
TEIL I

Erste Schritte

»Es gibt keinen Grund, warum Menschen zu Hause einen Computer haben sollten.«

Ken Olson, Gründer von Digital Equipment Corporation, 1977

»Aller Anfang ist schwer, doch in der Shell nicht so sehr.« So wird das Resümee dieses ersten Abschnittes lauten. Natürlich hakt es gerade am Anfang immer wieder an den unterschiedlichsten Stellen, doch die Fehler lassen sich meist sehr schnell lokalisieren und beheben. Nirgendwo hilft das Motto »Learning by doing« so stark wie in der Shell. Lassen Sie sich also nicht verwirren! Vieles ist einfacher, als es vielleicht auf dem Papier erscheint.

KAPITEL 1

Einführung

Herzlich willkommen in der Welt der Shellprogrammierung! Sie befinden sich zwar erst am Anfang dieses Buchs, doch kann ich Ihnen jetzt schon versprechen, daß Sie die Shell wegen ihrer kraftvollen Fähigkeiten schätzen werden.

1.1 An wen richtet sich dieses Buch?

Die Frage aller Fragen beim Kauf eines Buches ist meist: An welche Leserschaft richtet sich das Buch? Gerade in der Shellprogrammierung gibt es sehr unterschiedliche Ansätze. Während einerseits einige Bücher sehr grundlegender Natur sind und nicht nur die Shell, sondern auch das Betriebssystem erklären, gibt es auf der anderen Seite echte Programmierbücher, die weitreichende Programmierkenntnisse voraussetzen.

Dieses Buch positioniert sich in der Mitte dieser beiden Extreme. Das einzige Wissen, das beim Leser vorausgesetzt wird, sind grundlegende Kenntnisse in Unix. Er sollte einen Editor bedienen können, grundlegende Befehle wie *rm* kennen und Basiswissen über Unix besitzen – mit anderen Worten: Der Leser sollten Unix schon einmal benutzt haben. Mehr Wissen benötigt man in der Regel nicht, alles darüber hinausgehende erkläre ich meist von Anfang an. Natürlich kann die Kenntnis einer beliebigen Programmiersprache sehr hilfreich sein.

Neben den Voraussetzungen spielt eine andere Frage eine wichtige Rolle: Richtet sich das Buch an den Heimanwender oder an einen professionellen Administrator? Dieses Buch bemüht sich um den Spagat, sowohl dem Heimanwender als auch dem Administrator gerecht zu werden. Dies spiegelt sich vor allem in den zahlreichen Beispielen wieder. Allerdings darf nicht vergessen werden, daß sich dieses Buch mit der Programmierung der Shell beschäftigt, Themen wie der Zeileneditor oder der History-Mechanismus werden deshalb bewußt ausgelassen oder nur am Rande erläutert.

Dieses Buch spezialisiert sich auf keine der Unix-Shells, auch wird auf keines der vielen Unix-Systeme auf dem Markt gesondert eingegangen. Mein Bestreben ist es, den kleinsten gemeinsamen Nenner zu finden. Dies ist meiner Ansicht nach notwendig, da es durchaus üblich ist, daß in einem Netzwerk viele verschiedene Unix-Systeme eingesetzt werden. Als Administrator sieht man sich häufig vor dem Problem, Skripten zu schreiben, die überall lauffähig sind – daß dies gar nicht so einfach ist, wie man vielleicht vermutet, werden Sie noch erfahren. Dieses Buch wird Sie dabei so gut wie möglich unterstützen. Falls Sie solche Probleme nicht beschäftigen, werden Sie es hingegen zu schätzen wissen, daß alle in diesem Buch behandelten Shells sehr detailliert besprochen werden. Falls Sie also verstärkt nur eine der Shells einsetzen möchten, werden auch Sie sich zu Hause fühlen.

1.2 Aufbau des Buchs

Dieses Buch gliedert sich in insgesamt sechs Abschnitte:

I. Erste Schritte

Dieser Abschnitt enthält einige Bemerkungen zu diesem Buch und erklärt die Grundlagen der Shell.

