Linux im Rechenzentrum

öS 62,– sfr 7,50 **DM 7,50** H 10554 Januar 1998

1

iX

MAGAZIN FÜR PROFESSIONELLE INFORMATIONSTECHNIK

Anschluß für die Firma:
Internet-Provider in Deutschland
Kosten, Kontakte, Bandbreiten

Marktübersicht, Praxistips:
Datensicherung im Netz
Software von 0 bis über 100 000 Mark

PC-Verwaltung:
NetPC mit Zero Administration Kit

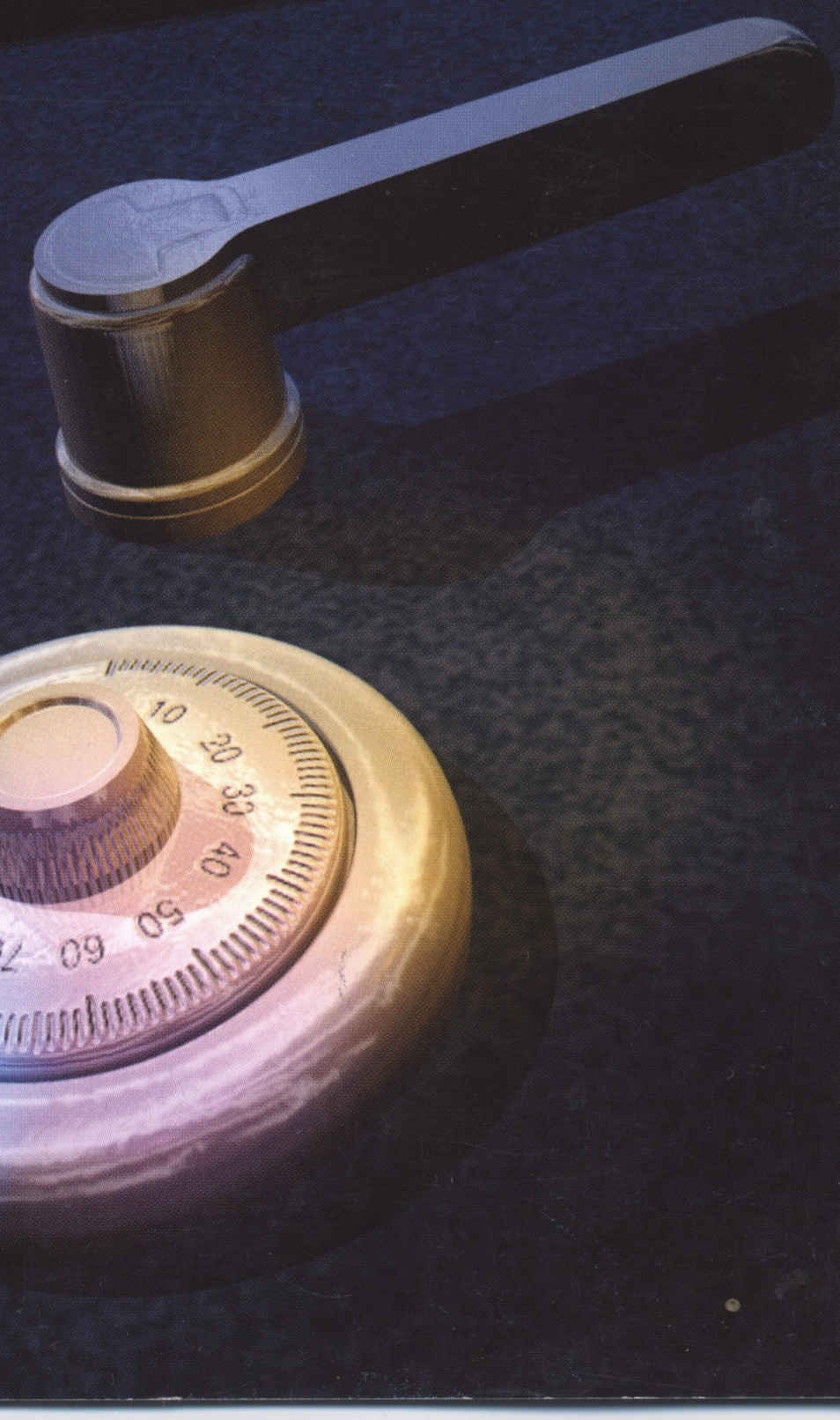
Softwareentwicklung:
Was OLE DB und ADO verändern
C++ nach Java portieren

