Digging Deeper Reaching Further

Libraries Empowering Users to Mine the HathiTrust Digital Library Resources

Module 2.2 Gathering Textual Data: Bulk retrieval Instructor Guide

Further reading: go.illinois.edu/ddrf-resources

Narrative

The following is a suggested narrative to accompany the slides. Please feel free to modify content as needed when delivering workshops. Additional information and examples that are provided to aid the instructor's understanding of the materials are in *Italic text* – whether to present them to workshop audiences is entirely up to the instructor.

Slide M2.2-1
In this module, we will be covering methods for gathering textual data from the web in bulk, including web scraping, APIs, file transfers, and will also introduce the command line interface.

Slide M2.2-2

In this module we will take a look at how researchers can access data in an automated way that allows them to download text in bulk. We will explore basic API protocols (less scary than it sounds!), and then we will do more hands-on using the command line and the command line tool wget. Finally we will see how Sam gathered data for his Creativity Corpus.

Slide M2.2-3

By the end of this module, you will have gained experience using the command line interface such that you are able to run a command to scrape text from a webpage.
Slide M2.2-4
For text analysis projects, most researchers will need more than 1 or 10 texts – they will be working with hundreds, thousands, or millions of texts! Getting all this data can be very time-consuming. Simply pointing and clicking is inefficient, so it's very necessary to automate when possible.

In this module, we will introduce options for automated data retrieval and practice some basic web scraping.

Slide M2.2-5

Let's spend a little time going through three different ways to automate the retrieval of text, which is especially important when you're working at scale.

First, providers can make data available through file transfer mechanisms.

- File Transfer Protocol (FTP) and Secure (or SHH) File Transfer Protocol (called SFTP) are ways of moving files from one place to another on the internet.
- Rsync, which is used by the HTRC, is another. Rsync is efficient because it sends only the differences between the files at the source location and the files at the destination location. You need to run rsync from the command line. Note that rsync can be used to download Extracted Features data.

Slide M2.2-6

Another way to bulk retrieve content is through web scraping, which means grabbing text on the web. Plenty of text exists on the web that could be used for analysis, and web scraping helps you to avoid getting that text by tedious copying-and-pasting.

There are a couple of ways to scrape text from the web:

- You can run commands in the command line (we'll discuss more about the command line later in this module).
- You can write a **script**, which is basically a file containing a set of programing statements that can be run using the command line. We will be using scripts in some of our hands-on activities in later modules.
- There are also some web scraping software that you can use, such as webscraper.io or Kimono.
- We'll come back to web scraping later in the module!

Additional information on commands for web scraping: both curl and wget are command line tools that can download contents from FTP, HTTP and HTTPS. Technically, curl is a command line tool for getting or sending files using URL syntax. It uses the library libcurl, which is a client-side URL transfer library. Wget is command line only and does not have a library.

Slide M2.2-7

Before we conclude this section, it is important to note that web scraping can put a large workload on the scraped server, and this can upset the data holder. Actually, some data providers are more than willing to share their data. Therefore, be a good citizen and ask first if they'll give you access before you start scraping on your own, and make sure to check if they have an API. If you still have to use web scraping on your own, it is suggested that you time your requests to add a delay between server hits. This is not only polite, but also signifies you are not a malicious attacker.

Slide M2.2-8

I just mentioned APIs, which is another way to automate the retrieval of text. Let's look at them more closely. API stands for application programming interface.

- APIs are basically instructions (written in code) for accessing systems or collections. They are digital pathways to or from content. For example, using Amazon's API, you can display information about items for sale via Amazon on sites not operated by Amazon. You might think of them like a mailbox: you open the door (sometimes with a key) and either retrieve or deposit items.
- Usually you will need to write code to retrieve content via APIs, but some of them have graphical user interfaces (GUI) that make the process more convenient.
- A number of digital content providers, including the HT, have APIs to make it easier to grab data and metadata.
- A few examples are the Twitter API, which allows tweets to be downloaded or displayed on non-Twitter websites, and the Chronicling America API, which opens access to newspapers digitized and made available by the Library of Congress.

