Backup Files And Folders Tool

# Code Explanation

## backupFoldersFiles.py

This Python script is designed to back up files from specified source directories to corresponding destination directories. It includes features like checksum verification, file permission preservation, and error logging. Let's break down the code section by section:

**1. Importing Modules**

python

Copy code

import os

import time

import shutil

import hashlib

import logging

import json

* **os**: For interacting with the operating system, e.g., file paths and directory operations.
* **time**: For sleeping between backup cycles and timestamping logs.
* **shutil**: For file operations like copying files.
* **hashlib**: For generating checksums (MD5) to verify file integrity.
* **logging**: For logging information and errors.
* **json**: For reading the configuration from a JSON file.

**2. Helper Functions**

**calculate\_checksum(file\_path)**

python

Copy code

def calculate\_checksum(file\_path):

"""Calculate MD5 checksum of a file."""

md5 = hashlib.md5()

with open(file\_path, 'rb') as f:

for chunk in iter(lambda: f.read(4096), b""):

md5.update(chunk)

return md5.hexdigest()

* This function calculates the MD5 checksum of a file, which is used to verify if a file has changed by comparing it with another file's checksum.

**get\_permissions(file\_path)**

python

Copy code

def get\_permissions(file\_path):

"""Get file permissions."""

return oct(os.stat(file\_path).st\_mode)[-3:]

* This function retrieves the file permissions in octal format (e.g., 755), which is useful for maintaining the same permissions in the backup.

**is\_file\_changed(src\_file, dest\_file)**

python

Copy code

def is\_file\_changed(src\_file, dest\_file):

"""Check if the file has changed by comparing checksum and modification time."""

if not os.path.exists(dest\_file):

return True

src\_checksum = calculate\_checksum(src\_file)

dest\_checksum = calculate\_checksum(dest\_file)

return src\_checksum != dest\_checksum

* This function checks if the file at dest\_file exists and, if it does, compares its checksum with the source file. If the checksums differ, it returns True (indicating the file has changed), otherwise False.

**3. Core Backup Function**

**backup\_files(src\_dir, dest\_dir)**

python

Copy code

def backup\_files(src\_dir, dest\_dir):

"""Backup files from src\_dir to dest\_dir with logging, excluding '.stfolder'."""

if not os.path.exists(dest\_dir):

os.makedirs(dest\_dir)

for root, dirs, files in os.walk(src\_dir):

# Skip the .stfolder directory

if '.stfolder' in dirs:

dirs.remove('.stfolder')

relative\_path = os.path.relpath(root, src\_dir)

dest\_path = os.path.join(dest\_dir, relative\_path)

if not os.path.exists(dest\_path):

os.makedirs(dest\_path)

for file in files:

src\_file = os.path.join(root, file)

dest\_file = os.path.join(dest\_path, file)

if is\_file\_changed(src\_file, dest\_file):

checksum = calculate\_checksum(src\_file)

permissions = get\_permissions(src\_file)

try:

shutil.copy2(src\_file, dest\_file)

logging.info(f"Backed up file: {src\_file} -> {dest\_file} | Checksum: {checksum} | Permissions: {permissions}")

except Exception as e:

error\_message = f"Failed to back up file: {src\_file} -> {dest\_file} | Error: {e}"

logging.error(error\_message)

with open(ERROR\_LOG\_FILE, 'a') as error\_log:

error\_log.write(f"{time.strftime('%Y-%m-%d %H:%M:%S')} {error\_message}\n")

else:

logging.info(f"File unchanged, skipping backup: {src\_file}")

* This function handles the actual backup process:
  + It creates the destination directory if it doesn't exist.
  + It walks through all files and subdirectories in the source directory.
  + It skips a special directory named .stfolder.
  + For each file, it checks if the file has changed using is\_file\_changed. If so, it backs up the file by copying it to the destination directory, preserving the file's metadata (e.g., permissions and modification time).
  + It logs the success or failure of each file copy operation.

