PYTHON BASED TOOL TO SYNCHRONISE DIRECTORIES

Vivek Punia

July 2023

Contents

1 Abstract

The "Synchronization Tool" is a Python-based utility that facilitates one-way synchronization between two directories: the "Source" directory and the "Replica" directory. This tool ensures that the contents of the Replica directory remain fully identical to the Source directory. The primary goal of this report is to provide an overview of the tool's functionalities, usage instructions, and its implementation details.

2 Introduction

Data synchronization between directories is a common requirement in various scenarios, such as backing up data, replicating files across different locations, or maintaining redundancy in critical systems. The Synchronization Tool is designed to address these use cases, offering an efficient and user-friendly solution.

3 Requirements

- **Python Version**
  The Synchronization Tool is developed using Python programming language. It has been tested and verified to work with Python 3.x (where x is the latest version at the time of development). Please ensure that you have Python 3.x installed on your system to run the tool.
- **Dependencies**
  The Synchronization Tool relies on the *schedule* library, which is responsible for scheduling periodic synchronization operations. To install the necessary dependencies, the user can execute the following command:

  ```
  pip install schedule
  ```

  Alternatively, users can use the provided requirements.txt file to install the required dependencies with:

  ```
  pip install -r requirements.txt
  ```

4 Usage

To use the Synchronization Tool, the user needs to follow the steps outlined below:

1. Clone the repository from GitHub:

   ```
   git clone https://github.com/puklu/Synchronize-directories.git
   ```

2. Navigate to the location where the project is cloned. Navigate into the "src" folder:

   ```
   cd src
   ```

3. Execute the script with appropriate command-line arguments to initiate synchronization.
   The tool accepts the following parameters:
   Mandatory

   (a) -src: Specifies the path to the Source directory.

   Optional Parameters (Recommended):

   (b) -rep: Specifies the path to the Replica directory. If not provided, the tool uses the root of the project as the default location for replication.

   (c) -sync: Specifies the synchronization interval in minutes. If not provided, the default interval is set to 30 minutes.

   (d) -logf: Specifies the path to the log file. If not provided, the tool uses the root of the project as the default location for the log file.

4.1 Examples

Here are two usage examples of the Synchronization Tool with different sets of parameters:

1. With all the parameters:

   ```
   python3 main.py
   -src <path/to/source-dir>
   -rep <path/to/replica-dir>
   -sync <sync_interval>
   -logf <path/to/logfile>
   ```

2. With only the mandatory parameter

```
python3 main.py -src <path/to/source-dir>
```

5 Methodology

The Synchronization Tool follows a well-defined methodology to ensure the correct and efficient synchronization of files between the Source and Replica directories. The methodology can be summarized as follows:

1. **Creating File Name and Hash Dictionaries**
   Before performing the synchronization, the tool creates dictionaries to store file names and their corresponding MD5 hashes for both the Source and Replica directories. These dictionaries are key-value pairs, where the key represents the file name, and the value is the MD5 hash of the file's content.
2. **Populating Dictionaries from Source and Replica**
   The tool scans the Source and Replica directories to populate the respective dictionaries with file names and their MD5 hashes. This step enables efficient content comparison without the need to repeatedly read file contents during synchronization.
3. **Content Comparison based on Name and MD5 Hash**
   After creating and populating the dictionaries, the tool performs one-way synchronization from the Source directory to the Replica directory. It uses the dictionaries to identify new, modified, and removed files in the synchronization process.
   - **File Name Comparison**
     For each file in the Source directory, the tool checks if a file with the same name exists in the Replica directory. If the file is not present in the Replica, it is considered new and needs to be copied.
   - **MD5 Hash Comparison**
     If a file with the same name exists in both the Source and Replica directories, the tool calculates the MD5 hash of both files. The MD5 hash acts as a unique identifier for the file content. If the MD5 hash of the file in the Source directory differs from the MD5 hash of the corresponding file in the Replica directory, it indicates that the file content has been modified. The tool then proceeds to copy the updated file from the Source to the Replica.
4. **Copying New and Modified Files**
   After identifying the new and modified files, the tool performs the actual copying of

these files from the Source directory to the Replica directory, ensuring an up-to-date and identical copy of the Source.

5. **Removing Files from Replica**
   To maintain the full synchronization between the Source and Replica directories, the tool also checks for files that exist in the Replica but are no longer present in the Source. If any such files are found, they are removed from the Replica directory to match the Source's content.

6. **Logging Actions**
   The Synchronization Tool incorporates a logging mechanism to keep a record of the various actions performed during the synchronization process. This log file serves as a valuable resource for users and developers to track the tool's behavior and review the synchronization activities.

