
Metadata Analysis at the Command-Line

Over the past few years the University of North Texas Libraries' Digital Projects Unit (DPU) has developed a set of metadata analysis tools, processes, and methodologies aimed at helping to focus limited quality control resources on the areas of the collection where they might have the most benefit. The key to this work lies in its simplicity: records harvested from OAI-PMH-enabled digital repositories are transformed into a format that makes them easily parsable using traditional Unix/Linux-based command-line tools. This article describes the overall methodology, introduces two simple open-source tools developed to help with the aforementioned harvesting and breaking, and provides example commands to demonstrate some common metadata analysis requests. All software tools described in the article are available with an open-source license via the author's GitHub account.

by Mark Phillips

Introduction

The UNT Libraries' Digital Libraries Division is responsible for the creation and quality review of the majority of metadata records in the UNT Libraries' Digital Collections. These collections contain items of similar format to other university library collections of comparable size. Items in the collections include digitized and born-digital photographs, letters, documents, maps, ledgers, technical reports, and theses and dissertations. The size and scope of these collections continue to grow at an increasing rate for the past three years measuring 83,000, 93,000, and 120,000 items added per year for the fiscal years 2010, 2011, and 2012. The continued growth in these collections means that there are a greater number of metadata records created by an increasing number of metadata creators, which in turn causes a wider variance in quality. The need to analyze and report statistics for these metadata records has led the UNT Libraries to develop new tools and processes to ensure that high quality metadata records are used throughout its digital library collections.

This article describes an approach in use at the UNT Libraries for harvesting metadata records from an OAI-PMH repository and then transforming them into a simpler text format, which can easily be consumed by a number of standard command-line tools available freely on most Unix and Linux based systems. Metadata quality, as defined for this article, falls into three major areas.

Collection Level Analytics – How many of something are in the entire collection of metadata records? For example, how many unique creators are represented by the collection? Which creator is associated with the most items? What item in the collection has the most creator, subject or coverage instances? These analytics are helpful in communicating metrics about the collection to others.

Metadata Completeness – How well does the collection's item-level metadata conform to various measures of completeness? What are required fields for a given subset of metadata, and how well do the collection's metadata records adhere to these requirements? How does this collection of metadata meet both metadata creator and metadata consumer ideals of value?

Metadata Value Quality – Based on local requirements, how well do the values within a given metadata record match those of standard or defined metadata specifications? For example, if a collection of metadata records utilizes the Extended Date Time Format for the date values in the collection, how well does the metadata collection meet the requirement of that format? Which values need to be changed in order to meet more of the requirements?

The tools and methodology explained in this article provide a way of identifying these areas of metadata cleanup to focus our attention, and help answer the question, "If there is a limited amount of time to spend on metadata cleanup, what is the best use of this time?"

Getting the data to the right tools

The command line in the Unix/Linux environment offers a number of tools which when used in different combinations provide a useful pipeline for manipulating data. The workflow described below makes use of many standard Unix/Linux tools:

- **sort** – sort each line of a text file
- **uniq** – report or omit repeated lines
- **wc** – count the number of words or lines
- **awk** – pattern-directed scanning and processing language
- **grep** – print lines matching a supplied pattern

These tools work by having the output (standard output or stdout) of one tool act as the input (standard input or stdin) of the next tool, and chaining these tools together by directing the output and input in a series of processes often referred to as a pipeline.

The tools generally operate on simple text formats. The most challenging aspect of working with these tools in a metadata analysis workflow is converting the metadata from a metadata repository into a format that can be easily consumed by the tools.

Many repositories have adopted the Open Archives Initiatives Protocol for Metadata Harvesting (OAI-PMH) [1] as a way to share metadata with others. At this time there are over 1,850 repositories worldwide that make metadata available with this protocol. The methodology described in this article makes use of OAI-PMH as the way to retrieve records from a repository.

A concise view of the methodology used by the UNT Libraries is to provide a straightforward workflow for harvesting records with a simple OAI-PMH harvester into an xml file referred to as a “repository file,” then using another small tool to convert this repository into a text format that is easily consumed by common command-line tools.

