Mission Statement
Safe cryptographic steganographic advanced filesystem

Arkadiusz Bulski

Abstract

Cameleonica is a prototype of a highly versatile filesystem. This filesystem is to cover a wide range of safety and security and usability features.

Mission statement explains what the project is all about, as it is the first phase in systems engineering, it sets a direction in which further development will tend to. Purposefully omitted are any technical details on how to achieve it. Features are mostly described through basic definitions and elaborate examples, not technical implementations. Usage scenarios showing obvious benefits are also presented. Scenarios show how different people benefit from different subsets of features.

Ideas and Observations describe a wide range of theoretic concepts that could be incorporated into a new filesystem design. Potential benefits are presented, and also reasons why these features were not popularized earlier. Expectations of their performance are based on theoretic models and measured empirical evidence.

[work in progress]

Introduction

> *Cameleonica is a safe cryptographic steganographic advanced filesystem.*

This succinct definition best describes the overall project (and is the official description for the project on GitHub). That is however exactly because it uses loaded terms. Each of the terms carries a very elaborate meaning which will be discussed below. Above statement can be expanded into:

> *It aims to guarantee confidentiality, authenticity, plausible deniability, transactions, snapshots, versioning, instantaneous copying, permanent deletion, high performance, low delays, compression, and hashing.*

These properties are more elaborately explained through definitions and examples:

Filesystem is a data structure that maps a hierarchy of regular files and directories into a flat storage. Access to the files is governed by rules that make up so called semantics. Most semantics are already well defined by POSIX standard. The fact that open files can be removed and replaced while still being open is a notable example. There are also other, de-facto standards that are built upon POSIX. Linux man-pages define even more fancy semantics. For example, *fallocate* allows to remove a range of bytes without leaving a hole, stitching together both sides. Another example, *creat/open* can create anonymous temporary files that are not vulnerable to outside access since they do not have a name to be addressed with. Also temporary files are

guaranteed to disappear after interruption. Replacing files is an atomic action, which is often used in modern software as means of ensuring that some consistent version remains on disk in case of a failure. All these semantics are interesting but are not the main selling point. They only build a foundation for a useful general purpose filesystem that is protected from at most interruption.

Cryptographic filesystem is a filesystem that provides two attributes: confidentiality and authenticity. Confidentiality is a property that easily translates to filesystems. If a filesystem is encrypted, it should not reveal any information about it's file structure, names, sizes, content or usage quota until a valid password is provided. Authenticity is a property that guarantees that the files read now are same as the files written earlier. That is, a valid password must be provided before any changes can be made to the file structure. No file can be moved, truncated, or it's content changed without a valid password. Random or malicious changes to files are not allowed to remain undetected. If bytes being read were modified without a valid password, an error must be returned upon read. It is not acceptable for a read operation to return garbage bytes in this situation.

Steganographic filesystem is a filesystem that provides more than one file structure using only one backend storage. Every file structure is encrypted with an independent valid password. Steganographic confidentiality is a property that demands that if one valid password is revealed, it does not aid in discovery of any other valid password or even existence thereof. In other words, existence of valid passwords remain concealed on top of their corresponding file structures. Steganographic confidentiality is a stronger notion than cryptographic confidentiality in this respect. Multiple independent valid passwords, or equivalently independent file structures, can exist at same time within a given filesystem. As a matter of contrast, cryptography usually deals with only one password at a time. More importantly, the mere existence of that one password is hardly a secret. In steganography, zero or one or more passwords may exist and their existence is a secret in it's own right.