Slide M2.2-9

Access to some APIs is through a publically available URL, and that's the kind we are going to experiment with today.

The HathiTrust Bibliographic API provides programmatic access to bibliographic metadata for volumes in the HathiTrust. This API is just for retrieving volume metadata -- there are other kinds of APIs for getting other kinds of data, but today we'll just be practicing this one.

The metadata is retrieved using a specially formatted URL and the volume ID number. We will learn about how to use the HathiTrust Bibliographic API in the next activity.

Slide M2.2-10

Let's do a short activity together to explore APIs. Alone or with a partner, use the HathiTrust API to retrieve the metadata for a volume of your choice.

- First, search HTDL for a volume of interest to our research question (hathitrust.org).
- In your search results, each volume will have a "Full view" or "Limited (search-only)" link, depending on whether your volume is in the public domain. Click on this link, and you will be directed to the volume's page.
- Click on the URL of the page and find the volume ID. It should consist of all the characters between "id=" and ";" (can be made up of numbers and letters).
    - Examples: the **bold** parts of the following URLs are volume IDs
        - https://babel.hathitrust.org/cgi/pt?id=**pst.000023137875**;view=1up;seq=9
        - https://babel.hathitrust.org/cgi/pt?id=**miua.4731703.proc.001**;view=1up;seq=5
        - https://babel.hathitrust.org/cgi/pt?id=**mdp.39015005337046**;
    - The first characters of the volume ID are the code that identifies the institution where the volume came from (i.e. hvd = Harvard) and must be included as a part of the id.
- Structure your API call accordingly. We will explain the structure of the API call in the next slide.

Slide M2.2-11

The structured API call in URL format looks like what you can see on the screen:

Structure: http://catalog.hathitrust.org/api/volumes

_____/brief **OR** /full

_____/<ID type>

_____/<ID.json>

There is a base URL, and then you locate the information you want by saying:

- Whether the metadata returned should be brief (shorter record) or full (complete record)

- What kind of ID you'll be using to find the information. The volume ID number can be either OCLC number, record number, HTID, Library of Congress call number, ISBN, or ICCN.
- And the ID for the volume of interest, followed by ".json" (Note: An ID can be made up of numbers and letters)

Example: https://catalog.hathitrust.org/api/volumes/full/htid/mdp.39015005337046.json

To complete the activity, structure your own API call accordingly. Paste your structured API call to your web browser as a URL and hit enter. This will retrieve the volume's metadata for you in your browser.

Note the JSON format. JSON (JavaScript Object Notation) is a lightweight data-interchange format that is human-readable. It is often used for storing and exchanging data. (Instructors should also note that when the API call is made, results may look different when viewed in Firefox than in other browsers.)

Slide M2.2-12

Next, we're going to try our hand at a little web scraping, and to get ready, we'll be talking about the command line next.

This is what the Bash shell (i.e. the command line interface) on Mac computers (called Terminal), looks like. We'll go into a bit more detail in the next few slides.

Slide M2.2-13

The **command line** is a text-based interface that takes in commands and passes them on to the computer's operating system to run in order to accomplish a wide range of tasks. The interface is usually implemented with a shell.

- On Mac computers, we usually use a shell called Terminal. On Windows computers, the command line interface is Command Prompt, but there could also be other shells available depending on your version of Windows. These shells can be different from each other, and I'll explain more about what I mean.
- The Terminal shell on Mac is a Bash shell, which is native to Unix. Therefore, you can run Unix commands on Terminal. However, the shell on a Windows system is unlikely to be a Bash shell, unless you've configured it in Windows 10 (*Bash on Ubuntu on*

Windows comes installed on Windows 10), so it may or may not be able to run Unix commands.