Once metadata records have been collected the next step is to convert from the format presented by the OAI-PMH repository into a format that is usable by the previously mentioned command-line tools. An example of the difference in representation can be outlined in the following two examples:

```
1  <record>
2    <header>
3      <identifier>info:ark/67531/metadc97952</identifier>
4      <datestamp>2012-08-17T12:16:00Z</datestamp>
5      <setSpec>partner:UNTCAS</setSpec>
6      <setSpec>collection:UNTSW</setSpec>
7      <setSpec>access_rights:public</setSpec>
8    </header>
9    <metadata>
10     <oai_dc:dc xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:oai_dc="http://www.oper
11       <dc:title>Fenología de Tayloria dubyi (Splachnaceae) en las turberas de la Reser
12       <dc:title>Phenology of Tayloria dubyi (Splachnaceae) in the peatlands of the Cap
13       <dc:title>Sub-Antarctic Biocultural Conservation Program</dc:title>
14       <dc:creator>Jofre, Jocelyn</dc:creator>
15       <dc:creator>Massardo, Francisca</dc:creator>
16       <dc:creator>Rozzi, Ricardo</dc:creator>
17       <dc:creator>Goffinet, Bernard</dc:creator>
18       <dc:creator>Marino, Paul</dc:creator>
19       <dc:creator>Raguso, Robert</dc:creator>
20       <dc:creator>Navarro, Nelso P.</dc:creator>
21       <dc:subject>bryophytes</dc:subject>
22       <dc:subject>Cape Horn Biosphere Reserve</dc:subject>
23       <dc:subject>phenology reproduction</dc:subject>
24       <dc:subject>Splachnaceae</dc:subject>
25       <dc:subject>sub-Antarctic Magellanic ecoregion</dc:subject>
26
27
```

```

28 <dc:description>This article discusses the phenology of Tayloria dubyi (Splachna
29 <dc:publisher>Sociedad de Biología de Chile</dc:publisher>
30 <dc:date>2010</dc:date>
31 <dc:type>Article</dc:type>
32 <dc:format>12 p.</dc:format>
33 <dc:format>Text</dc:format>
34 <dc:identifier>http://digital.library.unt.edu/ark:/67531/metadc97952/</dc:identi
35 <dc:identifier>ark: ark:/67531/metadc97952</dc:identifier>
36 <dc:source>Revista Chilena de Historia Natural, 2010, Santiago: Sociedad de Biol
37 <dc:language>Spanish</dc:language>
38 <dc:rights>Public</dc:rights>
39
  </oai_dc:dc>
  </metadata>
</record>

```

```

1 info:ark:/67531/metadc97952 {http://purl.org/dc/elements/1.1/}title Fenología de Taylori
2 info:ark:/67531/metadc97952 {http://purl.org/dc/elements/1.1/}title Phenology of Taylori
3 info:ark:/67531/metadc97952 {http://purl.org/dc/elements/1.1/}title Sub-Antarctic Biocul
4 info:ark:/67531/metadc97952 {http://purl.org/dc/elements/1.1/}creator Jofre, Jocelyn
5 info:ark:/67531/metadc97952 {http://purl.org/dc/elements/1.1/}creator Massardo, Franci
6 info:ark:/67531/metadc97952 {http://purl.org/dc/elements/1.1/}creator Rozzi, Ricardo
7 info:ark:/67531/metadc97952 {http://purl.org/dc/elements/1.1/}creator Goffinet, Bernar
8 info:ark:/67531/metadc97952 {http://purl.org/dc/elements/1.1/}creator Marino, Paul
9 info:ark:/67531/metadc97952 {http://purl.org/dc/elements/1.1/}creator Raguso, Robert
10 info:ark:/67531/metadc97952 {http://purl.org/dc/elements/1.1/}creator Navarro, Nelso I
11 info:ark:/67531/metadc97952 {http://purl.org/dc/elements/1.1/}subject bryophytes
12 info:ark:/67531/metadc97952 {http://purl.org/dc/elements/1.1/}subject Cape Horn Biosph
13 info:ark:/67531/metadc97952 {http://purl.org/dc/elements/1.1/}subject phenology repro
14 info:ark:/67531/metadc97952 {http://purl.org/dc/elements/1.1/}subject Splachnaceae
15 info:ark:/67531/metadc97952 {http://purl.org/dc/elements/1.1/}subject sub-Antarctic Ma
16 info:ark:/67531/metadc97952 {http://purl.org/dc/elements/1.1/}publisher Sociedad de Biol
17 info:ark:/67531/metadc97952 {http://purl.org/dc/elements/1.1/}date 2010
18 info:ark:/67531/metadc97952 {http://purl.org/dc/elements/1.1/}type Article
19 info:ark:/67531/metadc97952 {http://purl.org/dc/elements/1.1/}format 12 p.
20 info:ark:/67531/metadc97952 {http://purl.org/dc/elements/1.1/}format Text
21 info:ark:/67531/metadc97952 {http://purl.org/dc/elements/1.1/}identifier http://digit
22 info:ark:/67531/metadc97952 {http://purl.org/dc/elements/1.1/}identifier ark: ark:/67
23 info:ark:/67531/metadc97952 {http://purl.org/dc/elements/1.1/}source Revista Chilena
24 info:ark:/67531/metadc97952 {http://purl.org/dc/elements/1.1/}language Spanish
25 info:ark:/67531/metadc97952 {http://purl.org/dc/elements/1.1/}rights Public
26 info:ark:/67531/metadc97952 {http://purl.org/dc/elements/1.1/}description This article