Safe filesystem is a filesystem that refuses to become unusable or lose access to already existing files in event of abrupt interruption. Modern filesystems are able to reliably recover from an unexpected power loss or system crash encountered at any point in time. This safety guarantee comes without any extra settings turned on. It is commonly expected from any popular filesystem to handle interruptions gracefully and reliably. However, commonly established semantics of what state is returned to after recovery are far from being acceptable. For example, once a file was opened for writing, usually a set of discrete changes is applied. An interrupted write can be partially successful, leaving content neither in the state before opening, neither in the state expected after a complete write. Often even individual file writes are not atomic. This outcome is unacceptable. Content from before changes started should be available for recovery. Another scenario is when a user copies a file overwriting the destination file. Old destination file gets truncated to zero before new content is written into it, which may take several minutes. After interruption, user can expect old content to be gone while new content is only partially present, ending at undefined position. Even worse, eventual file size does not mean that all bytes up to that offset were persisted. Filesystems can persist file size before the content. This outcome is also unacceptable. Old content from before truncation should be available for recovery. Another scenario is when a program modifies a series of related files as supposedly one operation. Consider for example rotating pictures in a photo album. User could not only expect a rotation of a single image to be atomic, but could also expect the whole album to be processed seemingly at once. After interruption, user could demand the whole album to be reverted to original state. Mentioned problems are not new and can be solved using existing mechanisms. Snapshots are becoming quite popular lately, however they are too cumbersome to be of practical use. They have to be taken manually, often enough, and usually cover a whole volume at once. This is clearly not the right way to go. Versioning infrastructure should retain the state of file structure after every significant change, making save points automatically and transparently. Usable recovery scenario should be easily discoverable for any of mentioned scenarios. User should be able to recover from sets of changes made to sets of files.

Advanced filesystem is a filesystem with outstanding usability. Usability can be meant as performance (achieving speeds of underlying hardware), as functionality (instant copying), as security (deletions equivalent to permanent wiping), as introspectability (versioning, snapshots), as utilisation (lower disk usage), as integration (speeding up utilities). Let us briefly look at all these features.

Performance of modern filesystems is getting close to physical limitations of underlying hard-drives. Processing of big files would be expected to happen at high throughput close to underlying device throughput. Processing of small files would be expected to happen at high rate due to write-back caches. To achieve that in the long term, file and free space fragmentation must be actively avoided.

Functionality of modern filesystems is mostly ancient. Paradigm of opening a file for writing, then applying a set of discrete changes, one at a time has not changed since the dawn of time. There is still room for improvement however. Hard-links were invented as means of instantaneous forking of files. This however only works in a shared-state fashion, where subsequent changes are also shared. It is possible to achieve similar performance characteristics for copying operation (cloning files), resulting in independently writable forks sharing an immutable copy of common data. Cloning files may be used to apply sets of changes to a file as atomic transactions. This operation is so beneficial that even adding a new syscall (API) for cloning files would seem warranted.

Secure deletion is another area where modern filesystems are lacking. Wiping files is not practical. User has to manually take action, and if he fails to do it right, unwanted evidence remains on disk basically forever. Performance of a wipe operation is usually comparable to writing few times more than the amount of data to be wiped. This makes wiping huge files problematic. Reliability is also a problem as filesystems do not always make guarantees whether overwriting is done in-place or copy-on-write. Above that, truncated file cannot be securely wiped as some data blocks are no longer reachable. Filesystems do not track data blocks that were used by a file. File removal could be made quick and reliable using basic cryptography.

Versioning is another functionality that is commonly missing. Users do often enough start modifying their documents without considering that they may later want to revert their changes, merely as a matter of changing their mind. Also, programs do not always apply changes in a safe, atomic manner while the user is not necessarily aware of it. This poses a risk of losing data, one way or another. Risk could be minimized by underlying filesystem by automatically creating a continuous history of changes, an ongoing list of save-points that keep being added in front of the list and being scrubbed away from the end of the list after some time.

Disk utilisation can be increased through compression of selected files. Compression can happen transparently, with the user only noticing that he can store files with more total size than hard-disk capacity.

