:57 2011	Sun Aug 4 22:19:13 2013	Fri Jul 12 09:08:53 2013	0B12CF649EEEC652EE273D01BC9A9703B93422A018D190E6DD7F8DEBDE44ACC0	0
7	crontab	/etc/crontab	722	Mon Apr 2 04:28:25 2012	Sun Aug 4 22:12:32 2013	Fri Jul 12 09:08:53 2013	0E5C204385B21E15B031C83F37212BF5A4EE77B51762B7B54BD6AD973EBDF354	0
8	ld.so.cache	/etc/ld.so.cache	61993	Fri Jul 12 10:57:54 2013	Sun Aug 4 22:12:31 2013	Fri Jul 12 10:57:54 2013	8299E13790B8E923311724F0239775677350DF177677E9F1349228A2629E0AB5	0
9	gai.conf	/etc/gai.conf	3343	Thu Apr 19 19:10:10 2012	Sun Aug 4 22:19:13 2013	Fri Jul 12 09:08:53 2013	096C691054334DDF70DC68E1C19850BFA9E4D8B296302B233817B889D53BB6AE	0
10	blkid.conf	/etc/blkid.conf	321	Fri Mar 30 00:49:23 2012	Sun Aug 4 22:19:13 2013	Fri Jul 12 09:08:53 2013	B9470BA7B8FF692901C43AD392DE036F91821FD5FBB84B7DA7E91153DA24BEC7	0
11	legal	/etc/legal	267	Thu Apr 19 05:15:14 2012	Sun Aug 4 22:19:13 2013	Fri Jul 12 09:08:53 2013	9FA4AD4D7C2A346540C64C4C3619E389DB894116F99A0FBBCC75A58BF2851262	0
12	profile	/etc/profile	665	Wed Feb 13 17:07:50 2013	Sun Aug 4 22:13:47 2013	Fri Jul 12 09:08:53 2013	89748D3946F8DAC87EC9875EAA0204F4504A6972C6D2FC3501072F97E59ABF2D	0
13	insserv.conf	/etc/insserv.conf	839	Mon Apr 9 19:21:28 2012	Sun Aug 4 22:19:13 2013	Fri Jul 12 09:08:53 2013	0649D36A47AAC0E07BD0FFCD16E7EC26596C1C6335A457A97052ADD7E97EA18A	0
14	shells	/etc/shells	73	Wed Feb 13 17:07:35 2013	Sun Aug 4 22:19:13 2013	Fri Jul 12 09:08:53 2013	DDBEBD60D26DD772114734225C89B80EF95F3CFFA69031AE2B6AAEDA791C4652	0
15	colord.conf	/etc/colord.conf	699	Tue Oct 23 18:21:42 2012	Sun Aug 4 22:12:35 2013	Fri Jul 12 09:08:53 2013	3ED50EF6837283D72C9EF2D610D30C0684D4B83B67CBEAF6D204CB25E175E1AE	0
16	sysctl.conf	/etc/sysctl.conf	2083	Mon Dec 5 06:45:35 2011	Sun Aug 4 22:12:31 2013	Fri Jul 12 09:08:53 2013	39C941BBD3761EB9C9A0B371C4BE5523679067E407586C338F409DC7110A8F70	0
17	netscsid.conf	/etc/netscsid.conf	2064	Thu Nov 23 14:33:10 2006	Sun Aug 4 22:19:13 2013	Fri Jul 12 09:08:53 2013	F8B4C9171E592A5BC0286E24C2727DCF9883E7D94F057A0A819412B98BB2F8E0	0
18	fstab	/etc/fstab	664	Fri Jul 12 09:08:46 2013	Sun Aug 4 22:12:35 2013	Fri Jul 12 09:08:46 2013	3FAF726B34D17308C74E05F8DAD89D32A4A227AE457BA7955F1D6EEF76CA93B7	0
19	usb_modeswitch.conf	/etc/usb_modeswitch.conf	572	Wed Mar 7 11:06:18 2012	Sun Aug 4 22:19:13 2013	Fri Jul 12 09:08:53 2013	C60303BDA50FF8AA927731DEEC2EF60D16D393F35E3C11D8E29FA1312434D51E	0
20	pnm2ppa.conf	/etc/pnm2ppa.conf	7649	Wed Feb 13 17:10:50 2013	Sun Aug 4 22:19:13 2013	Fri Jul 12 09:08:53 2013	2ECC4EBB1364896BD8D08DC49562022706788BFA40588B75736F65DFDA95F068	0
21	modules	/etc/modules	198	Fri Jul 12 09:25:18 2013	Sun Aug 4 22:19:13 2013	Fri Jul 12 09:25:18 2013	6CDB05ECC01BACEABABCCE811A209C6B86188437140AA0D18402B2B12DB80858	0
