Assignment 1

CS532-s16: Web Sciences

Spring 2016

John Berlin

Generated on January 28, 2016

Question 1

Demonstrate that you know how to use "curl" well enough to correctly POST data to a form. Show that the HTML response that is returned is "correct". That is, the server should take the arguments you POSTed and build a response accordingly. Save the HTML response to a file and then view that file in a browser and take a screen shot.

Answer

To post data to form using `curl` we must first consult the man pages to see what options are available.

From the man pages of `curl`:

-d, --data <data>
(puHTTP) Sends the specified data in a *POST request to the HTTP server*, in the same way that a browser does when a user has filled in an HTML form and presses the submit button. This will cause curl to pass the data to the server using the content-type application/x-www-form-urlencoded. Compare to -F, --form.

The man pages of `curl` state it can be used to post data to a form using this syntax:

```
curl -d "formValX=x&formValY=y" http://www.someform.com
```

From my recent work with the WS-DL tool mink I believe we can use curl send a uri to WebCite for archival using this command:

```
curl -I -d "url=http://docs.python-requests.org/en/latest/&email=
    jberlin@cs.odu.edu" http://webcitation.org/archive
```

The curl commands posts and we receive the following response:

```
HTTP/1.1 200 OK
Date: Thu, 28 Jan 2016 01:40:12 GMT
Server: Apache/2.0.63 (Unix) mod_ssl/2.0.63 OpenSSL/0.9.8e-fips-
    rhel5 mod_auth_passthrough/2.1 mod_bwlimited/1.4 PHP/5.2.9
X-Powered-By: PHP/5.2.9
Content-Length: 5993
Content-Type: text/html
```