Know-how:
Videokonferenz-Lösungen

Electronic Commerce:
Micropayment im Internet

Sicherheit:
Crack 5 knackt Paßwörter
AS/400 im Internet

Number-Crunching:
Top500-Supercomputer



4 399148 507509 01

Links in HTML-Dokumenten expandieren

Grabscher

Michael Schilli

HTML-Dokumente sind gut und schön. Nervig wird's, wenn man das ganze Handbuch oder Tutorial komplett kopieren oder drucken will. Inhaltsverzeichnis laden – speichern, erstes Kapitel laden – speichern, zweites Kapitel … Ein Perl-Script hilft, derlei Aufgaben schneller und ohne Browser zu erledigen.

Webgrab.pl holt Web-Dokumente vom Netz und extrahiert auf Wunsch in ihnen enthaltene URLs. Mit Shell-Befehlen verknüpft, lassen sich so ohne jeden Mausklick verteilte Dokumente zusammensuchen.

Die Web-Seite, auf der Randal Schwartz seine hochinteressanten Perl-Artikel online präsentiert, holt der Aufruf

```
webgrab.pl -g http://www.stonehenge.com/merlyn/
WebTechniques/
```

vom Netz und gibt sie auf STDOUT aus. Der HTML-Text verzweigt über Links zu den einzelnen Artikeln.

```
webgrab.pl -e http://www.stonehenge.com/merlyn/
WebTechniques/ >out
```

schnappt sich nicht nur die Seite, sondern analysiert deren Inhalt und schreibt gefundene Hyperlinks der Tags *<A>*, *<IMG>* und *<AREA>* in die Datei *out*, beispielsweise:

```
mailto:perl-training-info@stonehenge.com
http://www.stonehenge.com/cgi/wtsearch?search=flock
http://www.stonehenge.com/icons/text.gif
http://www.stonehenge.com/merlyn/WebTechniques/
        col01.html
http://www.stonehenge.com/merlyn/WebTechniques/
        col01.listing.txt
```

Möchte man nun nicht alle Artikel einzeln laden, sondern, um Tipparbeit und Zeit zu sparen, die gesamte Sammlung von Schwartz' Texten auf einen Schlag, löscht man mittels eines Editors in *out* alles außer den *colxx.html*-Einträgen und reicht *out* anschließend an einen weiteren *webgrab.pl*-Aufruf weiter, der die Seiten nacheinander holt:

```
webgrab.pl -g -f out -t randal.tar
```

Die mitgegebenen Optionen und Parameter veranlassen *webgrab.pl*, den Inhalt aller in der Datei *out* abgelegten URLs zu holen. Der Schalter *-t* kanalisiert den HTML-Strom mehrerer Dokumente, statt ihn einfach auf der Standardausgabe auszugeben, in eine *tar*-Datei, die wiederum die Dateien

```
merlyn/WebTechniques/col01.html
merlyn/WebTechniques/col02.html
```

enthält. Mit diesem Verfahren kann auch eine Reihe von Dokumenten, die unter anderem Bilder enthalten, auf den lokalen Rechner kopiert werden. Expandieren der *tar*-Datei legt alle notwendigen Unterverzeichnisse und Dateien an. Nach einem 'Open File' mit einem Browser auf die Einstiegsdatei darf man nach Herzenslust in Manual-Seiten oder Spezifikationen schmökern. Über relative Hyperlinks funktioniert sogar das Blättern zwischen verknüpften Dokumenten.

