

Parsing Web Server Access Log File Fields (PDF Tutorial)

Pattern Matching and Regular Expressions: Using Perl to Parse Web Server Access Log File Fields

In the [last Perl-Tips post](#), I discussed the NCSA Extended Log Format for web servers. Please read that post before continuing with this one. The discussion here assumes the Extended Log Format for the web server access log.

Let's review the problem at hand. We have a website for which we are getting visitors. We want to do some analysis (web metrics, web analytics) for the website: who is visiting, how often, and which pages? To do this, our secondary goal has to be to transfer the information from the website's web server access log into a database. We are a few posts away from this goal. We have devised a temporary [XML format](#), which we'll use later to transfer the access log data into a database.

To create the WSML (Web Server Markup Language) XML output file, we need to parse the access log. To do that, we need to come up with the appropriate Perl regular expressions to properly extract the fields of each entry in the access.log. Regular expressions are a sort of wildcard/pattern rule that we can specify to extract all or some "fields" in a line of data. I can't give you a full discussion of regular expressions here. ([This](#) and [this](#) are two of the best books available on the topic.) You'll have to at least understand the basics of Perl pattern-matching before continuing. (Try checking some of the Perl *perldoc* documentation that should have come with your Perl distribution first.)

However, before we actually get into true regular expressions, let me show you a way to extract some of the information of the web server access log using the Perl `split()` function. We cannot accurately extract every field in every record, but we can extract some important ones. The [rest of this post](#) is in PDF format. (Before you read the PDF file, please read the previous posts.) The post following this one will get into using regular expressions to extract all of the fields in each access log record.

(c) Copyright 2005-present, Raj Kumar Dash, <http://perl-tips.blogspot.com>

PDF Tutorial – Parsing Select Web Server Access Log Record Fields Using Perl's `split()` Function

Disclaimer: *You are free to use the Perl code below as you wish, without giving me credit. While I have tested the code, I do not, however, take any responsibility for your use of the code, or for the loss of any data that you have if you use the code incorrectly. If you do not agree with this, please do not use the code.*

What we will do in this tutorial is:

- (1) Determine which fields of the web server access log that we are interested in.
- (2) Use the Perl `split()` function to roughly split each access log record into its component fields.
- (3) Produce a report that shows, for each unique IP address, what days and times they visited

To review, here is an example record from the access log of my web server:

```
131.104.175.199 - - [08/Sep/2005:17:16:10 -0400] "GET /blog/closeup-veryism.jpg
HTTP/1.1" 200 3282 "http://geoplotting.blogspot.com/" "Mozilla/4.0 (compatible; MSIE
6.0; Windows NT 5.1; SV1; MathPlayer 2.0)"
```

We are only interested in a few fields: the IP address and the date and time. We'll ignore the zone, as it does not change for a given website. We can write a complex Perl regular expression to extract the IP address, date and time from each record in the web server access log, or we can use the `split` function to break down each record roughly into a number of fields. Have a look at the example access log record above. Notice that, for the most part, each field is separated by a space. There are some groups of fields that are both separated by spaces and surrounded by double quotes. We are not concerned with these fields just yet. We want a quick-and-dirty extractor for 3 fields.

The Perl `split()` function is perfect for a quick-and-dirty field extractor here. We give the function a character (or sequence) to split fields with. For example, look at the following line of Perl code:

```
my @tokens = split(' ');
```

Keep in mind that the above code operates on the Perl system variable `$_`, which contains the current line of the input file. The above line of code operating on the example access record above would split the record into the following fields:

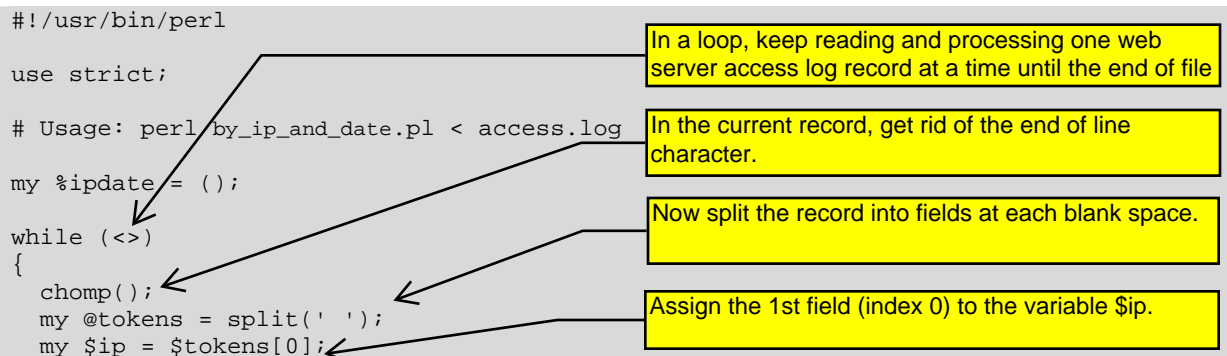
```
131.104.175.199
-
-
[08/Sep/2005:17:16:10
-0400]
"GET
/blog/closeup-verysm.jpg
HTTP/1.1"
200
3282
"http://geoplotting.blogspot.com/"
"Mozilla/4.0
(compatible;
MSIE
6.0;
Windows
NT
5.1;
SV1;
MathPlayer
2.0)"
```