22	ucf.conf	/etc/ucf.conf	1260	Mon May 2 08:19:32 2011	Sun Aug 4 22:19:13 2013	Fri Jul 12 09:08:53 2013	4B83C11BAF906B8BD8612883FF856421409C4926168BDF2C2B734DCD8915D23F	0
23	popularity-contest.conf	/etc/popularity-contest.conf	350	Fri Jul 12 09:25:27 2013	Sun Aug 4 22:19:13 2013	Fri Jul 12 09:25:27 2013	939F85694A077EEA739B1548DB8BBCB2BEA704A7AA20C7892B2B14FD1CC14F56	0
24	hdparm.conf	/etc/hdparm.conf	4728	Wed May 2 04:45:42 2012	Sun Aug 4 22:12:31 2013	Fri Jul 12 09:08:53 2013	FA2B9E5D4A45FB6103069058644CE3AB49D9DB9DFD7A815CF7F5E711CB9FC85C	0
25	brlapi.key	/etc/brlapi.key	33	Wed Feb 13 17:10:15 2013	Sun Aug 4 22:19:13 2013	Fri Jul 12 09:08:53 2013	825B4E41E5C4A7DC5692A0B4D65F025F3423A017F521A025369DD0C9CA17D8EA	0
26	environment	/etc/environment	79	Wed Feb 13 17:07:34 2013	Sun Aug 4 22:12:34 2013	Fri Jul 12 09:08:53 2013	2C7A9DF5A54CAD3E1285093D738C35442A80F73ABF57BE23408E69A9B8589B1A	0
27	ca-certificates.conf	/etc/ca-certificates.conf	6961	Wed Feb 13 17:09:41 2013	Sun Aug 4 22:19:13 2013	Fri Jul 12 09:08:53 2013	C5BE3F9144F9610121CC296A6FEFBC17C47FA93DA925E55C650BF6BC5B2699A1	0
28	bash_completion	/etc/bash_completion	58753	Fri Mar 30 20:10:19 2012	Sun Aug 4 22:14:39 2013	Fri Jul 12 09:08:53 2013	1674CCBB3B11A005AB1962CBBE420A882FC666534C8AFFB5DCEF5E217D1E2E66	0
29	securetty	/etc/securetty	3902	Sun Apr 8 22:40:08 2012	Sun Aug 4 22:12:35 2013	Fri Jul 12 09:08:53 2013	763CA30F966B08B2A2CF1D8743A9F0D9615D03CA7FD36AF634A516D4880E157C	0
30	rpc	/etc/rpc	887	Mon Feb 13 13:33:04 2012	Sun Aug 4 22:19:13 2013	Fri Jul 12 09:08:53 2013	93B45C797D096A12AADD5395FC9AE9E613A0F5CDA58DEEFA18C79FD40EEEB3F	0
31	adduser.conf	/etc/adduser.conf	2981	Wed Feb 13 17:07:34 2013	Sun Aug 4 22:19:13 2013	Fri Jul 12 09:08:53 2013	59E9F341BA73A2C65C2DC5E722313C32C1B6486DECDBB13D535B8651BE46C723	0
32	logrotate.conf	/etc/logrotate.conf	599	Tue Oct 4 12:19:31 2011	Sun Aug 4 22:19:13 2013	Fri Jul 12 09:08:53 2013	99B07CCE79121CFDCE2199848FFF44C9B4EA4AF69206CB4DD3331DB8F42523DE	0
33	rmt	/etc/rmt	268	Fri Mar 30 21:06:58 2012	Sun Aug 4 22:19:13 2013	Fri Jul 12 09:08:53 2013	0BDD822A1DF82458BDF623D6869F9D0817F6E66F517366A67A4770DBBE714DD2	0
34	nanorc	/etc/nanorc	8453	Fri Dec 3 14:40:16 2010	Sun Aug 4 22:19:13 2013	Fri Jul 12 09:08:53 2013	697C5B76B0A012E2C582024EDCC018267D5EB214A221A084060BE466927B764F	0
35	hostname	/etc/hostname	16	Fri Jul 12 09:12:55 2013	Sun Aug 4 22:12:32 2013	Fri Jul 12 09:12:55 2013	79C9B72CC419937257ED9EE68906CFCF245FEE8639D866A3EFA52F2174A4FB84	0
36	adjtime	/etc/adjtime	10	Fri Jul 12 09:14:14 2013	Sun Aug 4 22:19:13 2013	Fri Jul 12 09:14:14 2013	2A733BC539BDCC30E20FF23F1163DDEF9A56B4B6F6824A9D3C085E708C911A85	0