The form that was used is shown in Figure 1

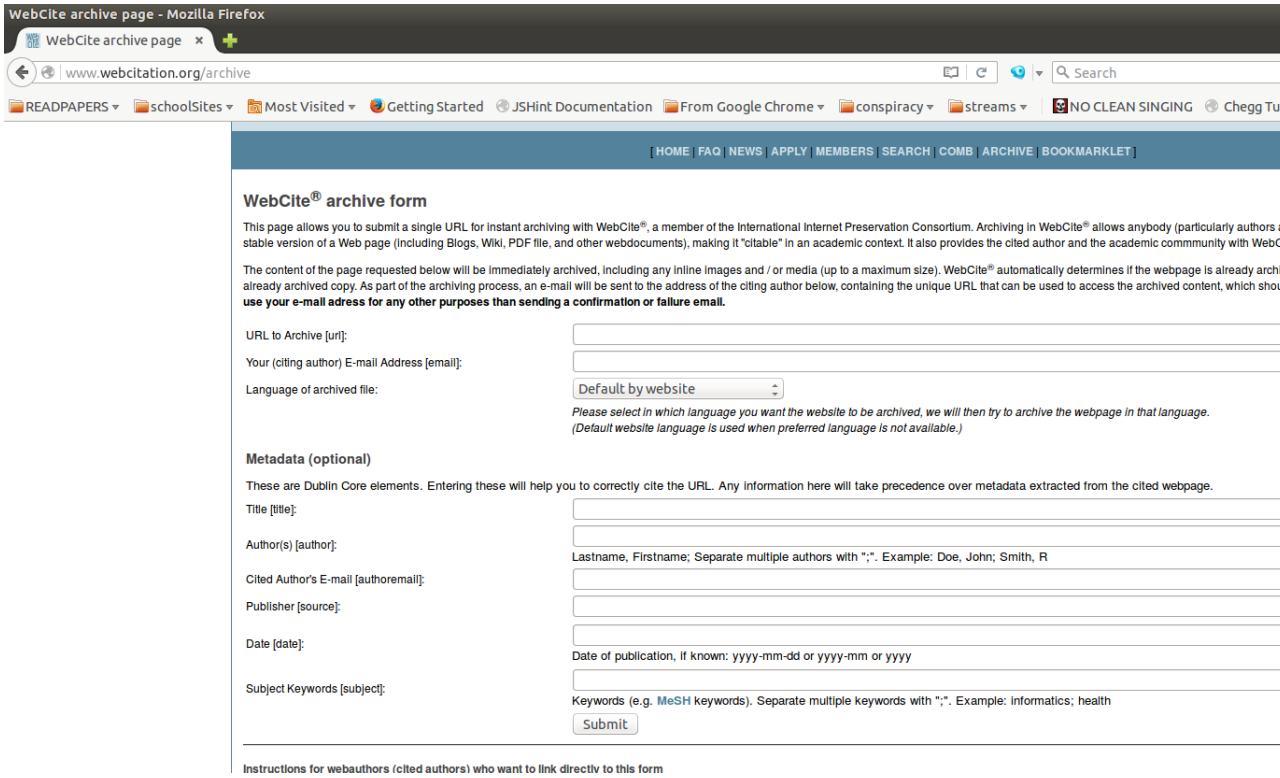

Figure 1: WebCite Archival Form

The -I option from the man pages for curl:

```
-I, --head
    (HTTP/FTP/FILE)  Fetch  the HTTP-header only!
    HTTP-servers feature the command HEAD which this uses to get nothing
    but the header of a document. When used on an FTP or FILE file,
    curl displays the file size and last modification time only.
```

As expected only the headers were returned to in the response. To recive back the html the -I option is removed and curl is executed with these arguments:

```
curl -d "url=http://docs.python-requests.org/en/latest/&email=
    jberlin@cs.odu.edu"  http://webcitation.org/archive
```

The rendered html returned by this curl command is shown in Figure 2

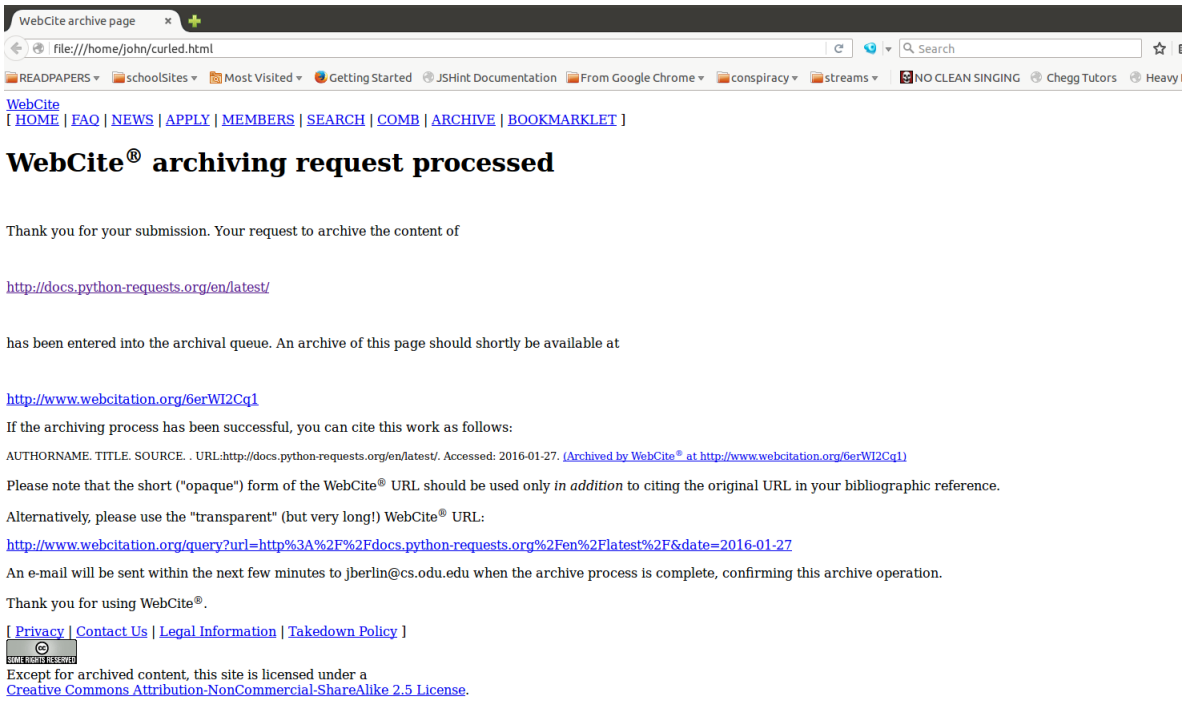

Figure 2: The output of the curl command, rendered in Mozilla Firefox

Question 2

Write a Python program that:

1. takes as a command line argument a web page
2. extracts all the links from the page
3. lists all the links that result in PDF files, and prints out the bytes for each of the links. (note: be sure to follow all the redirects until the link terminates with a "200 OK".)
4. show that the program works on 3 different URIs, one of which needs to be:
   http://www.cs.odu.edu/~mln/teaching/cs532-s16/test/pdfs.html

