Sync Folders - Veeam Test Task

Victor Oliveira

December 2023

1 Explanations and Insights of the Code

In this small report, I will present a brief explanation of the developed code and other possibilities that could be employed in the code. The script is written in *Python*, and the OS used in this test was Linux (Ubuntu 20.04), but it should also work in Windows.

1.1 Parsing Command Line Arguments

To parse the command line arguments, I decided to use the *argparse* library. With this library, we can specify in the command line which arguments we are passing. Besides that we can use the -h or –help complement to understand the correct structure of the command line. The arguments that should be passed are the source path, replica path, log file path, and the period of synchronization. In our case, an example of command line is the following:

```
python3 folder_sync.py -s "/home/victor/Documentos/Veeam Software/source"
-r "/home/victor/Documentos/Veeam Software/replica" -l "/home/victor/
Documentos/Veeam Software/log_file" -p 10
```

1.2 Period of Synchronization

The synchronization should be performed periodically, and in order to do that, I decided to use the "time.sleep(sync_period)" function. There are other ways to do this, which are a bit more complex and fancier. Nevertheless, for this task specifically, this method employed is enough.

1.3 Logging Operations

For every file created, deleted, or updated, a message is printed in the console output and logged in a predefined file. The information logged are: name of the file, operation performed, folder where it happened, and time. Right now, the log happens in the format of phrases, but we could log info with the help of a table - simplifying the visualization and the data handling. In the end, it is just a matter of preference.

1.4 Synchronizing the Folders

To compare both folders, I decided to use the "filecmp.dircmp(source, replica)" function. With this function, we can get: the files and folders that are only in the source folder; the files and folders that are only in the replica folder; the files and folders that are equal; and the files that have the same name but different content.

Thus, if the file or directory is only in the source folder, we copy it to the replica folder. Here, we could have used the "shutil.copytree(dir1, dir2)" function to copy a directory. However, with this approach, we wouldn't be able to log the creation of files. So, in order to log every file creation, I decided to implement a function that would copy the directory from source to replica.

The same happens for the directories that should be deleted (if they only exist in the replica folder). In such case, instead of using the "shutil.rmtree(dir)" to delete the directory at once, I implemented another function to remove the given directory, logging all the files removed in this process.

If the file in the replica folder is outdated, we just delete the old version and copy the new version. Finally, for the common directories, we explore them recursively until we reach the bottom.