II. Die Eingabeaufforderung

Hier befassen wir uns mit der Eingabeaufforderung der Shell – Wildcards oder Umlenkungen sind also zentrale Themen.

III. Der Werkzeugkasten

Im dritten Abschnitt beginnen wir, uns erstmals mit der eigentlichen Programmierung zu beschäftigen. Dabei werden nur grundlegende Programmier-elemente der Shell erklärt.

IV. Von der Pflicht zur Kür

Während sich der vorherige Abschnitt darauf konzentriert, grundlegende Programmierung, wie sie auch in anderen Sprachen möglich ist, zu erklären, wendet sich dieser Abschnitt mehr zur Sichtweise der Shell.

V. Der Feinschliff

Der fünfte Abschnitt läßt keine Feinheiten der einzelnen Shells mehr aus. Hier finden Sie unter anderem auch eine Befehlsreferenz und ein Kapitel zur Konfiguration.

VI. Über den Tellerrand

Der Shellprogrammierer an sich ist anderen Programmiersprachen sehr aufgeschlossen. Wenn eine andere Sprache irgendeine Aufgabe besser erledigen kann als die Shell, wird in der Regel diese benutzt. Deshalb erhalten Sie in diesem Abschnitt einen Überblick über andere Skriptsprachen und erfahren, wie und wann man diese gewinnbringend in einem Shellskript einsetzen kann.

Der Aufbau dieses Buches richtet sich nach dem Schema: von grob nach fein. Falls Sie also bereits Vorwissen über die Shell besitzen, können Sie, falls Sie das wünschen, getrost einige Kapitel überspringen.

Falls Sie beim Lesen dieses Buches (vorerst) den Umgang mit nur einer Shell erlernen möchten, sollten sie bei jeder Kapitelüberschrift einen Blick in die Tabelle darunter werfen. Dort ist jeweils angegeben, mit welchen Shells sich das Kapitel beschäftigt. Falls sich keine Information neben der Kapitelüberschrift finden läßt, gilt das Kapitel für alle Shells.

1.3 Schreibkonventionen

In diesem Buch gelten die üblichen Schreibkonventionen zur Syntaxbeschreibung.

Schreibweise	Bedeutung
<...>	Ist ein Platzhalter für einen Wert, der immer angegeben werden muß.
[...]	Beschreibt einen optionalen Wert. Man muß diesen Platzhalter also nicht unbedingt mit einem Wert füllen.
... ...	Gibt Alternativen an.

Tabelle 1.1

Aus Gründen der Übersichtlichkeit wird oft eine etwas ungenaue, aber lesbarere Schreibweise eingehalten. Als Beispiel sei der Befehl *rm*, der Dateien löscht, genannt. Eine Beschreibung der Aufrufsyntax dieses Unix-tools könnte folgendermaßen aussehen:

```
rm [-fir] <Dateien>
```

[-fir] gibt an, daß *rm* die Optionen *-f*, *-i* und *-r* akzeptiert, keine von diesen Optionen *muß* jedoch angegeben werden. Ganz anders sieht es mit *<Dateien>* aus. Durch die spitzen Klammern wird klar, daß *rm* mindestens ein

KAPITEL 1

Dateiname übergeben werden muß. Natürlich ist die Formulierung *<Dateien>* sehr ungenau, exakter wäre:

```
rm [-fir] <Datei1> [<Datei2> [...]]
```

Sie werden mir zustimmen, wenn ich behaupte, daß diese Beschreibung das eigentlich Entscheidende an der Syntax verschleiert.

