Working Towards Preservation in a Non-Preservation Environment: Identifying, Replicating, and Restructuring File Organization

Winnie Schwaid-Lindner, Louisiana State University

Abstract

When creating digital files and organizing them into folders, it is both human nature and best practice to choose a structure that feels intuitive, so that files can be added, removed, or modified easily. Although this is great for access, this flexibility makes the structure unsuitable for preservation. This is especially a challenge when the objects that are added, removed, or modified are preservation files. This article is a case study and tutorial for how to identify, replicate, and restructure folders for preservation, using only features that your computer already has. Although rooted in digital preservation and featuring an included technical solution, the intended audience of this case study is librarianship in the broadest sense. This article was written so that anyone, regardless of their digital stewardship experience, technical abilities, or type of institution, will be able to follow and replicate the process themselves.

Keywords: digital stewardship; digital preservation; file structure; digital storage; file organization; case study

The stewardship of digital materials entails managing digital objects through different stages of their content lifecycle to ensure their long-term preservation and access (McCurry, 2014). The many lifecycle stages represent categories of stewardship actions that vary in depth and frequency. Some are carried out occasionally, such as migrating data to a different format, some on a continuous basis, such as planning for preservation, and some in a sequential order, such as the "ingest" and "preservation actions" sections (Higgins, 2008). This paper focuses on the intersection of the "preservation action" stage in relation to "ingest", when digital objects have been received and transferred into a system, yet require additional attention as part of preparation for long-term preservation. Furthermore, I hope that disseminating this paper will aid others throughout their own "preservation planning" stage, which entails managing the processes relating to preserving digital objects and continuously reconsidering the capabilities of those processes.

At Louisiana State University (LSU) Libraries and many other institutions, large and small, the lifecycle actions are often applied to multiple digital objects at a time. In this approach, actions geared towards the processing or long-term preservation of digital objects are applied to groups of digital objects related to each other by their content. These groups can vary in size from expansive, consisting of thousands of the digital objects in a collection, to fewer than ten digital objects that are grouped together according to the extent of their original physical material. Regardless of what the grouping itself looks like or which aspect of the content defined the group, it is maintained in the form of a digital file folder, and the objects within that folder progress through lifecycle stages with each other.

Basing the file storage environment around these content-grouped files and folders is practical for certain aspects of the digital object lifecycle. During the ingestion stage of the

lifecycle model, when digital objects are received, created, or otherwise added to a digital collection, the file storage environment (and the folders within) are built to be intuitive to use, including folders describing the status or desired use of objects, such as how one may have a "favorites" section within a larger "funny cat videos" folder so that it's easy to find and access the desired content quickly and easily. Additionally, the environment is built to be flexible, so that new files or folders can be easily added or moved. However, these same characteristics that make the environment good for the ingest and processing-based actions make it undesirable for certain long-term preservation conventions, which necessitate a static, regimented storage environment, such as BagIt, a file packaging and structure specification designed for the transfer and storage of digital objects (Kunze et al., 2018).

So then, how does one navigate the overlap, to perform long-term preservation actions for an organizational environment built to change, for a folder based around aspects of the content that is consistently ingesting?

There is an abundance of literature on how to create a file structure and organization for access and usability as part of data management best practices (Data Management Best Practices — Documentation, 2017), as well as packaging specifications, such as the aforementioned BagIt which produces packaged "bags" or the Oxford Common File Layout, which are considered best practice for long-term object stewardship. The overlap on a small scale has received some attention, such as Wellcome Trust's solution to storing multiple versions of the same bag if the original is modified (Chan, 2020), but even this approach expects that the digital objects are already structured in a bag by preservation workflow systems, and although it can accommodate occasional changes, this method is not reasonable for weekly additions of preservation-quality files to hundreds of folders. Therefore, there is a gap in the literature on how to address the

intersection for a large collection experiencing large-scale changes before reaching long-term preservation workflow systems, as this case study's collection requires.

This article describes a case study outlining my solution to this predicament for the Louisiana Newspapers collection at LSU Libraries, which has been ingesting digital objects for nearly twenty years, and will continue to for the foreseeable future. In addition, it includes the code written as part of this case study, so that others may easily replicate the approach for their own collections.