```

The first example displays a common oai_dc record produced by an OAI-PMH repository. The second example shows the way that many of the command line tools expect to receive text: flattened and consistently delimited.

By utilizing OAI-PMH for this workflow, the concepts, methods, and tools are more portable and work with any repository that supports this protocol. This approach can therefore be more broadly applied than a workflow tightly integrated into an individual platform's infrastructure.

The tools developed for this workflow are described below with examples of common metadata analysis operations following the tool description.

Introduction to the tools

OAI-PMH Harvester

The first tool is an OAI-PMH harvester written in Python and based significantly on the Two Page OAI Harvester from OCLC Research [2]. The harvester, while simple, offers a set of options that cover many standard use cases for harvesting metadata records from an OAI-PMH repository. The harvester takes as input the URL for an OAI-PMH repository and the name of the output file for storing the results of the harvest. The output of the OAI-PMH response is stored as a single xml file enclosed in a set of <repository> tags.

```
1 | mphilipps$ python pyoaiharvest.py -l http://digital.library.unt.edu/explore/collections/t
```

The above command will harvest all records from the OAI-PMH repository available at the url <http://digital.library.unt.edu/explore/collections/UNTSW/oai/> and save them as a file named `untsw.dc.xml`. The default metadata format of `oai_dc` is requested from the repository and transmitted back to the harvester. For a full list of command line options see the help screen for the script:

```
1 | mphilipps$ python pyoaiharvest.py -h
2 | Usage: pyoaiharvest.py [options]
3 |
4 | Options:
5 |   -h, --help                show this help message and exit
6 |   -l LINK, --link=LINK      URL of repository
7 |   -o FILENAME, --filename=FILENAME
8 |                           write repository to file
9 |   -f FROMDATE, --from=FROMDATE
10 |                            harvest records from this date yyyy-mm-dd
11 |   -u UNTIL, --until=UNTIL
12 |                            harvest records until this date yyyy-mm-dd
13 |   -m MDPREFIX, --mdprefix=MDPREFIX
14 |                            use the specified metadata format
15 |   -s SETNAME, --setName=SETNAME
16 |                            harvest the specified set
```

The tool supports requesting a specific `setSpec`, a different metadata format, or limiting to a date range for updating collections of existing records.

Repository Breakers

Once a set of metadata records have been harvested, the next step of processing metadata records is converting them into a text format that can act as input to common command line tools. The tool `dc_breaker.py` is used for this function. This tool efficiently consumes the output format from the `pyoaiharvester.py` script as input and provides a set of options for converting this into formats easily used by other command-line tools.

```
1 | mphilipps$ python dc_breaker.py untsw.dc.xml
2 | 1000 records processed
3 |
4 | {http://purl.org/dc/elements/1.1/}contributor: |=====| 1032/1835 |
5 | {http://purl.org/dc/elements/1.1/}coverage: |==| 172/1835 |
6 | {http://purl.org/dc/elements/1.1/}creator: |=====| 1834/1835 |
7 | {http://purl.org/dc/elements/1.1/}date: |=====| 1807/1835 |
8 | {http://purl.org/dc/elements/1.1/}description: |=====| 1832/1835 |
9 | {http://purl.org/dc/elements/1.1/}format: |=====| 1835/1835 |
10 | {http://purl.org/dc/elements/1.1/}identifier: |=====| 1835/1835 |
11 | {http://purl.org/dc/elements/1.1/}language: |=====| 1835/1835 |
12 | {http://purl.org/dc/elements/1.1/}publisher: |=====| 681/1835 |
13 | {http://purl.org/dc/elements/1.1/}relation: |=====| 351/1835 |
14 | {http://purl.org/dc/elements/1.1/}rights: |=====| 1671/1835 |
15 | {http://purl.org/dc/elements/1.1/}source: |=====| 1426/1835 |
16 | {http://purl.org/dc/elements/1.1/}subject: |=====| 1835/1835 |
17 | {http://purl.org/dc/elements/1.1/}title: |=====| 1835/1835 |
18 | {http://purl.org/dc/elements/1.1/}type: |=====| 1835/1835 |
19 |
20 | dc_completeness 79.258856
21 | collection_completeness 84.250681
22 | www_completeness 99.604905
23 | average_completeness 87.704814
```

