Mit Stellenmarkt

bfr 207,– hfl 10,95 lfr 205,– öS 70,– sfr 8,50 **DM 8,50** H 10554

1

Januar 2001

c't

MAGAZIN FÜR PROFESSIONELLE INFORMATIONSTECHNIK

11 MBit/s ohne Kabelsalat:

Drahtlose LANs

Vergleichstest, Technik, Linux-Anbindung

XML-Know-how:

XSLT-Tutorial

Marktübersicht:

Windows 2000 remote installieren

E-Commerce:

Sparen mit XML-Katalogen

Programmieren:

UML mit Rational Rose

Gtk meets Java

Mietsoftware:

Zukunftsmarkt oder Hype?

Getestet:

SunBlade 1000

Netscape Navigator 6

Reliant Cluster für Linux

Anschluss für die Firma:

Internet-Provider

4 399148 508506 01

c't Magazin für professionelle Informationstechnik

http://www.heise.de/ix/

HEISE

Bookmarks für die Shell

Fit im Alter

Michael Schilli

Wer sich keine armlangen Pfade für seine Unix-Projekte merken kann, dem hilft das hier vorgestellte Bookmark-Skript ins richtige Verzeichnis zu manövrieren.

Der Webserver steht in */ext/apps/nes40/https-mike*, das gerade entwickelte Projekt unter */home/mike/CVS/aol/projekt*. Außerdem spielen vielleicht noch die Logdateien in */disk2/data/log/access* eine Rolle. Wer kann sich das alles merken? Wer will jedes Mal 26 Zeichen tippen, um den Webserver neu zu starten, wenn er wegen eines fehlerhaften C-Plugin im Entwicklungsprojekt mal wieder abgestürzt ist?

Nun bietet die Shell zwar Variablen, die man mit Verzeichnisnamen belegen kann, und einmal besuchte Pfade lassen sich mit Befehlen wie *pushd* und *popd* speichern und wieder laden – doch eine idiotensichere Lösung ist das nicht.

Perl als Gedächtnisstütze

Wie wäre es mit einem Perl-Skript, das – wie ein Web-Browser – häufig besuchte Pfade in einer Bookmark-Liste speichert? Ein Skript, das die Elemente der Liste zur Auswahl stellt und auf Tastendruck auch gleich ins gewünschte Verzeichnis wechselt?

Die Sache hat nur einen Haken: Kein Programm oder Perl-Skript der Welt kann das aktuelle Verzeichnis der Shell ändern. Das heißt, Programme können natürlich intern mit *chdir()* in andere Verzeichnisse wechseln, doch es ist unmöglich, das aktuelle Verzeichnis auch *nach* Ablauf des Programms beizubehalten. Dies liegt daran, dass die Unix-Shell aufgerufene Programme in neu erzeugten Kindprozessen ausführt – Verzeichniswechsel dort wirken sich nicht auf den Vaterprozess aus.

Deshalb ist *cd* in der Shell kein Programm in */usr/bin* oder anderswo, sondern ein internes Kommando, das die Shell direkt interpretiert, ohne nach einem gleichnamigen Programm zu suchen. Fast alle Shells bieten aber mit Alias-Namen eine Möglichkeit, Kommandos zu definieren, um das aktuelle Verzeichnis dauerhaft zu wechseln. In der unter Linux üblichen *bash*-Shell entsteht zum Beispiel mit

```
alias cdt='cd /tmp'
```

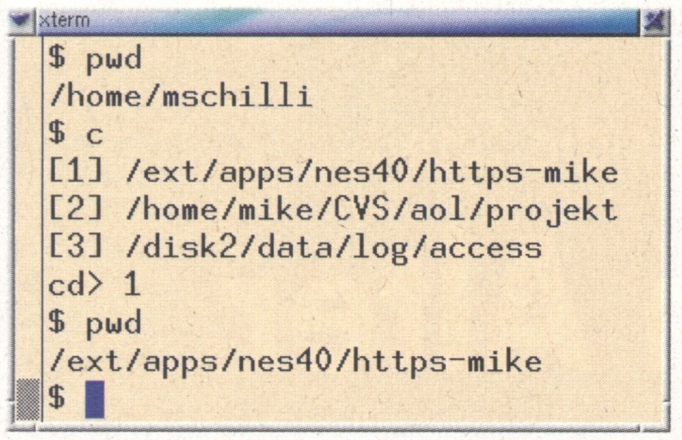
ein neues Kommando *cdt*, das beim Aufruf den Shell-Befehl *cd /tmp* absetzt und damit ins Verzeichnis */tmp* wechselt. Hier handelt es sich nicht um einen Kindprozess, sondern um reine Textersetzung, und deswegen steht die Shell nach Ausführung des Kommandos im angeforderten Verzeichnis */tmp*.

Angenommen, es gäbe ein Perl-Skript *cdbm* (für *cd Bookmark*), das eine Reihe von Pfaden anzeigt, den Benutzer einen davon auswählen lässt und das Ergebnis ausgibt. Dann definiert

```
alias c='cd `cdbm`'
```

ein neues Kommando *c*, das beim Aufruf wegen der Backquotes (‘…’) das Skript *cdbm* ausführt und dessen ausgegebenes Verzeichnis dem *cd*-Kommando übergibt, welches wiederum zum angegebenen Pfad wechselt.

