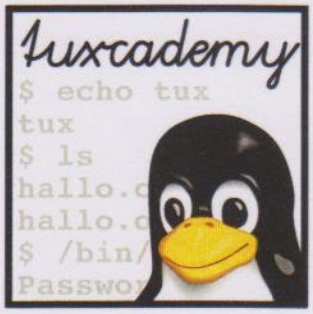
Version 4.0

Linux für Fortgeschrittene

Der Linux-Werkzeugkasten



tuxcademy – Linux- und Open-Source-Lernunterlagen für alle
www.tuxcademy.org · info@tuxcademy.org

Inhalt

1 Allgemeines über Shells 13
1.1 Shells und Shellskripte . . . 14
1.2 Shelltypen . . . 14
1.3 Die Bourne-Again-Shell . . . 17
1.3.1 Das Wichtigste . . . 17
1.3.2 Login-Shells und interaktive Shells . . . 18
1.3.3 Dauerhafte Konfigurationsänderungen . . . 20
1.3.4 Tastatur-Belegung und Abkürzungen . . . 21

2 Shellskripte 23
2.1 Einleitung . . . 24
2.2 Aufruf von Shellskripten . . . 24
2.3 Aufbau von Shellskripten . . . 26
2.4 Shellskripte planen . . . 27
2.5 Fehlertypen . . . 28
2.6 Fehlererkennung . . . 29

3 Die Shell als Programmiersprache 33
3.1 Variable . . . 34
3.2 Arithmetische Ausdrücke . . . 40
3.3 Bearbeitung von Kommandos . . . 41
3.4 Kontrollstrukturen . . . 42
3.4.1 Überblick . . . 42
3.4.2 Der Rückgabewert von Programmen als Steuergröße . . . 42
3.4.3 Alternativen, Bedingungen und Fallunterscheidungen . . . 44
3.4.4 Schleifen . . . 49
3.4.5 Schleifenunterbrechung . . . 51
3.5 Shellfunktionen . . . 54
3.6 Das Kommando `exec` . . . 55

4 Praktische Shellskripte 59
4.1 Shellprogrammierung in der Praxis . . . 60
4.2 Rund um die Benutzerdatenbank . . . 60
4.3 Dateioperationen . . . 65
4.4 Protokolldateien . . . 67
4.5 Systemadministration . . . 73

5 Interaktive Shellskripte 77
5.1 Einleitung . . . 78
5.2 Das Kommando `read` . . . 78
5.3 Menüauswahl mit `select` . . . 80
5.4 »Grafische« Oberflächen mit `dialog` . . . 84

6 Der Stromeditor sed 93
6.1 Einsatzgebiete . . . 94
6.2 Adressierung . . . 94
6.3 sed-Anweisungen . . . 96
6.3.1 Ausgeben und Löschen von Zeilen . . . 96
6.3.2 Einfügen und Verändern . . . 97
6.3.3 Zeichen-Transformationen . . . 98
6.3.4 Suchen und Ersetzen . . . 98
6.4 sed in der Praxis . . . 99

7 Die awk-Programmiersprache 105
7.1 Was ist awk? . . . 106
7.2 awk-Programme . . . 107
7.3 Ausdrücke und Variable . . . 109
7.4 awk in der Praxis . . . 113

8 SQL 123
8.1 Warum SQL? . . . 124
8.1.1 Überblick . . . 124
8.1.2 SQL einsetzen . . . 126
8.2 Tabellen definieren . . . 127
8.3 Datenmanipulation und Abfragen . . . 129
8.4 Relationen . . . 133
8.5 Praktische Beispiele . . . 136

9 Zeitgesteuerte Vorgänge – at und cron 141
9.1 Allgemeines . . . 142
9.2 Einmalige Ausführung von Kommandos . . . 142
9.2.1 at und batch . . . 142
9.2.2 at-Hilfsprogramme . . . 144
9.2.3 Zugangskontrolle . . . 145
9.3 Wiederholte Ausführung von Kommandos . . . 145
9.3.1 Aufgabenlisten für Benutzer . . . 145
9.3.2 Systemweite Aufgabenlisten . . . 147
9.3.3 Zugangskontrolle . . . 148
9.3.4 Das Kommando crontab . . . 148
9.3.5 Anacron . . . 148

10 Lokalisierung und Internationalisierung 153
10.1 Überblick . . . 154
10.2 Zeichencodierungen . . . 154
10.3 Spracheneinstellung unter Linux . . . 158
10.4 Lokalisierungs-Einstellungen . . . 160
10.5 Zeitzonen . . . 163