Manchmal kommen in diesem Buch Auflistungen vor, dessen Elemente durch beispielsweise Kommata getrennt werden. Hier verwende ich folgende Notation:

```
<Wert1>, [Wert2], [...]
```

Auch diese Beschreibung ist ungenau, da sie vortäuscht, daß die Kommata immer angegeben werden müssen. Allerdings ist auch hier die exakte Variante wesentlich unleserlicher:

```
<Wert1>[, <Wert2>[, <Wert3> [...]]]
```

Diese Schreibkonventionen sind in der Computerwelt allgemein üblich und anerkannt, leider sind sie jedoch bei der Shell nicht unbedingt immer die beste Wahl. Das Problem entsteht dadurch, daß die Shell so viele Sonderzeichen für sich beansprucht, daß manchmal nicht ganz eindeutig ist, welche Zeichen eingegeben werden müssen und welche nur Platzhalter sind. Betrachten Sie dazu folgendes Beispiel:

```
<Kommando> > <Datei>
```

Hierbei handelt es sich um die Beschreibung einer einfachen Umlenkung, wie Sie sie wahrscheinlich schon einmal benutzt haben. Das Problem dieser Syntaxbeschreibung ist die fehlende Übersicht. Das eigentlich entscheidende Größer-Zeichen geht vollkommen unter. Deshalb lasse ich in solchen Fällen *<...>* meist weg:

```
kommando > datei
```

Wenn es notwendig ist, schreibe ich Platzhalter zusätzlich *kursiv*. Ich hoffe, daß es trotz aller Probleme keine Mißverständnisse gibt.

An einigen Stellen im Buch kommen Tastenkombinationen vor. Diese schreibe ich in eckigen Klammern:

Schreibweise	Bedeutung
[Enter]	Return-Taste
[Tabulator]	Tabulator-Taste, befindet sich links neben dem »Q«
[Strg]-[C]	[Strg]-Taste (heißt auf amerikanischen Tastaturen [Ctrl]) und gleichzeitig die [C]-Taste drücken

Tabelle 1.2

1.4 Begriffe

Begriffe sind im Idealfall eindeutig und leicht verständlich. In der Praxis ist jedoch meistens beides nicht zutreffend. Viele Begriffe werden in mehreren Zusammenhängen mit unterschiedlicher Bedeutung verwendet und sind zudem auch noch schwer zu verstehen. Um Mißverständnisse auszuräumen, sollten also einige Grundbegriffe zuerst einmal definiert werden.

1.4.1 Die ewige Diskussion um Fachbegriffe

Zur Einführung ein abschreckendes Beispiel:

»Bei der Command Substitution sind alle Metacharacter und damit auch alle Expansions gültig.«

Ich versuche in diesem Buch, soweit wie möglich unnötige Fachbegriffe zu vermeiden. Doch gerade bei der Shellprogrammierung treten so viele Spezialbegriffe auf, daß es zumindest notwendig ist, diese zu erwähnen und zu erklären. Nicht gerade einfacher machen es einem da die englischen Fachausdrücke, die zwar oft kürzer als die deutschen Übersetzungen sind, aber für Einsteiger eine große Hürde darstellen können. Deshalb ist es mein Ziel, passende und sinnvolle deutsche Begriffe den englischen vorzuziehen. Natürlich kann eine solche Auswahl und die Entscheidung, ob ein deutscher Begriff »gut« oder »schlecht« ist, nur sehr subjektiv sein.

In den meisten Fällen richte ich mich bei der Begriffswahl nach dem üblichen Sprachgebrauch. Ist ein englischer Begriff bereits so in den Sprachgebrauch übergegangen, daß das deutsche Wort mehr Verwirrung stiftet als Klärung, behalte ich den Originalausdruck. Beispiele hierfür sind Arrays (im Vergleich dazu: »Reihung« oder »Feld«) oder das Quoting (zu deutsch: »Abdeckung«). Findet sich jedoch ein passender deutscher Begriff und gibt es keine zwingenden Gründe, die sich durch den Mainstream (auch dieser Anglizismus ist bereits im Duden zu finden) ergeben, bevorzuge ich den deutschen Begriff, in der Hoffnung, daß ich dadurch mehr Klarheit als Verwirrung schaffe. Ich bin mir natürlich dessen bewußt, daß besonders bei diesem Thema die Meinungen sehr weit auseinandergehen.