The Dilemma

The current folder groupings for Louisiana Newspapers exist to enable the folders and their contents to change frequently, but to perform the long-term preservation actions needed for our digital objects, we need the folder (and storage environment) to be static rather than the intuitive and flexible nature of the current system. Currently it is not, and couldn't be made to be more static without detrimentally affecting other lifecycle actions, like access (being able to find what you're looking for, like one's aforementioned favorite funny cat video). The grouping system, as it currently stands, is unsuitable for a static approach.

Parameters

When considering potential solutions to this dilemma, I identified requirements that the revised file/folder grouping strategy must fulfill. Many of these can apply to any institution or person in a similar predicament, while some are additional considerations that are specific to LSU Libraries.

The broad specifications are that the solution must:

- *Be static.* The file folders (often referred to as "directories") and their contents can't change over time. Having static folders allows the system to be usable with BagIt. For

those without a BagIt requirement, it also enables less processing-intensive and energy-intensive methods to validate the directory as a whole, and to use (lossless) folder compression.

- *Not alter the source folder/directory.* The context or structure of the digital objects in the original directory should never be altered.
- *Be space conscious.* We shouldn't use more additional storage than we need to at any given time, even if we're using it only temporarily.
- *Preserve all file metadata.* All data about the digital object, such as file attributes and date information, should be retained.

I also considered the storage system and the specific collection that were the subject of this case study. The digital collection at hand, Louisiana Newspapers, currently consists of 2,310,424 files across 104,226 folders/directories, and takes up roughly 9.96 terabytes of storage space. The majority of those files are XMLs (1,100,766 files), PDFs (1,079,311 files), and to a lesser extent, TIFs (61,094 files). Additionally, there are also several thousand raw canon camera files, adobe sidecar files (for use with photoshop), JPEGs, and JPEG2000 files.

The storage system structure is primarily hierarchical based on newspaper title (for a fictional and simplified example, newspaper titles *The Louisiana Newspapers News* or *The Louisiana Journals Journal*), with subfolders for date or newspaper issue arrangement (such as the equally fictional "January 2005" or "feb-2006"). Additional higher-level folders indicate the category or status of the newspaper title, such as "Current" for newspapers that are actively producing issues, and "Ready for the LDL" for newspaper titles that are ready for ingest into the Louisiana Digital Library (LDL).

A visual representation of this simplified fictional example:

```
-- Newspapers\
   |-- Current\
      |-- Louisiana Journals Journal\
         |-- feb-2006\
            |-- feb-2006_01-a.tif
            |-- feb-2006_01-b.tif
      |-- Ready for the LDL\
         |-- Louisiana Newspapers News\
            |-- January 2005\
               |-- 2005-01-a.tif
               |-- 2005-01-b.tif
```

These category/status folders are undeniably advantageous for navigating the storage space and categorizing aspects relevant for other actions (like interacting with the digital library), but are not relevant for long-term preservation of the digital objects. Additionally, if *The Louisiana Journals Journal* ceased publication, that folder in its entirety would be moved out of "Current" and into “Historical”, so even though the contents of the *Louisiana Journals Journal* folder would remain the same, the exact location and filepaths would change. Therefore, the static version of the folder hierarchy must be restructured to exclude these higher intermediate levels, moving all the newspaper title folders to the same level. In order to accomplish this, the restructured organization would call for the *Louisiana Journals Journal* folder to be one level higher (as it excludes one intermediate folder, "Current"), while the *Louisiana Newspapers News* folder would be two levels higher (excluding two intermediate folders, "Current" and "Ready for the LDL").

Using the same fictional abridged example, the visualization of the updated hierarchy would be:

```
-- Newspapers\
   |-- Louisiana Journals Journal\
      |-- feb-2006\
         |-- feb-2006_01-a.tif
         |-- feb-2006_01-b.tif
   |-- Louisiana Newspapers News\
      |-- January 2005\
         |-- 2005-01-a.tif
         |-- 2005-01-b.tif
```

In addition, the resulting strategy must be compatible with our existing system, in that it must work with our LOCKSS network's requirements, which requires the use of BagIt (Johnson, 2021), and be managed from a local computer running either Windows 10 Pro or Ubuntu (Linux) with 8 GB of RAM and 460 GB of total disk space. In addition, all bespoke computer automation must be written in python, due to its ability to handle these kinds of tasks, as well as the author's specific expertise using python compared to other programming languages. Additionally, it must work with the LSU Libraries network storage drives and use no more than 1TB of storage at a time.