11 Die Grafikoberfläche X11 169
11.1 Grundlagen . . . 170
11.2 X11 konfigurieren . . . 175
11.3 Displaymanager . . . 182
11.3.1 Grundlegendes zum Starten von X . . . 182
11.3.2 Der Displaymanager LightDM . . . 183
11.3.3 Andere Displaymanager . . . 185
11.4 Informationen anzeigen . . . 186
11.5 Der Schriftenserver . . . 188
11.6 Fernzugriff und Zugriffskontrolle . . . 191

12 Linux für Behinderte 193
12.1 Einführung 194
12.2 Tastatur, Maus und Joystick 194
12.3 Die Bildschirmdarstellung 195

A Musterlösungen 197

B Reguläre Ausdrücke 213
B.1 Überblick 213
B.2 Extras 214

C LPIC-1-Zertifizierung 217
C.1 Überblick 217
C.2 Prüfung LPI-102 218
C.3 LPI-Prüfungsziele in dieser Schulungsunterlage 218

D Kommando-Index 223

Index 225

1 Allgemeines über Shells

Inhalt

1.1 Shells und Shellskripte 14
1.2 Shelltypen 14
1.3 Die Bourne-Again-Shell 17
1.3.1 Das Wichtigste 17
1.3.2 Login-Shells und interaktive Shells 18
1.3.3 Dauerhafte Konfigurationsänderungen 20
1.3.4 Tastatur-Belegung und Abkürzungen 21

Lernziele

- Grundlagen über Shells und Shellskripte lernen
- Login-, interaktive und nichtinteraktive Shellprozesse unterscheiden können
- Die Methoden zur Konfiguration der Bash beherrschen

Vorkenntnisse

- Vertrautheit mit der Kommandooberfläche von Linux
- Umgang mit Dateien und einem Texteditor

1.1 Shells und Shellskripte

Shell
Kommandointerpreter

Die **Shell** erlaubt Ihnen den direkten Umgang mit dem Linux-System: Sie können Kommandos formulieren, die die Shell auswertet und ausführt – meist durch Starten von externen Programmen. Man spricht von der Shell auch als »Kommandointerpreter«.

Die meisten Shells haben ferner Eigenschaften von Programmiersprachen – Variable, Kontrollstrukturen wie Verzweigungen und Schleifen, Funktionen und mehr. Das macht es möglich, auch komplexe Folgen von Shellkommandos und externen Programmaufrufen in Textdateien abzulegen und als **Shellskripte** interpretieren zu lassen. Damit lassen sich schwierige Operationen wiederholbar und dauernd auftretende Vorgänge mühelos gestalten.

Shellskripte

Das Linux-System verwendet Shellskripte für viele interne Aufgaben. Zum Beispiel werden die »Init-Skripte« in `/etc/init.d` in aller Regel als Shellskripte realisiert. Das gilt auch für viele Systemkommandos. Linux erlaubt es, den Umstand, dass ein »Systemprogramm« gar kein direkt ausführbares Binärcode-Programm ist, sondern ein Shellskript, vor seinen Benutzern zu verstecken, jedenfalls soweit es um die Aufrufsyntax geht – wenn Sie es genau wissen wollen, können Sie natürlich herausfinden, was Sache ist, zum Beispiel mit dem Kommando `file`:

```
$ file /bin/ls
/bin/ls: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV),▷
 ◁ for GNU/Linux 2.2.0, dynamically linked (uses shared libs),▷
 ◁ stripped
$ file /bin/egrep
/bin/egrep: Bourne shell script text executable
```

Übungen

1.1 [!2] Welche Vorteile könnte es haben, ein Kommando als Shellskript und nicht als Binärprogramm zu realisieren? Welche Nachteile? Unter welchen Umständen würden Sie sich für ein Shellskript anstatt eines Binärprogramms entscheiden?

1.2 [3] Wie viele Kommandos in den Verzeichnissen `/bin` und `/usr/bin` Ihres Linux-Systems sind als Shellskripte realisiert? Geben Sie eine Kommando-Pipeline an, mit der Sie diese Frage beantworten können (Tipp: Verwenden Sie die Kommandos `find`, `file`, `grep` und `wc`.)

1.2 Shelltypen

bash
csh
tcsh
ksh

Die Standardshell unter Linux ist die *Bourne-Again-Shell* (`bash`) des GNU-Projekts. Sie ist weitestgehend kompatibel zur ersten brauchbaren Shell der Unix-Geschichte (der Bourne-Shell) und zum POSIX-Standard. Traditionell sind auch andere Shells üblich, etwa die C-Shell (`csh`) von BSD und deren Weiterentwicklung, die Tenex-C-Shell (`tcsh`), die Korn-Shell (`ksh`) und einige mehr. Diese Shells unterscheiden sich mehr oder weniger stark in ihren Eigenschaften und Möglichkeiten und in ihrer Syntax.