1.4.2 Einige Begriffe aus der Computerwelt

In der Computerwelt haben sich einige Fachbegriffe so eingebürgert, daß sie fast selbstverständlich benutzt werden. Falls Sie auf einen Begriff stoßen, der Ihnen unbekannt ist, können Sie in diesem Kapitel dessen Bedeutung nachschlagen:

KAPITEL 1

Manpage beziehungsweise man-Page

Das Dokumentationssystem von Unix fußt auf den sogenannten »Manual Pages«. Fast jedes Programm besitzt einen solchen Hilfetext. Die Dokumentation eines Unixbefehls erreicht man dabei über

```
man [Sektion] <Manpage>
```

Da manchmal Namensüberschneidungen auftreten, sind die Hilfetexte in Sektionen aufgeteilt. Falls in diesem Buch auf eine Manpage verwiesen wird, füge ich deshalb immer in Klammern die Nummer der Sektion an, beispielsweise findet man die Manpage von *rm* unter *rm(1)*.

Pfad

Ein Pfad gibt den Ort einer Datei oder eines Verzeichnisses im Dateisystem an. Man unterscheidet bei den Pfaden zwischen absoluten und relativen Pfaden. Relative Pfade gehen von der aktuellen Position im Dateisystem aus, beispielsweise ist *src/main.c* eine Datei, die sich in einem Unterverzeichnis namens *src* befindet. Ein absoluter Pfad ist unabhängig vom aktuellen Verzeichnis, er enthält den vollständigen Pfad. Ein Beispiel hierfür ist */usr/bin*.

<i>/usr/bin/vi</i>	Absoluter Pfad
<i>../src/main.c</i>	Relativer Pfad
<i>datei</i>	Relativer Pfad
<i>./datei</i>	Relativer Pfad

Unix-System

UNIX ist ein registriertes Warenzeichen von »The Open Group« (siehe <http://www.unix.org>). In diesem Buch wird der Begriff Unix nicht in diesem Sinn verwendet, sondern als Oberbegriff für eine Gruppe von kommerziellen Betriebssystemen, die teilweise als Unix-System zertifiziert sind. Außerdem für solche freien Betriebssysteme (beziehungsweise Open Source), die dieser kostspieligen Zertifizierung nicht unterzogen wurden und deshalb richtigerweise als unix-artige oder unix-ähnliche Betriebssysteme bezeichnet werden. Dieser unhandliche Begriff soll hier vermieden werden, weshalb in diesem Buch vereinfachend von *Unix-Systemen* gesprochen wird.

Als Grundstock eines Unix-Systems wird häufig System V von AT&T genannt. Momentan aktuelle Unix-Systeme im oben geschilderten Sinn sind beispielsweise GNU/Linux, FreeBSD, NetBSD, OpenBSD, AIX, IRIX, Solaris/SunOS, BeOS und Mac OS X.

Die in diesem Buch aufgeführten Shells und Hilfsprogramme gibt es zum Teil aber auch in der Posix-Emulation *Cygwin* von Windows, so daß die Aussagen dann auch hier gelten.

Zeichenkette

Eine Zeichenkette ist eine Aneinanderreihung von Zeichen. Dies kann ein Satz, ein Wort oder auch ein einzelnes Zeichen sein. Der englische Fachbegriff hierfür lautet »String«. Beispiele für Zeichenketten sind:

Hai

Der weiße Hai

h

12345

-wichtig-

1.4.3 Was ist eine Shell?

Die Shell ist die direkte Schnittstelle des Betriebssystems zum Benutzer. Mit ihrer Hilfe werden die wichtigsten alltäglichen Aufgaben wie das Aufrufen von Programmen oder das Kopieren von Dateien bewältigt. Im Idealfall wird die komplette Funktionalität des Betriebssystemkerns zur Verfügung gestellt. Der Name »Shell« kommt aus dem Englischen und bedeutet »Muschel«, da die Shell anschaulich gesehen die äußerste Hülle um die verschiedensten Schichten eines Betriebssystems bildet.