Guiding Questions and Resulting Strategy

Based on the dilemma and parameters, we identified four related conceptual and logistic questions, the answers to which outline the strategy.

1. How do we organize or group the files so that the folders will be static?

Grouping the files into static folders according to content attributes (such as alphabetically or by subject, topic, or publication date) can't be done, as the number of items with each attribute can grow over time. Even publication dates, which initially seem static, are flexible within the digital collection as a whole. Although newspapers will not publish new issues for past dates (as time traveling is not yet possible), LSU Libraries still digitizes newspapers from past years, leading the number of files with any given publication year to grow

over time.

Therefore, any grouping scheme must be related to the progression of time, as well as dependent on an aspect of the file that is static. The resulting solution is the "last modified" time and date of each file. For the number of files and storage space we have, grouping annually by the year last modified is practical. This additionally ensures that files which are updated over time will always have the most recent version included in that year's batch. In short, long-term preservation of digital newspapers in our example relies not on the newspaper issue's location in the library's hierarchical digital filing structure, nor on its date of publication, but rather on the file's last date of modification.

For a fictional example, let's say a *Louisiana Newspapers News*' metadata file was created on December 31st, 2022 but was left blank until information was added in February 2023, while I had generated the 2022 grouping of files in early January 2023. If we were to group files based on their file creation date (instead of last modified date), the version of that metadata file uploaded into preservation storage would be how it appeared in January 2023 - entirely blank. Furthermore, because the file was created in 2022, the 2023 batch of files made the following year would not include this file either. This results in the designated long-term preservation copy of this file devoid of all the information that warranted undergoing preservation in the first place.

However, taking the same situation but using a file's last modified date instead of its creation date, the empty version of the file as it appeared in January 2023 would still be saved in January 2023, but a completed version of the file would also be saved as part of the 2023 batch which would be generated in 2024.

2. What is the structure and file composition of these folders?

Since we've now identified the criteria by which to group the files and folders, the next question is naturally how to identify the folder structure, which also entails determining which files are allocated to which folder. As the grouping criteria are based on an individual file's last modified year, the method for sorting files into folders must similarly be on a file-by-file basis.

These groups should use a file's last modified year as the primary sorting criterium, and then should contain subfolders for the newspaper titles that have files in that last modified year. Newspapers that don't have any files last modified in that year will not be represented, and status subfolders, such as "Current" or "Ready for the LDL" will be excluded, bringing all of the newspaper title folders to the same hierarchical level.

Using the same fictional example, the visual representation:

```
-- 2005\
   |-- Louisiana Newspapers News\
      |-- January 2005\
         |-- 2005-01-a.tif
         |-- 2005-01-b.tif
-- 2006\
   |-- Louisiana Journals Journal\
      |-- feb-2006\
         |-- feb-2006_01-a.tif
         |-- feb-2006_01-b.tif
```

As the folders are based on the last modified date of each individual file, the number of files in each group will be inconsistent and specific to that year. For example, at the time of writing this, the Louisiana Newspapers had some years with fewer than 1,000 files, while recent years (below the dotted line) have upwards of 20,000 files, and the year 2022 dwarfs all previous years with more than 2 million files (due to LSU Libraries' partnership with newspapers.com):

Last Modified Year	*Number of Files*
2005	106
2006	583
2007	643
2019	33,764
2020	20,131
2021	23,948
2022	2,143,051