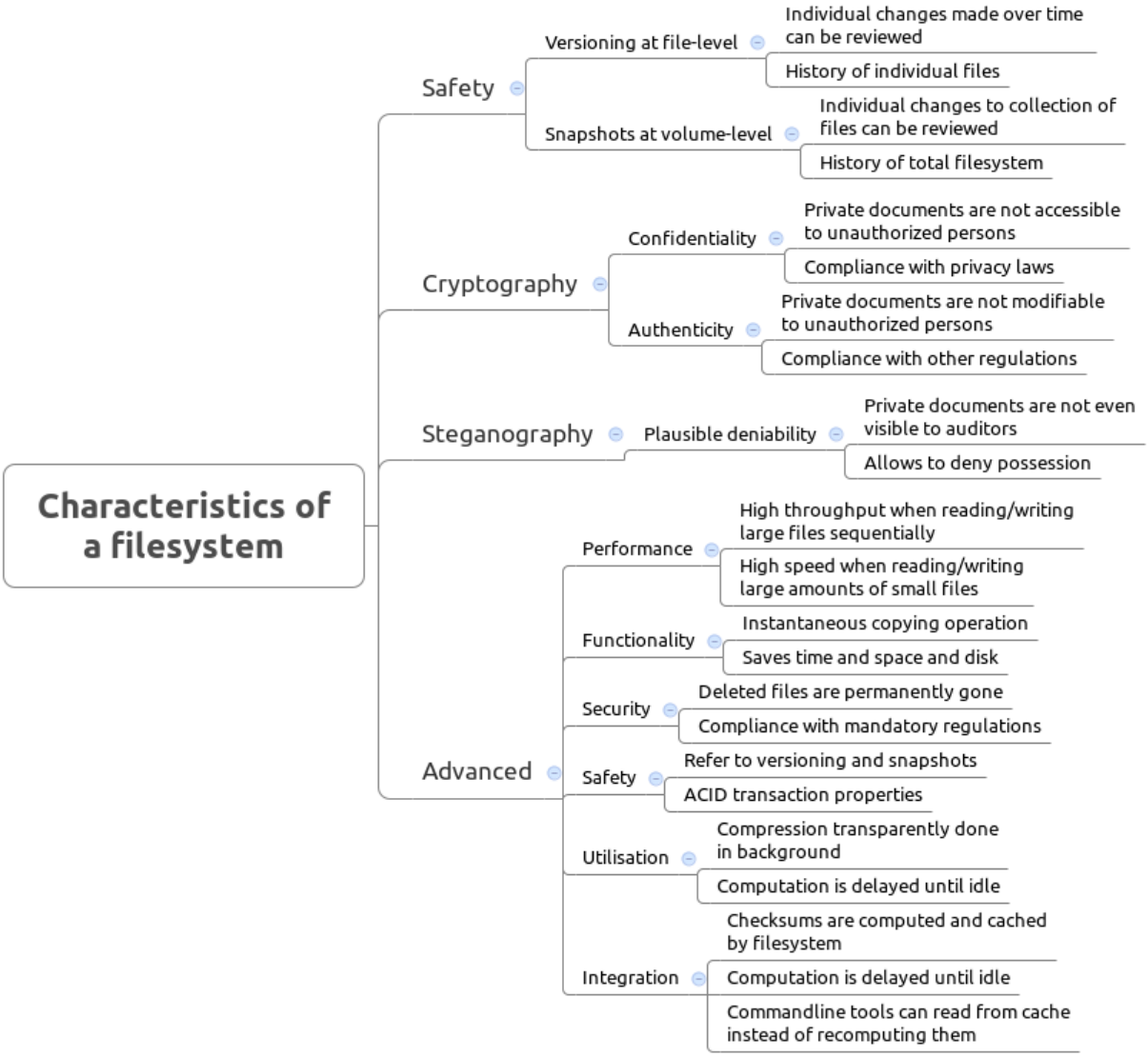
Integration is a feature where system can process files on behalf of a user space process, but having more capabilities. Filesystem is in a position where it can do more or more efficiently because it has access to internal information that processes do not have.

Compressed representation of file content can be directly accessible to compressing software making re-compression unnecessary. Current filesystems use compression only internally and do not expose encoded representation to user processes. When a utility compresses a file, a compressed representation is read from disk, decompressed internally by the filesystem, only to be then handed over to the program that will re-compress it again, with perhaps the same algorithm. It would be beneficial for the filesystem to just handed over compressed representation. This could make compression software significantly faster.

