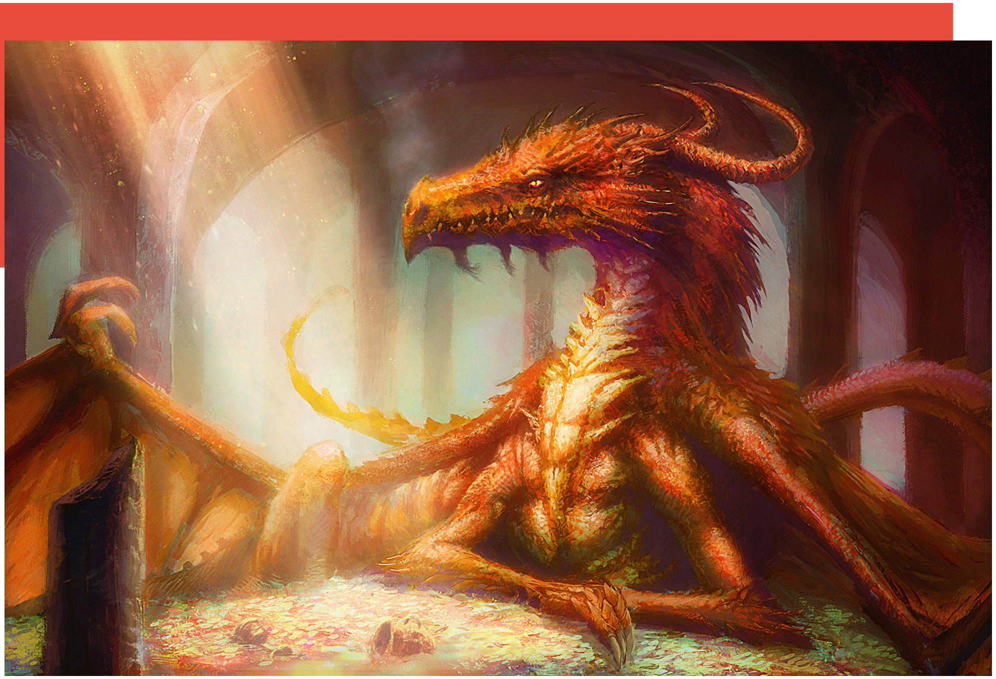
File Checksums in Python: The Hard Way

Shane Kerr <shane@time-travellers.org>

Amsterdam Python Meetup Group
2018-04-25

Data Hoarding



- **I hate losing data.**
- **I don’t trust the cloud.**
- **Disks are big now!**
- **But… bad things happen to good data.**
- **We can use checksums to detect problems.**
- **Ideal world: everything “just works”.**
  - Block or file system would detect & correct media issues.
- **Not true for Linux RAID, ext4, XFS.**
- **btrfs is relatively new, ZFS is encumbered.**

File Checksums in Bash:
The Easy Way

```
find . -type f -print0 | xargs -0 sha1sum > chksum
```

- **Doesn’t handle metadata**
- **No parallelism**
- **Not THE HARD WAY**

Python Tool

```
python3 fileinfo.py file1 [file2 [...]] > fileinfo.dat
```

- **Output format:**
  - ASCII, line-by-line
  - Context dependent, sort of command-driven
  - Would not recommend 😉

Basic Algorithm (Still Not the Hard Way)

```
for root, dirs, files in os.walk(dir_name):
    for name in dirs + files:
        join_path = os.path.join(root, name)
        full_path = os.path.normpath(join_path)
        st = os.lstat(full_path)
        if stat.S_ISREG(st.st_mode):
            h = hashlib.sha224()
            with open(full_path) as f:
                h.update(f.read())
            hash = h.digest()
        else:
            hash = None
        output(full_path, st, hash)
```

Which Python Version?

- **Python (a.k.a. Python 3, or rather CPython 3)**
- **Legacy Python (CPython 2)**
  - Started program 5 years ago, today might not bother
- **pypy**
  - Hoping for performance gain, but actually slower
- **Jython**
  - Just for fun
- ~~**Iron Python**~~
  - Missing crypto, weird `stat` values, alternate Unicode

File Name Issue: Localization

- **File systems don’t have language settings**
  - ext4 is (often) UTF-8, NTFS & VFAT are (basically) UTF-16
- **Python standard libraries try to be smart**
  - Ask for files in `b'/home/shane'`, get bytes.
  - Ask for files in `'/home/shane'`, get strings (or exceptions).
- **Escape output to look vaguely like Python strings**
  - `\x9A`, `\u81F3`, `\U12003ABF`
- **Legacy Python**
  - Everything is string-ish.

Timestamp Issues: Python and File Times (1)

- **Modern file systems store HIGHLY PRECISE timestamps**

```
$ ls -l --time-style=full-iso /etc/passwd
-rw-r--r-- 1 root root 2494 2018-04-22 22:31:47.470945551 +0200 /etc/passwd
```

- **Python usually returns time as a floating point number**
  - This is an IEEE 765 double: a 64-bit float, with only enough for 6-digits of precision on a timestamp.
- **Python 3 *also* returns nanosecond timestamps**
  - Not available on Legacy Python.