Abbildung 1 zeigt das neue Kommando in Aktion: Ein einleitendes *pwd* gibt das Verzeichnis */home/mschilli* aus. Der Aufruf *c* listet drei mit Nummern versehene Verzeichnisse auf. Der



Das Bookmark-Skript in Aktion (Abb. 1)

Mit Pattern-Suche direkt wechseln (Abb. 2)

Benutzer tippt *1* für das erste Verzeichnis ein, worauf die Shell nach */ext/apps/nes40/https-mike* wechselt, wie die Ausgabe des folgenden *pwd*-Befehls zeigt.

Besser wechseln mit Argument

Schöner wärs freilich noch, wenn *cdbm* Argumente entgegennähme – sodass die dargestellte Liste zum Beispiel schon auf Verzeichnisse reduziert wäre, die einem vorgegebenen Pattern entsprechen. *cdbm log* veranlasste dann *cdbm* dazu, nur Dateien anzubieten, in denen das Pattern *log* vorkommt. Das käme vor allem bei langen Verzeichnislisten, die *cdbm* hartkodiert in einem Array speichert, gelegen.

Dabei macht die *alias*-Anweisung einen Strich durch die Rechnung, denn es ist unmöglich, durch den Aufruf von *c PATTERN* das *PATTERN* in

```
alias c='cd `cdbm PATTERN`'
```

zu ersetzen – hierzu müsste *alias* Argumente verarbeiten. Aber das tut es nicht, da es sich um eine reine Textersetzung handelt. Zu Hilfe kommt die *function*-Anweisung der *bash*, die schlauer ist als *alias* und Parameter versteht. Die Definition

```
function c () { cd `cdbm $1`; pwd; }
```

erzeugt eine Funktion *c*, die auf das Kommando *c PATTERN* hin *cdbm PATTERN* aufruft und in das zurückgegebene Verzeichnis mit *cd* hineinspringt. (Wer vorher den *alias* definiert hat, muss ihn übrigens nun mit *unalias c* wieder zurücknehmen.) Gleich darauf zeigt sie mit dem Kommando *pwd* noch das Zielverzeichnis an. Benutzung von *c* ohne *PATTERN* resultiert im Aufruf von *cdbm* ohne Parameter, dem Anzeigen der Auswahlliste und schließlich dem Sprung ins gewählte Verzeichnis – ganz wie vorher.

Bleibt die Auswahlliste wegen eines auf keinen Eintrag passenden Pattern leer, gibt *cdbm* einen Punkt (.) zurück, um die Shell-Funktion mit *cd .* dazu zu bewegen, im aktuellen Verzeichnis zu bleiben. Falls das Muster nur auf ein Element der Auswahlliste passt, stellt *cdbm* keine langen Fragen, sondern gibt dieses gleich aus, sodass die Shell-Funktion direkt dorthin springt.

Abbildung 2 zeigt diesen Fall. *log* passt nur auf das Verzeichnis */disk2/data/log/access*, wohin das Kommando *c log* springt. Bleiben zwei oder mehr Einträge übrig, bietet *cdbm* dem Benutzer die verkürzte Liste mit passenden Einträgen zur Auswahl an.

Der oben vorgestellte *function*-Befehl sollte in die *.bashrc*-Datei im Home-Verzeichnis wandern und schon steht nach dem Einloggen mit *c* ein neues Kommando bereit. Um in ein Verzeichnis zu wechseln, das *cdbm* in der angezeigten Liste enthält, tippt man einfach *c* ein (mit oder ohne Pattern) und wählt mit einer Zahl das entsprechende Verzeichnis aus.

Auswahl per Hash und Muster

Das Listing zeigt die kurze Implementierung von *cdbm*. Zeile 3 definiert das Array *@DIRS* mit allen Verzeichnissen der Bookmark-Liste. Der besseren Wartbarkeit wegen steht dort ein *qw*-Konstrukt, dessen Begrenzer (in diesem Fall runde Klammern) Array-Elemente ohne Anführungszeichen und trennende Kommata enthalten dürfen. *qw* trennt die Einzelelemente an Leerzeichen oder Zeilenumbrüchen. Kommen Verzeichnisse hinzu, die *cdbm* bedienen soll, fügt man einfach neue Zeilen mit deren Namen ein.

Falls der Benutzer ein Pattern angab, um die Liste einzuschränken, ist das Element *$ARGV[0]* definiert, und Zeile 9 verzweigt in den Rumpf der *if*-Bedingung. Dort reduziert das Kommando *grep* das Array *@DIRS* auf Einträge, die auf das vorgegebene Pattern passen. Der Modifizierer */o* sorgt dafür, dass *perl* den regulären Ausdruck nur einmal kompiliert (*o* für *once*), obwohl eine Variable darin steht – *ARGV[0]* ändert sich ja bekanntlich nicht. Das spart wertvolle Femtosekunden.