The example above runs the tool without any options selected. This will generate an output that can be helpful for seeing the overall utilization of fields within a collection of metadata records. It shows the fifteen elements of the `oai_dc` metadata scheme, as well as a visualization of the percentage of records in the repository file which have at least one value in this element. Next, the output presents a column showing the number of records in the repository that contain this field related to the total number of records in the collection and, finally, a percentage of utilization of this field in the collection.

Next, the tool provides a set of completeness scores, which give another type of overview of the collection. The first score is a percentage of completeness assuming that all fifteen of the oai_dc elements are required, the second is a measure of completeness as a collection. This percentage takes into account the fields used within the collection and generates a percentage of completeness based on those fields being present. For example, if a collection of metadata records uses the fields title, creator, description, and data exclusively, only these fields are used in the calculation of collection completeness. Finally, the percentage of completeness is based on the recommendation by the Kernel Metadata and Electronic Resource Citations (ERCs) [3] community, which state that the who, what, where, when of an item are required for adequate access and citation of an item. In this case who is mapped to creator, what is mapped to title, where is mapped to identifier and when is mapped to date. In addition to these completeness values there is an average of the three scores presented. These completeness measures are useful as an overview of collection level metadata and in showing improvement resulting from metadata cleanup activities.

Other sample uses of this tool are printing to standard output each value of a specific element either by itself or in the format of identifier <tab> value.

```
1 | mphilipps$ python dc_breaker.py -e creator untsw.dc.xml
2 |
3 | Tummala, Dinesh
4 | Li, Xinrong
5 | Nguyen, Son
6 | Akl, Robert G.
7 | Garlick, Ryan
8 | Akl, Robert G.
9 | Li, Wenming
10 | Kavi, Krishna
11 | Akl, Robert G.
12 | Alhabsi, Amer H.
13 | Al-Rizzo, Hussain M.
14 | Akl, Robert G.
15 | Akl, Robert G.
16 | Parvez, Asad
17 | Nguyen, Son
18 | Akl, Robert G.
19 | Naraghi-Pour, Mort
20 | Hegde, Manju
21 | Haidar, Mohamad
22 | Akl, Robert G.
23 | Al-Rizzo, Hussain
24 | Chan, Yupo
25 | ...
```

This example shows each instance of the creator element in the repository with one instance per line.

```
1 | mphilipps$ python dc_breaker.py -e creator -i untsw.dc.xml
2 |
3 | info:ark/67531/metadc30827      Tummala, Dinesh
4 | info:ark/67531/metadc30827      Li, Xinrong
5 | info:ark/67531/metadc30820      Nguyen, Son
6 | info:ark/67531/metadc30820      Akl, Robert G.
7 | info:ark/67531/metadc30828      Garlick, Ryan
8 | info:ark/67531/metadc30828      Akl, Robert G.
9 | info:ark/67531/metadc30824      Li, Wenming
10 | info:ark/67531/metadc30824      Kavi, Krishna
11 | info:ark/67531/metadc30824      Akl, Robert G.
12 | info:ark/67531/metadc30829      Alhabsi, Amer H.
13 | info:ark/67531/metadc30829      Al-Rizzo, Hussain M.
14 | info:ark/67531/metadc30829      Akl, Robert G.
15 | info:ark/67531/metadc30826      Akl, Robert G.
16 | info:ark/67531/metadc30826      Parvez, Asad
17 | info:ark/67531/metadc30826      Nguyen, Son
18 | info:ark/67531/metadc30823      Akl, Robert G.
19 | info:ark/67531/metadc30823      Naraghi-Pour, Mort
20 | info:ark/67531/metadc30823      Hegde, Manju
21 | info:ark/67531/metadc30835      Haidar, Mohamad
22 | info:ark/67531/metadc30835      Akl, Robert G.
23 | info:ark/67531/metadc30835      Al-Rizzo, Hussain
24 | info:ark/67531/metadc30835      Chan, Yupo
```

This is very similar to the previous example but it prepends the records identifier from the oai record identifier to the beginning of each line and separates the identifier from the value by a tab character.

