öS 62,– sfr 7,50 **DM 7,50** H 10554

Mai 1997

5

Mehr als EMail:
MS Exchange 5.0

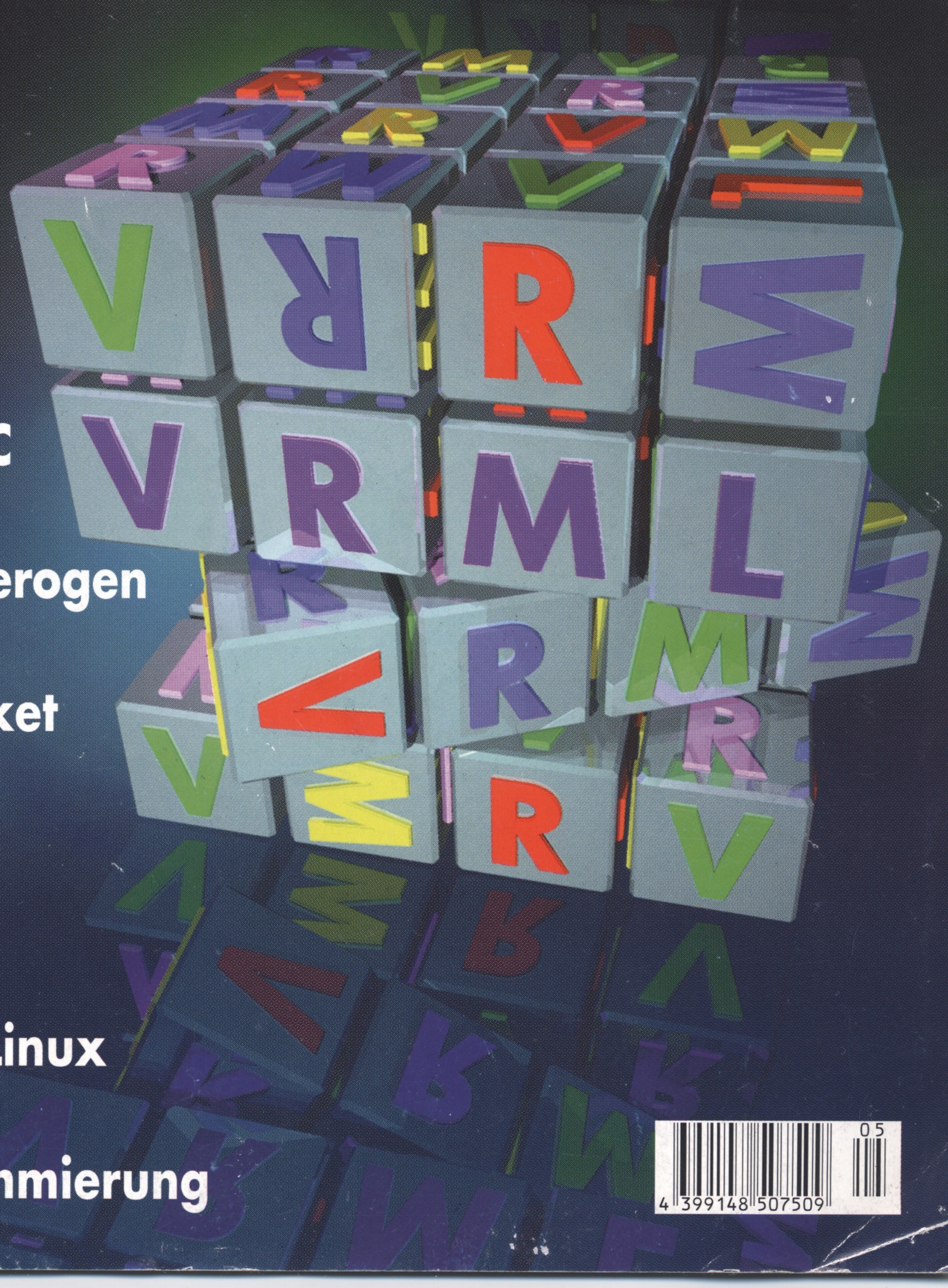
iX

MAGAZIN FÜR PROFESSIONELLE INFORMATIONSTECHNIK

Neue Serie:
Perl-Know-how
Plattformübergreifendes Scripting

VRML 2, Netscape Communicator und die Alternativen:

