HTML Dokumente mit Perl verwalten, HTML::TagReader

by Guido Socher (homepage)

About the author:

Guido liebt Perl, weil es eine flexible und schnelle Script−Sprache ist. Das Motto von Perl, "Es gibt mehr als nur einen Weg", reflektiert die Freiheit und die Möglichkeiten, die man hat, wenn man Open Source Software benutzt.

Abstract:

Jemand, der eine Webseite mit mehr als 10 HTML Seiten verwalten will, wird schnell merken, dass man einige Programme zur Unterstützung dieser Aufgabe braucht.
Aus historischen Gründen bietet die meiste Software Funktionen, um Dateien Zeile für Zeile oder Zeichen für Zeichen zu lesen. In SGML/XML/HTML Dateien haben Zeilen leider keine Bedeutung. SGML/XML/HTML Dateien sind Tag−basiert (Sprich "Täg", nicht zu verwechseln mit Tag oder Nacht). HTML::TagReader ist ein effizientes Modul, um eine Datei Tag für Tag zu lesen.

Dieser Artikel nimmt an, dass du Perl schon sehr gut beherrscht. Du kannst z.B. meine Perl Tutorien, (Januar 2000, LinuxFocus) lesen um Perl zu lernen.

Einführung

Historisch gesehen sind fast alle Dateien Zeilen basiert. Beispiele sind die Unix Konfigurationsdateien wie /etc/hosts, /etc/passwd .... Es gibt sogar noch Betriebssysteme, in denen das Betriebssystem selbst Funktionen hat, um Daten zeilenweise zu lesen. SGML/XML/HTML Dateien basieren auf Tags. Zeilen haben hier keine Bedeutung. Texteditoren und Menschen funktionieren aber irgendwie immer noch zeilenweise.

Speziell größere HTML Dateien werden im allgemeinen aus mehreren Zeilen HTML−Code bestehen. Es gibt sogar Programme wie z.B. "Tidy", die HTML einrücken und in Zeilen umbrechen, damit der HTML−Code lesbar wird. Wir benutzen Zeilen, obwohl HTML auf Tags basiert und nicht auf Zeilen. Man kann das mit C−Code vergleichen. Theoretisch könnte man den gesamten C−Code in eine einzige Zeile schreiben. Niemand macht das. Es wäre unlesbar.

Man erwartet daher, dass ein HTML−Syntaxchecker schreibt "Fehler in Zeile ..." und nicht "Fehler nach Tag 4123". Das ist so, weil man mit einem Texteditor ganz einfach zu einer bestimmten Zeile gehen kann.

Was man also braucht, ist ein **einfaches Verfahren um eine HTML−Datei Tag für Tag zu lesen und gleichzeitig mitzuhalten, in welcher Zeile man sich befindet**.

Eine mögliche Lösung

Die übliche Methode, eine Datei in Perl zu lesen, ist, den *while(<FILEHANDLE>)* Operator zu benutzen. Das wird die Datei zeilenweise lesen und jede Zeile an $_ übergeben. Warum macht Perl das? Perl hat eine interne Variable namens INPUT_RECORD_SEPARATOR ($RS oder $/), in der definiert ist, dass "\n" das Ende einer Zeile ist. Wenn man $/=">" setzt, dann wird Perl ">" als "Zeilenende" benutzen. Der folgende Befehl wird HTML Text umformatieren, so dass er immer in ">" endet:

perl −ne 'sub BEGIN{$/=">";} s/\s+/ /g; print "$_\n";' file.html

Eine HTML Datei, die so aussieht

```
<html><p>some text here</p></html>
```

wird zu:

```
<html>
<p>
some text here</p>
</html>
```

Wichtig ist hier nicht die Lesbarkeit! Für den Softwareentwickler ist es wichtig, dass die Daten Tag für Tag an die Funktionen übergeben werden. Damit wird es z.B. einfach, nach "<a href= ..." zu suchen, selbst wenn der Original HTML−Code "a" und "href" auf zwei verschiedenen Zeilen hatte.

Das Verändern von "$/" (INPUT_RECORD_SEPARATOR) verursacht keinen zusätzlichen Verarbeitungsaufwand und ist sehr schnell. Es ist auch möglich, den Match−Operator und Regular Expressions als Iterator zu benutzen. Das ist etwas komplizierter und langsamer, wird aber auch gerne benutzt.

Wo ist das Problem?? Im Titel dieses Artikels steht HTML::TagReader aber jetzt haben wir die ganze Zeit über eine viel einfachere Lösung gesprochen, die keine zusätzlichen Module erfordert. Irgendetwas muss faul an der bisherigen Lösung sein:

- Fast alle HTML Dateien, die es gibt, sind fehlerhaft. Es gibt z.B. Millionen von Seiten, die C−code Beispiele enthalten und auf HTML−Codeebene so aussehen:
  `if ( limit > 3) ....`
  anstelle von
  `if ( limit > 3) ....`
  In HTML sollte "<" einen Tag beginnen und ">" ihn beenden. Keines dieser Zeichen sollte so im Text auftauchen. Die meisten HTML−Browser stellen die Seite, falls möglich, trotzdem richtig dar und verstecken damit den Fehler.