When working with metadata record cleanup it is sometimes necessary to identify which records in a set do not have a value. This can be accomplished by adding the `-p` flag which displays if an instance of the specified element is present in the record.

```
1 | mphilipps$ python dc_breaker.py -e creator -p untsw.dc.xml
2 |
3 | info:ark/67531/metadc27055 True
4 | info:ark/67531/metadc27052 True
5 | info:ark/67531/metadc27051 True
6 | info:ark/67531/metadc27057 True
7 | info:ark/67531/metadc27050 True
8 | info:ark/67531/metadc27058 True
9 | info:ark/67531/metadc27054 True
10 | info:ark/67531/metadc27059 True
11 | info:ark/67531/metadc27056 True
12 | info:ark/67531/metadc27053 True
13 | info:ark/67531/metadc27175 True
14 | info:ark/67531/metadc27172 True
15 | info:ark/67531/metadc27171 True
16 | info:ark/67531/metadc27177 True
17 | info:ark/67531/metadc111250 False
18 | ...
```

This displays the identifier for the record followed by a True if the record has value for that element or False if the record does not have value for the specified element.

Finally an option to dump all data in the repository file into a tab-delimited format is available by using the flag `-d`.

```
1 | mphilipps$ python dc_breaker.py -e creator -d untsw.dc.xml
2 |
3 | info:ark/67531/metadc27055 {http://purl.org/dc/elements/1.1/}title [handmade paper
4 | info:ark/67531/metadc27055 {http://purl.org/dc/elements/1.1/}creator Spear, S
5 | info:ark/67531/metadc27055 {http://purl.org/dc/elements/1.1/}subject surface
6 | info:ark/67531/metadc27055 {http://purl.org/dc/elements/1.1/}subject laid pap
7 | info:ark/67531/metadc27055 {http://purl.org/dc/elements/1.1/}subject velvetee
8 | info:ark/67531/metadc27055 {http://purl.org/dc/elements/1.1/}description This det
9 | info:ark/67531/metadc27055 {http://purl.org/dc/elements/1.1/}contributor Wolfe, I
10 | info:ark/67531/metadc27055 {http://purl.org/dc/elements/1.1/}date 1987~
11 | info:ark/67531/metadc27055 {http://purl.org/dc/elements/1.1/}type Artwork
12 | info:ark/67531/metadc27055 {http://purl.org/dc/elements/1.1/}format 1 art on
13 | info:ark/67531/metadc27055 {http://purl.org/dc/elements/1.1/}format Image
14 | info:ark/67531/metadc27055 {http://purl.org/dc/elements/1.1/}identifier http://c
15 | info:ark/67531/metadc27055 {http://purl.org/dc/elements/1.1/}identifier ark: arl
16 | info:ark/67531/metadc27055 {http://purl.org/dc/elements/1.1/}language No Langu
17 | info:ark/67531/metadc27052 {http://purl.org/dc/elements/1.1/}title [full moon in fr
18 | info:ark/67531/metadc27052 {http://purl.org/dc/elements/1.1/}creator Spear, S
19 | info:ark/67531/metadc27052 {http://purl.org/dc/elements/1.1/}subject surface
20 | info:ark/67531/metadc27052 {http://purl.org/dc/elements/1.1/}subject silk (te
21 | info:ark/67531/metadc27052 {http://purl.org/dc/elements/1.1/}subject laid pap
22 | ...
```

Metadata Analysis Examples

The examples below make use of records harvested from the UNT Scholarly Works and UNT Theses and Dissertation collection hosted by the UNT Digital Library [4]. The OAI-PMH repository links are available at the following URLs:

UNT Scholarly Works – <http://digital.library.unt.edu/explore/collections/UNTSW/oai/>

UNT Theses and Dissertations – <http://digital.library.unt.edu/explore/collections/UNTETD/oai/>

These metadata collections are saved as untsw.dc.xml and untetd.dc.xml respectively for the purpose of these examples.