3. How do we actually create these batches?

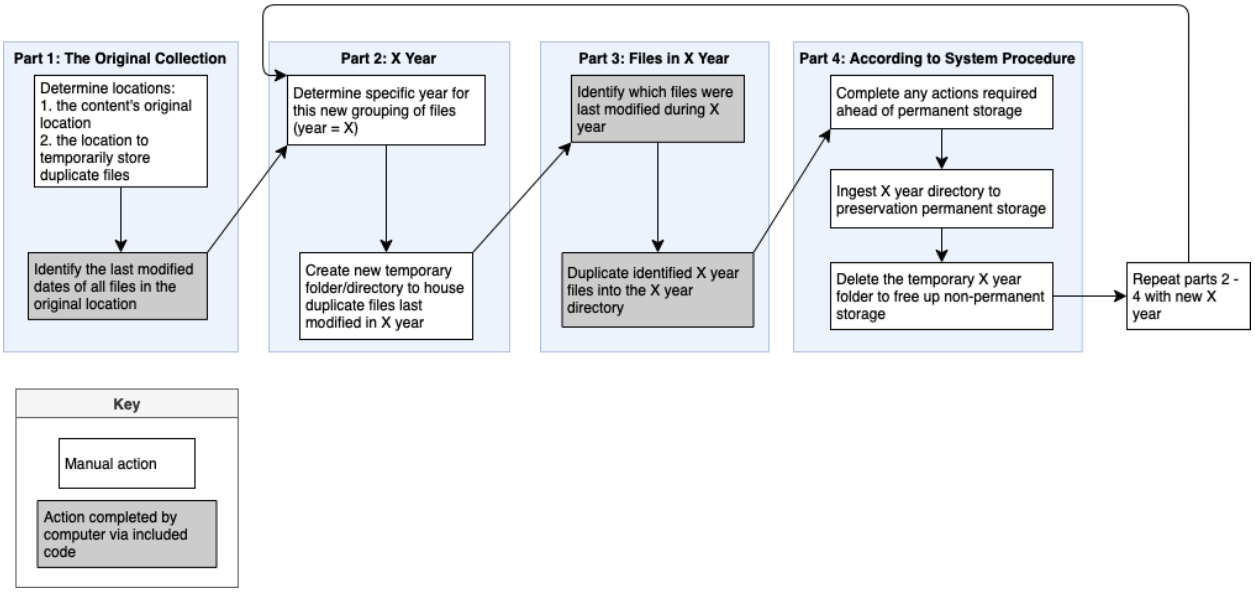
Now that we know what each folder is, as well as its respective contents, we must determine the method to realize this new structure. Since the guiding parameters require that we not alter the existing digital newspaper filing structure and that we be space conscious, it's logical that we must form these folders/groups by creating duplicates of each original file, one at a time, instead of by moving the originals, and that we must create only one group at a time. After an annual group has been constructed, undergone the necessary processing, and been uploaded to our preservation storage, the duplicated folder will be deleted to free up storage to create the next batch.

The need to retain all file metadata narrows down the field of options for duplicating files, although the remaining options each come with their own challenges. For Mac and Linux computers there's rsync, which has a built-in function for duplicating a list of specific files, as our strategy requires. Although rsync may be the perfect solution for other institutions and collections, it was unfeasible for the Louisiana Newspapers. Rsync requires that either the source file or file destination be your computer's local storage, whereas both the source and destination for the Louisiana Newspapers are the LSU Libraries' networked storage. Local storage is not a viable option, simply because of the amount of storage required for the size of the collection. The Windows parallel of rsync is robocopy, which is capable of transferring both to and from remote locations, but is designed to transfer an entire folder at a time and doesn't have a built-in option

to transfer a series of individual files across several different folders. However, with the options that robocopy does have, this proved an easier obstacle to clear than rsync's respective challenges.

Implementation

General Workflow

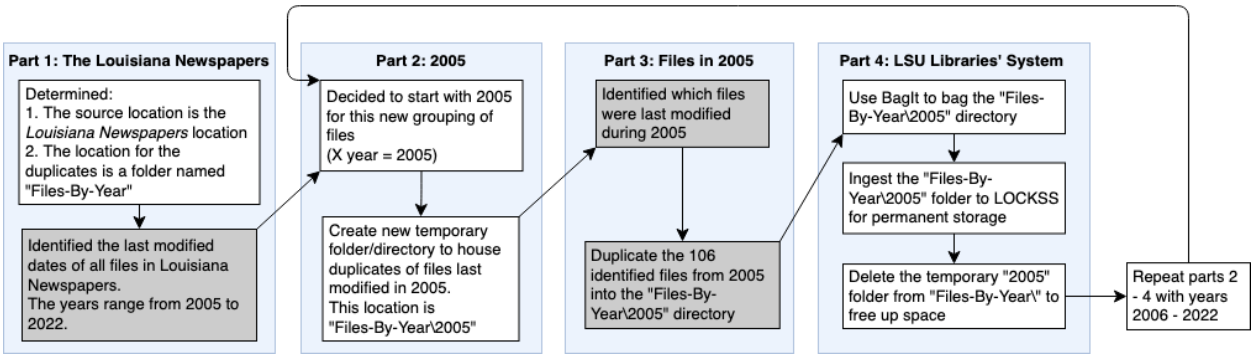


The general workflow consists of four parts, comprising both manual actions (in white) that are completed manually by the person overseeing this process, such as making decisions or indicating folder locations, and actions completed by the computer using code I've written in python. The included appendix provides access to these scripts as well as clear usage instructions for readers' direct use and is summarized and explained in the following subsections.

