HASH CODES

A hash is a long number, that looks something like this: f001aca2a1fbab63ce506a737b8070cc84c6d928

It is a number - it's written in hexadecimal - see Hexadecimal mini-topic if you are not familiar.

Hashes (aka hash codes, or message digests, or digests) are generated for a file by running an algorithm that is designed to always produce the same output hash for the same file, but wildly different outputs if any aspect of the file - even one bit - is changed.  There's various hash algorithms that can be used, some common ones are SHA1 ( pronounced 'sha one' with sha as in 'shadow' ) and MD5.

Any program that knows the hash algorithm can compute the hash for a file. For comparison purposes, everyone has to use the same hash algorithm, for example, SHA1. So if you have a file on your web server, and calculate and post the SHA1 hash for that file;  when I download it, I can calculate the SHA1 hash on my computer and compare the two hashes. If they are the same, then the file I have, has the exact same bytes as the file on your server. If the hashes are different, then the file has changed in transit - perhaps corrupted during the download? I'll know that I need to get a new copy, and/or figure out what went wrong - maybe someone is tampering with my network connection?

Determining if two things (two files, two sets of files, two bundles of application code, two hard drives) are the same or different is a useful thing to do in many aspects of IT. For example:

* Is my project the same as when I last worked on it, or did I make changes?  Is my copy of the project the same, or different, to my colleagues copy?
* Has this file changed since I last worked on it - should it be saved?
* Has this important file been changed - accidentally (like corrupted during a download), or maliciously (like malware inserting malicious code)?

All of these issues can be addressed by calculating hashes for the relevant files; and then re-calculating hashes as needed.

Git uses SHA1 hashes to uniquely identify commits. It uses hashes to determine if you've made any changes to your project or not; and calculates a hash code as a unique identifier for each commit.

More uses for hashes:

* The 'signatures' that antivirus software is checking? Hashes of known malware. Part of a malware scan is calculating the hash for every file on your drive, and if any match the hash of known malware, then that file is malware.
* Forensics: create a hash of a hard drive, flash drive, cellphone storage... when it is collected as evidence. A copy can be made of the media, and the hash can be checked at any time. Necessary to verify that the evidence has not been changed.
* Uniquely identify a mobile app's code package, for example, an Android .apk file or iOS .ipa file. Used to block non-app store apps being installed, and identify an app to other services they connect to. Mobile apps may use a back-end service and this service may wish to verify that only a designated app can connect.
* Verifying integrity of other files, for example checking program code is unaltered; verifying financial accounts or network log files haven't been changed.
* Hashes also have the property of irreversibility. You can't figure out the original data that was used to make the hash. So, passwords are often hashed, and the hash is stored. The original password is discarded. When a user enters their password for a system, the password is hashed, and the hash is compared to the stored hash. If the password is correct, the hashes should match. [Unfortunately, there's various ways to get around this and compromise password security; <https://labs.mwrinfosecurity.com/blog/a-practical-guide-to-cracking-password-hashes> or ask me if interested in learning more ]

Most programming languages have support for computing hashes, using various algorithms.

So we can write a little Python program to generate hashes for files....  Try this out with a file; then change one character, and try again - you should see two totally different hashes. If you then revert the file to the original state, you'll get the original hash back.

import hashlib  
import sys  
  
if len(sys.argv) < 2:  
 print('Please provide a file name e.g. python sha1 myfile.txt')  
  
else:  
 filename = sys.argv[1]  
 with open(filename, 'rb') as f:  
 bytes = f.read()  
 sha1 = hashlib.sha1(bytes).hexdigest()  
 print('The SHA1 hash of {} is {}'.format(filename, sha1))

You can compute a hash for any set of bytes you want. For example,

print(hashlib.sha1(b'hello').hexdigest() # Prints'aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d'

print(hashlib.sha1(b'jello').hexdigest() # Prints'2ced3ee86f82bf91c15cc30605df6d3ddf0769ff'

Alternatives: If you are on Mac or Linux, try typing

**openssl sha1 *filename***

where *filename* is a file on your computer. Use a file that you can modify. You should see the hash of that file.

Now, open the file, and change one character. Example: change an 'a' to a 'b'.  Run the **openssl sha1 filename** command again. What do you notice about the hashes? Are they the same? Similar? Totally different?

If you are on Windows, sorry, no command line tool. But you can use an online Hash generator, like <http://onlinemd5.com/>

More information on hashing: <https://en.wikipedia.org/wiki/Cryptographic_hash_function>