Aus Anwendersicht ist die Shell eines Betriebssystems das Programm, das nach dem Login beziehungsweise nach dem Bootvorgang gestartet wird. Demnach ist also nicht nur die *command.com* von MS-DOS als rudimentäres Textmodus-Programm, sondern auch die grafische Oberfläche von Windows (*explorer.exe*) eine Shell. Dazwischen sind Welten, Bedienkomfort und Funktionsumfang unterscheiden sich doch erheblich. Noch größer wird diese Diskrepanz durch den Vergleich einer Unix-Shell mit der *explorer.exe*.

Wenn man bei Unix von der Shell spricht, meint man meist die */bin/sh*, doch strenggenommen ist bei einem grafischen Login der jeweilige X-Window-Fenstermanager die Shell. Um Mißverständnisse zu vermeiden, verwende ich jedoch die gängige Definition, die den Begriff Shell mit der Eingabeaufforderung gleichsetzt.

1.4.4 Was ist ein Shellskript?

Die Shell eines Unix-Systems ist mehr als nur eine simple Kommandozeile. Sie kennt viele Sprachelemente wie Fallunterscheidungen oder Schleifen und steht somit bekannten Hochsprachen in nichts nach. Selbstverständlich stehen alle diese Befehle auch an der Eingabeaufforderung zur Verfügung, doch sinnvoll nutzen lassen sich diese Kommandos erst, wenn sie in Programmen Anwendung finden. Solche Programme, die eigentlich eine Aneinanderreihungen mehrerer Befehle sind, nennt man im allgemeinen »Shellskripte«. In der deutschen Literatur liest man auch häufig den etwas hölzernen Begriff »Kommandoprogramm«.

Nicht übersehen werden darf jedoch, daß Shellskript nicht gleich Shellskript ist. Es ist vielmehr ein Gattungsbegriff genauso wie zu den Hochsprachen Pascal, C oder Java zählen. Wenn also ein Shellskript unter einer Shell funktioniert, ist nicht zwangsläufig davon auszugehen, daß es auch auf anderen läuft, auch wenn man sich auf eine gewisse Basissyntax in der Regel verlassen kann.

Vielleicht fragen Sie sich, warum es sich überhaupt lohnt, sich mit der Programmiersprache der Shell zu beschäftigen. Immerhin gibt es viele andere Skriptsprachen, die teilweise wesentlich mächtiger sind. Um Vorurteile aus dem Weg zu räumen, zeigt folgende Auflistung einige Vorteile der Shell gegenüber anderen Skriptsprachen:

- ◆ Shellskripten können sehr schnell geschrieben werden, da sie im einfachsten Fall nicht viel mehr sind als bloße Anreihungen mehrerer Kommandos. Man muß als Nutzer der Eingabeaufforderung also nichts lernen, solange keine Programmier-elemente benötigt werden.
- ◆ Die Shell ist die erste Wahl, wenn es darum geht, die Fähigkeiten von Unix voll auszuschöpfen. Insbesondere ist es nicht nur sehr einfach, andere Programme aus einem Shellskript zu starten, es ist sogar *wesentlicher Bestandteil* dieser Programmiersprache.
- ◆ Man kann Shellskripten so programmieren, daß sie auf *jedem* Unix-System lauffähig sind. Somit ist die Shell die einzige Programmiersprache, die auf jedem Unix verfügbar ist, egal welchen Alters.
- ◆ Der Umgang mit Dateien und Verzeichnissen ist in keiner Programmiersprache so einfach wie in der Shell. Besonders beeindruckend ist, daß keine neuen Befehle erlernt werden müssen – es können ja die wohlbekanntesten Unixbefehle verwendet werden.

Neben den überwältigenden Stärken gibt es selbstverständlich auch Schwächen. Diese werde ich in diesem Buch an gegebener Stelle erläutern.