Part 1 is based on the source collection or highest level of the new file organization hierarchy, and the results of Part 1 determine all iterations of Parts 2, 3, and 4. The second part is based on any given year, indicated here as *X year*. This can be any year that appears in the "Identify the last modified dates" step of Part 1. Part 3 narrows further and pertains to the specific files that were last modified in *X year*, while Part 4 is based on policies and procedures

that extend beyond the source collection and apply to your entire system. Upon concluding Part 4, the workflow repeats for each remaining *X year*.

Case Study Specific Workflow



Part 1: The Louisiana Newspapers

The source location (the folder that holds the entirety of the Louisiana Newspapers files) is easily selected, and the destination location for the duplicates must meet the criteria of having enough space for the duplicates of any year, as well as being a location that I am able to read, write, and remove files from. For LSU Libraries, this location is on networked storage, the "P Drive" and is given the fictional name "Files-By-Year". When looking at the full location, this will appear as "P:\Files-By-Year"

Identifying the last modified dates of all the files is accomplished programmatically. A script I've written will:

1. Look through every file and folder in the source directory (the Louisiana Newspapers) and record the location (filepath) of every individual file
2. Find the last modified year for each of the files, which can take a while. If the computer encounters an error, it will try again. If the second attempt fails as well, the script will record the last modified year for that file as "9000", so it's easy to identify all of the failed files, and then continue on to the next file.

3. Export a list of all of the files and their respective last modified years to a CSV spreadsheet file on my desktop, so the computer doesn't need to recalculate all of the last modified years again.
4. Count how many files there are with any given last modified year and will display the final totals in chronological order as well as the grand total.

Part 2: 2005

As can be seen in the above output, the first chronological year of the Louisiana Newspapers files' last modified dates is 2005, so this is the year I have selected for the first grouping. Now that the year has been determined, I created a new folder in the duplicates destination folder ("Files-By-Year") for the batch of 2005 files. The full path of this directory is "P:\Files-By-Year\2005".

Part 3: Files in 2005

At this point I needed to determine which specific 106 files out of the 2,310,424 total files were the ones that were last modified in 2005. In order to do this, I wrote another script, which:

1. Opens the CSV file from Part 1 that contains each Louisiana Newspapers file and its respective last modified year
2. Looks line-by-line at each last modified year until it finds the year 2005 listed for a file, and then records that filepath
3. Saves this list of the path from each file last modified in 2005 to a TXT file on my desktop.

I've now identified all of the 2005 files that require duplication and can proceed to the duplication process. As noted in the "Guiding Questions and Resulting Strategy" section above,

robocopy is the method selected for duplicating the identified files; however, it's designed for copying entire folders at a time and doesn't have a method to transfer a list of specific files across different folders. Additionally, robocopy works by replicating the source folder/directory structure in the duplicated location, which is beneficial for ensuring that the hierarchy within the individual newspaper titles folders is maintained but is undesirable in our situation because this means it will also replicate the status and category folders, like "Current" or "Ready for the LDL". Therefore, the script I wrote is designed to go file-by-file through the TXT file output from the previous step and alter the destination of each duplicated file to remove those directories from the filepath.

The resulting script will:

1. Open the saved TXT file of 2005 filepaths
2. Look through each filepath listed, one at a time, and determine that file's:
    a. Specific filename (such as "2005-01-a.tif")
    b. Source directory path where the file is located ("P:\Newspapers\Current\Ready for the LDL\Louisiana Newspapers News\January 2005")
    c. Destination directory path, which excludes any of the status folders ("P:\File-By-File\2005\Louisiana Newspapers News\January 2005")
3. Uses robocopy to duplicate that file into the revised destination directory path, using the following robocopy options:
    a. Maintaining the file's data, attributes, and timestamps
    b. Using multithreading to speed up processing time
    c. If it encounters an error, waiting one second and then retrying the duplication one more time

   d. Giving verbose output, so that we can examine the output if there are any errors, and recording the full output to a logfile

4. Additionally, the script I've written will identify duplications that result in errors and record the text of the error to an additional TXT file on my desktop recording only file error output.

