# Linux/UNIX Hash Verifier Command line Utility in Kotlin

Thomas A. McKernan
Department of Computer Science
University of Dayton
Dayton, Ohio USA
mckernant1@udayton.edu

## ABSTRACT

This is a hash verifier in linux/unix. The purpose of this utility is to be able to tell whether a file has been changed since the hash was created. This command line tool is written in Kotlin with Gradle as a dependency manager. Gradle allows us to integrate with tons of existing plugins. The plugin used for this CLI is picocli, which is an annotation based CLI framework.

## KEYWORDS

Command line tool, Kotlin, Linux, hashing.

**ACM Reference Format:**
Thomas A. McKernan. 2019. Linux/UNIX Hash Verifier Command line Utility in Kotlin. In *Proceedings of CPS 452: Emerging Programming Languages.* ACM, New York, NY, USA, 2 pages.

## 1 INTRODUCTION

When working on a UNIX system it is often useful to be able to see what files have changed. This tool saves a hash of a files text with an associated name. This hash can be checked and reset at any point in time.

There are many ways to do this including looking at timestamps. However, timestamps are not a good way to see if the contents of a file have changed. The changes could be to the whitespace in the file, or the changes could have been undone. The timestamp would not reflect these changes. Also the linux command `touch` changes the timestamp of files if they already exist. In summary timestamps are a bad way to check for file modification.

This hashing system allows you to save the hashes of many files or folders quickly. These hashes are saved separately for each file in a directory.

## 2 SCOPE

The scope of this tools does not include the securing of the hashes it stores. Given that this tool is built for UNIX systems, this tool relies on UNIX ownership and permissions to function securely.

This tool also relies on the Java 8 standard library for all file and path interactions, hashing and hash checking. If these methodologies were to change, the tool would not function correctly on hashes that were created before the change.

The overall scope of this project is only to store hashes in a file and be able to verify those hashes at any point.

## 3 IMPLEMENTATION

In terms of implementation there are a lot of options to choose from. Java offers lots of third party libraries that can take care of various aspects of this application. In this application, there are 3 external libraries that are essential to cutting down the amount of code that was written. This tool attempts to follow applicable linux guidelines for tool writing [3]. For example, the output can easily be piped into `grep` to easily search for files.

### 3.1 Simplicity

Kotlin is a newer JVM language with a lot of syntax upgrades and convenient helper functions[1]. One of the most helpful of these structures is the `data class`. These are simple class definitions that can be used in combination with the `kotlinx.serialization` library to create convenient JSON objects.

```
import kotlinx.serialization.Serializable
@Serializable
data class Options(
  val includeWhitespace: Boolean,
  val includeTimestamp: Boolean,
  val algorithm: String
)
```

When put into the kotlinx JSON parser this data class gets easily transformed into

```
{
  includeWhitespace: true,
  includeTimestamp: false,
  algorithm: "SHA-256"
}
```

This simple transformation easily allows reading and writing to files. The hash data is stored in JSON form so it can be easily read from and written to by the program.

In addition the picoclihttps://picocli.info/ framework is an annotation based solution to creating CLI's in Java. It translates wonderfully over to Kotlin and takes care of all the heavy lifting such as argument and option parsing. It also easily allows you to build in subcommands and options as seen below.

```
@Option(names = ["-a", "--algorithm"])
private var algorithm = "MD5"
```


*CPS 452: Emerging Programming Languages, Fall 2019, University of Dayton, Ohio 45469–0232 USA*

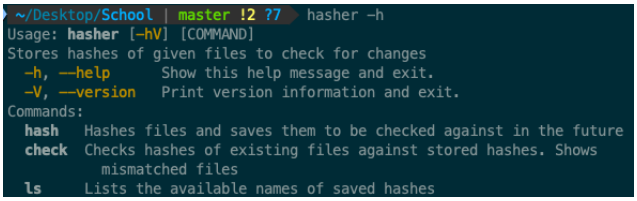


**Figure 1: A visual representation of the Picocli help menu.**

**Table 1: Hasher Available Commands**

| Subcommand | Description |
|---|---|
| hash *[opts] <name> <file>* | Creates hash with name. |
| ls *[opts] [name]* | Lists hashes/files from a hash. |
| check *[opts] [name]* | Verifies a hash. |

Picocli also allows boolean flag mixing so if you have two flags `-t` and `-w`, they can be combined as `-tw` or `-wt` without any additional code.