Man kann bei Shells im Wesentlichen zwei große inkompatible Strömungen identifizieren, nämlich die Bourne-artigen Shells (`sh`, `ksh`, `bash`, ...) und die C-Shell-artigen Shells (`csh`, `tcsh`, ...). Die Bash versucht, die wichtigsten C-Shell-Eigenschaften zu integrieren.

Welche Shell für was?

Sie können als Benutzer selbst entscheiden, welche Shell Sie für was verwenden möchten. Entweder Sie starten die Shell Ihrer Wünsche durch den entsprechenden

Programmaufruf oder Sie stellen sich die Shell als Login-Shell ein, die das System beim Anmelden für Sie startet. Zur Erinnerung, der Typ der Login-Shell wird in der Benutzerdatenbank (meist in `/etc/passwd`) festgelegt. Ihren Login-Shell-Eintrag können Sie mit Hilfe des Kommandos `chsh` abändern:

```
$ getent passwd tux
tux:x:1000:1000:Tux der Pinguin:/bin/bash:/home/tux
$ chsh
Password: geheim
Changing the login shell for tux
Enter the new value, or press return for the default
        Login Shell [/bin/bash]: /bin/csh
$ getent passwd tux
tux:x:1000:1000:Tux der Pinguin:/bin/csh:/home/tux
$ chsh -s /bin/bash
Password: geheim
$ getent passwd tux
tux:x:1000:1000:Tux der Pinguin:/bin/bash:/home/tux
```

Mit der Option `-s` können Sie direkt die gewünschte Shell angeben; wenn Sie das nicht tun, werden Sie interaktiv gefragt.

Für `chsh` stehen Ihnen alle Shells zur Verfügung, die in der Datei `/etc/shells` erwähnt werden (jedenfalls soweit sie tatsächlich als Programme auf Ihrem System installiert sind – viele Distributoren tragen alle Shells der Distribution in die Datei ein, egal ob Sie das Paket eingespielt haben oder nicht.)

`/etc/shells`

Sie können herausfinden, welche Shell Sie gerade verwenden, indem Sie sich den Wert der Shellvariablen `$0` anschauen:

Aktuelle Shell

```
$ echo $0
/bin/bash
```

Unabhängig davon, mit welcher Shell Sie interaktiv arbeiten, können Sie natürlich auch frei wählen, welche Shell(s) Sie für Ihre Shellskripte nutzen. Hier kommen verschiedene Erwägungen ins Spiel:

- Es ist natürlich verlockend, die interaktiv benutzte Shell auch für Shellskripte einzusetzen. Schließlich brauchen Sie nur das, was Sie sonst auf der Kommandozeile tippen würden, in eine Datei zu schreiben. Das hat nur zwei mögliche Nachteile: Zum einen setzt das voraus, dass überall da, wo Sie ein Shellskript später laufen lassen wollen, Ihre Shell installiert ist – je nachdem, welche Shell Sie benutzen, ist das kein Problem oder doch. Zum anderen sind manche Shells zwar interaktiv gut, aber nicht besonders für Skripte geeignet, oder umgekehrt.

  Die C-Shell hat zwar für den interaktiven Gebrauch ihre treue Fan-Gemeinde, für den Einsatz in Shellskripten ist sie jedoch aufgrund diverser Implementierungsfehler und syntaktischer Inkonsistenzen universell verpönt (siehe hierzu [Chr96]). Obwohl die Tenex-C-Shell da nicht ganz so schlimm ist, sind gewisse Vorbehalte sicherlich nach wie vor angebracht.

- In vielen Fällen tun Sie gut daran, bei den Anforderungen an die zu verwendende Shell Bescheidenheit walten zu lassen, etwa indem Sie sich in Ihren Skripten auf den von POSIX genormten Funktionsumfang für Shells beschränken und die (komfortablen) Bash-Erweiterungen links liegen lassen. Dies ist zum Beispiel wichtig für Init-Skripte – in vielen Linux-Systemen ist die Bash zwar die Standard-Shell, aber es lassen sich Umgebungen denken, in denen sie zugunsten etwas weniger fetter Shells weggelassen wird.

Ein Skript, das sich als »Bourne-Shell-Skript« ausgibt und vom System auch so gestartet wird, sollte also keine »Bashismen« enthalten; wenn Sie wirklich Bash-Eigenschaften ausnutzen, sollten Sie Ihr Skript explizit als »Bash-Skript« kenntlich machen. Wie das geht, sehen Sie in Kapitel 2.