1.5 Die Shells und das Kompatibilitätsproblem

Unix ist jetzt über dreißig Jahre alt. Obwohl sich dieses System im Laufe der Zeit sehr stark weiterentwickelt hat, blieben die Grundideen und der generelle Aufbau erhalten. Dasselbe gilt auch für die Shell. Sie bekam zwar in diesen drei Jahrzehnten viele Nachfolger, doch diese behielten weitgehend die Abwärtskompatibilität zur Urshell bei. Wie so oft, steckt jedoch der Teufel im Detail, wie wir noch sehen werden.

1.5.1 Die Geschichte der Shell

Die Geschichte der Shell als kontinuierlichen, geradlinigen Weg zu bezeichnen, ist sehr gewagt. In Wirklichkeit gab es nach der ersten Shell von Unix einige Zeit zwei Nachfolgershells, die um die Gunst der Benutzer warben. Glücklicherweise machte das System 4 von AT&T Schluß mit diesem Wettrennen und installierte standardmäßig alle drei Shells, die es damals gab: die Bourne-Shell, die C-Shell und die Korn-Shell. Blicken wir nun zurück und verfolgen die Entwicklung der Unix-Shell in ihren verschiedenen Etappen.

Alles begann mit der Bourne-Shell...

Bourne-Shell

Die Bourne-Shell, benannt nach ihrem Erfinder Steve Bourne, ist der Ursprung aller modernen Shells auf heutigen Unix-Systemen. Sie war 1978 aufgrund ihrer sehr mächtigen Programmiersprache eine Sensation. Im Gegensatz zur wahrscheinlich ersten Unix-Shell, der Thompson-Shell, die auf Unix Version 1 bis 6 lief, konnte man äußerst geschickte Shellskripten schreiben. Außerdem unterstützte sie genauso wie ihr Vorgänger die mächtigen Funktionen der Ein- und Ausgabeverarbeitung. Die erste Implementierung von Steve Bourne lief auf Unix Version 7.

Die Kritikpunkte an der Bourne-Shell konzentrieren sich hauptsächlich auf die mangelnden interaktiven Fähigkeiten. Mehr als Programme aufzurufen, ist am Prompt der Bourne-Shell nicht möglich. Deshalb wurde der Ruf nach Erleichterung bei den Benutzern immer lauter.

C-Shell

Die C-Shell ist ein Kind der BSD-Entwicklungslabore, als Vater wird oft Bill Joy genannt. Da den Erschaffern der C-Shell die Schwächen der Bourne-Shell sowie deren Vorgänger durchaus bewußt waren, wurde bei ihrer Entwicklung 1979 verstärkt auf eine bequeme Eingabeaufforderung Wert gelegt. Dieser Vorsatz führte unter anderem zur Einführung eines History-Mechanismus, mit dessen Hilfe bereits eingegebene Kommandos mit wenigen Tastendrücker nochmals wiederholt werden können. Auch verfügt die C-Shell über eine eigene Prozeßverwaltung, die die Arbeit mit

mehreren Hintergrundprozessen stark vereinfacht. Leider ist die C-Shell nicht abwärtskompatibel zur Bourne-Shell, da anstatt der Urshell die Programmiersprache C zum Vorbild genommen wurde. Auch wenn C-Programmierer keine Probleme mit dem Einstieg in die C-Shell haben werden, sind die Fähigkeiten dieser Skriptsprache heutzutage nicht mehr mit den aktuellen Shells erforderlich. Wegen der fehlenden Abwärtskompatibilität zur Bourne-Shell wird diese Shell in diesem Buch nicht in Verbindung mit den anderen Bourne-Shell-kompatiblen Shells vorgestellt. Da diese ungewöhnliche Shell in manchen Teilgebieten noch weiterhin verwendet wird, finden Sie in Kapitel 17 ab Seite 747 eine Einführung in die C-Shell und die Tcsh, einen populären Clone der C-Shell.