Answer

```
#!/usr/bin/env python3
import argparse
import re
from collections import deque

import requests
from bs4 import BeautifulSoup

reg_s = "((([A-Za-z]{3,9}:(?:\/\/)?)(?:[\-;:&=\+\$,\w]+@)?[A-Za-z0-9\.\-]+|" + \
        "(?:www\.|[\-;:&=\+\$,\w]+@)[A-Za-z0-9\.\-]+)((?:\/[\+~%\/\.\w\-]*)?" + \
        "\??(?:[\-\+=&;%@\.\w]*)#?(?:[\.\!\/\\\w]*))?)"

# my standard url regex found a while ago
url_re = re.compile(reg_s, re.IGNORECASE)


def print_headers(r):
    print("printing headers for url=", r.url)
    for headerk, headerv in r.headers.items():
        print(headerk + ":" + headerv)
    print("++++++++++++++++++++++++++++++")


def ispdf(rq):
    if rq.headers['Content-type'].lower() in 'application/pdf':
        return True
    else:
        return False


def strip_href(text, que, saw):
    try:
        s = BeautifulSoup(text, 'html5lib')
    except:
        # just because if this fails there are problems
        s = BeautifulSoup(text)
    all_a = s.find_all('a', href=True)
    for link in map(lambda a: a['href'], all_a):
```

```
39          if link not in saw:
40              if url_re.match(link):
41                  que.append(link)
42              else:
43                  print("The input uri %s failed to pass my regex " %
                        link, reg_s)
44      return s
45
46
47  if __name__ == '__main__':
48      parser = argparse.ArgumentParser()
49      parser.add_argument("uri", type=str, help="the url to extract
            from")
50      parser.add_argument("-v", help="verbose", action="store_true")
51      parser.add_argument("-ph", help="print all headers", action="
            store_true")
52
53      args = parser.parse_args()
54
55      # your on Ubuntu and you will like it
56      useragent = 'Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:44.0)
            Gecko/20100101 Firefox/44.01'
57      session = requests.Session()
58      session.headers.update({'User-Agent': useragent})
59      seen = set()
60      q = deque()
61      q.append(args.uri)  # ("http://tinyurl.com/gv8jnso")
62      uri = q.popleft()
63
64      if re.match(url_re, uri):
65          if args.v:
66              print("it matches")
67      else:
68          print("The input uri %s failed to pass my regex " % uri,
                reg_s)
69
70      request = session.get(uri)
71      """:type : requests.Response """
72      if ispdf(request):
73          print("The link %s was a link to an actual pdf file " % uri
                ,
74                ("but it was misleading %s its length is " % request.
                      url)
75                if uri != request.url else "its length is",
76                request.headers['Content-Length']
77                )
78          if request.is_redirect or request.is_permanent_redirect:
79              print("It was dirty reditect ;)")
80          exit()
81      soup = strip_href(request.text, q, seen)
82      if args.ph:
83          print_headers(request)
84
85      while True:
86          try:
87              uri = q.popleft()
88          except IndexError:
```

```
89              break
90          if re.match(url_re, uri):
91              if args.v:
92                  print("the uri %s was valid" % uri)
93          else:
94              print("Bad url %s" % uri)
95              continue
96          seen.add(uri)
97
98          request = session.get(uri)
99          """:type : requests.Response """
100
101         if args.ph:
102             print_headers(request)
103
104         if ispdf(request):
105             if request.ok:
106                 print("The link %s was a link to an actual pdf file
                        " % uri,
107                       ("but it was misleading %s its length is " %
                              request.url)
108                       if uri != request.url else "its length is",
109                       request.headers['Content-Length']
110                       )
111                 continue
112
113         if args.v:
114             print("so far I have processed these uris: ", seen)
115             print("I still have these to go", q)
116
117         if request.ok:
118             print("\nHey were are ok! %i" % request.status_code, "
                    Done going down the rabit hole for %s\n" % uri)
119         elif request.is_permanent_redirect or request.is_redirect:
120             soup = strip_href(request.text, q, seen)
121
122     session.close()
```

Listing 1: Python program for extracting links from a web page to look for pdf files

The code is shown in Listing 1 starting on page 4:

- Takes an argument uri specifying the web page to extract links from (line 49)
- Takes an optional argument to print extra data (line 50)
- Takes an optional argument to print the headers of all received requests (line 51)
- Open a session for (line 57), Set our user agent (line 58), Download starting URI (line 70), check to see if won on first uri(line 72), otherwise get all hrefs(line 81), and consume each one in web page informing user if pdf was found(lines 87-120)

The core libraries used are

- **argparse**: I do not like sys.args and closest thing to Commons CLI
- **re**: for regular expression support. Do not want some arbitrary href=file.js or malformed uri
- **requests**: better urllib, it will handle sneaky redirects automatically
- **bs4**: get the web soup even tho I like my soup hot and sour

strip_href (lines 31 to 44) takes the html text and retrieves all *a* tags and extracts the uri content and adds it the queue. Return the hot soup after extraction.