- If you have a Windows computer and find you need a Unix-like shell, we recommend downloading Cygwin, which is a Unix-like shell for Windows. You can also try Git BASH, another a Unix-like shell.

To avoid systems issues, and to keep you from having to download software, we're using a web-based command line interface program called PythonAnywhere that has built-in Bash consoles.

Slide M2.2-14

PythonAnywhere is a browser-based programming environment. You can write code, save it in PythonAnywhere, and also run it from your browser. It comes with a built-in Bash shell command line interface, and it does not interact with your local file system.
We will be using PythonAnywhere in a couple of our modules.

Slide M2.2-15

Before we start looking at some basic Bash commands, here are some basic tips for working in a shell:

- A directory is essentially a folder, so when you need to "change directories" it's like clicking from "Desktop" to "Downloads".
- Cases, spaces, and punctuation matter. If you get an error, check first to make sure you entered the command correctly. Be especially careful about spaces.
- Hitting tab while you are typing on the command line attempts to finish the command or file name you are entering. This is particularly helpful when you need to type in long file names, and we strongly suggest using this command whenever you will need to type in a long file name throughout the workshop.
- Hitting the up arrow will cycle through the last commands you entered, but will not allow you to move between lines in a multi-line command. You will need to use the sideways arrows to navigate one command.
- When you view files in a shell, you will need to quit viewing before you can enter any new commands. Use "q" to quit viewing a file.

Slide M2.2-16

Now let's watch a video together that introduces some basic Bash commands.

http://go.illinois.edu/ddrf-bash-video *(Instructor should click on the link on the slide to access the video*

Main points:

- *The "pwd" command shows you which directory you're in.*
- *The "ls" command shows you all your files and directories.*
- *Use the "mkdir" command to make a new directory. Type in "mkdir" followed by a space and then what you want to call your new directory, for example "mkdir project1".*
- *The "cd" commands lets you change the directory you are in.*

  ex. cd /home/your username
- *The "less" command lets you view but not edit a file. The contents of the file will be displayed directly in the window. Hit "q" to quit viewing.*

  ex. less README.txt

Slide M2.2-17

In this hands-on activity, we're going to practice some basic commands in PythonAnywhere. Open a Bash console and try out these commands:

- Use **`pwd`** to see your current directory.
- Use **`ls`** to list the files and directories in your directory.

Slide M2.2-18

Next, we're going to use some other commands to unzip the activity_files.zip file that you have uploaded to PythonAnywhere and move them to your user directory.

Open a Bash console in PythonAnywhere and practice the following:

- Unzip activity files (activity_files.zip) in your user directory:

  ```
  unzip activity_files.zip
  ```
- Files were unzipped to a new directory called activity_files.
- Move the files from activity_files to your user directory. Substituting your Python Anywhere user name for *your_username*:

  ```
  mv activity_files/* /home/your_username
  ```

Slide M2.2-19

Let's use the ls command again in your Bash console to check if the files were successfully unzipped and moved. After typing **`ls`** and hitting enter, you should be able to see your list of files and directories. They should include the following files and directories:

> 1930 README.txt activity_files.zip remove_tag.py top_adjectives.py washington_4.txt
> 1970 activity_files remove_stopwords.py stopwords.txt washington word_count.py

Slide M2.2-20

In this activity, we're going to practice more basic commands in PythonAnywhere. Open a Bash console and try out these commands on the slides:

- Make a directory called 'test'

  `mkdir test`

- Change into the directory you just made, then back out of it

  `cd test`

  `cd ..`

After completing the activity here, please make sure you have changed out of your washington directory and moved back into your main directory. You can run the command **`pwd`** to double check. If you are still in the washington directory, type "**`cd ..`**" to move back up to your main directory. *(Instructors should remind participants about this once they complete this activity.)*

Slide M2.2-21

Let's get ready to add some files to our directory by taking a look at the command line tool for webscraping wget.