Korn-Shell

Die Korn-Shell, benannt nach ihrem Schöpfer David Korn, wurde bei AT&T für das berühmte System V entwickelt. Sie versteht sich als direkter Nachfolger der Bourne-Shell und ist deshalb fast vollständig zu dieser abwärtskompatibel. Wagt man einen Blick in den Quelltext dieser Shell, findet man an vielen Stellen einen Copyright-Hinweis wie

»sh by S. R. Bourne, rewritten by David Korn«

Die Korn-Shell ist also nicht nur abwärtskompatibel zur Bourne-Shell, sondern eine direkte Weiterentwicklung. Die Vorteile der Korn-Shell liegen auf der Hand: Der Zeileneditor macht die Arbeit an der Eingabeaufforderung zum Vergnügen (Stichwort: [Pfeil-Oben] holt das letzte Kommando zurück), Aliase sparen Tastendrucke und der Programmierer erhält einige sehr mächtige Programmiererelemente. Alles in allem ist die Korn-Shell in allen Bereichen ihren Vorgängern überlegen. Problematisch für die Shellprogrammierung ist nur, daß es mehrere Versionen der Korn-Shell gibt. Zu allem Überfluß wurden die meisten dieser Versionen nicht einmal offiziell mit einer Versionsnummer versehen. Um trotzdem zwischen den einzelnen Revisionen unterscheiden zu können, hat sich deshalb eingebürgert, das Jahr der Veröffentlichung als Versionsnummer zu »mißbrauchen«. Folgende Bezeichnungen werden in diesem Buch für die einzelnen Versionssprünge verwendet:

<i>Shell</i>	<i>Beschreibung</i>
Korn-Shell	Die erste Version der Korn-Shell wurde im Jahr 1983 veröffentlicht, heutzutage ist sie nur noch selten anzutreffen.
Korn-Shell86	Wurde im Jahr 1986 veröffentlicht; unterstützt einige wichtige neue Programmiererelemente wie <code>// ... //</code> oder Coprozesse.
Korn-Shell88	Wurde im Jahr 1988 veröffentlicht; enthält einige wenige Detailverbesserungen.
Korn-Shell93	Wurde im Jahr 1993 veröffentlicht; dieser Quantensprung enthält beachtliche Geschwindigkeitsverbesserungen, viele neue Programmiererelemente und einen noch mächtigeren Zeileneditor.

Tabelle 1.3

Beachten Sie bitte, daß bei der ersten Korn-Shell-Version in diesem Buch die Jahreszahl nicht angegeben ist. Dies entspricht der gängigen Schreibweise, ist jedoch manchmal nicht ganz eindeutig. In allen Fällen, in denen es zu Verwechslungen kommen könnte, verwende ich deshalb die Begriff »ursprüngliche Korn-Shell« oder »Original-Korn-Shell«.

Bourne-Again-Shell (Bash)

Auf GNU/Linux-Systemen hat sich die Bash als Standard bewährt. Diese versteht sich, wie der Name schon vermuten läßt, als konsequente Weiterentwicklung der Bourne-Shell. Wegen der Implementierung vieler grundlegender Features der Korn-Shell kann man jedoch schon fast von einem Korn-Shell-Clone sprechen. Die Tatsache, daß sie zu keiner bekannten Shell wirklich kompatibel ist, wird immer wieder kritisiert und macht es deshalb für uns notwendig, sie in diesem Buch gesondert zu betrachten. Dabei unterscheiden wir zwischen der Version 1 und der Version 3 (zum Zeitpunkt der Drucklegung dieses Buches war Version 3.00 aktuell). Obwohl die Version 1 – ab nun immer Bash1 geschrieben – seit Einführung der Version 2 überholt ist, ist sie in der Praxis auf Systemen mit wenig Speicher noch oft anzutreffen. Deshalb berücksichtigt dieses Buch sowohl die Bash1 als auch die Bash3.