3D im Web



Erfahrungsbericht:

Unix-NT-Migration

AH, ESP und SKIP:

IPv6-Sicherheit

Getestet:

Linux auf PowerPC

LSF-Einsatz:

Lastverteilung heterogen

Khoros:

Visualisierungspaket

Intranet:

Netscape LiveWire
Java-Benchmarks

Kommerzielle RDBMS:

Datenbanken für Linux

Apple-Zukunft:

OpenStep-Programmierung

iX Magazin für professionelle Informationstechnik
http://www.heise.de/ix/
HEISE

4 399148 507509 05

Webseiten kontrollieren

Altbacken oder aktuell?

Michael Schilli

Ändert sich der Inhalt einer Webseite, gibt es drei Möglichkeiten, dies zu erfahren: Entweder ein freundlicher Kollege macht einen darauf aufmerksam, man klappert selbst in regelmäßigen Zeitabständen interessierende URLs mit einem Browser ab oder man nutzt das Perl-Skript *urlchk.pl*.

Wer hinsichtlich bestimmter Webseiten immer auf dem aktuellen Stand sein will, dürfte häufiges manuelles Kontrollieren der entsprechenden URLs ziemlich ermüdend finden. Das Perl-Skript *urlchk.pl* zeigt, wie sich diese Arbeit automatisieren läßt: Es nimmt als Parameter eine Reihe von URLs entgegen, überprüft die zugehörigen Dokumente und meldet etwaige Veränderungen. Hierzu holt es Informationen über die angegebenen Seiten ein und vergleicht diese mit früher aufgezeichneten Werten. So führt der Aufruf

```
urlchk.pl http://www.perl.com
```

im Fall einer Veränderung von http://www.perl.com zur Meldung

```
http://www.perl.com changed
```

während das Skript sonst stumm bleibt. Für Testzwecke schaltet die Option *–v* in den 'verbose'-Modus, in dem das im übrigen sehr wortkarge Programm auch dann einige zusätzliche Informationen produziert, wenn sich nichts geändert hat:

```
$ urlchk.pl –v http://www.perl.com \
  http://www.netscape.com
Checking http://www.perl.com ... changed.
Checking http://www.netscape.com ... unchanged.
```

Welche Datei das Skript in diesen Fällen zu sehen bekommt, entscheidet der jeweilige HTTP-Server. In seinen Konfigurationsdateien ist festgelegt, ob er *index.html*, *homepage.html* oder gar nichts liefert, wenn der Client nur einen Verzeichnisnamen übergibt. Als weitere Option ist *–once* verfügbar. Sie veranlaßt *urlchk.pl*, nur einmal wirklich nachzusehen und bei weiteren Aufrufen am selben Tag kommentarlos zurückzukehren. In das *.profile* (respektive *.bashrc*, *.kshrc* oder *.cshrc*) der Login-Shell eingebunden, kontrolliert der Aufruf

```
urlchk.pl –once http://www.microsoft.com \
  http://www.heise.de \
  http://www.addison-wesley.de
```

nur frühmorgens beim ersten Einloggen die angegebenen URLs, während er sonst weder den Rechner noch das Internet belastet.

Das Listing zeigt *urlchk.pl*. Zeile 10 definiert mit der Variablen *$pfile* eine Datei, in der das 'Gedächtnis' des

Perl-Know-how

Unter diesem Titel erscheinen in lockerer Folge Hinweise zum Einsatz von Perl. Ein Schwerpunkt liegt dabei auf Internet- und WWW-Applikationen. Wer eigene Perl-Skripts, auch zu anderen Themen, oder Anregungen hat, ist eingeladen, sich zu wenden an

ck@ix.heise.de oder
hb@ix.heise.de

iX-TRACT

- Manuelles Überprüfen interessanter Webseiten auf Neuerungen ist mühsam und langweilig. Ein Perl-Skript automatisiert diese Arbeit.
- *urlchk.pl* nutzt zwei Felder des HTML-Headers, um festzustellen, ob sich das entsprechende Dokument seit der letzten Kontrolle geändert hat.
- Frei verfügbare, aktuelle Perl-Versionen gibt es nicht nur für Unix, sondern auch für Windows (NT/95) und MacOS. Manche Unix-Funktionen sind in diesen Varianten nicht verfügbar.