Find Most Prolific Creators in a set of Records

```
1 | mphilipps$ python dc_breaker.py -e creator untsw.dc.xml | sort | uniq -c | sort -n -r |
2 |
3 | 107 Mihalcea, Rada
4 | 93 Cundari, Thomas R.
5 | 90 Falsetta, Vincent
6 | 89 Phillips, Mark Edward
7 | 85 Spear, Shigeko
8 | 77 Moen, William E.
9 | 61 Marshall, James L., 1940-
10 | 59 Murray, Kathleen
11 | 58 Marshall, Virginia R.
12 | 57 Eve, Susan
```

This pipeline of commands will display back the ten most frequent creators. Here is what the pipeline is doing:

1. First the `dc_breaker` script is used to extract creator elements from each record.
2. The output of all creators is sorted by the `sort` command.
3. The output of the `sort` command is then made unique by the unix `uniq` command which is given the flag to append the number of times that value is present.
4. The resulting set is sorted again this time by the count of each value from highest value to lowest using the `-n` and `-r` options.
5. Finally the first ten values are returned along with their counts.

Find the number of unique creators in a dataset

```
1 | mphilipps$ python dc_breaker.py -e creator untsw.dc.xml | sort | uniq | wc -l
2 | 1592
```

This example extracts all of the creators in the repository, sorts the values, makes them unique and finally passes these to the `wc` (word count) utility with the `-l` option selected to count only lines. This returns the number of unique creators in a dataset, which can answer questions such as “How many creators have content in the UNT Scholarly Works Repository?”

Find the number of subjects per record

```
1 | mphilipps$ python dc_breaker.py -e subject -i untsw.dc.xml | cut -f 1 | sort | uniq -c |
2 |
3 | 24 info:ark/67531/metadc111236
4 | 21 info:ark/67531/metadc29400
5 | 21 info:ark/67531/metadc111261
6 | 20 info:ark/67531/metadc111271
7 | 20 info:ark/67531/metadc111269
8 | 20 info:ark/67531/metadc111200
9 | 17 info:ark/67531/metadc29401
10 | 17 info:ark/67531/metadc29393
11 | 17 info:ark/67531/metadc111212
12 | 17 info:ark/67531/metadc111177
13 | 16 info:ark/67531/metadc29399
14 | 16 info:ark/67531/metadc111216
15 | 16 info:ark/67531/metadc111206
16 | ...
```

This example extracts all subjects in the repository with the identifier appended to the beginning of the line. This identifier

is extracted from the line using the cut utility and then sorted, made unique and finally sorted again. The result is a list containing the number of subjects in a record followed by that records identifier. This output is useful when a collection has requirements pertaining to the number of subjects per record such as, “a minimum of three subjects per record, or no more than six subjects per record”.

Find the average number of subjects per record

```
1 | mphillips$ python dc_breaker.py -e subject -i untetd.dc.xml | cut -f 1 | sort | uniq -c |
2 |
3 | Average = 5.79016
```

This example builds on the previous by adding the use of the awk utility to calculate the average number of subjects per record in the repository. This information is especially useful in analyzing the overall subject usage patterns across various collections.

Find the records without any Creators

```
1 | mphillips$ python dc_breaker.py -e creator -p untsw.dc.xml | grep False
2 | info:ark/67531/metadc111250 False
```

The output of this example is a record identifier followed by True or False based on the presence of the specified field, this time creator. After piping this output to grep and searching only for lines that have False in them, we can see which records do not have any creator values. If each record in a collection should have a creator element present, this output makes it easy to find records lacking this value.

List creators sorted by creator value

```
1 | mphillips$ python dc_breaker.py -e creator untsw.dc.xml | sort
2 |
3 | AECO Economic and Community Development Class Fall, 2009
4 | Aars, Christian
5 | Abel, Mickey
6 | Abel, Mickey
7 | Acevedo, Mitzi
8 | Adada, Rami
9 | Adalar, Mehmet
10 | Adams, Mark
11 | Adams, Mark
12 | ...
13 | Zhang, Cankui
14 | Zhang, Jubo
15 | Zhang, Xue
16 | Zhang, Xue
17 | Zhang, Xue
18 | Zhang, Xue
19 | Zhang, Xue
20 | Zhao, Yong
21 | Zhi, Miaochan
22 | Zhou, Tie
23 | Zhou, Tie
24 | Zhou, Xin
25 | Zhou, Xin
26 | Zhou, Xin
27 | Zhu, Yuntian
```

This example demonstrates a common usage of the dc_breaker tool in which all values of a field are printed to the screen and then sorted. It is useful for identifying slight misspellings between adjacent values. This is particularly helpful with the creator and contributor fields when trying to normalize names values that are similar, which should be replaced with an authoritative version.