Im Zuge des Einsatzes von Linux als *embedded system* (also extrem abgespeckte Umgebung etwa als Betriebssystem für einen Router oder digitalen Videorecorder) und mit dem zunehmenden Interesse an Dingen wie der `initrd` sind »alternative« Shells wieder in den Vordergrund gerückt, die sich benehmen wie die »POSIX-Shell« und die Bash-Erweiterungen nicht anbieten. Einige Namen sind `ash`, `dash` oder `busybox` (letztere bringt außer der Shell-Funktionalität auch noch die meisten sonst externen Hilfsprogramme eingebaut mit). Ebenfalls erwähnenswert ist die `sash` oder *System Administrator's Shell*, die ebenfalls die wichtigsten externen Kommandos eingebaut hat und, statisch gelinkt, gerne als »Notnagel« für harte Fälle der Systemrettung vorgehalten wird.

Portabilität: Shell vs. Programme

Übrigens: Häufig ist nicht die Shell das Problem, das die Portabilität eines Shellskripts einschränkt, sondern die externen Programme, die das Skript aufruft. Diese müssen auf dem »Zielsystem« natürlich auch vorhanden sein und sich genauso benehmen wie auf Ihrem System. Solange Sie nur Linux verwenden, ist das in der Regel nicht so schlimm – aber wenn Sie auf einem Linux-System Skripte für proprietäre Unix-Systeme erstellen, können Sie Überraschungen erleben: Die GNU-Versionen der Standardwerkzeuge wie `grep`, `cat`, `ls`, `sed`, ... haben nämlich nicht nur mehr und leistungsfähigere Optionen als die Implementierungen, die Sie auf vielen proprietären Unixen finden, sondern oft auch weniger Fehler und willkürliche Grenzen (etwa bei der Länge von Eingabezeilen). Auch hier ist also oft Mäßigung angesagt, wenn Sie nicht sicher sind, dass Sie immer die GNU-Werkzeuge zur Verfügung haben.

Betrachten Sie zur Illustration ein beliebiges `configure`-Skript eines freien Programmpakets (`configure`-Skripte sind Shellskripte, die auf maximale Portabilität getrimmt sind). Tun Sie das im eigenen Interesse aber nicht unmittelbar vor oder nach der Nahrungsaufnahme.

Die folgenden Kapitel beschränken sich auf die Bourne-Again-Shell, was aber nicht heißt, dass Sie das Gelernte nicht auch zum Großteil auf andere Shells anwenden können. Spezielle Bash-Eigenschaften sind, wo möglich, als solche gekennzeichnet.

Übungen

1.3 [!1] Versuchen Sie herauszufinden, welche Shells auf Ihrem System installiert sind.

1.4 [!2] Rufen Sie – falls Sie sie haben – die Tenex-C-Shell (`tcsh`) auf, geben Sie dort ein Kommando mit einem kleinen Tippfehler ein und drücken Sie ↩. – Wie kommen Sie wieder zu Ihrer alten Shell zurück?

1.5 [!1] Weisen Sie sich eine andere Shell als Login-Shell zu (zum Beispiel die `tcsh`). Prüfen Sie, ob es geklappt hat, etwa indem Sie sich auf einer anderen virtuellen Konsole anmelden. Ändern Sie danach Ihre Login-Shell wieder zur Bash zurück.

1.6 [2] Installieren Sie die `sash` (falls Ihre Distribution sie anbietet) und führen Sie einen neuen Benutzer `rootsash` ein, der die `sash` als Login-Shell hat, aber sonst die Rechte von `root` genießt.

1.3 Die Bourne-Again-Shell

1.3.1 Das Wichtigste

Die Bourne-Again-Shell (oder Bash) wurde im Rahmen des GNU-Projektes der *Free Software Foundation* von Brian Fox und Chet Ramey entwickelt und vereinigt Funktionen der Korn- und der C-Shell.

Da die Korn-Shell eine Weiterentwicklung der Bourne-Shell ist und die Bash quasi eine Rückbesinnung der traditionell eher mit der C-Shell assoziierten BSD-Welt auf die Bourne-Konzepte darstellt, ist der Name »Bourne-Again-Shell« – englisch-phonetisch nicht von der »wiedergeborenen Shell« zu unterscheiden – angebracht.