Hashing can also be done on behalf of common utilities. Filesystem can compute checksums on file content and provide system utilities with result checksums instead of all the data necessary to compute it. Computation can both be done in advance when system is idle or doing scrubbing or defragmenting, on the fly during a sequential write, or on demand when a user process requests it. Once computed, results can be cached for future use. System utilities can be modified to take advantage of filesystem provided computation when available and revert to usual method of reading file otherwise. Filesystem guarantees that cached checksum is correct

even if other processes also contributed to it's computation, by concurrent requests. User space sharing of checksums would not be safe because programs could forget to change checksums, or change them out of sync with file itself, or purposefully corrupt it. Furthermore, checksum request can be computed on a particular revision, as if a snapshot was created just for the purpose of checksumming, making sure that checksum is not affected by any writing being done while checksumming is in progress. Concurrent checksumming of a same file would not be possible on user space side, and correctness would not be possible at all.

Summary of the features and their applications:



Usage scenarios

People with different needs can benefit from using a versatile filesystem. There are several reasons why someone might want to use any given feature. Provided below are possible scenarios of people benefiting from different subsets of functionality:

Graphic designer. He works at an advertising company. Everyday he edits images and photographs provided by his agency. At regular meetings he exchanges materials with his coworkers, where everybody copy source materials from a shared drive and also copy their produced work to their superior's drive. Processing of medium sized files at high rate is beneficial, as it regularly saves time. After morning meeting, he started working on his last assignment. He would like to browse history of his changes to see what remains to be done. Regular snapshots and individual file versioning makes reviewing done work easy. He can now resume his work. While editing an image, his editor crashed. He would like to revert to a state few changes back but the image was saved after they were made and there is no undo possible after editor was closed. Even worse, editor crashed before saving was done, corrupting the only copy of the file. Filesystem keeps a continuous history of recent changes and fine grained undo is possible. He can revert to any state after a successful close operation. After a day full of work, he copies his files onto another drive, overwriting several files. Right after he started copying files power went off. After power was restored, some files were already partially overwritten and thus not usable. Replacing operation triggered a save-point however, and old version of entire file structure was quickly recovered.

Human rights activist. She works for a local newspaper. Country in which they operate is ruled by an oppressive government. Her daily job activities revolve around gathering incoming reports and writing articles to be printed. She drives a lot around the country and encounters random checkpoints. She must protect her sources and cannot allow her notes to be seized by an opportunist militia soldier. If randomly searched, her laptop is encrypted and does not provide access to unauthorized persons. She refuses to decrypt it until a warrant is presented and her agency lawyer arrives. She is briefly questioned about her business and let go. After getting back home she gets detained by a security force and questioned about any involvement with anti-government organizations. She admits having no involvement. She is presented with a warrant to search her computers. She agrees to comply and provides a valid password that decrypts some documents on her laptop. To the auditors it is clear that the provided password is indeed valid and she complied with the order to decrypt her laptop. Her laptop is throughly checked and only expected agency documents are discovered. Her interrogators do not stop accusing her of being a suspected member and keep searching her laptop for incriminating evidence. Indeed, she was regularly in contact with rebel forces and her laptop contains illegally obtained documents. If these documents were found, or even a hint of their existence was found, she would likely be taken to jail. Secret documents are kept on a separate file structure which is unlocked by a different password that she did not disclose or even mention. Filesystem itself does not reveal how many more file structures exist on the hard disk, if any. Ultimately, no evidence is found on her computer showing that she hides any documents and she is cleared of suspicion.