As you can see, this method is not very accurate. It fragments the fields too far, because we are ignoring the double quotes. But it's fast and we can easily get the three fields that we want. In fact, it's as simple as accessing the array `@tokens` for indexes 0, 3 and 4:

```
my $ip      = $tokens[0]; # Index 0 is the ip address
my $dateinfo = $tokens[3]; # Index 3 is the date and time, but it has an extra square bracket at
                           # the beginning of the string.
$dateinfo   =~ s/^\[//;   # Use the Perl substitute command to get rid of the square bracket.
                           # We don't care about the colons in the string.
$dateinfo{$ip}{$dateinfo}++; # Use a double-level hash table to save the date info. The date info
                              # is secondary. Each time the same visitor requests a website
                              # resource (web page, document, image, script), the count for that ip
                              # and date/time combination is incremented by 1. We don't really care
                              # about the count. It's just a convenient way to record the
                              # combinations.
```

Now in truth, the double-level hash isn't the best choice, but it's fast and easy to create. We could have used an array of dates for each IP address, but it's a hassle to check dates so we don't store duplicates. And fast-and-dirty is all we're interested in right now. Here is a complete sample Perl program:

```
#!/usr/bin/perl
use strict;
# Usage: perl by_ip_and_date.pl < access.log
my %ipdate = ();
while (<>)
{
  chomp();
  my @tokens = split(' ');
  my $ip = $tokens[0];
```



In a loop, keep reading and processing one web server access log record at a time until the end of file

In the current record, get rid of the end of line character.

Now split the record into fields at each blank space.

Assign the 1st field (index 0) to the variable `$ip`.

```

my $dateinfo = $tokens[3];
$dateinfo =~ s/^\[//;
$dateinfo{$ip}{$dateinfo}++;
}

my $d;
my $ip;
my $date;
my $nvis = 0;

for $ip (sort keys %ipdate)
{
    my %dates = %{$ipdate{$ip}};
    print "$ip";
    foreach my $d (sort keys %dates)
    {
        print " [$d]";
    }
    print "\n";

    $nvis++;
}
print "\n\nTotal number of unique visitors: $nvis\n";

```

Assign the 4th field (index 3) to the variable \$dateinfo. Then get rid of the square bracket, '[', that is at the beginning of the string.

Increment a counter for the current (ip address, date info) combination, and store the result in a double-level hash table.

Sort the ip addresses.

For each unique ip address, get the secondary hash table of date/time counts. (The date/time strings are the keys, the counts are the values.)

For each ip, print the ip and the list of date/time keys. Note that the inner foreach loop sorts the date for the current ip. However, because "Aug" (august) is lexically before, say, Jun (june), the dates are not sorted the way that we really want.

I've tested this Perl script in both Cygwin/Linux and Windows XP. I've written similar code and had it work fine on Linux, Win 98 and Win NT. This script produces one line of output for each unique IP address. Here is an example usage of the script:

```
perl by_ip_and_date.pl < access.log
```

Here's a sample line of output

```
131.104.175.199 [01/Sep/2005:15:10:46] [03/Sep/2005:18:14:26]
```

So the above visitor visited twice in September.

Summary

The Perl script above is a quick-and-dirty way to determine the visiting habits of each unique visitor. The code works for its intended purpose, but because the `split()` function, as used, cannot accurately break down each access log record into the appropriate fields. To do extract all of the fields in each access record, we will need to use regular expressions. We'll do this in the next post.

(c) Copyright 2005-present, Raj Kumar Dash, <http://perl-tips.blogspot.com>