Skripts liegt: der später ins Leben gerufene persistente Hash *%MEM*. Als Implementierung der *dbm*-Datei, die den Hash nach Programmende sichert, wählt das Skript die GNU-Variante *GDBM_File*. Deren Funktionalität enthält das in Zeile 13 eingebundene Modul *GDBM_File.pm*; es ist Bestandteil der Perl-Standard-Distribution.

Hilfestellung durch WWW-Modul

Zeile 12 bindet den *LWP::Useragent* ein, ein praktisches Perl-Werkzeug, mit dem sich Webseiten kinderleicht vom Netz holen lassen. Er ist Teil der Bibliothek *libwww* von Gisle Aas und über das Comprehensive Perl Archive Network (CPAN) zu beziehen (siehe Kasten 'Quellen'). Zeile 14 holt mit *Fcntl.pm* die POSIX-Konstanten *O_CREAT* und *O_RDWR* ins Skript. Sie veranlassen den in Zeile 19 folgenden *tie*-Befehl, die Datei, die den persistenten Hash *%MEM* beherbergt, zum Lesen sowie zum Schreiben zu öffnen und sie notfalls anzulegen. Zeile 16 aktiviert den 'Output-Autoflush'. Dieser Modus puffert die Standardausgabe nicht bis zum nächsten Zeilenende, sondern läßt auch Texte ohne Newline sofort erscheinen. Dies macht die Ausgaben des Skripts im 'verbose'-Modus etwas lebendiger, da so auf 'Checking URL …' erst nach der Verzögerung durch den Netzzugriff das Ergebnis *(un)changed* folgt.

Kommandozeilenparameter verarbeiten die Zeilen 23 bis 26; sie setzen für *–v* die Variable *$verbose* sowie *$once_a_day* für *–once*. Zwei *grep*-Befehle analysieren die Kommandozeile und

```
 1  #!/usr/bin/perl -w
 2  ######################################################
 3  # urlchk.pl - URLs auf Veränderungen hin überprüfen
 4  ######################################################
 5  # urlchk.pl [-v] [-once] URL
 6  #   -v:    verbose
 7  #   -once: nur einmal täglich
 8  ######################################################
 9
10  $pfile = "$ENV{HOME}/.urlchk"; # Pfad für Persistenz-Datei
11
12  use LWP::UserAgent;             # WWW Zugriffe
13  use GDBM_File;                  # Persistenter Hash
14  use Fcntl;                      # O_CREAT, O_RDWR usw.
15
16  $| = 1;                         # Ausgaben nicht puffern
17
18                                  # Persistenten Hash öffnen
19  tie(%MEM, GDBM_File, $pfile, O_CREAT|O_RDWR, 0644) ||
20        die "Cannot open $pfile";
21
22                                  # Kommandozeilenparameter analysieren
23  $verbose    = 0;                # Ausgaben erwünscht?
24  $once_a_day = 0;                # Nur einmal täglich?
25  @ARGV = grep { !(/^-v$/ && ($verbose = 1)) } @ARGV;
26  @ARGV = grep { !(/^-once$/ && ($once_a_day = 1)) } @ARGV;
27
28                                  # Nur einmal täglich
29  if($once_a_day && $MEM{'julday'} == (localtime(time))[7]) {
30    print "No more checks today\n" if $verbose;
31    exit 1;
32  }
33                                  # Datum speichern
34  $MEM{'julday'} = (localtime(time))[7];
35
36
37  while($url = shift @ARGV) {     # URLs abklappern
38
39    print "Checking $url ... " if $verbose;
40
41    $ua = LWP::UserAgent->new(); # UserAgent erzeugen
42
43                                  # URL prüfen
44    die "Invalid URL" unless
45      $ua->is_protocol_supported($url =~ /^(.+?):/);
46
47                                  # URL festlegen
48    $request = HTTP::Request->new('HEAD', "$url");
49
50                                  # Netzzugriff ausführen
51    $response = $ua->request($request) || die "Request failed: $url";
52
53                                  # Fehlerprüfung
54    die "Cannot get $url" unless $response->is_success;
55
56                                  # Zeitstempel holen und vergleichen
57    if(defined($dat = $response->header('Last-Modified'))) {
58      result($url, $dat ne $MEM{"$url.dat"});
59      $MEM{"$url.dat"} = $dat;
60
61    } else {
62                                  # Kein Zeitstempel - Länge ermitteln
63      unless(defined($len = $response->header('Content-Length'))) {
64                                  # Kein Längenfeld im Header -
65                                  # Dokument holen und Länge bestimmen
66        $request  = HTTP::Request->new('GET', "$url");
67        $response = $ua->request($request) ||
68        die "Request failed: $url";
69        die "Cannot get $url" unless $response->is_success;
70        $len = length($response->content());
71      }
72
73      result($url, $len ne $MEM{"$url.len"});
74      $MEM{"$url.len"} = $len;
75    }
76  }
77
78  untie(%MEM);                    # Persistenten Hash schließen
79
80
81  ######################################################
82  sub result {                    # Ergebnis ausgeben
83  ######################################################
84      my $url     = shift;
85      my $changed = shift;
86
87      if($changed) {
88        print "$url " unless $verbose;
89        print "changed.\n";
90      } else {
91        print "unchanged.\n" if $verbose;
92      }
93  }
```