Zeile 13 prüft mit dem in skalaren Kontext gestellten Array *@DIRS*, ob dieses mittlerweile leer ist. Falls ja, druckt die folgende Zeile eine Warnung und Zeile 15 gibt einen einzelnen Punkt (.) aus, um wie erläutert den Verzeichniswechsel zu unterbinden. Zeile 17 kontrolliert, ob nur ein einzelnes Array-Element den Ansprüchen des Benutzers genügt und verzweigt in diesem Fall in den *elsif*-Rumpf. Dort fackelt *cdbm* nicht lange, sondern gibt das passende Verzeichnis aus, damit die um das Skript gewickelte Shell-Funktion sofort dorthin wechseln kann.

Blieb in *@DIRS* mehr als ein Verzeichnis zur Auswahl übrig, läuft die in Zeile 21 beginnende *foreach*-Schleife über das Array von Pfadelementen und bietet den gerade aktuellen Pfad für jeden Schleifendurchgang in der Variablen *$_* an. Die Zählervariable *$count* nummeriert die Einträge von *1* an aufsteigend. Falls *$count* noch nirgends initialisiert wurde, steht die Variable nach *++$count* auf *1* und der Rückgabewert ist ebenfalls *1*.

Zeile 23 ordnet in einer Hash-Tabelle *%files* diesen Zahlenwert wieder dem ursprünglichen Verzeichnisnamen zu, damit das Skript später anhand der Nummer auf das ausgewählte Ver-

LISTING 1

```
 1 #!/usr/bin/perl -w
 2
 3 my @DIRS = qw(
 4     /ext/apps/nes40/https-mike
 5     /home/mike/CVS/aol/projekt
 6     /disk2/data/log/access
 7 );
 8
 9 if(defined $ARGV[0]) {
10     @DIRS = grep /$ARGV[0]/o, @DIRS;
11 }
12
13 if(@DIRS == 0) {
14     warn "No match";
15     print ".\n";
16
17 } elsif (@DIRS == 1) {
18     print "$DIRS[0]\n";
19
20 } else {
21     foreach (@DIRS) {
22         print STDERR "[", ++$count, "] $_\n";
23         $files{$count} = $_;
24     }
25
26     print STDERR "[1-", scalar @DIRS, "]> ";
27     $input = <STDIN>;
28     chop($input);
29
30     if(exists $files{$input}) {
31         print "$files{$input}\n";
32     } else {
33         warn "Invalid input";
34         print ".\n";
35     }
36 }
```

zeichnis schließen kann. Die Anweisung *$files{$count} = $_;* legt den Hash *%files* an, falls er noch nicht existiert.

Ausgabe auf *STDERR* statt *STDOUT*

Wichtig ist, dass die *print*-Funktion in Zeile 22 nicht in die Standardausgabe, sondern auf den Fehlerkanal *STDERR* schreibt. Schriebe sie nach *STDOUT*, landete die Liste mit den Auswahlmöglichkeiten der Shell auf der Befehlszeile des *cd*-Kommandos. Nach Abschluss der *foreach*-Schleife fordert das *print*-Kommando in Zeile 26 den Benutzer mit der Ausgabe von *[1-N]>* (N ist die Nummer des letzten Verzeichnisses) zur Entscheidung auf. Da die Eingabe in derselben Zeile stattfinden soll, steht kein Zeilenumbruch im String. Führte man dies nicht auf *STDERR*, sondern *STDOUT* aus, müsste man übrigens mit der Autoflush-Anweisung *$| = 1* dafür sorgen, dass Perl die Ausgaben sofort mit jedem *print*-Kommando ausgibt, statt sie zu puffern. Der Zeilenumbruch *\n* löst unter *STDOUT* üblicherweise die Puffer-Entleerung aus. Auf *STDERR* wird auch ohne *\n* sofort geschrieben, schließlich soll kein Fehler verloren gehen.

Zeile 27 nimmt eine Benutzereingabe entgegen. Da das zeilenweise Einlesen mit *<STDIN>* immer ein Newline-Zeichen mitnimmt, putzt der *chop*-Befehl in Zeile 28 dieses gleich wieder weg. Zeile 30 prüft, ob die Eingabe auf einen gültigen Hash-Eintrag weist – falls ja, greift die nächste Zeile schließlich mit dem eingegebenen numerischen Wert als Schlüssel in den Hash *%files* hinein, fördert den zugehörigen Dateinamen hervor und gibt ihn aus. Gibt der Benutzer eine unsinnige Zahl ein, existiert *$files{$input}* nicht – und der *else*-Zweig gibt eine Warnung nach *STDERR* aus und setzt aus den weiter oben schon besprochenen Gründen das zurückgelieferte Verzeichnis von *cdbm* auf das gegenwärtige (.): (ck)

MICHAEL SCHILLI

ist als Web-Engineer für AOL/Netscape in Mountain View, Kalifornien, tätig. Er arbeitet an 'Perl Lernen', einem Buch für Perl-Anfänger, das voraussichtlich im Frühjahr 2001 bei Addison-Wesley erscheinen wird.