- Wenn man "$/" ändert, dann betrifft das das ganze Programm. Möchte man nun eine andere Datei zeilenweise lesen, während man die HTML Datei liest, so hat man ein Problem.

Mit anderen Worten es ist nur in Spezialfällen möglich "$/" (INPUT_RECORD_SEPARATOR) zu benutzen.

Trotzdem habe ich hier ein nützliches Beispielprogramm, das genau das benutzt, was wir bisher besprochen haben. Es setzt jedoch "$/" auf "<" weil die meisten Browser ein fehlplaziertes "<" nicht so gut handhaben könne wie ein ">". Deshalb gibt es viel weniger Seiten, in denen ein "<" an der falschen Stelle sitzt. Das Programm nennt sich tr_tagcontentgrep (Klick zum Ansehen). In dem Code kann man auch sehen, wie man die Zeilennummer beim Lesen der Datei mithalten kann. tr_tagcontentgrep kann man benutzen, um nach einem String innerhalb eines Tags zu "grep–en". Z.B. nach "img". Das funktioniert dann auch, wenn der Tag sich über viele Zeilen erstreckt. So etwas wie:

tr_tagcontentgrep –l img file.html
index.html:53: <IMG src="../images/transpix.gif" alt="">
index.html:257: <IMG SRC="../Logo.gif" width=128 height=53>

HTML::TagReader

HTML::TagReader löst die Probleme mit dem Überschreiben des INPUT_RECORD_SEPARATOR und bietet eine viel schönere Schnittstelle, um Tags von Text zu unterscheiden. Es ist nicht so schwergewichtig wie ein vollständiger HTML::Parser und bietet genau das, was man zum Verarbeiten von HTML–Code braucht: Eine Methode, um Tag für Tag zu lesen.

Genug Worte. So benutzt man es. Zuerst schreibt man
use HTML::TagReader;
, um das Modul zu laden. Danach ruft man
my $p=new HTML::TagReader "filename";
auf, um die Datei "filename" zu öffnen und erhält eine Objektreferenz zurück in $p. Nun kann man $p–>gettag(0) oder $p–>getbytoken(0) aufrufen, um den nächsten Tag zu lesen. gettag gibt nur Tags zurück (Das Zeug zwischen < und >). getbytoken hingegen gibt uns auch den Text zwischen den Tags und sagt uns, ob das jetzt ein Tag ist oder Text. Mit diesen Funktionen ist es ganz einfach, HTML Dateien zu lesen. Das ist essenziell, um eine größere Website zu unterhalten. Eine vollständige Beschreibung der Syntax findet sich in der man–Page von HTML::TagReader.

Hier ist ein echtes Beispielprogramm. Es gibt den Titel von HTML Dokumenten aus:

```
#!/usr/bin/perl -w
use strict;
use HTML::TagReader;
#
die "USAGE: htmltitle file.html [file2.html...]\n" unless($ARGV[0]);
my $printnow=0;
my ($tagOrText,$tagtype,$linenumber,$column);
#
for my $file (@ARGV){
  my $p=new HTML::TagReader "$file";
  # read the file with getbytoken:
  while(($tagOrText,$tagtype,$linenumber,$column) = $p->getbytoken(0)){
  if ($tagtype eq "title"){
    $printnow=1;
    print "${file}:${linenumber}:${column}: ";
    next;
  }
  next unless($printnow);
  if ($tagtype eq "/title" || $tagtype eq "/head" ){
    $printnow=0;
```

```
    print "\n";
    next;
  }
  $tagOrText=~s/\s+/ /; #kill newline, double space and tabs
  print $tagOrText;
  }
}
# vim: set sw=4 ts=4 si et:
```

Wie es funktioniert? Wir lesen die HTML Dateien mit $p–>getbytoken(0) und wenn wir ein <title> oder <Title> oder <TITLE> (sie werden alle als $tagtype eq "title" erkannt) haben, dann setzen wir ein Flag ($printnow), um mit der Ausgabe anzufangen. Wenn wir </title> finden, hören wir mit dem Drucken auf. Man benutzt das Programm so:

htmltitle file.html somedir/index.html
file.html:4: the cool perl page
somedir/index.html:9: joe's homepage