37	services	/etc/services	19281	Mon Feb 13 13:33:04 2012	Sun Aug 4 22:19:13 2013	Fri Jul 12 09:08:53 2013	59133B8B89014A1E77165A4D17FB5E821E652AFB70A948DB847B6B5CFCE72854	0
38	hosts.deny	/etc/hosts.deny	880	Wed Feb 13 17:09:54 2013	Sun Aug 4 22:19:13 2013	Fri Jul 12 09:08:53 2013	0A67F45E230BBCB8F900A10C02A772D8614E46D0BB727CA34F1E3BF0FA694821	0
39	mailcap.order	/etc/mailcap.order	449	Mon Nov 15 03:07:32 2010	Sun Aug 4 22:19:13 2013	Fri Jul 12 09:08:53 2013	852FA1BA6AE9E06176848E0CA5E9B7BB620416D4EFEAB1C8A6EF988DECC18374	0
40	inputrc	/etc/inputrc	1721	Tue Nov 22 10:46:41 2011	Sun Aug 4 22:14:39 2013	Fri Jul 12 09:08:53 2013	D23EBA72CD72037B1106B73C6E7C11A101BC7BA09CB25F9EE7157B792C528F09	0
41	ltrace.conf	/etc/ltrace.conf	15752	Sat Jul 25 11:13:02 2009	Sun Aug 4 22:19:13 2013	Fri Jul 12 09:08:53 2013	9189EF035E0A4B8D357885DA0F5FC314A17B7C559C4FD60F45982AA24AC18E9D	0
42	debconf.conf	/etc/debconf.conf	2969	Thu Mar 15 09:21:13 2012	Sun Aug 4 22:19:13 2013	Fri Jul 12 09:08:53 2013	FE7E76D4162E80E0BC8C24BC638C56AE92C07A80DB750CBF0A87E0904E143F4E	0
43	hosts	/etc/hosts	230	Fri Jul 12 09:12:55 2013	Sun Aug 4 22:12:31 2013	Fri Jul 12 09:12:55 2013	B841DB7C5683E3C1FFDC6255DCF8439ADC5C8C16A484FC581DB302C30597EF15	0
44	pam.conf	/etc/pam.conf	552	Wed Feb 8 20:43:10 2012	Sun Aug 4 22:19:13 2013	Fri Jul 12 09:08:53 2013	8AA7F3472EC88A24A572D6FFD9748CE3DA223FBA3B2545098EAAAE768B6408C4	0
45	magic.mime	/etc/magic.mime	111	Tue Nov 1 06:40:46 2011	Sun Aug 4 22:19:13 2013	Fri Jul 12 09:08:53 2013	58219EC4BFE06D84640B4E86341FEB3099CB078146C9EEE73EC55152819DF247	0
46	manpath.config	/etc/manpath.config	5173	Fri Dec 28 11:24:04 2012	Sun Aug 4 22:19:13 2013	Fri Jul 12 09:08:53 2013	3959834AB299A8CF57AE2523330F6DAB7E4867FCAF3E12E6C42787D650A0F8BB	0
47	lsb-base-logging.sh	/etc/lsb-base-logging.sh	3279	Thu Aug 11 08:59:53 2011	Sun Aug 4 22:12:31 2013	Fri Jul 12 09:08:53 2013	80B2A941F9B10D15454CDDDE672969AE89AC8E51233BEF55F58320DE12A1F2AD	0
48	hosts.allow	/etc/hosts.allow	580	Wed Feb 13 17:09:54 2013	Sun Aug 4 22:19:13 2013	Fri Jul 12 09:08:53 2013	98358B8D97C4B7165E0621EF18620EC59D178A1599CC698CF54976E76E9B61AA	0
49	anacrontab	/etc/anacrontab	395	Sun Jun 20 04:11:02 2010	Sun Aug 4 22:12:32 2013	Fri Jul 12 09:08:53 2013	7EB994BFB0F66295E917813701DA0989988EEEE2132FA1958CB00527BD8D6627	0
50	passwd	/etc/passwd	1662	Fri Jul 12 09:14:19 2013	Sun Aug 4 22:12:31 2013	Fri Jul 12 09:14:19 2013	CFE52C65F87FFBAAD265B0BDB4262485BEFCFEE6CD43332F6AB3F2CC7E02F8C	0
51	os-release	/etc/os-release	141	Fri Jan 25 06:31:29 2013	Sun Aug 4 22:19:13 2013	Fri Jul 12 09:08:53 2013	348ADC79C3834B524B682830791783B688D5BA9C32B96D1D165F5CA1A8A44ED0	0