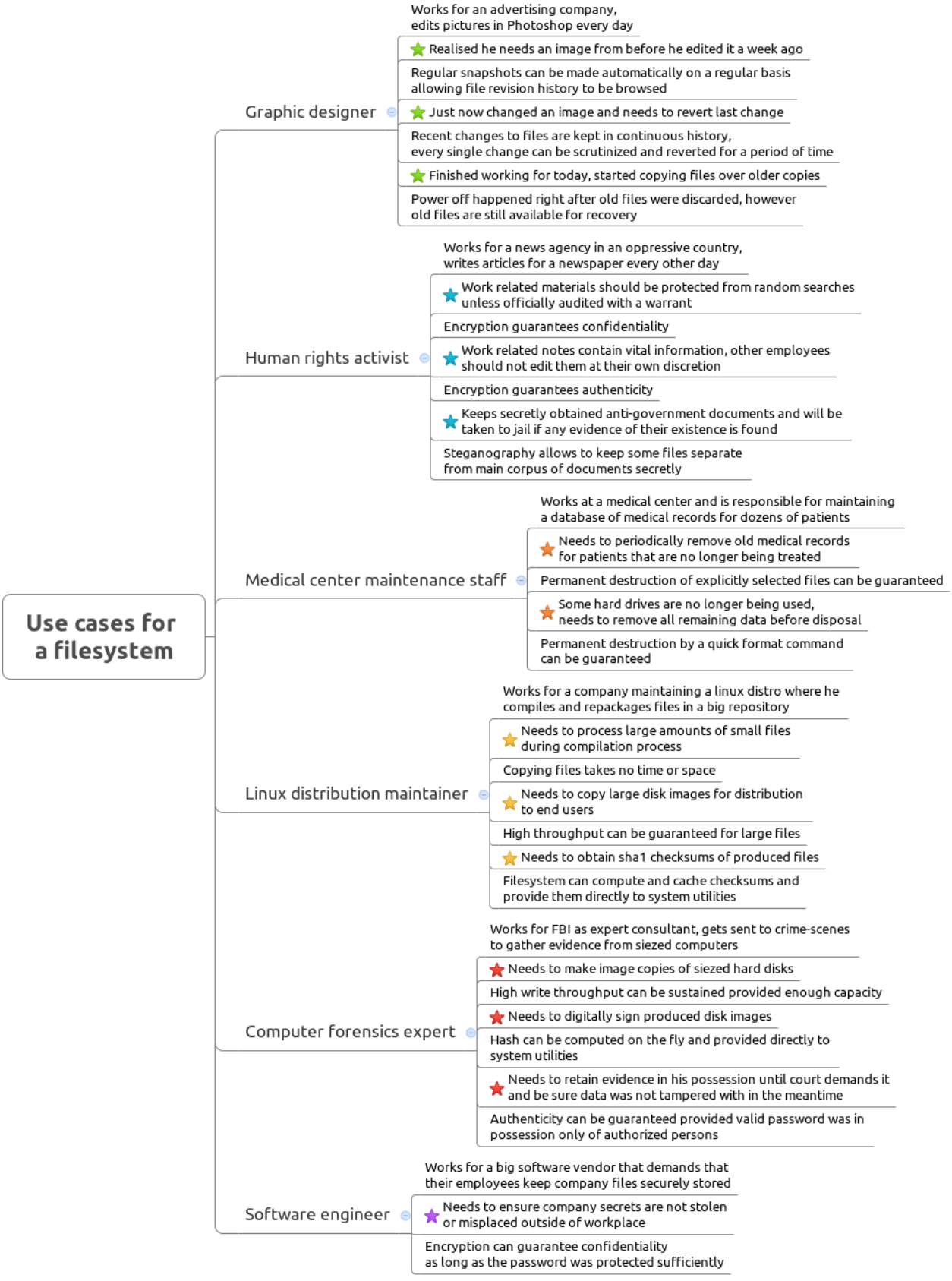
Medical center maintenance staff. He works at a major hospital that processes dozens of patients every day. His job is to maintain a database of medical records of current patients. Regularly he has to delete old records of former patients. Government regulations demand that these records are permanently purged when no longer needed. Medical center also has an obligation to guard privacy of it's patients and keep their records confidential. If these records would resurface later the center would get fined for breaking regulations or sued by patients for not providing privacy. He can rest assured that deleted files are permanently gone, as the filesystem guarantees it by design. Aside of regular file purging, there is a need to retire some hard-drives that were used for years. He needs to remove any remaining files from them before

he can send them for disposal. Again, data needs to be purged from disks permanently or the center would be liable. He can rest assured that quick formatting done on the hard drives destroyed master keys permanently as the filesystem guarantees it. Also strong passphrase can be stored on a separate device like a pendrive, so even quick formatting is not needed.