Dokumentation auf einen Rutsch

Um sich beispielsweise die gesamte XEmacs-Dokumentation im PostScript-Format zu besorgen, kann man die fol-

gende Kombination zweier Aufrufe von *webgrab.pl* benutzen:

```
webgrab.pl -v -g `webgrab.pl -e \
http://nis-www.lanl.gov/info/xemacs/xemacs_toc.html` \
| html2ps > emacs.ps
```

html2ps ist ein Perl-Script, das HTML in PostScript wandelt [1]. Es ist erhältlich bei *http://www.tdb.uu.se/~jan/html2ps.html*. Im Fall der XEmacs-Handbücher sollte man allerdings *html2ps* lieber die HTML-Dokumente einzeln vorwerfen – auf einem mit 64 MByte ausgestatteten Linux-Rechner brach das Script nach mehr als 30 Minuten Arbeit wegen zu wenig Arbeitsspeicher ab. Für informative Ausgaben während der Übertragung sorgt die Option *-v*. Gedächtnisschwäche beugt *-h* vor und hilft mit einer Liste erlaubter Parameter weiter.

Noch ein Tip: Ist ein angegebener URL keine Datei, sondern ein Verzeichnis, besteht *webgrab.pl* auf einem angehängten '/', sonst funktioniert die Berechnung des absoluten Pfades nicht. Also: *http://path/* statt *http://path*.

Zeile 3 bindet das Modul *Getopt::Std* ein, dessen Funktion *getopts()* in Zeile 11 die Kommandozeile nach den Optionen *-e, -f, -g, -h* und *-v* durchsucht; gegebenenfalls setzt sie den entsprechenden Skalar *$opt_** auf 1. Der Doppelpunkt hinter der *t*-Option veran-laßt *getopts()*, nach *-t filename* oder *-tfilename* zu suchen und, falls vorhanden, *$opt_t* auf *filename* zu setzen. Zeile 14 bricht das Script mit der Meldung zur Benutzung ab, falls der Anwender weder *-g* noch *-e* angegeben hat.

Auf der Kommandozeile übergebene URLs lesen die Zeilen 18 bis 24 ein. Bei gesetzter *-f*-Option hinterläßt *webgrab.pl* einfach den Dateinamen auf der Kommandozeile, denn das magische *while(<>)*-Konstrukt liest die Datei zeilenweise aus und beherrscht sogar '-', um über STDIN hereinkommende URLs zu verarbeiten.

```
 1 #!/usr/bin/perl
 2
 3 use Getopt::Std;                # Kommandozeilen-Parameter-Erfasser
 4 use LWP::Simple;                # WWW-Utility
 5 use HTML::TreeBuilder;          # HTML-Parser
 6 use Archive::Tar;               # Tar-Archivierer
 7
 8 sub info { print STDERR @_ if $opt_v; }   # Ausgabe Verbose-Modus
 9 sub err  { print STDERR @_; }             # Fehler-Ausgabe
10
11 getopts("ef:ght:v");            # Kommandozeilen-Parameter erfassen
12
13 usage() if(defined $opt_h);     # Help-Option gesetzt
14 usage() unless grep {defined} ($opt_e, $opt_g);  # Extract oder Get
15
16 my $tar = Archive::Tar->new() if $opt_t;         # Tar-Objekt erzeugen
17
18 if(defined $opt_f) {               # URLs aus einer Datei ...
19   push(@ARGV, $opt_f);             # Datei einfach an die Kommandozeile
20                                    # anhaengen.
21   while(<>) { chop; push(@urls, $_); }
22 } else {
23   push(@urls, @ARGV) || usage();  # oder URLs von der Kommandozeile
24 }
25
26 foreach $url (@urls) {
27   info "# GET URL $url ... ";
28   ($doc = LWP::Simple::get $url) ||
29      do { err "FAILED on ($url)\n"; next };
30   info "OK\n";
31
32   if($opt_t) {                          # Nicht ausgeben, => tarfile
33     my $path = URI::URL->new($url)->path;
34     $path =~ s,/$,/index.html,g; $path =~ s,^/,,g;
35     $tar->add_data($path, $doc);        # Daten ins Archiv
36     next;
37   }
38
39   if($opt_g) { print "$doc"; next; }   # Dokument ausgeben
40
41                                        # Links extrahieren
42   my $tree = HTML::TreeBuilder->new->parse($doc);
43
44                                        # <A>, <AREA> und <IMG>
45   for (@{$tree->extract_links(qw/a area img/)}) {
46     my $l = URI::URL->new($_->[0]);    # href-Attribut
47     ($s = $l->abs($url)) =~ s/#.*//g;  # URL absolut, #.. raus
48      print "$s\n" unless $links{$s}++; # Ausgeben falls neu
49   }
50
51     $tree->delete();                   # Parse-Baum loeschen
52 }
53
54 if($opt_t) {
55   $tar->write($opt_t);                 # Tarfile erzeugen
56   info "$opt_t ready.\n";              # Meldung im verbose-Mode
57 }
58
59 sub usage {
60 ###################################################################
61     $0 =~ s,.*/,,g;             # Pfad entfernen
62
63     print <<EOT;
64 usage: $0 -g [-f URLfile] [-t tarfile] URL ...  # get URLs
65        $0 -e [-f URLfile] URL ...               # extract links
66 options:
67        -h: help
68        -v: verbose
69 EOT
70     exit 1;
71 }
```