Natürlich ist es möglich, den tr_tagcontentgrep von oben mit HTML::TagReader zu implementieren. Es ist ein bisschen kürzer und einfacher zu schreiben:

```
#!/usr/bin/perl -w
use HTML::TagReader;
die "USAGE: taggrep.pl searchexpr file.html\n" unless ($ARGV[1]);
my $expression = shift;
my @tag;
for my $file (@ARGV){
  my $p=new HTML::TagReader "$file";
  while(@tag = $p->gettag(0)){
    # $tag[0] is the tag (e.g <a href=...>)
    # $tag[1]=linenumber $tag[2]=column
    if ($tag[0]=~/$expression/io){
      print "$file:$tag[1]:$tag[2]: $tag[0]\n";
    }
  }
}
```

Das Script ist sehr kurz und hat nicht viel Fehlerbehandlung, aber ansonsten ist es voll funktionsfähig. Um nach Tags zu suchen, die den String (regexp) "gif" enthalten, benutzt man:

taggrep.pl gif file.html
file.html:135:15: <img src="images/2doc.gif" width=34 height=22>
file.html:140:1: <img src="images/tst.gif" height="164" width="173">

Noch ein Beispiel? Hier ist ein Programm, das all die <font...> und Tags aus HTML Seiten entfernt. Diese font Tags werden manchmal in Unmengen von schlecht entwickelten grafischen HTML Editoren benutzt und verursachen Probleme, wenn man die Seiten mit unterschiedlichen Webbrowsern anschaut. Dieses einfache Script entfernt alle font Tags. Man kann es ändern, so dass es nur die entfernt, die fontface oder size setzen und color unverändert lassen.

```
#!/usr/bin/perl -w
use strict;
use HTML::TagReader;
# strip all font tags from html code but leave the rest of the
# code un-changed.
die "USAGE: delfont file.html > newfile.html\n" unless ($ARGV[0]);
my $file = $ARGV[0];
```

```
my ($tagOrText,$tagtype,$linenumber,$column);
#
my $p=new HTML::TagReader "$file";
# read the file with getbytoken:
while(($tagOrText,$tagtype,$linenumber,$column) = $p->getbytoken(0)){
  if ($tagtype eq "font" || $tagtype eq "/font"){
    print STDERR "${file}:${linenumber}:${column}: deleting $tagtype\n";
    next;
  }
  print $tagOrText;
}
# vim: set sw=4 ts=4 si et:
```

Wie du sehen kannst, ist es sehr einfach, ein nützliches Programm mit nur wenigen Zeilen zu schreiben. Der Quellcode von HTML::TagReader (siehe Referenzen) enthält einige Applikationen von HTML::TagReader:

- tr_blck −− Prüft HTML Seiten auf zerbrochene relative Links
- tr_llnk −− Listet alle Links in HTML Seiten
- tr_xlnk −− Expandiert Links auf Verzeichnisse in Links auf die Indexdatei im Verzeichnis.
- tr_mvlnk −− Modifiziert die Tags in HTML Seiten mit Perl Befehlen.
- tr_staticssi −− expandiert die SSI Direktiven #include virtual und #exec cmd und produziert eine statische HTML Seite.
- tr_imgaddsize −− fügt width=... und height=... zu <img src=...> hinzu

tr_xlnk und tr_staticssi sind sehr nützlich, wenn man eine CD−ROM von einer Webseite machen möchte. Ein Webserver liefert z.B. http://www.linuxfocus.org/index.html, selbst wenn man nur http://www.linuxfocus.org/ (ohne index.html) getippt hat. Wenn man jedoch einfach alle Verzeichnisse und Dateien auf CD brennt und mit dem Browser dann nach (file:/mnt/cdrom/) geht, dann sieht man ein Verzeichnis und das passiert nicht nur beim ersten Mal, sondern jedes Mal, wenn man auf einen Link klickt, der nicht auf eine HTML Datei zeigt. Die Firma, die die ersten LinuxFocus CDs gemacht hat, hatte diesen Fehler begangen und es war furchtbar, die CD zu benutzen. Nun erhalten sie die Daten expandiert mit tr_xlnk und die CD funktioniert super.

Ich bin sicher, dir wird HTML::TagReader gefallen. Frohes Programmieren!

Referenzen

- Die man−Page von HTML::TagReader
- Perl Tutorial: Perl III (Jan. 2000)
- Das tr_tagcontentgrep Programm (das, welches nicht HTML::TagReader benutzt): tr_tagcontentgrep (txt) oder tr_tagcontentgrep (html)
- Der Quell−Code von HTML:TagReader:
  http://cpan.org/authors/id/G/GU/GUS/
  oder
  http://main.linuxfocus.org/~guido/
- Tidy ist essenziell, wenn man Webseiten entwickelt: tidy, ein Syntackchecker für HTML
  Wie man tidy benutzt? Einfach:
  `tidy -e file.html`
  druckt Fehlermeldungen
  `tidy -im -raw file.html`
  das wird die Datei editieren und sie hübsch einrücken. Tidy wird dabei auch versuchen, Fehler zu korrigieren (soweit Tidy erraten kann, was wohl gemeint war).

2005−01−11, generated by lfparser_pdf version 2.51