In diesem Skript ist gegebenenfalls die verwendete Bibliothek anzupassen: statt GDBM sollte SDBM immer funktionieren. Unter Windows NT gibt es kein $HOME, der Mac mag '/' nicht als Verzeichnistrenner.

Zwischen den Welten

Obwohl Perl aus der Unix-Welt stammt, ist es nicht mehr nur dort heimisch. Mehr oder minder aktuelle Versionen gibt es inzwischen für Windows (NT/95) und MacOS (siehe nächste Seite). Wer sich für die NT-Version interessiert, sollte sich die lauffähige Version *ports/win32/Perl5/beta/Pw32i304.EXE* vom nächsten Comprehensive Perl Archive Network (CPAN) besorgen; hartgesottene Hacker erhalten dort den Quellcode zum Übersetzen als *ports/win32/Perl5/beta/Pw323s304.zip*.

Perl-Code ist im allgemeinen unabhängig vom Betriebssystem, offensichtlich ist jedoch schon auf den ersten Blick der Unterschied beim Ausführen von Skripts. Während unter Unix ein

```
#!<vollständiger Pfad>/perl
```

In der ersten Zeile den Kernel auffordert, die folgenden Zeilen dem Perl-Interpreter zur Ausführung zu überlassen, bietet Windows diese Möglichkeit nicht. Das Installationsskript trägt zwar eine Verknüpfung zwischen der Dateiendung 'pl' und dem Interpreter ein, so daß ein Doppelklick auf entsprechende Dateien sie automatisch an Perl zur Ausführung übergibt. Trotzdem empfiehlt es sich, Perl-Programme von der Kommandozeile aus mit

```
perl <script>
```

zu starten: Jede vom Skript erzeugte Ausgabe, die unter Unix auf *stdout* oder *stderr* landet, wäre unter Windows nicht lange genug sichtbar. Das läßt sich sehr schön zeigen, wenn man ein beliebiges Perl-Skript beispielsweise im Explorer per Doppelklick startet: Das DOS-Fenster öffnet sich kurz und verschwindet mit dem Ende des Skripts sofort wieder.

Unterschiede zwischen der Windows- und der Unix-Version von Perl zeigen sich vor allem in den Bereichen Netzwerk, Prozeßkommunikation und Dateisystem. So fehlen unter Windows *...hostent()*, *...protoent()* und ähnliche Routinen, *msg...*, *shm...* und *sem...* für den Nachrichtenaustausch, Shared Memory sowie Semaphoren. *chmod* und *link* gibt es nicht, da Windows andere Zugriffsrechte hat und keine Links kennt. Welche Funktionen nicht vorhanden sind, ist der Datei *doc\win32\status* zu entnehmen. Durch Inspektion des *Config*-Hash ist leider nicht herauszubekommen, ob ein bestimmter Aufruf möglich ist – er enthält nur einen Teil der unter Unix verfügbaren Einträge, und ausgerechnet die für diese Aufgabe nötigen fehlen. Defensives Programmieren ist also gefragt, sollen die Skripts portabel bleiben. Eine Verteidigungslinie könnte so aussehen:

```
eval 'gesthostent()';
die "Fehler $@" if ($@);
```