At the conclusion of this script, all 106 files have been duplicated successfully! However, it is worth noting that it's during this last script - the duplication process - when bottlenecks are most likely to occur. Depending on processing speed, the number of files being copied, and the size of those files, this process can vary from a few minutes for a few PDFs, to several days for many preservation-quality image files. It also uses a lot of the workstation's processing power, so that the computer will run more slowly than normal. If the bottleneck is significant, I'd recommend either starting the script right before a long weekend, or if the matter is time sensitive, altering the batch sizes to a smaller increment of time, such as bi-yearly or monthly. The increase in number of temporal groups will take more human processing time to iterate through a greater number of batches but will reduce processing time for your computer.

Part 4: LSU Libraries' System

At this point, the actions are determined by our preservation procedures more generally, and include using BagIt to bag the 2005 directory, and then upload it to our LOCKSS network for permanent storage. Once uploaded to the LOCKSS network, we are confident that the 2005 grouping of these files has successfully reached our preservation storage, and therefore I can delete the local copy of the 2005 folder, so that I can proceed to restarting the process with the 2006 files.

Conclusion

The solution outlined in this case study has helpful implications, not only for similar cases at LSU Libraries, but for any circumstances where there are preservation quality files stored in a non-preservation environment, such as born-digital accessions, active media production environments, or even simply DIY-minded individuals creating yearly backups of their own personal files.

The overall strategy and approach outlined is specific in required parameters but meets them from a sweeping approach, so that the exact solution and its scalability can be tweaked based on any institution's system and processing power, while the exemplar case study and included code aimed to implement the strategy in a manner that is applicable and amenable for small- to medium-sized institutions.

References

Chan, A. (2020, February 11). *How we store multiple versions of BagIt bags*. Medium. https://stacks.wellcomecollection.org/how-we-store-multiple-versions-of-bagit-bags-e68499815184

Data Management Best Practices—Documentation. (2017). [Documentation]. Axion Data Science. https://www.axiomdatascience.com/best-practices/DataManagementBestPractices.html

Higgins, S. (2008). The DCC curation lifecycle model. *International Journal of Digital Curation*, *3*(1), 134–140. https://doi.org/10.2218/ijdc.v3i1.48

JasonGerend. (2022, August 9). *Robocopy*. https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/robocopy

Johnson, C. (2021, February 19). *HOWTO: Package files for staging on the Drop Server—Adpnwiki*. http://www.adpn.org/wiki/HOWTO:_Package_files_for_staging_on_the_Drop_Server#Third.2C_Bag_the_Folder=

Kunze, J. A., Littman, J., Madden, L., Scancella, J., & Adams, C. (2018). *The BagIt File Packaging Format (V1.0)* (Request for Comments RFC 8493). Internet Engineering Task Force. https://doi.org/10.17487/RFC8493

Louisiana Newspapers | LSU Libraries. (n.d.). LSU Libraries Special Collections. https://lib.lsu.edu/special/CC/louisiana-newspapers/familytree/all

McCurry, J. (2014, April 2). *Digital Stewardship: The one with all the definitions*. The Collation. https://collation.folger.edu/2014/04/digital-stewardship-the-one-with-all-the-definitions/

OCFL Specifications. (n.d.). Oxford Common File Layout. https://ocfl.io/

Rsync(1) [redhat man page]. (n.d.). https://www.unix.com/man-page/redhat/1/rsync/

Appendix

Requirements

To access and use the supplied scripts, you will need to have:

1. a Windows computer

2. a connection to the internet

3. either: have permission to download an application from the Microsoft Store, have python already installed, or have someone who can install python for you.

Accessing the Scripts

Once you meet the above requirements, click this link to go to my project page on GitHub (github.com/wn-ie/irrfileorg), and then follow the simple instructions presented there. The reason for providing access this way (through GitHub) is so I'm able to make updates to both the instructions and the code, ensuring that you will always have the most up to date and easy to use version.