
COMPSCI 340 S2 C 2015
SOFTENG 370 S2 C 2015
Operating Systems
Assignment 2 – File Synchronisation
7% of course mark
Due Friday 25th of September 9:30 pm

## Introduction

If you ever work on files on different machines you can easily end up with multiple copies of the same file in different locations which are inconsistent with each other. There are several types of solution to this problem. The simplest is by being very careful and copying all files backwards and forwards as they get modified. The widespread use of Dropbox and similar cloud solutions is another approach but this requires an internet connection. It is much safer if the operating system provides some assistance. Windows provides briefcases to do this sort of thing with removable media. There are some limitations in how you work with briefcases and in this assignment you will attempt to produce a different solution.

One problem with synchronising directories that have been modified on different machines is that the time information associated with a file, in particular the time the file was last modified, cannot be relied upon to determine if one version of a file is more recent than another. There are all sorts of ways in which the modified times can be incorrect.

In this assignment you are going to produce a file synchronisation service that doesn't depend on modification times, but does use the times to resolve some difficulties.

To do this you will need to become proficient in a number of different technologies, in particular file manipulation and setting file attributes, and the use of JSON and file digest creation.

## What you have to do

You have to write a program called `sync` that synchronises two directories (all the files contained within and under those directories). It should be run by typing:

```
./sync dir1 dir2
```

where `dir1` and `dir2` are the paths to the directories to synchronise, from the current directory.

After the program is run on the two directories the contents of the directories should be identical. (This is not quite true because the service will use some hidden files in the directories that will normally be slightly different. All unhidden files will be identical.)

The program must work by maintaining information about the files in each directory. The information must include an SHA-256 digest for each file along with the modified times that were valid when the digest was generated.

Using a unique digest based on the contents of the file means that we can verify that two versions of the file are the same, regardless of the times associated with the files. The digest and associated time information is to be stored in a hidden file inside each directory. These files will be referred to as sync (sounds like "sink") files in the rest of this handout. Sync files are always called "`.sync`". On Unix file systems, if a file name starts with a "`.`" it is normally hidden. (You can do this assignment on Windows if you want, but the sync files will not be hidden from the user, also the markers will be using Linux in the labs in order to test your program.)

After checking to see if the directories are valid (and possibly creating a directory if one doesn't exist) the program should scan the directories and either create the sync files if they don't exist or update them to reflect the current values of the files.

When both directories have up-to-date sync files (based on the current contents of each directory) the sync files should be used to merge the directories together. During the merge, the sync files will need to be altered to reflect the new state of the files in each directory. The merge may also require changes to the files in one or both directories.

## The form of the sync files

The sync files should maintain a history of changes to files. Every time the `sync` program is run on a directory it should examine the files in the directory and either update or verify the current information in the sync file.

Each ordinary file in a directory has an entry in the sync file. A digest is calculated for each file using the SHA-256 algorithm. This algorithm calculates a unique number for a file based on its contents. If a file differs from another version in any way the SHA-256 value will be different.

Here is the content of an example sync file:

```
{
  "file1_1.txt": [
    [
      "2015-08-26 12:03:44 +1200",
      "d6072668c069d40c27c3f982789b32e33f23575316ebbbc11359c49929ac8adc"
    ],
    [
      "2015-08-26 12:03:42 +1200",
      "a2ebea1d55e6059dfb7b8e8354e0233d501da9d968ad3686c49d6a443b9520a8"
    ]
  ],
  "file1_2.txt": [
    [
      "2015-08-26 12:03:42 +1200",
      "c62b8de531b861db068eac1129c2e3105ab337b225339420d2627c123c7eae04"
    ]
  ],
  "file2_1.txt": [
    [
      "2015-08-26 12:03:42 +1200",
      "3032e7474e22dd6f35c618045299165b0b42a9852576b7df52c1b22e3255b112"
    ]
  ]
}
```

The sync files must be in JSON (JavaScript Object Notation) format as in this example. Each file has a list of pairs recorded with its name. Each pair records the the modification time of the file and the SHA-256 digest (those hexadecimal strings) associated with it.

Almost all current programming languages have libraries which allow the generation of SHA-256 digests and writing and reading JSON files. (Before you choose a language to implement the assignment make sure it runs on Linux in the labs - Java, Python and Ruby are possibilities. You may need to add some extra classes or files to your submission so that your solution runs properly. This is your responsibility.)

If a file has more than one pair of records associated with it then the file has been in more than one state when the `sync` program has been run. The first pair of records for a file is the current pair. e.g. For `file1_1.txt` the current modified time is `"2015-08-26 12:03:44 +1200"` and the digest is `"d6072668c069d40c27c3f982789b32e33f23575316ebbbc11359c49929ac8adc"`.

If the digest already exists in the sync file as the current value, then the file has not been changed since the last synchronisation. In this case the modification dates should be compared - the current modification date and the one in the sync file. If the current one is different from the one in the file, it should be changed to the value stored in the sync file; different programming languages have different methods for this (the access time is not important for this assignment, only the modification time).

If a directory does not contain any synchronisation information about a file, or the digest is different from the most recent one in the sync file, a new pair `[modificationTime, digest]` should be added to that file's information.

The sync file maintains information about changes over time. No entries are ever removed from the sync file, new information is always added.

If a file is deleted from a directory it should still have an entry in the sync file. The entry should have a digest of “deleted” rather than an SHA-256 value. The current time can be used as the time value for this entry. A deleted file is only discovered if there was an existing entry in the sync file for the file.

## Merging directories

After the sync files for each directory have been updated, the two sync files are compared and changes must be made to synchronise the directories.