Die Windows-Implementierung zeichnet sich jedoch nicht nur durch abwesende Features aus – sie stellt im Modul *Win32* auch einige speziell auf dieses Betriebssystem zugeschnittene Funktionen bereit. Dazu gehören solche, die Zugriff auf die NT-Registry gestatten ebenso wie jene zum Starten und Beenden von Prozessen. In diesem Modul finden sich gleichfalls Routinen, die die Nutzung von Semaphoren erlauben, Benutzerverwaltung per Skript ermöglichen und auf Änderungen im Dateisystem reagieren.

Mit einem Detail müssen sich Perl-Entwickler bei der Portierung zwischen Unix und Windows immerhin nicht herumärgern: Dateinamen in den Quellen dürfen so bleiben wie sie sind, denn für Windows-Perl ist '/' so gut wie '\'.

Wer sich für Perl unter Windows interessiert, findet hier weitere Informationen, unter anderem Verweise auf Mailing-Listen:

http://www.endcontsw.com/people/evangelo/Perl_for_Win32_FAQ.html

http://www.hip.activeware.com/

Christian Kirsch

entfernen eventuell gefundene Optionen gleich aus dem Array *@ARGV*, indem der zugehörige Codeblock *0* zurückliefert, falls eine Option erkannt und der entsprechende Wert gesetzt wurde.

Header-Infos minimieren Netzverkehr

Anschließend folgt die 'Einmal-am-Tag'-Logik. Hierzu enthält der persistente Hash *%MEM* unter dem Schlüssel *julday* den Tag des Jahres (1–366) zum Zeitpunkt des letzten Skriptaufrufs. Ist dieser gleich dem aktuellen julianischen Datum, wurde das Skript am selben Tag bereits aufgerufen und bricht ab. Sonst speichert *urlchk.pl* den neuen Wert im Hash und fährt mit der Bearbeitung fort.

In Zeile 37 beginnt die eigentliche Analyse der interessierenden Webseiten. Um überflüssigen Netzverkehr zu vermeiden, versucht das Skript zunächst, anhand der Informationen im HTTP-Header herauszufinden, ob sich seit der letzten Überprüfung der Zeitstempel (Header-Feld *Last-Modified*) verändert hat. Fehlt dieser, weil ihn der HTTP-Server nicht mitliefert, greift es auf das Header-Feld *Content-Length* zu, welches die Länge des Dokuments enthält. Das ist sinnvoll, da sich mit dem Inhalt einer Webseite in den meisten Fällen deren Größe ändert. Gibt es auch diesen Wert nicht im Header, holt *urlchk.pl* das Dokument selbst und bestimmt seine Länge.

Anschließend vergleicht das Skript die so ermittelten Werte mit den in *%MEM* parat liegenden Ergebnissen früherer Anfragen. Dort sind das letzte Modifikationsdatum einer Webseite oder ihre Länge in Bytes unter den Schlüsseln *URL.dat* beziehungsweise *URL.len* gespeichert, wobei *URL* für den URL des Dokuments steht:

```
$MEM{'URL.dat'} = 'Thu, Jan 30 00:27:25 PST 1997';
$MEM{'URL.len'} = 13221;
```

In Zeile 41 ensteht ein neues Objekt vom Typ *UserAgent*. Seine Methode *is_protocol_supported* prüft, ob der Agent das im URL spezifizierte Protokoll beherrscht und bricht das Programm ab, falls dies nicht der Fall ist. *is_protocol_supported* verarbeitet keine URLs, sondern Protokollnamen wie *http*, *ftp* oder *gopher*. Deshalb extrahiert der Ausdruck

```
$url =~ /^(.+?):/
```

aus dem URL vorher noch sämtliche Zeichen vor dem ersten Doppelpunkt – man beachte den non-greedy Operator *+?*, der in diesem Fall nur eine minimale Anzahl von Nicht-Doppelpunkten zuläßt.