- Wget helps you transfer files from a server. In other words, it helps you scrape the contents of a website. Note that if there are links of the page you want to get, wget can grab that content, too. (*It recursively pulls content from page links.)*
- In the command line interface, you can also use options to modify the operation of a command and tailor it to do what you specifically want. Options are often written as single "flagged" letters, such as "-d" or "-r". Options for wget include:
  - "-l" (dash L). This option specifies how far in the hierarchy of the URL of the website you want to go when you are scraping content.
  - "--limit-rate=" sets the file transfer speed.

- **We'll demonstrate how to use these options in our hands-on activity.**

Slide M2.2-22

Now, let's return to our sample reference question. The student wants to analyze how concepts such as liberty change in political speech over time, so one possible approach to gathering textual data for this research question is to scrape some political speeches from an open resource, such as WikiSource, to start building a corpus.

Slide M2.2-23

For this hands-on activity, we will run a command to scrape text from a webpage version of George Washington's *Fourth State of the Union Address*. It's just one example of a speech we can scrape from this site.

- *Depending on the source of the data, there are different ways you may go about getting access to it. Some data providers build pathways to accessing content via APIs or systems for transferring data with rsync or ftp. In cases when there is no established mechanism for getting data, you may need to scrape it from the web. This is a complimentary technique to building a workset that we did earlier.*

Slide M2.2-24

We will need the URL of the webpage and PythonAnywhere (or another Bash shell of your choice) for this activity.

- Our command is:

  wget -l 1 --limit-rate=20k https://en.wikisource.org/wiki/George_Washington%27s_Fourth_State_of_the_Union_Address --output-document=washington_4.txt

  (Once again, instructors should remind participants to make sure they are in their main directory before running the wget command.)

- Here are the pieces of the command *(instructors should explain each part of the command before asking participants to run it):*
  - First we have the core of the command: wget.
  - Then we say "-l 1" (dash L, one), which means we will stay on the first "level" of the URL and not follow the links found within it.

- Then we specify the rate limit, which is how quickly we want to download the content. The speed we give here, 20k, is considered a thoughtful speed. Remember that web scraping taxes the server where the website is stored, so we want to be considerate about how we grab the data.
- Next we give the URL we are interested in scraping.
- Finally we specify that we want to write what we grab to a file called "washington_4.txt." Otherwise the command automatically names the file for you based on the URL.

Slide M2.2-25

(Screenshots are for backup only – instructor will demo live)

Here's what the webpage looks like.

Slide M2.2-26

Now, log in to PythonAnywhere and open a Bash shell and enter the following command:

```
wget -l 1 --limit-rate=20k
https://en.wikisource.org/wiki/George_Washington%27s_Fourth_State_of_the_Union_Address --output-document=washington_4.txt
```

Recommend that they copy-and-paste from the handout for ease if they get stuck.

Note that the URL If copied from Firefox may include the apostrophe in the URL instead of "escaping" it with the %27s, which will break the command.

Slide M2.2-27

If the command ran successfully, the shell should look like the screenshot on the slide.

Slide M2.2-28

Let's see what the scraped text looks like. Click on the PythonAnywhere logo on the top left of the page to go back to your dashboard. Click on the "Browse files" button in the Files column, or click on the "Files" option in the upper right corner of the page. You should be able to find a new file called "washington_4.txt". This file is our scraped version of the text from the Webpage. Click on the file to open it, or go to your Bash console and use the command `less washington_4.txt` to view it in your console.

Slide M2.2-29

Here is our scraped text! You may notice that there is still some strange stuff in it; we'll talk about that soon. If you viewed the file using the less command, please press "q" to quit viewing when you finish.

Slide M2.2-30

Let's take a little time to go over what happened in this activity. Essentially, you executed a command to scrape the text from the URL. The command grabbed everything on the web page, so this is only the first step of the process.

What would need to be removed from the files before analysis?

Next, you will want to prepare the data for analysis – taking out the HTML tags, for example. Then you can begin to analyze it using the text analytic method of choice.

