File Integrity Checker  
A Python tool to ensure file integrity through hash-based verification

15.01.2025

# Overview

## Introduction

|  |  |
| --- | --- |
|  | **Purpose**:  The File Integrity Checker is a Python-based tool that ensures the integrity of files in a directory by generating and verifying their hash values. It helps detect unauthorized changes to files and ensures data security.  **Importance**:  File integrity monitoring is vital for data security, especially in sensitive applications like financial systems, healthcare, and software development. |

## Project Scope

|  |  |
| --- | --- |
|  | **Overview**:  This tool uses cryptographic hash functions like SHA-256 or MD5 to compute file hashes. It supports two primary operations:   1. **Generate**: Create a hash file for all files in a directory. 2. **Check**: Compare current file hashes with those stored in the hash file to detect changes   **Workflow**:   1. Generate hash values for files. 2. Store hash values in a JSON file. 3. Verify files against the hash file. |

## Features & Requirements

|  |  |
| --- | --- |
|  | List the main features:   * Supports multiple hash algorithms: MD5, SHA-1, SHA-256, etc. * Generates a JSON file containing file hashes. * Detects modified, added, or deleted files. * Command-line interface for ease of use. * Scalable for directories with multiple files.   Technical requirements:   * **Operating System**: Windows, Linux, or macOS. * **Python Version**: Python 3.x. * **Libraries Used**:   + argparse   + hashlib   + os   + json |

## Installation & Usage Instructions

|  |  |
| --- | --- |
|  | **Clone or Download the Repository**:  git clone <https://github.com/your-repository-link.git>  **Install Dependencies**:  pip install -r requirements.txt  **Run the Script**: Use PyCharm or the terminal to execute the Python script. **Usage Instructions****1.Generate Hashes** python main.py generate /path/to/directory --hash-algo sha256 --hash-file h  **Check Integrity**:  python main.py check /path/to/directory --hash-algo sha256 --hash-file hashes.json |

## Project Code

|  |  |
| --- | --- |
|  | import os import hashlib import argparse import json   def compute\_hash(file\_path, hash\_algo='sha256'):  *"""Compute the hash of a file using the specified algorithm."""* hash\_func = hashlib.new(hash\_algo)  with open(file\_path, 'rb') as f:  while chunk := f.read(8192):  hash\_func.update(chunk)  return hash\_func.hexdigest()   def generate\_hashes(directory, hash\_algo, hash\_file):  *"""Generate hashes for all files in the directory and save them to a JSON file."""* hashes = {}  for root, \_, files in os.walk(directory):  for file in files:  file\_path = os.path.join(root, file)  hashes[file\_path] = compute\_hash(file\_path, hash\_algo)   with open(hash\_file, 'w') as f:  json.dump(hashes, f, indent=4)  print(f"Hashes generated and saved to {hash\_file}.")   def check\_hashes(directory, hash\_file):  *"""Check file integrity by comparing current hashes with stored hashes."""* if not os.path.exists(hash\_file):  print(f"Hash file {hash\_file} not found.")  return   with open(hash\_file, 'r') as f:  saved\_hashes = json.load(f)   current\_hashes = {}  for root, \_, files in os.walk(directory):  for file in files:  file\_path = os.path.join(root, file)  current\_hashes[file\_path] = compute\_hash(file\_path)   for file, saved\_hash in saved\_hashes.items():  current\_hash = current\_hashes.get(file)  if current\_hash is None:  print(f"File missing: {file}")  elif current\_hash != saved\_hash:  print(f"File modified: {file}")  else:  print(f"File intact: {file}")   for file in current\_hashes:  if file not in saved\_hashes:  print(f"New file detected: {file}")   if \_\_name\_\_ == "\_\_main\_\_":  parser = argparse.ArgumentParser(description="File Integrity Checker")  parser.add\_argument("action", choices=["generate", "check"], help="Action to perform")  parser.add\_argument("directory", help="Directory to process")  parser.add\_argument("--hash-algo", default="sha256", help="Hash algorithm to use (default: sha256)")  parser.add\_argument("--hash-file", default="hashes.json", help="File to save/load hash values (default: hashes.json)")   args = parser.parse\_args()   if args.action == "generate":  generate\_hashes(args.directory, args.hash\_algo, args.hash\_file)  elif args.action == "check":  check\_hashes(args.directory, args.hash\_file) |

## Example Outputs & Limitations and Future Enhancements

|  |  |
| --- | --- |
|  | **Generate Output**:  Json:  {  "/Users/dell/FileIntegrityChecker\\hashes.json": "89b39694f880ee8d329773895af3b49c07f56f1f1054de0bf1b7cd881571199b",  "/Users/dell/FileIntegrityChecker\\Paper Template General.docx": "e5d0a577e1b734e82343639aab9f6a95d29725f643f512b86f7db34561dd10ce"  }  **Check Output**:  file1.txt: OK  file2.png: MODIFIED **Limitations**:  * Currently supports only JSON as the hash file format. * No graphical user interface (GUI). * Requires Python installed on the system.  **Future Enhancements**:  * Add support for XML and CSV formats. * Develop a GUI for non-technical users. * Implement email alerts for integrity violations. |

## Specific Exclusions from Scope

|  |  |
| --- | --- |
|  | Real-time file integrity monitoring or automated alerts. |

## Conclusion

|  |  |
| --- | --- |
|  | The File Integrity Checker is a lightweight tool for monitoring file changes. It provides a robust way to ensure file security and can be extended for enterprise use |

# Approval and Authority to Proceed

We approve the project as described above, and authorize the team to proceed.

|  |  |  |
| --- | --- | --- |
| Name | Title | Date |
| Preetam Dutta (Author) | File Integrity Checker using Python Script | 15.01.2025 |
|  |  |  |
|  |  |  |