Zeile 47 definiert ein Objekt der Klasse *HTTP::Request*, die der *LWP::UserAgent* bereits kennt und die so ohne einen expliziten *use*-Befehl bereitsteht. Als HTTP-Zugriffsmethode legt es *HEAD* fest, da nicht das Dokument selbst, sondern nur die Header-Informationen interessieren.

Drei Zeilen später findet der eigentliche Zugriff aufs Internet statt. Fehlt das Änderungsdatum, prüft der *else*-Zweig ab Zeile 61, ob wenigstens die Dokumentenlänge verfügbar ist. Wenn nicht, erzeugt das Skript ein neues Objekt vom Typ *HTTP::Request*, diesmal mit *GET* als Zugriffsmethode, um das Dokument selbst zu holen und anschließend seine Länge zu ermitteln. Der *untie*-Befehl in Zeile 76 sichert zum Abschluß den aktuellen Hash in der Datei und löst die Verbindung zwischen den beiden.

urlchk.pl analysiert übrigens nicht nur URLs, die auf Webseiten verweisen. Dank der transparenten Behandlung des FTP-Protokolls durch die *libwww* verarbeitet es zudem mittels 'ftp://…' definierte Dokumente. Auch der Inhalt von Verzeichnissen liegt in Form von Dokumenten vor, die (manchmal) ein Modifikationsdatum enthalten und (immer) eine Länge haben. So stellt

```
urlchk.pl ftp://ftp.leo.org/pub/comp/ \
  programming /languages/perl/CPAN/src/5.0
```

fest, ob es im Perl-Distributionsverzeichnis etwas Neues gibt. Hierbei wäre es einerlei, ob eine neue Release nun *perl-5.003a* oder *perl-5.004* hieße – das Verzeichnis enthielte eine neue Datei und käme mit veränderter Länge zurück. Lauffähig ist *urlchk.pl* mit den Unix- und Windows-Versionen von Perl; auf dem Mac kam es zu bisher nicht erklärbaren Fehlermeldungen. An den Ergebnissen von CGI-Abfragen kann das Skript scheitern, etwa

```
urlchk.pl 'http://www.bank24.de/cgi-bin/ \
  cgichart?type=5&dsatz=1&sort=1&'
```

Die Antwort auf diese Frage nach dem aktuellen Kurs der 3Com-Aktie enthält keinen der beiden Header, und ihre Länge ändert sich höchstens, wenn der Kurs sich drastisch verbessert oder verschlechtert. (ck)

MICHAEL SCHILLI

ist Softwareentwickler bei Black Sun Interactive, San Francisco.

INGO BORDACH

studiert Informatik und ist freier Mitarbeiter der CUBiC GmbH. Schwerpunkte seiner Tätigkeit sind Unix-Systemverwaltung und Netzwerkadministration.

STEFAN PANTKE

ist Geschäftsführer der CUBiC GmbH, die sich mit der Entwicklung individueller Netzwerksystemlösungen insbesondere auf Java-Basis beschäftigt.

Literatur

[1] Randal L. Schwartz: Monatliche Kolumnen in 'Web Techniques'; http://www.teleport.com/~merlyn/WebTechniques

[2] Michael Schilli: Effektives Programmieren mit Perl 5; Addison-Wesley 1996; ISBN 3-8273-1095-4

Perl auf dem Mac

Es ist eine verbreitete Meinung, daß Apples Macintosh-Rechner nur für DTP und Grafikbearbeitung geeignet ist. Tatsächlich existiert ein sehr breites Angebot an Tools für TCP/IP-Netzwerker und an Programmiersprachen. Unter anderem sind Implementierungen von Perl 4 und Perl 5 verfügbar. Das aktuelle Perl 5 ist unter dem Namen 'MacPerl' in der Version 5.1.3r2 auf dem FTP-Server ftp://ftp.perl.com/CPAN/ports/mac/ zu finden. Es wurde von Matthias Neeracher (neeri@iis.ee.ethz.ch) entwickelt und setzt die Betriebssystemversion 7.x voraus. MacPerl arbeitet mit 'Internet Config' zusammen, das den Zugriff verschiedener Applikationen auf Internet-Einstellungen (zum Beispiel die Adresse des WWW-Proxies) vereinheitlicht und vereinfacht.