**4. Main Execution Loop**

python

Copy code

if \_\_name\_\_ == "\_\_main\_\_":

while True:

# Load configuration from file

with open('backup\_config.json', 'r') as config\_file:

config = json.load(config\_file)

# Configuration

SOURCE\_DIRS = config['source\_dirs']

DEST\_DIRS = config['dest\_dirs']

LOG\_FILE = config['log\_file']

ERROR\_LOG\_FILE = config['error\_log\_file']

SLEEP\_TIME = config['sleep\_time']

RUN\_ENABLED = config['run\_enabled'] # Y = will run , anything else no run

# Logging configuration

logging.basicConfig(filename=LOG\_FILE, level=logging.INFO, format='%(asctime)s %(message)s')

if RUN\_ENABLED=="Y":

for src\_dir, dest\_dir in zip(SOURCE\_DIRS, DEST\_DIRS):

backup\_files(src\_dir, dest\_dir)

else:

logging.info(f"Run Disabled, Exiting Process: {\_\_name\_\_}")

time.sleep(SLEEP\_TIME)

* The script enters an infinite loop (while True:) where it:
  + Loads the configuration from a JSON file (backup\_config.json). This configuration includes:
    - Source and destination directories.
    - Log and error log file paths.
    - Sleep time between backup cycles.
    - A flag (RUN\_ENABLED) to enable or disable the backup process.
  + Configures logging to output messages to the specified log file.
  + If the backup process is enabled (RUN\_ENABLED == "Y"), it calls backup\_files() for each pair of source and destination directories.
  + If the backup process is disabled, it logs a message and exits.
  + The script then sleeps for the specified time (SLEEP\_TIME) before repeating the process.

**Summary**

* **Purpose**: The script is designed to back up files from one or more source directories to corresponding destination directories while checking if the files have changed.
* **Key Features**:
  + Checksums are used to detect changes.
  + File permissions are preserved.
  + Logging is implemented for both successful and failed operations.
  + The script is controlled via a JSON configuration file and runs in a loop, periodically checking and backing up files as configured.

## Json File

This JSON configuration file is used by the backup script to control how it operates. Each key-value pair in this file represents a specific configuration setting for the script. Here's an explanation of each part:

**JSON Structure**

json

Copy code

{

"run\_enabled": "N",

"source\_dirs": [

"C:\\Users\\Milind\\Music\\"

],

"dest\_dirs": [

"D:\\AutoBackup\\Music\\"

],

"log\_file": "backupFoldersFiles.log",

"error\_log\_file": "error.log",

"sleep\_time": 30

}

**Key-Value Pairs**

1. **"run\_enabled": "N"**
   * **Purpose**: Controls whether the backup process should run or not.
   * **Value**:
     + "Y": The script will perform the backup process.
     + "N": The script will not perform the backup process and will exit after logging that the run is disabled.
2. **"source\_dirs": ["C:\\Users\\Milind\\Music\\"]**
   * **Purpose**: Specifies the source directories that contain the files you want to back up.
   * **Value**:
     + "C:\\Users\\Milind\\Music\\": This is the directory on the user's system where the files to be backed up are located. You can list multiple directories here, and the script will back up files from each listed directory.
3. **"dest\_dirs": ["D:\\AutoBackup\\Music\\"]**
   * **Purpose**: Specifies the destination directories where the backed-up files should be stored.
   * **Value**:
     + "D:\\AutoBackup\\Music\\": This is the directory on the user's system where the backed-up files will be stored. It corresponds to the source directory listed under source\_dirs. Each source directory should have a corresponding destination directory listed in the same order.
4. **"log\_file": "backupFoldersFiles.log"**
   * **Purpose**: Defines the file where the script will log information about the backup process.
   * **Value**:
     + "backupFoldersFiles.log": The name of the log file where the script records details about which files were backed up, skipped, or if any errors occurred.
5. **"error\_log\_file": "error.log"**
   * **Purpose**: Specifies the file where the script will log errors that occur during the backup process.
   * **Value**:
     + "error.log": The name of the error log file where the script records details about any errors or issues that prevent files from being backed up.
6. **"sleep\_time": 30**
   * **Purpose**: Sets the time interval (in seconds) between each backup cycle.
   * **Value**:
     + 30: The script will wait for 30 seconds after completing one backup cycle before starting the next cycle.

**Summary**

* This configuration file tells the backup script:
  + Not to run the backup process ("run\_enabled": "N").
  + Where to find the files to back up ("source\_dirs").
  + Where to store the backed-up files ("dest\_dirs").
  + Which file to use for logging ("log\_file").
  + Which file to use for error logging ("error\_log\_file").
  + How long to wait between backup cycles ("sleep\_time": 30 seconds).