Since requests library handles the redirects for us a check to see if there was some trickery the statement uri != resquest.url checks if the uri found in the extraction is not the same as was the final link. If its not a redirect happened for sure.
Executing: python3 followLinks.py http://www.cs.odu.edu/ mln/teaching/cs532-s16/test/pdfs.html gives the output

```
The link http://www.cs.odu.edu/~mln/pubs/ht-2015/hypertext-2015-temporal-violations.pdf
was a link to an actual pdf file  its length is 2184076

The link http://www.cs.odu.edu/~mln/pubs/tpdl-2015/tpdl-2015-annotations.pdf
was a link to an actual pdf file  its length is 622981

The link http://arxiv.org/pdf/1512.06195 was a link to an actual pdf file
but it was misleading
http://arxiv.org/pdf/1512.06195.pdf its length is  1748961

The link http://www.cs.odu.edu/~mln/pubs/tpdl-2015/tpdl-2015-off-topic.pdf
was a link to an actual pdf file  its length is 4308768

The link http://www.cs.odu.edu/~mln/pubs/tpdl-2015/tpdl-2015-stories.pdf
was a link to an actual pdf file  its length is 1274604

The link http://www.cs.odu.edu/~mln/pubs/tpdl-2015/tpdl-2015-profiling.pdf
was a link to an actual pdf file  its length is 639001


The link http://bit.ly/1ZDatNK was a link to an actual pdf file
but it was misleading
http://www.cs.odu.edu/~mln/pubs/jcdl-2015/jcdl-2015-temporal-intention.pdf
its length is  720476
```

```
The link http://www.cs.odu.edu/~mln/pubs/jcdl-2015/jcdl-2015-mink.pdf
was a link to an actual pdf file  its length is 1254605

The link http://www.cs.odu.edu/~mln/pubs/jcdl-2015/jcdl-2015-arabic-sites.pdf
was a link to an actual pdf file  its length is 709420

The link http://www.cs.odu.edu/~mln/pubs/jcdl-2015/jcdl-2015-dictionary.pdf
was a link to an actual pdf file  its length is 2350603
```

Now when executing it with a link straight to a pdf such as http://arxiv.org/pdf/1512.06195.pdf

```
The link http://arxiv.org/pdf/1512.06195.pdf
was a link to an actual pdf file  its length is 1748961
No other links are in a pdf have nice day
```

For the third and final link I saw one in the full output of the program when running on the required link that looked oddly funky lets try it: http://bit.ly/jcdl-pdf

```
Hey were are ok! 200 Done going down the rabit hole
for http://twitter.com/webscidl

The link
http://www.cs.odu.edu/~mln/pubs/ht-2015/hypertext-2015-temporal-violations.pdf
was a link to an actual pdf file  its length is 2184076

The link http://www.cs.odu.edu/~mln/pubs/tpdl-2015/tpdl-2015-annotations.pdf was a link to a

The link http://arxiv.org/pdf/1512.06195 was a link to an actual pdf file
but it was misleading
http://arxiv.org/pdf/1512.06195.pdf its length is  1748961

The link http://www.cs.odu.edu/~mln/pubs/tpdl-2015/tpdl-2015-off-topic.pdf was a link to an

The link http://www.cs.odu.edu/~mln/pubs/tpdl-2015/tpdl-2015-stories.pdf
was a link to an actual pdf file  its length is 1274604

The link http://www.cs.odu.edu/~mln/pubs/tpdl-2015/tpdl-2015-profiling.pdf
was a link to an actual pdf file  its length is 639001


The link http://bit.ly/1ZDatNK was a link to an actual pdf file
but it was misleading
```

```
http://www.cs.odu.edu/~mln/pubs/jcdl-2015/jcdl-2015-temporal-intention.pdf
its length is  720476

The link http://www.cs.odu.edu/~mln/pubs/jcdl-2015/jcdl-2015-mink.pdf
was a link to an actual pdf file  its length is 1254605

The link http://www.cs.odu.edu/~mln/pubs/jcdl-2015/jcdl-2015-arabic-sites.pdf
was a link to an actual pdf file  its length is 709420

The link http://www.cs.odu.edu/~mln/pubs/jcdl-2015/jcdl-2015-dictionary.pdf
was a link to an actual pdf file  its length is 2350603
```

Oddly familiar would you not agree with me.