FIGURE 3.12

Linux Execution Results pfish Result File.

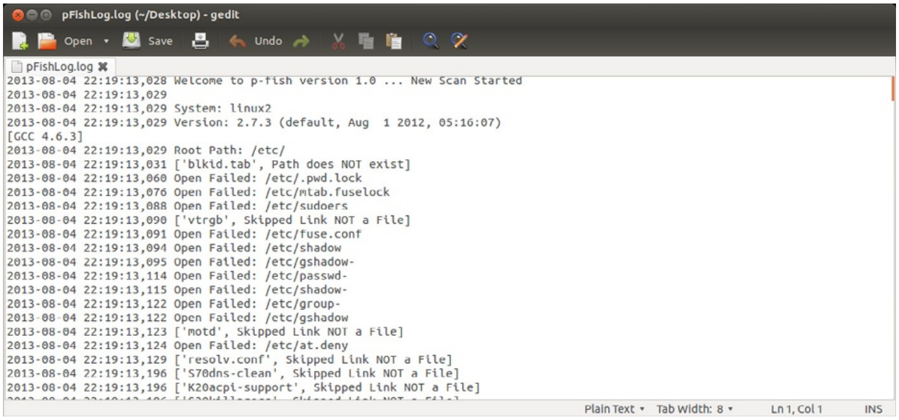


FIGURE 3.13

Linux Execution Results pFishLog File.

SUMMARY QUESTIONS

1. If you wanted to add additional one-way hashing algorithms, which functions would you need to modify? Also, by using just the Python Standard Library what other one-way hashing algorithms are readily available.
2. If you wanted to eliminate the need of the two global variables how could you easily accomplish this by using classes? What function would you convert to a class and what methods would you need to create?
3. What other events or elements do you think should be logged? How would you go about doing that?
4. What additional columns would you like to see in the report and how would you obtain the additional information?
5. What additional information (such as Investigator name or case number) should be included in the log. How would you obtain that information?

LOOKING AHEAD

In Chapter 4, I will be continuing with the cookbook section by tackling searching and indexing of forensic data.

Additional Resources

Hosmer C. Using SmartCards and digital signatures to preserve electronic evidence. In: SPIE proceedings, vol. 3576. Forensic Technologies for Crime Scene and the Laboratory I. The paper was initially presented at the investigation and forensic science technologies symposium; 1998. Boston, MA, http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=974141 [01.11.1998].

Kim G. The design and implementation of tripwire: a file system integrity checker. Purdue ePubs computer science technical reports, 1993. http://docs.lib.purdue.edu/cstech/1084/.