webgrab.pl: Web-Seiten ohne Browser holen.

Zeile 3 bindet das Modul *Getopt::Std* ein, dessen Funktion *getopts()* in Zeile 11 die Kommandozeile nach den Optionen *-e, -f, -g, -h* und *-v* durchsucht; gegebenenfalls setzt sie den entsprechenden Skalar *$opt_** auf 1. Der Doppelpunkt hinter der *t*-Option veran-laßt *getopts()*, nach *-t filename* oder *-tfilename* zu suchen und, falls vorhanden, *$opt_t* auf *filename* zu setzen. Zeile 14 bricht das Script mit der Meldung zur Benutzung ab, falls der Anwender weder *-g* noch *-e* angegeben hat.

Auf der Kommandozeile übergebene URLs lesen die Zeilen 18 bis 24 ein. Bei gesetzter *-f*-Option hinterläßt *webgrab.pl* einfach den Dateinamen auf der Kommandozeile, denn das magische *while(<>)*-Konstrukt liest die Datei zeilenweise aus und beherrscht sogar '-', um über STDIN hereinkommende URLs zu verarbeiten.

URLs per Kommandozeile und Datei

Ist *-f* nicht angegeben, stehen die URLs auf der Kommandozeile, und Zeile 23 übernimmt alle nach der Optionenanalyse verbliebenen Parameter in das *@urls*-Array. Für jeden URL holt nun der *get*-Befehl aus dem *LWP::Simple-Modul* (Zeile 28) das entsprechende Dokument vom Web und legt es als String in *$doc* ab.

Bei gesetzter *-g*-Option (und nur, falls nicht *$opt_t* für den *tar*-Kanal angegeben wurde) schreibt Zeile 39 den Inhalt von *$doc* einfach nach STDOUT und setzt die Bearbeitung mit dem nächsten Dokument fort.

Für *-e* zerlegt *webgrab.pl* den HTML-Text mittels eines Parsers ab Zeile 42 in einen Syntaxbaum, fischt Tags wie *<A>*, *<IMG>* und *<AREA>* heraus und untersucht deren HREF-Attribute. *HTML::TreeBuilder->new* in Zeile 38 liefert eine Referenz auf ein neues *TreeBuilder*-Objekt, dessen Methode *parse* den HTML-Text im String *$doc* analysiert und (intern) den Syntaxbaum erzeugt.