List creator values sorted by the number of times they occur


```

1 | mphilipps$ python dc_breaker.py -e creator untsw.dc.xml | sort | uniq -c
2 |
3 |     1 AECO Economic and Community Development Class Fall, 2009
4 |     1 Aars, Christian
5 |     2 Abel, Mickey
6 |     1 Acevedo, Mitzi
7 |     1 Adada, Rami
8 |     1 Adalar, Mehmet
9 |     2 Adams, Mark
10 |    1 Aerts, Andrea
11 |     ...
12 |     2 Zeug, Nicole M.
13 |     1 Zhang, Cankui
14 |     1 Zhang, Jubo
15 |     5 Zhang, Xue
16 |     1 Zhao, Yong
17 |     1 Zhi, MiaoChan
18 |     2 Zhou, Tie
19 |     3 Zhou, Xin
20 |     1 Zhu, Yuntian

```

This example adds the `uniq` tool to the pipeline from the previous example. The `uniq` tool with the `-c` (count) option takes a sorted list (supplied by the `sort` command) and sums the value instances before returning a row for each unique value preceded by the number of times it is present in the dataset. This shows the number of time each value occurs in the dataset, from which it is possible to derive the relative importance of one value over another.

```

1 | mphilipps$ python dc_breaker.py -e contributor untetd.dc.xml | sort | uniq -c
2 |     8 Mikler, Armin
3 |    48 Mikler, Armin R.
4 |     1 Mikler, Susie
5 |     1 Ramisetty-Mikler, Suhasini
6 |     1 Ramisetty-Mikler, Susie

```

This example shows that one value is most likely incorrect based on the fact that there are so many instances of the other version of the value ("Mikler, Armin", 8), ("Mikler, Armin, R.", 48).

List field values sorted by anagram hashes

Another variation on the previous models is helpful in certain situations. When comparing names in a dataset you may come across versions of names that are inverted (Last, First) as well as natural order (First Last). When identifying possible metadata changes based on adjacency in a list, these values would not appear next to each other based on a normal sort order.

Mark Phillips
Phillips, Mark

By feeding these values into a simple anagram hash function and then sorting the values based on the hash it is possible to get these values to group together and helpful to identify related problems.

aihkmipsr Mark Phillips
aihkmipsr Phillips, Mark

```

1 | import sys
2 | import re
3 |
4 | def anagram_string(string):
5 |     string = string.lower()
6 |     string = re.sub("[\W\d]", "", string)
7 |     return "".join(set(string))
8 |
9 | # input comes from STDIN
10 | for line in sys.stdin:
11 |     # remove leading and trailing whitespace
12 |     print "%s\t%s" % (anagram_string(line.strip()), line.strip())

```

```

1 | mphilipps$ python dc_breaker.py -e contributor untetd.dc.xml | python example-018.txt | s
2 |
3 |     1 abeihlsru  Hariss, Beulah
4 |     1 abeihlsru  Harris Beulah
5 |     1 abeihlsru  Harris, Beulah A.
6 |     6 abeihlsru  Harriss, Beulah
7 |     1 abeihlsru  Harriss, Beulah A.
8 |     1 abeihlsru  Harriss, Beulah A
9 |     58 abeihlsru Harriss, Beulah A.

```

Field Validation

Standard Format – Extended Date Time Format (EDTF)

These tools can also be helpful in validating a given value in a record based on a list of known values, a standard format, or a well-known feature. The following examples demonstrate the usage of these tools for this purpose:

```

1 | mphilipps$ python dc_breaker.py -e date untsw.dc.xml | python valid_edtf.py -
2 |
3 | 1987~    True
4 | 1987~    True
5 | 1981~    True
6 | 1987~    True
7 | 1990~    True
8 | 1987~    True
9 | 1987~    True
10 | 1987~    True
11 | 1987~    True
12 | 1986~    True
13 | 2003     True
14 | 2002     True
15 | 2002     True
16 | 2003     True
17 | 2001     True
18 | 2003     True
19 | 2002     True
20 | ...

```

A little background is required for this example. The UNT Libraries are moving toward the adoption of the Extended Date/Time Format (EDTF) [5] for encoding date information in their digital library. In order to identify values that comply with this specification a simple validator was created that compares a given string against the EDTF draft specification. The output of this script is in the following format:

```

2012 True
2001~ True
2012-12 True
2012-15 False
2012/12/12 False

```

It is then possible to filter the output to just instances that are invalid and investigate those in the dataset as examples of values to change.