Einen ausführlichen Überblick über die Möglichkeiten der Bash im interaktiven Gebrauch gibt die Linup-Front-Schulungsunterlage *Linux-Grundlagen für Anwender und Administratoren*. Wir wiederholen hier nur kurz das Wichtigste:

Variable Die Bash unterstützt – wie die meisten Shells – die Verwendung von **Variablen**, die gesetzt und wieder abgerufen werden können. Variable werden auch zur Konfiguration zahlreicher Aspekte der Shell selbst verwendet, beispielsweise sucht die Shell ausführbare Programme für Kommandos in den Verzeichnissen, die in `PATH` aufgezählt werden, oder orientiert sich für die Ausgabe einer Eingabeaufforderung am Wert der Variablen `PS1`. Mehr über Variable finden Sie im Abschnitt 3.1. Variablen

Aliasnamen Das Kommando `alias` gibt Ihnen die Möglichkeit, eine längere Kommando-Folge abzukürzen. Beispielsweise definieren Sie durch

```
$ alias c='cal -m; date'
$ type c
c is aliased to `cal -m; date'
```

das neue »Kommando« `c`. Immer, wenn Sie jetzt »`c`« aufrufen, wird die Bash das Kommando `cal` mit der Option `-m` gefolgt vom Kommando `date` für Sie ausführen. Dieses **Alias** können Sie auch wieder in anderen Alias-Definitionen benutzen. Sie können sogar durch ein Alias ein bestehendes Kommando »umdefinieren«. Mit Alias

```
$ alias rm='rm -i'
```

wird beispielsweise das Kommando `rm` entschärft. Allerdings ist das von zweifelhaften Nutzen: Haben Sie sich einmal an die »sichere« Variante gewöhnt, so kann es sein, dass Sie auch dort damit rechnen, wo das Alias nicht gesetzt ist. Es ist daher besser, wenn Sie sich bei potenziell gefährlichen Kommandos einen neuen Namen ausdenken.

Durch `alias` (ohne Argumente) können Sie sich alle zur Zeit definierten Aliase anzeigen lassen, und mit `unalias` können Sie einzelne oder alle Aliase löschen. `unalias`

Funktionen Wenn Sie mehr als reine textuelle Ersetzung wie bei Aliasen benötigen, so können Ihnen Funktionen weiterhelfen. Mehr über Funktionen erfahren Sie in Abschnitt 3.5.

Das Kommando set Mit dem Bash-internen Kommando `set` können Sie nicht nur alle Shell-Variablen anzeigen (wenn Sie es ohne Parameter aufrufen), sondern auch Options-Schalter der Bash setzen. So können Sie mit »`set -C`« (oder gleichwertig: »`set -o noclobber`«) verhindern, dass Sie durch Ausgabe-Umlenkung versehentlich eine existierende Datei überschreiben. `set`

Mit »`set -x`« (oder »`set -o xtrace`«) können Sie sehen, was die Bash für Schritte unternimmt, um zu ihrem Ergebnis zu kommen:

```
$ set -x
$ echo "My $HOME is my castle"
+ echo 'My /home/tux is my castle'
My /home/tux is my castle
```

Wenn anstelle des »-« bei den Optionen ein »+« gesetzt ist, wird die entsprechende Option abgeschaltet.

1.3.2 Login-Shells und interaktive Shells

Shell ist nicht gleich Shell. Natürlich unterscheidet sich die Bash von einer C-Shell oder Korn-Shell, aber schon das Verhalten eines Bash-Prozesses kann von dem eines anderen Bash-Prozesses abweichen, je nachdem wie die Bash jeweils aufgerufen wurde.

Prinzipiell gibt es drei Varianten: Login-Shell, interaktive Shell und nichtinteraktive Shell. Diese Formen unterscheiden sich darin, welche Konfigurationsdateien eingelesen werden.

Login-Shell Diese Form der Shell erhalten Sie direkt nach dem Anmelden am System. Das die Shell startende Programm, also `login`, »`su -`«, `ssh` usw., gibt der Shell ein »-« als erstes Zeichen des Programm-Namens mit. Dadurch weiß die Shell, dass sie eine Login-Shell sein soll. Wenn Sie sich frisch angemeldet haben, sollte es etwa so aussehen:

```
$ echo $0
-bash
$ bash                                  Aufruf der Bash von Hand; keine Anmeldung
$ echo $0
bash
```

Alternativ können Sie die Bash auch mit der Option `-l` aufrufen, damit diese sich als Login-Shell fühlt.

`/etc/profile` Jede Bourne-artige Shell (nicht nur die Bash) führt als erstes die Kommandos in der Datei `/etc/profile` aus. Dadurch können systemweit beim Anmelden zum Beispiel Umgebungsvariable oder die *umask* gesetzt werden.

Wenn Sie auf Ihrem System sowohl die Bash als auch die Bourne-Shell verwenden, müssen Sie darauf achten, in `/etc/profile` nur Konstruktionen zu verwenden, die auch die Bourne-Shell versteht. Alternativ dazu können Sie von Fall zu Fall prüfen, ob gerade eine Bourne-Shell oder eine Bash die Datei bearbeitet. Wenn es sich um eine Bash handelt, ist die Umgebungsvariable `BASH` definiert.