Mit einer Liste von Tags (*a, area, img*) als Parameter liefert die Methode *extract_links()* eine Referenz auf eine Liste von *($link, $ref)*-Paaren. *$link* ist hierbei der URL des Links, *$ref* zeigt auf ein *HTML::Element*-Objekt. Zeile 46 verwendet lediglich

Benötigte Module

Die vielverwendete *libwww* holt HTML-Dokumente vom Netz und wandelt relative URLs in absolute. Sie ist vom CPAN unter *CPAN/modules/by-module/WWW/libwww-perl-5.13.tar.gz* zu beziehen. Das *Archive::Tar*-Modul gibt's unter *CPAN/modules/by-module/Archive/Tar-0.06.tar.gz* Es befindet sich noch im Betastatus und mault 'Compression not available', falls das Modul *Compress::Zlib* nicht vorhanden ist. Sonst funktioniert es tadellos.

Deutsche CPAN-Spiegel sind unter anderem:

ftp://ftp.leo.org/pub/comp/programming/languages/perl/CPAN/

ftp://ftp.rz.ruhr-uni-bochum.de/pub/CPAN/

ftp://uni-hamburg.de/pub/soft/lang/perl/CPAN/

ftp://uni-erlangen.de/pub/source/Perl/CPAN/

ftp://ftp.gmd.de/packages/CPAN/

den gefundenen URL (*$_->[0]*) und erzeugt ein URL-Objekt. Die *abs()*-Methode ersetzt relative durch absolute URLs.

Zeile 47 schneidet den URL an eventuell vorkommenden #-Zeichen ab, da diese Teildokumente spezifizieren und *webgrab.pl* nur mit ganzen Dokumenten arbeitet. Der Hash *%links* in Zeile 48 speichert für jeden gefundenen URL nur einen Eintrag und sorgt so dafür, daß das Script keine Duplikate ausgibt.

Da ein HTML-Baum zirkuläre Referenzen enthalten darf und Perls Garbage-Collector diese nicht automatisch abräumt, löscht die *delete*-Methode in Zeile 51 das *TreeBuilder*-Objekt und gibt den für den Syntaxbaum beanspruchten Speicherplatz frei.

Im *tar*-Archiv bewahren

Holt *webgrab.pl* eine Reihe von Dokumenten vom Netz, stellt sich die Frage: wie speichern? Ein *tar*-Archiv bietet sich an, das die Dokumente entsprechend ihren Zugriffspfaden auf dem Web-Server in einer Archivdatei ablegt. Kürzlich stieg das Modul *Archive::Tar* von Calle Dybedahl in die heiligen Hallen des CPAN auf. Es bietet (neben den 'normalen' Funktionen von *tar*) eine Methode, die eine Zeichenkette in einem *tar*-Archiv speichert, als entstamme sie einer Datei des Filesystems – ideal für die vorliegende Aufgabe (siehe Kasten 'Benötigte Module').

Zeile 16 in *webgrab.pl* erzeugt ein *Tar*-Objekt, dessen *add_data*-Methode in Zeile 35 unter *http://host/path/file.html* gefundene HTML-Daten (gespeichert in *$doc*) mit der Pfadangabe *path/file.html* ins Archiv verfrachtet. Zeile 34 entfernt führende Pfadtrenner und benennt namenlose Dateien in *index.html* um. Die *write*-Methode in Zeile 55 schreibt das Archiv endlich in eine *tar*-Datei. (ck)

MICHAEL SCHILLI

arbeitet als Web-Engineer bei AOL Productions, San Mateo. Er ist Autor von 'Effektives Programmieren mit Perl 5'.

Literatur

[1] Thomas Merz; Philosophischer Unterschied; html2ps: HTML-Seiten in PostScript und PDF konvertieren; *iX* 12/97, S. 158 ff.