Linux distro maintainer. He works for a company maintaining a linux distro repository. Everyday he compiles and archives whole collections of source code and binaries. When he compiles, scripts often copy files which actually takes no space and no time. This takes away some time from compilation time. After a day worth of work, he sends a big disk image with upgrades. Big files are being copied at high throughput, which again saves time. Afterwards he needs to obtain sha1 checksums. Filesystem computed checksums already during copy operation. When he uses a standard command-line utility to obtain a checksum it gets results immediately from cache.

Computer forensics expert. He works for FBI as a consultant. He is often being sent to crime-scenes to secure evidence. When a computer gets seized his job is to make disk images of confiscated hard-drives. Disk images take a huge amount of space. High throughput is necessary to make copies in acceptable amount of time. Also a fraction of the disk image is full of zero bytes allowing the filesystem to compress some of the data. Hash of the image gets calculated on the fly during the long process of sequential writing. Hash of the image then gets digitally signed and handed over to the court. After this is done, he needs to keep a copy in his possession until further notice. Court later demands evidence to be presented and jury needs to be assured that these copies were not modified after being obtained. The expert can remain calm as he was the only user with a password and the filesystem can guarantee that no one else could modify the files in his possession. If defense asks for a proof that the image presented in court is the same as the image obtained during a search, expert can present a hash computed by the filesystem or the image itself.

Software engineer. He works for a major software vendor. His company has a policy that employees can bring work home only on encrypted devices. After work he took a copy of current project on a company laptop and drove back home. When shopping, his car got burglarized and the company laptop was stolen. Filesystem was encrypted and he can be sure that no company secrets fell into wrong hands. Company executives are relieved that there was no major loss and a new laptop was issued to the employee.


Use cases for a filesystem
Graphic designer
Works for an advertising company, edits pictures in Photoshop every day
Realised he needs an image from before he edited it a week ago
Regular snapshots can be made automatically on a regular basis allowing file revision history to be browsed
Just now changed an image and needs to revert last change
Recent changes to files are kept in continuous history, every single change can be scrutinized and reverted for a period of time
Finished working for today, started copying files over older copies
Power off happened right after old files were discarded, however old files are still available for recovery
Human rights activist
Works for a news agency in an oppressive country, writes articles for a newspaper every other day
Work related materials should be protected from random searches unless officially audited with a warrant
Encryption guarantees confidentiality
Work related notes contain vital information, other employees should not edit them at their own discretion
Encryption guarantees authenticity
Keeps secretly obtained anti-government documents and will be taken to jail if any evidence of their existence is found
Steganography allows to keep some files separate from main corpus of documents secretly
Medical center maintenance staff
Works at a medical center and is responsible for maintaining a database of medical records for dozens of patients
Needs to periodically remove old medical records for patients that are no longer being treated
Permanent destruction of explicitly selected files can be guaranteed
Some hard drives are no longer being used, needs to remove all remaining data before disposal
Permanent destruction by a quick format command can be guaranteed
Linux distribution maintainer
Works for a company maintaining a linux distro where he compiles and repackages files in a big repository
Needs to process large amounts of small files during compilation process
Copying files takes no time or space
Needs to copy large disk images for distribution to end users
High throughput can be guaranteed for large files
Needs to obtain sha1 checksums of produced files
Filesystem can compute and cache checksums and provide them directly to system utilities
Computer forensics expert
Works for FBI as expert consultant, gets sent to crime-scenes to gather evidence from siezed computers
Needs to make image copies of siezed hard disks
High write throughput can be sustained provided enough capacity
Needs to digitally sign produced disk images
Hash can be computed on the fly and provided directly to system utilities
Needs to retain evidence in his possession until court demands it and be sure data was not tampered with in the meantime
Authenticity can be guaranteed provided valid password was in possession only of authorized persons
Software engineer
Works for a big software vendor that demands that their employees keep company files securely stored
Needs to ensure company secrets are not stolen or misplaced outside of workplace
Encryption can guarantee confidentiality as long as the password was protected sufficiently