Slide M2.2-31

Can you edit the script to scrape Washington's second State of the Union speech and save it to a new text file? Can you view your scraped file using the less command?

Slide M2.2-32

There are other ways we could have approached our web scraping activity.

- The first would have been to use Beautiful Soup, which is a Python-based web scraping tool. It has more options for what on the page you want to scrape (within the HTML) and it is good for getting clean, well-structured text.
- We could have also written our scraping task into a **script:**
    - A script is a list of directions for your computer to follow, and we are going to use some Python scripts in the upcoming modules.
    - A script lets you iterate through more content and scrape content in bulk.
    - Just be sure to time your request! Remember that web scraping uses resources on the server you are scraping, so you want to be polite and non-malicious.

Slide M2.2-33

Creating worksets (collections) is just one way of working with text data in the HathiTrust, which we introduced in Module 2.1. Users can also download data in bulk from the HT and HTRC to work on their own machines, and this chart outlines some different HT and HTRC datasets that can be acquired.

- If a researcher needs full text, a HT custom data request can give them bulk access to page images and OCR text limited to volumes in the public domain.
- When a researcher can make do with “abstracted text”, they can retrieve HTRC extracted features, which are per-volume files of select metadata and data elements. The researcher will be able to get JSON files of select data such as word counts and metadata of all available volumes.
- Finally, very advanced researchers can access HT data (public domain only for now) through the HTRC’s Data API, which is used in a special environment called the Data Capsule only. The Data Capsule is for advanced researchers, and it isn’t conducive to the workshop environment, so we won’t be discussing it in this workshop other than mentioning it now.

Both the custom data requests and extracted features are retrieved via file transferring – there is no provided user interface. We have already introduced file transferring earlier (we talked about FTP, SFTP, and rsync), and we will be discussing the HTRC Extracted Features in detail in one of our later modules.

Slide M2.2-34

Now let’s return to our Creativity Boom case study again. What did Sam do get the text that he needed in bulk?

- After creating a final list of volumes in the HDTL, Sam used rsync to retrieve the HTRC Extracted Features files that he wanted to analyze.
- Remember that rsync is a command line utility, like wget is, that transfers files between computers.

Slide M2.2-35

We’ve mentioned the HTRC Extracted Features dataset in the last two slides, so let’s quickly take a look at what it is. We’ll be getting hands-on experience with it later.

- The HTRC Extracted Features dataset is per-volume files of select metadata and data elements.
- The metadata includes things like bibliographic metadata (author, title, publication date) pulled from the MARC record, as well as metadata that has been inferred by a computer (how many blank lines there are on each page, for example).
- The data includes things like words and the number of times they occur per page.

- The files are formatted in JSON, the same format we saw earlier with the HT Bibliographic API.
- An example of the structured HTRC Extracted Features data is on the screen.

Slide M2.2-36

Let's wrap up this lesson with a discussion based on a short reading.

The Santa Barbara Statement on Collections as Data was initiated during an Institute of Museum and Library Services national forum at the University of California Santa Barbara (March 1-3 2017). It provides a set of high level principles to guide collections as data work.

We will read some parts of the Statement and discuss the development of digital collections in libraries.

(Short discussion for 5-10 minutes.)

Slide M2.2-37

Some main points from the Statement are shown on the slide.

Digital libraries and digital collections contain many textual documents, which can potentially be made available as data for computational use. Principle 2 of the Statement states "Collections as data development aims to encourage computational use of digitized and born digital collections."

Read full Statement here: https://collectionsasdata.github.io/statement/

Slide M2.2-38

Discussion questions:

Think about the collections available at your library. Does your library provide access to digitized materials in a way that is conducive to text analysis? If so, how? If not, why? How could it be more conducive to text analysis?

(Short discussion for 5-10 minutes.)

Slide M2.2-39

That's all we have for this lesson, and we will be happy to take any questions from you.

Slide M2.2-40
(*Show references so attendees know where to find them later.)*