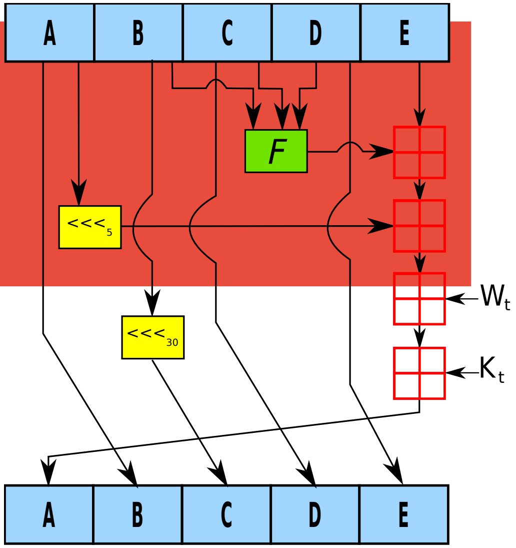
Timestamp Issues: Python and File Times (2)

- **Reading a file changes the Unix `atime` attribute**
  - Because of course reading a file should update it. 😐
  - Not pretty when we record `atime`, then read the file.
- **Using the O_NOATIME flag avoids this**
  - Not available on FreeBSD (or macOS).
  - We silently mask error, if it occurs.

Timestamp Issues: Python and File Times (3)

- **FAT file systems use a *2-second* resolution**
  - Every USB stick you buy is formatted with FAT
- **On Linux we detect files are on a FAT system**
  - We indicate in our output file
- **On other systems... ¯_(ツ)_/¯**

Which Algorithm?



- **Checksum?**
- **CRC?**
  - CRC-16? CRC-32? (both in the standard `binascii` library)
- **Hash function?**
- **Cryptographic hashing?**
  - MD5? (Possible but people would make fun of me.)
  - SHA-2? SHA-3? BLAKE2?
- **Used SHA-224 (SHA-2)**
- **Today would use BLAKE2 (but more later...)**

Multiprocessing Model

- **Pass an object around with state**
- **Split into major CPU-bound workloads:**
  1. Main thread (finds files, executes `stat` calls)
  2. Worker threads (calculate hash values of files)
  3. Serializer thread (outputs value in correct order)
- **All threads starts on program start**
- **Usually use `multiprocessing` not `threading`**
  - Runs multiple processes, which avoids Python’s GIL
- **Special path for single-core processing**
  - Eliminates work of passing objects around

inode cache

- **Unix has *hard links***
  - Actually just different paths that refer to the same file.
  - Files are uniquely identified by an *inode*.
- **Hash calculation is expensive**
  - Math is hard. Oh, and reading files requires a lot of I/O.
- **Track inodes seen**
  - We then only have to output the inode.
  - Checker can just verify inode matches.

Various Experiments

- **Binary output**
  - Provides no benefit after compressing file
- **Date values cache**
  - Provides no benefit after compressing file
- **Use external checksum program**
  - 25x slow-down
- **Use hex or base32 for output**
  - Hard to read, no benefit after compressing file

Progress Display

- **Waiting for 100’s of GB of file hashes… *boring***
- **Use `stderr` for progress (optionally)**
- **`\r` (carriage return) takes you back to column 1**
  - Each time you want new output output `\r` first
  - May need to output spaces over previous output
- **In our case, we output file counts and rates**
- **Not as sexy as ANSI-color output, but not bad**

File Checksums in Python: The Hard Way (Finally!)

On GitHub:

https://github.com/shane-kerr/fileinfo

- **1300 lines of heavily-commented code**
- **Some tests (about 700 lines)**
- **Not `flake8` or `pylint` clean**
- **No Sphinx documentation**
- **Doesn’t actually validate the results**

File Checksums in Python: The Tape Archive Way

On GitHub:

https://github.com/shane-kerr/fv

- **Similar technique, but using `tar`**
- **Stores checksums in a comment**
- **400 lines of lightly-commented code**
- **No tests, no documentation**
- **No multiprocessing**
  - Left as an exercise to the student 😉
- **DOES actually validate the results**

File Checksums in Python: The Database Way

Not (yet) on GitHub

- **Put data in database (SQLite by default)**
- **Allows stop/restart of scan and check**
- **1000 lines of uncommented code**
- **No tests, no documentation**
- **No validation**
- **Entertaining problem: restarting hash functions**
  - Can be done with `ctypes` or `ffi` or the like
  - Not for BLAKE2 though...

Image Attributions

https://commons.wikimedia.org/wiki/File:Smaug_par_David_Demaret.jpg

https://www.redbubble.com/people/swish-design/works/28828074-i-unicode

https://nl.m.wikipedia.org/wiki/Bestand:SHA-1.svg

https://commons.wikimedia.org/wiki/File:Knox-ut-research-1942.jpg