`.profile` Danach sucht die Bash die Dateien `.bash_profile`, `.bash_login` und `.profile` im Heimatverzeichnis des Benutzers. Nur die *erste* gefundene Datei wird abgearbeitet.

Auch dieses Verhalten hat seinen Ursprung in der Bourne-Shell-Kompatibilität der Bash. Wenn Sie auf Ihr Heimatverzeichnis von verschiedenen Rechnern aus zugreifen können, von denen manche die Bash und andere nur die Bourne-Shell unterstützen, können Sie Bash-spezifische Konfigurationseinstellungen in der Datei `.bash_profile` hinterlegen, so dass diese eine Bourne-Shell, die nur `.profile` liest, nicht aus dem Tritt bringen. (Aus der `.bash_profile` können Sie dann die `.profile` einlesen.) – Alternativ dazu können Sie in der Datei `.profile` den oben angedeuteten Ansatz mit der `BASH`-Variable benutzen.

Der Name `.bash_login` kommt aus der C-Shell-Tradition. Eine etwa vorhandene `.login`-Datei für die C-Shell lässt die Bash aber links liegen.

Beenden Sie eine Anmelde-Shell, so arbeitet sie die Datei `.bash_logout` im Heimatverzeichnis ab, sofern sie existiert.

Interaktive Shell Wenn Sie die Bash ohne Dateinamen-Argumente (aber möglicherweise mit Optionen) aufrufen und ihre Standard-Ein- und -Ausgabe mit einem »Terminal« verbunden ist (`xterm` und Konsorten reichen), so versteht sie sich als interaktive Shell. Als solche liest sie beim Starten die Dateien `/etc/bash.bashrc` und `.bashrc` in Ihrem Heimatverzeichnis und führt die darin enthaltenen Kommandos aus.

interaktive Shell
/etc/bash.bashrc
.bashrc

Ob eine interaktive Shell `/etc/bash.bashrc` liest, ist in Wirklichkeit eine Konfigurationsoption. Bei den wesentlichen Distributionen, beispielsweise den SUSE-Distributionen und Debian GNU/Linux, ist diese Option allerdings eingeschaltet.

Beim Verlassen einer interaktiven Shell, die keine Anmelde-Shell ist, werden keine Dateien ausgewertet.

Nichtinteraktive Shell Nichtinteraktive Shells werten gar keine Dateien beim Starten oder Beenden aus. Zwar können Sie einer solchen Bash durch die Umgebungsvariable `BASH_ENV` einen Dateinamen zum Auswerten mitgeben, diese Variable ist aber zumeist nicht gesetzt.

Eine Shell ist dann nichtinteraktiv, wenn sie benutzt wird, um ein Shell-Skript auszuführen, oder wenn ein Programm sich einer Shell bedient, um ein anderes Programm zu starten. Das ist der Grund, warum Aufrufe der Art

```
$ find -exec ll {} \;
find: ll: No such file or directory
```

fehlschlagen: `find` startet eine nichtinteraktive Shell, um `ll` auszuführen. Obwohl `ll` auf vielen Systemen verfügbar ist, handelt es sich nur um ein Alias für »`ls -l`«. Als Alias muss es aber in jeder Shell definiert werden, da Aliase nicht »vererbt« werden. Eine Konfigurationsdatei mit Aliasdefinitionen wird aber normalerweise nicht eingelesen.

Distributionsspezifische Besonderheiten Die strikte Trennung zwischen Anmelde-Shell und normalen interaktiven Shells erzwingt, dass Sie gewisse Einstellungen sowohl in `.profile` als auch in `.bashrc` vornehmen müssen, damit sie in jeder Shell wirksam werden. Um diese fehleranfällige Doppelarbeit zu vermeiden, haben viele Distributionen in der mitgelieferten Datei `.profile` eine Zeile wie

```
## ~/.profile
test -r ~/.bashrc && . ~/.bashrc
```

die das Folgende bewirkt: Wenn `.bashrc` existiert und lesbar ist (`test...`), dann (`&&`) wird die Datei `.bashrc` abgearbeitet (»`.`« ist ein Shell-Kommando, das die Datei so einliest, als würde ihr Inhalt an dieser Stelle eingetippt).

Möglicherweise hat Ihre Distribution die Bash auch so übersetzt, dass sie noch andere Dateien einliest. Dies können Sie mit `strace` überprüfen; es listet Ihnen alle Systemaufrufe auf, die ein anderes Programm benutzt, darunter auch den `open`-Aufruf zum Öffnen einer Datei. Erst damit können Sie wirklich sicher sein, welche Dateien ausgewertet werden.