   - **Log File Location**
     By default, the log file is saved in the root directory of the project. Users can also provide a custom path using the -logf command-line parameter while executing the script.
   - **Logged Actions:** The following actions are logged in the file
     - **File Creation:**
       Each time a new file is copied from the Source directory to the Replica directory, the log records the action along with the file name and the destination path in the Replica directory.
     - **File Update:**
       When an existing file in the Source directory has been modified, the tool detects the change and updates the corresponding file in the Replica directory. The log file captures this action, including the file name and the updated content.
     - **File Deletion:**
       If a file is deleted from the Source directory, the tool recognizes the absence of the file during synchronization and removes the corresponding file from the Replica directory. The log file registers this action, indicating the file name and the removal.
     - **Empty Folder Creation/Deletion:**
       The tool also logs the creation or deletion of empty folders in the Replica directory that mirror the Source directory's empty folder structure.

- **Log Formatting:** The log file follows a structured format with timestamps for each action, making it easy to track when each operation occurred.

6 Testing

I tested the program with various scenarios to ensure its robustness and reliability. By covering different test cases, you can identify potential issues and make necessary improvements. Below, I've summarized the test scenarios along with their outcomes:

1. **Replica Directory Doesn't Exist Initially**
   Scenario: When the Replica directory does not exist before starting the synchronization.
   Expected Outcome: The program should create the Replica directory and copy all files and folders from the Source directory to the Replica directory.
   Actual Outcome: As expected.

2. **Source Directory is Empty**
   Scenario: When the Source directory is empty with no files or subdirectories.
   Expected Outcome: The program should just create an empty directory at the location of Replica directory.
   Actual Outcome: As expected.

3. **File Deletion in Source Directory**
   Scenario: When a file is deleted from the Source directory.
   Expected Outcome: After synchronization, the same file in the Replica directory should be deleted to match the Source's content.
   Actual Outcome: As expected.

4. **File Update in Source Directory**
   Scenario: When a file in the Source directory is updated with new content.
   Expected Outcome: After synchronization, the same file in the Replica directory should be updated to match the updated content in the Source.
   Actual Outcome: As expected.

5. **Empty Folder Addition in Source Directory**
   Scenario: When an empty folder is added to the Source directory
   Expected Outcome: After synchronization, the same empty folder should be added to the Replica directory to mirror the Source.
   Actual Outcome: As expected.

6. **Manual File Deletion in Replica Directory**
   Scenario: When a file is manually deleted from the Replica directory.
   Expected Outcome: After synchronization, the same file should be copied again from the Source directory to the Replica directory.
   Actual Outcome: As expected.

7. **Manual File Addition in Replica Directory**
   Scenario: When a file is manually created in the Replica directory.
   Expected Outcome: After synchronization, the manually created file should be deleted from the Replica directory if it does not exist in the Source.
   Actual Outcome: As expected.

8. **Manual Empty Folder Addition in Replica Directory**
   Scenario: When an empty folder is manually created in the Replica directory.
   Expected Outcome: After synchronization, the manually created empty folder should be deleted from the Replica directory if it does not exist in the Source.
   Actual Outcome: As expected.

9. **Only source folder location is provided as a parameter**
   Scenario: User provides only the mandatory parameter while running the program.

   ```
   python3 main.py -src <path/to/source-dir>
   ```

   Expected Outcome:

   - After synchronization, a replica directory by the name of *replica_dir* should be created at the root of the project.
   - After synchronization, a logfile by the name of *logfile.log* should be created at the root of the project.
   - Folders should be synchronised every 30 minutes.

   Actual Outcome: As expected.

10. **A non numeric value is provided as sync interval**
    Scenario: When a non numeric value is provided for the sync interval parameter (-sync).
    Expected Outcome: Error should be thrown informing the user.
    Actual Outcome: As expected.

11. **A non existent Source folder is provided to the tool.**
    Scenario: When the user passes the path of a Source folder that doesnt exist.

Expected Outcome: Error should be thrown informing the user.
Actual Outcome: As expected.

7 Conclusion

The use of dictionaries to store file names and their corresponding MD5 hashes enhances the efficiency and accuracy of content comparison during the synchronization process. By following the methodology, the Synchronization Tool ensures a reliable and consistent replication of the Source directory to the Replica directory. The tool's simplicity in usage, content comparison, and file management make it a valuable addition to any data management workflow, especially for data backup, replication, and redundancy purposes. The logging feature of the Synchronization Tool contributes to its transparency and traceability, allowing users and developers to monitor and assess its behavior during synchronization. The log file provides an audit trail of the tool's operations, making it a valuable tool for tracking changes and ensuring data integrity during the synchronization process.