If a file has the same current digest in both directories but different modification dates, the earliest modification date (from one of the sync files) is applied to both versions of the file, and the sync file entry is updated to reflect this.

If a file has different digests in both directories then the files are different. This must be handled in a number of ways. If the current digest of one of the versions is the same as an earlier digest in the other version then this version has been superseded (see the assumptions). The older version of the file needs to be replaced by the more recent version and the sync file entry updated. The copied version of the file must be given the modification time specified in the updated sync file.

If both digests are unique (this can happen legitimately when the program is first run) then we have two different possible versions of the file. Several different solutions can be used for this situation, ranging from trying to reconcile the differences in the files, to asking the user for advice. In this assignment you should use the modified times to identify the most recent version of the file in this situation. Not ideal!

## Dealing with deletions

When a file is deleted in one directory but not in the other it has to be deleted in both. As a deletion is only detected when the sync program is run, it is possible that the file has been deleted in one directory and updated in the other. In this case the deletion wins. That is, the file is deleted in both directories.

An interesting case occurs when a file is deleted in both directories, and some time after synchronising the directories, a new copy of the file is created in one of them. In this case you must make sure that the newly created file is copied to the other directory on synchronisation, not deleted. You can see an example of this below. The `file1_2.txt` file has been deleted at one synchronisation and then recreated before a following synchronisation. The matching synchronisation file from the other directory looked like this before the last synchronisation:

```
{
  "file1_2.txt": [
    [
      "2015-08-26 16:53:53 +1200",
      "deleted"
    ],
    [
      "2015-08-26 16:52:56 +1200",
      "a8299b688d515f86123fcb56b797f9a10988d5c359f3b99734ca43f67913887d"
    ]
  ]
}
```

Now a file with the same name has been created again.

```
{
 "file1_2.txt": [
    [
      "2015-08-26 16:58:27 +1200",
      "48a402c3603731924b066619095f248a53e72cdd121c271fe63759ea6695f976"
    ],
    [
      "2015-08-26 16:53:53 +1200",
      "deleted"
    ],
    [
      "2015-08-26 16:52:56 +1200",
      "a8299b688d515f86123fcb56b797f9a10988d5c359f3b99734ca43f67913887d"
    ]
  ]
}
```

### Useful hints

Sync files only maintain information on the files in the associated directory, they do not maintain information on subdirectories. Each subdirectory has its own sync file. If synchronised directories include subdirectories, then the subdirectories must be synchronised as well.

Don't copy files unnecessarily. If the same file is in both directories then don't do a copy. You still need to ensure that the most recently modified times are correct (both in the sync files and in the file's attributes).

### Assumptions

If two files have the same signature, they are the same version. (In reality this may not be the case because the file could have been altered and then changed back to its original state.)

All permissions needed to create files and directories will be available, so you don't have to worry about file creation errors.

The pathnames sent to the sync program will be independent. You don't have to worry about infinite recursion because of one directory being inside the other.

No ordinary files in one directory will match the names of directories in the other directory. No symbolic links will create cycles in the directories. You don't have to deal with deleted directories, but you do have to deal with deleted files. No sync files will be deleted or modified by the test programs.

The file modification times are not reliable.

### Language

You may use any programming language you like (as long as it can be run on Linux in the labs by the markers). It is your responsibility to provide a simple makefile or shell script which will produce a runnable version of your program when the marker runs it. In particular this makefile might set your program to be executable and rename it to be "`sync`" so that it runs by typing "`./sync dir1 dir2`".

e.g. If you wrote your program in Python the makefile might contain:

```
run:
	cp sync.py sync
	chmod +x sync
```

In this case the `sync.py` file would have to include "`#!/usr/bin/env python3`" as its first line to let Linux know which interpreter to use.

You may give the marker a single line of instructions to run this makefile or script so that it completely produces a runnable version of your `sync` program in the current directory.

## Program Mark allocation (20 marks)

If the names passed on the command line are not directories, displays an error message and stops, (but see the next point).

`2 marks`

If one directory does exist and the other directory does not exist (and it is not the name of an ordinary file) then the non-existing directory is created and the new directory synchronised with the existing one.

`2 marks`

Sync files are created if they do not exist.

`1 mark`

Sync files are updated after modifications.

`1 mark`

Correctly synchronises two completely different directories (without subdirectories). (Originally, no matching contents.)

`2 marks`

Correctly synchronises two directories with some overlapping content (without subdirectories). Some files occur in both before the first synchronisation is performed.

2 marks

Correctly synchronises two directories after the contents of one of the directories is changed (without subdirectories).

3 marks

Correctly deals with deletions (without subdirectories).

3 marks

Correctly does all the above with subdirectories.

4 marks

## Questions (10 marks)

Answer the following questions. Put the answers in a simple text file called `a2Answers.txt`.

1. How well would this synchronisation implementation cope with a large number of files? Explain your answer.

   2 marks

2. Several of the assumptions cannot be enforced. We rely on the user being careful. This is partly because of the limitations of most current file systems. Take two of the assumptions above (you may also specify other assumptions that I haven't explicitly mentioned), and describe ways in which the assumptions can be made unnecessary. Your solutions should make the synchronization process safer or more reliable. At least one of the solutions should mention some extra facility that must be added to the file system to make the solution possible.

   8 marks

## Submitting the assignment

**Make sure your name and upi is included in every file you submit.**

Use the assignment drop box to submit. adb.auckland.ac.nz

Any work you submit must be your work and your work alone – see the information on academic integrity http://www.auckland.ac.nz/uoa/home/about/teaching-learning/honesty.