Question 3

```
Consider the "bow-tie" graph in the Broder et al. paper (fig 9):
   http://www9.org/w9cdrom/160/160.html

   Now consider the following graph:

   A --> B
   B --> C
   C --> D
   C --> A
   C --> G
   E --> F
   G --> C
   G --> H
   I --> H
   I --> J
   I --> K
   J --> D
   L --> D
   M --> A
   M --> N
   N --> D
   O --> A
   P --> G

   For the above graph, give the values for:

   IN:
   SCC:
   OUT:
   Tendrils:
   Tubes:
   Disconnected:
```

- **IN**: [M,O,P] nodes(pages) that can reach the SCC, but cannot be reached from it
- **SCC**: [A,B,C,G] central core, all of whose pages can reach one another along directed links
- **OUT**: [D,H]: pages that are accessible from the SCC, but do not link back to it
- **Tendrils**: [I,J,K,L] pages that cannot reach the SCC, and cannot be reached from the SCC

- **Tubes**: [N] passage from a portion of IN to a portion of OUT without touching SCC
- **Disconnected**: [E,F] pages who do not fit the descriptions above or has no links to the core or other pages

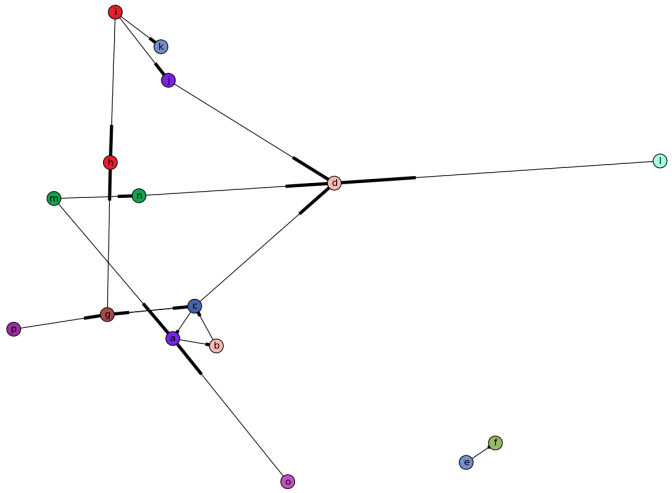

Figure 3: A different means of looking at the graph

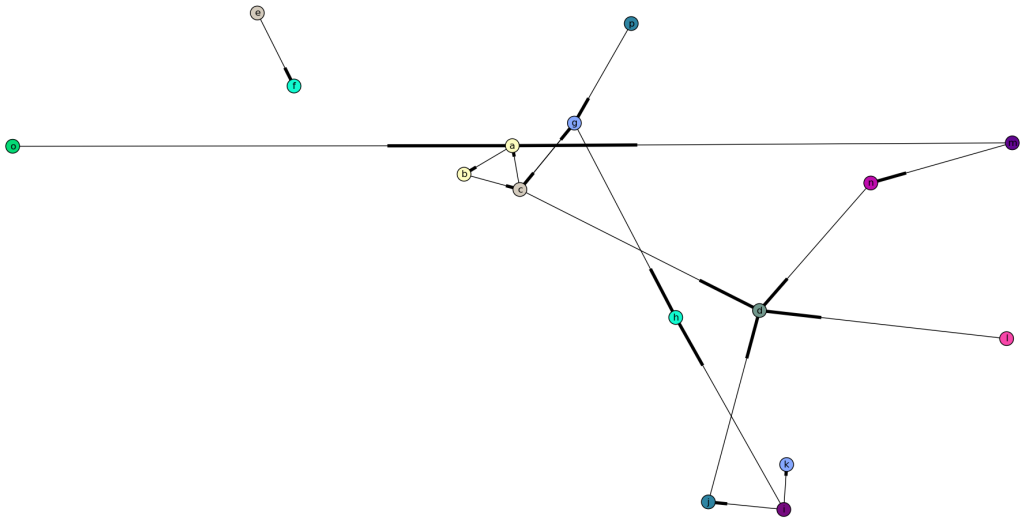

Figure 4: A different means of looking at the graph2