Does the creator field have a comma (Are names properly inverted)

```

1 | mphilipps$ python dc_breaker.py -e contributor untetd.dc.xml | grep -v '&quot;;&quot;';
2 |
3 | William. D. Deering
4 | Jesus Rosales-Ruiz
5 | Brian Richardson
6 | Sue Bratton
7 | May. Andrew
8 | Nann Goplerud
9 | Webb James F.
10 | Blackburn S. A.
11 | Harris Beulah

```

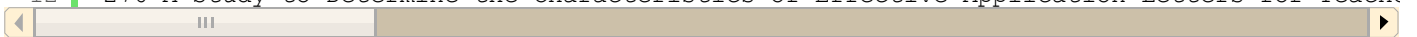
```
12 | Harris B. B.  
13 | Silvey J. K. G.  
14 | Silvey J. K. G.  
15 | Sharp L. A.  
16 | [Silvey J. K. G.]
```

In many digital library metadata settings names are notated in inverted fashion, separated by a comma. For example a name like “Mark Phillips” is notated as “Phillips, Mark” This example shows a quick way to identify values in a repository that are not formatted in this way. It prints the instances of creator names in the metadata records, which do not have commas. (This assumes that the presence of a comma is an indication of correct formatting.)

Longest title

Another use for this set of tools is to answer fairly obscure but important questions that arise in the library universe. One question that came up in the creation of the UNT Digital Library was, “what is the length of the longest titles we have in the system?” The user interface designer for the metadata display pages for records needed this. By using these tools it was relatively easy to find this answer.

```
1 | mphillips$ python dc_breaker.py -e title untetd.dc.xml | awk '{ print length($0) &quot;  
2 |  
3 | 354 Greek texts and English translations of the Bible: a comparison and contrast of the  
4 | 353 A Comparative Study of the Habits, Attitudes, and Opinions in Regard to Cigarette Sm  
5 | 349 Synthesis and Characterization of Platinum(II) (2-(9-anthracenylylidene)-4,5-bis(diph  
6 | 336 A Performer's Analysis of Lili Boulanger's Clairières dans le ciel: Song Cycle for  
7 | 332 Jules Massenet's Musical Prosody Focusing on His Eight Song Cycles And A Collection,  
8 | 325 The 1986 National Endowment for the Arts Commission: An introspective analysis of tv  
9 | 302 The Influence of Selma Meerbaum-Eisinger's Death on Xaver Paul Thoma's Composition o  
10 | 282 Supplemental Studies for Mastering Extended Techniques in Three Late Twentieth-Centu  
11 | 276 Gradus ad Parnassum of Modern Flute Technique: An Explication of Musical Intention a  
12 | 276 A Study to Determine the Characteristics of Effective Application Letters for Teache
```



This example uses a combination of awk to append the number of characters in each title to the beginning of each line, numerically sort the lines in reverse order with sort, and finally use the head utility to print the top 10 lines of this new list.

Conclusion

Harvesting metadata records from OAI-PMH repositories and then transforming these records into simple statements easily consumed by common command-line tools has significantly improved the workflow for identifying problems in metadata record collections and has allowed the UNT Libraries to utilize metadata enhancement resources in the most beneficial way possible. We have found that maintainers of metadata collections are eager to modify metadata records needing cleanup once identified using the methods described above. By using the methodology described in this article, the UNT Libraries is able to focus technology development resources on tools that can easily be integrated into this command-line environment. Currently in development are tools that help to further identify name collisions in creator, contributor, and publisher fields as well as validators for various fields and values present in the metadata used at the UNT Libraries.

Obtaining code mentioned in this article

All code mentioned in this article is available freely at the author’s GitHub repository: <http://github.com/vphill/>.

Notes

[1] Open Archives Initiative – Protocol for Metadata Harvesting (OAI-PMH) <http://www.openarchives.org/pmh/>

[2] 2PageOAI – <http://www.oclc.org/research/activities/oi2page.html>

[3] Kernel Metadata and Electronic Resource Citations (ERCs) <http://dublincore.org/groups/kernel/spec/>

[4] UNT Digital Library – About the Technology <http://digital.library.unt.edu/about/digital-library/technology/>

[5] Extended Date/Time Format (EDTF) <http://www.loc.gov/standards/datetime/>

About the author

Mark Phillips is Assistant Dean for Digital Libraries at the University of North Texas Libraries.

Subscribe to comments: [For this article](#) | [For all articles](#)

This work is licensed under a Creative Commons Attribution 3.0 United States License.

