**CODE EXPLANATION**

import os

import sys

import time

import hashlib

import shutil

In this part, the script starts by importing necessary Python modules. These modules are used for interacting with the operating system, working with time, calculating hash values, and performing file operations.

```

def calculate\_md5(file\_path):

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

md5\_hash = hashlib.md5()

while chunk := f.read(8192):

md5\_hash.update(chunk)

return md5\_hash.hexdigest()

```

This is a function named `calculate\_md5` that calculates the MD5 hash of a given file. It reads the file in chunks and updates the hash value using the MD5 algorithm. The final hash value is returned as a hexadecimal string.

```

def log\_event(log\_file, message):

with open(log\_file, "a") as f:

f.write(message + "\n")

```

The `log\_event` function appends a given message to a log file. It opens the log file in append mode, writes the provided message along with a newline character to separate events, and then closes the file.

```

def synchronize\_folders(source\_folder, replica\_folder, log\_file):

try:

if not os.path.exists(replica\_folder):

os.makedirs(replica\_folder)

```

The `synchronize\_folders` function is the core of the synchronization process. It accepts three arguments: `source\_folder` (path to the source directory), `replica\_folder` (path to the replica directory), and `log\_file` (path to the log file). Inside the `try` block, it checks if the replica folder exists and creates it if it doesn't.

```

for root, \_, files in os.walk(source\_folder):

relative\_path = os.path.relpath(root, source\_folder)

replica\_root = os.path.join(replica\_folder, relative\_path)

if not os.path.exists(replica\_root):

os.makedirs(replica\_root)

```

This loop iterates through the source folder using `os.walk`, which traverses directories and files recursively. For each directory (`root`), it calculates the relative path with respect to the source folder and constructs the corresponding directory structure in the replica folder.

```

for file in files:

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

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

if not os.path.exists(replica\_file\_path) or \

calculate\_md5(source\_file\_path) != calculate\_md5(replica\_file\_path):

shutil.copy2(source\_file\_path, replica\_file\_path)

log\_event(log\_file, f"Copied: {source\_file\_path} -> {replica\_file\_path}")

print(f"Copied: {source\_file\_path} -> {replica\_file\_path}")

```

Inside this nested loop, for each file in the `files` list of the current directory, it constructs the paths for both the source and replica files. It checks if the replica file doesn't exist or if the MD5 hashes of the source and replica files are different. If either condition is met, it copies the file using `shutil.copy2`, logs the event using `log\_event`, and prints a message to indicate the copy operation.

```

for root, \_, files in os.walk(replica\_folder):

relative\_path = os.path.relpath(root, replica\_folder)

source\_root = os.path.join(source\_folder, relative\_path)

for file in files:

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

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

if not os.path.exists(source\_file\_path):

os.remove(replica\_file\_path)

log\_event(log\_file, f"Removed: {replica\_file\_path}")

print(f"Removed: {replica\_file\_path}")

```

This loop iterates through the replica folder in a similar fashion. For each file in the `files` list, it calculates the path in the source folder. If the source file doesn't exist, it removes the corresponding replica file, logs the event, and prints a message Which is about the removal.

```

except Exception as e:

log\_event(log\_file, f"An error occurred: {e}")

```

In case any exception occurs during the synchronization , this block catches the exception, logs an error message with the details to the log file, and handles the error perfectly.

```

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

if len(sys.argv) != 5:

print("Usage: python sudhan.py source\_folder replica\_folder interval\_in\_seconds log\_file\_path")

sys.exit(1)

source\_folder = sys.argv[1]

replica\_folder = sys.argv[2]

interval = int(sys.argv[3])

log\_file = sys.argv[4]

while True:

synchronize\_folders(source\_folder, replica\_folder, log\_file)

time.sleep(interval)

```

The final part of the code checks if the script is being run directly (not imported as a module). If it's being run directly, it checks whether the correct number of command-line arguments (five in this case) have been provided. If not, it prints a usage message and exits with an error code.

If the arguments are provided correctly, the script assigns them to `source\_folder`, `replica\_folder`, `interval`, and `log\_file` variables. Then, inside an infinite loop, it repeatedly calls the `synchronize\_folders` function to synchronize the folders, waits for the specified interval using `time.sleep`, and repeats the process. Thus synchronization happens periodically based on the provided interval.