Practical Perl Tools

So Long and Thanks for All the Fish

DAVID N. BLANK-EDELMAN

David has over 30 years of experience in the systems administration/DevOps/SRE field in large multiplatform environments. He is the curator/editor of the O'Reilly Book *Seeking SRE: Conversations on Running Production Systems at Scale* and author of the O'Reilly Otter Book (*Automating Systems Administration with Perl*). He is a co-founder of the wildly popular SREcon conferences hosted globally by USENIX. David currently works for Microsoft as a senior cloud advocate focusing on site reliability engineering.

After 12 continuous years of writing this column with only one missed month, it is time for this column to shuffle off this mortal coil and leave room in *;login:* for a different column.

I am so, so grateful to:

- You, the reader. It's been a thrill to be able to talk with you each issue about something interesting in the land of Perl.
- USENIX, who gave me the challenge to stretch myself each issue to find that interesting topic.
- The countless authors and contributors in the Perl world that I've had the pleasure of writing about.

You may (or may not) be wondering: just how many Practical Perl Tools columns have been published in that 12-year span? I know I was. I thought it might be fitting to show you one last Perl program that I wrote to help me find all of the previous columns and also answer this question. Ready for one last dance?

For this code, we return to an old friend that has appeared in this column before, `WWW::Mechanize`. This module makes it easy to fetch web pages and parse them for specific links. The first part of the code sets up where we are going to pull the information from and grabs the first page.

```
use strict;
use WWW::Mechanize;
use open qw(:std :utf8); # quash warnings due to UTF-8 chars

# where are the issues found?
my $start = 'https://www.usenix.org/publications/login';

# for finding my articles
my $name  = 'blank-edelman|practical-perl-tools';

my $mech  = WWW::Mechanize->new;

# fetch the issues page
$mech->get( $start );
```

That page is both a listing of all of the issues and the root for all of the subsequent pages we will want to fetch. In the code we're going to see, we are careful to only retrieve URLs that start with this prefix.

Now let's find all of the issues we will want to check for an article:

```
            my @issues = $mech->find_all_links(
             tag => "a",
             url_abs_regex =>
               qr/$start\/[a-z-]+20(0[6-9]|1[0-8])/,
               text_regex => qr/.+/,
            );
```

The `find_all_links()` method is doing all of the heavy lifting, but we should explain the arguments it is receiving. The "tag" argument is pretty easy to guess: we're only looking for the anchor HTML tag, things of the form `<a href='something'>text</a>`. The next two arguments are a little more obtuse.

The first, `url_abs_regex`, is a regular expression meant to only find certain links on the page. It serves two purposes in this case: only select links that begin with `$start`, and also limit which years will be selected. I happen to know I began writing the column in 2006, so it only finds 2006–2009 and 2010–2018.

The `text_regex` deals with a quirk in the source of the issues page. Each issue actually has two anchor tabs, one for the picture of the cover, the second is the link for the text name (e.g., "Summer"). This regex makes sure we only grab one of the two, the one that has any characters in the text portion of the URL. This means we choose:

<a href="/publications/login/spring2018">Spring</a>

instead of:

<a href="/publications/login/spring2018"><img src="https://www.usenix.org/sites/default/files/styles/login_thumbnail/public/login/covers/1801_login_cover_170x221.png?itok=VBVKlmFO" width="100px" height="130" alt="" /></a>

The end result of the call to `find_all_links` is a list of `WWW::Mechanize::Link` objects that will point to all of the possible issues we'll want to scan for this column.

Now let's iterate over all of the issue links we found:

```
   my $issue_count = 0;
   foreach my $issue (@issues){

         $mech->get($issue->url_abs());

         my $article_link = $mech->find_link(
                 url_regex=>qr/$name/,
         );

         if (defined $article_link){
                 print  $article_link->text() .
                 ":\n" .
                 $article_link->url_abs(),"\n\n";
            $issue_count++;
         }
   }
   print "$issue_count issues in total!\n";
```

For each issue link we have, we fetch the contents of that link, then look for links in that page which could be my column. If we find one, we print the name of the column and its URL.

It would be pretty simple to grab the actual PDF of the column at this point if we wanted to create an archive of the content. This would consist of another `get()`, `find_link()` to locate the PDF on the page, `get()` that URL, and finally a call to `save_content()` to write it to a file. Permit me one last "exercise for the reader" if you will.

The output of our code looks like this:

```
Practical Perl Tools: Top of the Charts:
https://www.usenix.org/publications/login/spring2018
/blank-edelman

Practical Perl Tools: It's a Relationship Thing:
https://www.usenix.org/publications/login/summer2018
/blank-edelman

Practical Perl Tools: GraphQL Is Pretty Good Anyway:
https://www.usenix.org/publications/login/fall-2018
-vol-43-no-2/blank-edelman

Practical Perl Tools: Off the Charts:
https://www.usenix.org/publications/login/spring2017
/practical-perl-tools-charts

Practical Perl Tools: Perl on a Plane:
https://www.usenix.org/publications/login/summer2017
/blank-edelman
...
66 issues in total!
```

And there's the answer. Thank you, dear reader, for being with me for 66 columns.

Take care.