Übungen

1.7 [!2] Überzeugen Sie sich davon, dass Ihre Bash die Dateien `/etc/profile`, `/etc/bash.bashrc`, `~/.profile` und `~/.bashrc` wie beschrieben verwendet. Untersuchen Sie alle Abweichungen.

1.8 [1] Woran merkt ein Shellprozess, dass er eine »Login-Shell« sein soll?

1.9 [3] Wie können Sie eine Shell als »Login-Shell« verwenden, die nicht in `/etc/shells` steht?

1.3.3 Dauerhafte Konfigurationsänderungen

Individuelle Änderungen Individuelle Anpassungen Ihrer Arbeitsumgebung halten nur bis zum Ende des Shell-Prozesses vor. Wollen Sie Ihre Änderungen auch beim nächsten Anmelden vorfinden, so müssen Sie dafür sorgen, dass die Shell sie automatisch beim Start ausführt. Umgebungsvariablen, Aliase, Shell-Funktionen, die *umask* usw. müssen in einer der in Abschnitt 1.3.2 genannten Dateien gesetzt werden – die Frage ist nur, in welcher.

Im Fall von Aliasen ist die Antwort leicht: Da sie nicht vererbt werden können, müssen sie von jeder Shell neu gesetzt werden. Daher müssen Sie Aliase in der Datei `~/.bashrc` definieren. (Damit der Alias aber auch in der Anmelde-Shell funktioniert, müssen Sie ihn *zusätzlich* in die Datei `~/.profile` eintragen, wenn dort nicht, wie oben beschrieben, die Datei `~/.bashrc` eingelesen wird.)

Gute Kandidaten für die Datei `.bashrc` sind auch Variablen, die das Verhalten der Shell steuern (`PS1`, `HISTSIZE` u. a.), aber sonst nicht von Interesse sind, d. h. eben keine Umgebungsvariablen. Wenn Sie wollen, dass jede Shell »frisch« startet, so müssen Sie diese Variablen jedesmal neu setzen, was daher in `.bashrc` geschehen sollte.

Anders sieht die Sache bei Umgebungsvariablen aus. Diese werden üblicherweise einmal gesetzt, genau wie Änderungen der Tastaturbelegung und ähnliches. Es genügt daher, sie in `.profile` zu definieren. Bei Konstruktionen wie

```
PATH=$PATH:$HOME/bin
```

die die Variable `PATH` um eine weitere Komponente verlängert, verbietet sich die Datei `~/.bashrc`, weil sonst der Wert der Variable immer weiter verlängert würde.

Wenn Ihr Rechner in den Runlevel 5 startet, also die Anmeldung über den X-Displaymanager abgewickelt wird, haben Sie nicht direkt eine Anmelde-Shell. Damit die Einstellungen trotzdem berücksichtigt werden, sollten Sie diese in die Datei `~/.xsession` eintragen oder aus dieser heraus Ihre `.profile` einlesen.

Systemweite Änderungen Als Systemadministrator können Sie Einstellungen für alle Benutzer in den Dateien `/etc/profile` und `/etc/bash.bashrc` vornehmen. Damit diese Anpassungen eine Software-Aktualisierung des Systems überleben, haben viele Distributoren besondere Vorkehrungen getroffen: SUSE zum Beispiel empfiehlt die Verwendung der Dateien `/etc/profile.local` und `/etc/bash.bashrc.local`, die von den entsprechenden Schwesterdateien eingelesen werden.

`/etc/skel` Eine andere Möglichkeit der globalen Änderung stellt das Verzeichnis `/etc/skel` dar, das »Rohgerüst« eines Heimatverzeichnisses, das neue Benutzer kopiert bekommen, wenn Sie `useradd` mit der Option `-m` aufrufen. Alle darin enthaltenen Dateien und Verzeichnisse werden damit zur Grundausstattung des neuen Heimatverzeichnisses.

Wenn Sie in `/etc/skel` eine Datei `.bashrc` des Inhalts

```
## Systemweite Einstellungen; bitte nicht veraendern:
test -r /etc/bash.local && . /etc/bash.local

## Individuelle Einstellungen hier einfuegen:
```

ablegen, so können Sie Änderungen am System vornehmen (in `/etc/bash.local`), die für alle Benutzer wirksam werden.

Natürlich können Sie in `/etc/skel` auch noch vorgefertigte Konfigurationsdateien für beliebige andere Programme, Verzeichnisstrukturen usw. ablegen.