### 3.2 Concurrency

In Kotlin, there are a variety of concurrency options that are available. Kotlin includes the Java `Thread` class as well as its own concurrency library. In a comparision of these options [2] the best one that is available to us is the `async` function from the `kotlix.coroutines` library. This function acts like a future where the result of the async function is a `Deferred` value that can be awaited on.

```
val job: Deferred<String> = async { func() }
val result: String = job.await()
```

This operation is preferable here because if we created a thread for each file that needed to be hashed this tool would be exceedingly slow. However, because `kotlinx` uses coroutines and not threads this is a clearly superior option.

## 4 SECURITY

One application of this tools is for security. You can take a hash of any executable or file on your system and check whether it has been changed since the last hash. This has many implications including the security of the algorithms and the security of the files in which the hashes are stored as well as how the hashes are retrieved.

### 4.1 Algorithm Comparison

The allowed hashes for this tool are any of the algorithms available in Java's `MessageDigest` library. These algorithms include `MD5` and `SHA-1`. We will now compare these two hashing algorithms [4].

In a comparison of these algorithms we see that for small files the algorithms perform at about the same speed. However, for larger files `MD5` is much quicker then `SHA` [4].

This article also analyzes the security of the two hashes. Security is based on the length of the hash as well as how many steps it takes to create the hash. `SHA` generates a longer hash and takes more steps then `MD5` so it is technically more secure [4].

Sample MD5 hash

`2a38e09cc04091103590efb45c27114d`

Sample SHA-1 hash

`af0d166a1566e5201687dbb9b237246e5d61afde`

### 4.2 Possible Vulnerabilities

If you intend to use this as a security measure on a multi-user linux system you must take precaution to guard your hashes and environment variables. If another user is able to edit the hasher configuration file they could replace the hash with whatever their newly edited version is. In addition, since the file is retrieved through environment variables it is important to make sure the file that is being edited is the correct one. To prevent this hasher prints out the file it is retrieving hashes from.

## 5 FUTURE IMPROVEMENTS

As with all projects, there is always room to add more functionality. For this specific tool, there are several improvements that would make work a lot easier for users.

The first is adding an option to ignore hidden files and directories when when hashing all files in a directory. This is useful because there are a lot of folders such as `.git` that occur in multiple places and are usually not useful for looking them.

The second is to add a regex option that will ignore text matching the regex in each file. This is useful for removing comments from the hash in any given language that the files are in.

The third is to add a max depth option for searching through a directory. This is useful for making sure directories such as node modules or gradle build are kept under control.

The last is to add a global settings in the JSON settings file. This would allow users to set a default for any of the available options. In case users always want to use a certain hashing algorithm or always want whitespace to be included.

## 6 CONCLUSION

In conclusion, after analyzing the scope, implementation, and security of this application this tool serves the specific purpose of saving the hashes of files at a given point, and being able to verify against this hash at any other point. Yet this specific purpose can have a wide variety of applications, including on multi-user linux systems to a single user's specific environment.

## REFERENCES

[1] [n.d.]. Kotlin Programming Language. https://kotlinlang.org/
[2] Z. Li and E. Kraemer. 2013. Programming with Concurrency: Threads, Actors, and Coroutines. In *2013 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum*. 1304–1311. https://doi.org/10.1109/IPDPSW.2013.193
[3] D. Spinellis. 2005. Tool writing: a forgotten art? (software tools). *IEEE Software* 22, 4 (July 2005), 9–11. https://doi.org/10.1109/MS.2005.111
[4] Z. Wang and L. Cao. 2013. Implementation and Comparison of Two Hash Algorithms. In *2013 International Conference on Computational and Information Sciences*. 721–725. https://doi.org/10.1109/ICCIS.2013.195