Mehr Komfort als der Stammvater

Im Gegensatz zu Unix-Implementierungen ist MacPerl eine integrierte Entwicklungsumgebung, die Texteditor, Syntax-Checker, Runtime-System und Debugger enthält. Das System öffnet nach erkannten Syntaxfehlern direkt den Texteditor und markiert die fragliche Zeile. Das verkürzt den Edit-Compile-Run-Zyklus erheblich und beschleunigt die Entwicklung von Perl-Skripts. Insbesondere für Neueinsteiger ist die kontextsensitive Online-Hilfe im POD-Format (Plain Old Document) sehr hilfreich. Diese Hilfedateien besitzen eine ähnliche Struktur wie *man*-Seiten, Hyperlinks zwischen POD-Dateien sind aber leider nicht vorhanden.

MacPerl ist erfreulich gut in die Netzwerkwelt von System 7.x integriert. Neben den Standard-Socket-Familien UDP und TCP gibt es Unterstützung für PPC (Program to Program Communication) und ADSP (Apple Data Stream Protocol, AppleTalk). Somit kann ein Perl-Programm Daten mit einer beliebigen Macintosh-Applikation austauschen, wenn dies im konkreten Fall auch sehr viel Detailwissen über die Toolboxen des System 7.x voraussetzen dürfte. MacPerl wurde um weitere Mac-spezifische Funktionen erweitert. Beispielsweise sind Routinen zur Erzeugung von Dialogboxen und zur Bedienung der seriellen Schnittstelle verfügbar. Perl-Programme sind nicht nur als normale Textfiles speicherbar, sondern auch als *droplets* und *runtimes*. Letztere sind eigenständige Programme, die Perls Runtime-System und das Skript enthalten. *Droplets* hingegen sind ebenfalls eigene Programme, sie erfordern aber noch eine externe Runtime-Datei. *Runtimes* sind somit das Mittel der Wahl zur Distribution abgeschlossener Applikationen. MacPerl ist via AppleScript steuerbar (scriptable) und AppleScript-Code kann aufgezeichnet werden (recordable). Aufgrund der Unix-Herkunft von Perl ergeben sich naturgemäß einige Probleme, die eine ausführliche und Mac-spezifische README-Datei dokumentiert. Programme, die das '/' als Pfadbegrenzer benutzen, führen stets zu Fehlern auf dem Mac, da dieser statt dessen den Doppelpunkt ':' erwartet. Hier ist Handarbeit gefragt – oder ein pfiffiges Modul, das alle Dateinamen korrigiert, bevor sie das Betriebssystem zu sehen bekommt.

Aufgrund der Architektur von Apples System 7.x sind diverse Unix-Funktionen nicht ausführbar. Das MacOS ist keine Multiuser-Umgebung, so daß entsprechende Aufrufe zum Beispiel zur Verwaltung von Zugriffs- und Besitzrechten nicht mit sinnvoller Funktionalität implementierbar sind. Ebenso ist der Start von Programmen via *exec* nicht möglich. Allgemein sind alle Funktionen heikel, die Unix-Prozesse erzeugen und steuern. Durch defensives Softwaredesign ist Perl-Code allerdings recht leicht portabel zu gestalten.

MacPerl 5 ist zu finden in *CPAN/ports/mac*, häufig gestellte Fragen beantwortet die FAQ *CPAN/doc/FAQs/mac/MacPerlFAQ.html*. Für MacOS angepaßte Quellen und Korrekturen der CPAN-Distribution finden sich auf ftp://mors.gsfc.nasa.gov/pub/MacPerl/

Ingo Bordach, Stefan Pantke

QUELLEN

Das benötigte Modul libwww und Perl selbst sind auf einem der deutschen CPAN-Spiegel erhältlich:

ftp://ftp.leo.org/pub/comp/programming/languages/perl/CPAN/
ftp://ftp.rz.ruhr-uni-bochum.de/pub/CPAN/
ftp://ftp.uni-hamburg.de/pub/soft/lang/perl/CPAN/

libwww liegt dort in modules/by-module/WWW, IO und libnet in NET, MD5 in MD5.

Für Windows NT gibt es hier eine speziell gepatchte Version der libwww:
ftp://cogen.mit.edu/software/perl/libwww-win32-fix.zip