Übungen

1.10 [!1] Installieren Sie das Alias `hallowelt` für das Kommando »`echo Hallo Welt`« so, dass es in Ihrer Login-Shell und allen interaktiven Shells zur Verfügung steht.

1.11 [!1] Installieren Sie das Alias `hallowelt` für das Kommando »`echo Hallo Welt`« so, dass es in allen Login-Shells und interaktiven Shells im ganzen System zur Verfügung steht.

1.12 [2] Sorgen Sie dafür, dass Sie `hallowelt` auch in Ihren nichtinteraktiven Shells aufrufen können.

1.3.4 Tastatur-Belegung und Abkürzungen

Die Bash verwendet verschiedene Tastaturkürzel, um das Editieren von Kommandos bei der Eingabe zu erlauben und besondere Eigenschaften anzusteuern.

Noch weitergehende Einstellungsmöglichkeiten erlaubt Ihnen die Datei `.inputrc` in Ihrem Heimatverzeichnis bzw. `/etc/inputrc` für systemweite Einstellungen. Diese Datei ist die Konfigurationsdatei für die Readline-Bibliothek, die von der Bash und anderen Programmen benutzt wird. Beispielsweise liegt es an der Readline-Bibliothek, dass Strg+r das Rückwärtssuchen in der History erlaubt. `.inputrc`

Die Einstellungen für Readline gelten sowohl für die virtuellen Terminals als auch für die grafische Oberfläche. Die ganzen Details verrät Ihnen `readline(3)`, wir beschränken uns hier auf einige Beispiele. Wenn Sie die Zeilen

```
Control-t:      tab-insert
Control-e:      "cal\C-m"
```

in die Datei `.inputrc` eintragen, dann wird das Drücken von Strg+t einen Tabulator-Vorschub erzeugen, was Sie normalerweise in der Bash nicht bekommen, da die Tabulator-Taste schon belegt ist. Das Drücken von Strg+e startet ein Makro, in diesem Fall die Zeichen »cal« gefolgt von Strg+m (dafür steht »`\C-m`«), was dem Drücken von ↩ entspricht.

Die Änderungen in der Datei werden allerdings nur wirksam, wenn Sie die Umgebungsvariable `INPUTRC` auf `$HOME/.inputrc` gesetzt haben und eine neue Bash starten oder wenn Sie das Kommando `bind -f ~/.inputrc` ausführen.

Kommandos in diesem Kapitel

ash	Eine leichtgewichtige POSIX-konforme Shell	ash(1)	16
busybox	Eine Shell, die Versionen vieler Unix-Werkzeuge integriert hat	busybox(1)	16
chsh	Ändert die Login-Shell eines Benutzers	chsh(1)	14
dash	Eine leichtgewichtige POSIX-konforme Shell	dash(1)	16
file	Rät den Typ einer Datei anhand des Inhalts	file(1)	14
find	Sucht nach Dateien, die bestimmte Kriterien erfüllen	find(1), Info: find	14
sash	„Stand-Alone Shell" mit eingebauten Kommandos, zur Problembehebung	sash(8)	16
strace	Protokolliert die Systemaufrufe eines Programms	strace(1)	19

Zusammenfassung

- Die Shell erlaubt den kommandoorientierten Umgang mit dem Linux-System. Die meisten Shells haben Eigenschaften von Programmiersprachen und gestatten die Erstellung von »Shellskripten«.
- Shellskripte werden im Linux-System an vielen Stellen verwendet.
- Es gibt eine Vielzahl von Shells. Die Standardshell bei den meisten Linux-Distributionen ist die »Bourne-Again-Shell«, kurz »Bash«, aus dem GNU-Projekt.
- Die Bash kombiniert Eigenschaften der Bourne- und Korn-Shell mit der der C-Shell.
- Die Bash (wie die meisten Shells) verhält sich unterschiedlich, je nachdem, ob sie als Login-Shell, als interaktive oder nichtinteraktive Shell gestartet wurde.
- Benutzerspezifische Voreinstellungen für die Bash können in einer der Dateien `~/.bash_profile`, `~/.bash_login` oder `~/.profile` sowie in der Datei `~/.bashrc` gemacht werden.
- Systemweite Voreinstellungen für alle Benutzer können in den Dateien `/etc/profile` und `/etc/bash.bashrc` gemacht werden.
- Abweichende Tastaturbelegungen sind in `~/.inputrc` und `/etc/inputrc` konfigurierbar.

Literaturverzeichnis

Chr96 Tom Christiansen. »Csh Programming Considered Harmful«, Oktober 1996. `http://www.faqs.org/faqs/unix-faq/shell/csh-whynot/`