Auf eine explizite Beschreibung der Bash3 verzichte ich, da sich beim Sprung von Version 2.05 auf Version 3.00 nicht mehr besonders viel geändert hat. Schuld daran ist die ungewöhnliche Release-Politik der Bash-Entwickler: Sie verstehen die kleinen Releases lediglich als Zwischenstände der Erweiterung der Bash. Die Herausgabe der Version 3.00 dokumentiert dabei die Vervollständigung der Weiterentwicklung und enthält deshalb nur wenig Neuerungen gegenüber Version 2.05.

Eine Folge dieser Politik ist das gravierende Problem, daß selbst mit kleinen Versionssprüngen große Änderungen verbunden sind. Da eine detail-

lierte Unterscheidung (beispielsweise zwischen 2.03 und 2.04) viel zu platzraubend wäre, kann es vorkommen, daß manche Beispiele in diesem Buch von Ihrer Bash nicht ausgeführt werden können – gegebenenfalls müssen Sie also die neueste Version auf Ihrem System installieren.

Die Bash ist wohl eines der umstrittensten Programme von Linux. Für die Linux-Anhänger ist sie das wichtigste und mächtigste Instrument für den Umgang mit dem System, für Linux-Spötter ist sie nur eine inkompatible Verballhornung der guten alten Bourne-Shell. Immer wieder werden der Speicherhunger, die vielen »unnötigen« Erweiterungen und die Anmaßung, »Bourne« im Namen zu tragen, angeprangert. Allerdings ist es wiederum unumstritten, daß die Bash zu einer der beliebtesten Shells – unix-übergreifend – reifte und durch sie der Siegeszug von Linux erst richtig möglich wurde. Übrigens war die Bash, damals Vorzeigeprodukt des GNU-Projekts, das erste Programm, das Linus Torvalds auf seinem Linux-Kernel zum Laufen brachte.

Z Shell

Zu guter Letzt befassen wir uns mit einem echten Außenseiter, dessen Fangemeinde jedoch – zu Recht – stetig wächst: die Z Shell, ursprünglich von Paul Falstad entwickelt. Ihre Stärken lassen sich kaum mit ein paar Sätzen ausdrücken, viel zu viele Features wurden in diese eierlegende Wollmilchsau gepackt. Die Besonderheit der Z Shell ist die Tatsache, daß sie alle drei großen Shells in sich vereint. Neben der Bourne-Shell werden insbesondere die C-Shell und die Korn-Shell emuliert. Auch wenn die Z Shell bereits viele Erweiterungen der Korn-Shell93 enthält, ist sie momentan lediglich zur Korn-Shell88 kompatibel.

Diesem Buch liegt die stabile Version 4.2.5 zugrunde. Leider gilt für die Z Shell das gleiche wie für die Bash: Selbst kleine Versionssprünge können große Änderungen mit sich bringen. Bitte haben Sie jedoch Verständnis, daß ich aus Gründen der Übersichtlichkeit auf ältere Versionen der Z Shell keine Rücksicht nehmen kann.

Sprachregelung in diesem Buch

Da es sich dieses Buch zur Aufgabe gemacht hat, gleich mehrere Shells zu besprechen, ist es nicht immer ganz leicht, zu beschreiben, welche Shells welche Funktionalität besitzen. Der Einfachheit halber wird deshalb sehr oft auf das Alter der Shells zurückgegriffen. »Alle Shells ab Korn-Shell« bedeutet beispielsweise, daß das folgende in allen Varianten der Korn-Shell, sowie in der Bash und der Z Shell gilt, also in allen Shells außer der Bourne-Shell. »Ab Korn-Shell86« bezieht sich auf die Korn-Shell86 und die Korn-Shell93. Falls Bash ohne Versionsnummer angegeben ist, gilt der entsprechende Kontext für Bash1 und Bash3.

1.5.2 Welche Shells habe ich auf meinem System?

Sie haben im letzten Kapitel einige Shells kennengelernt, doch wissen Sie bisher noch nicht, welche Shells auf Ihrem System zur Verfügung stehen. Leider reicht es nicht, zu sagen, daß auf einem Unix-System normalerweise Bourne-Shell und Korn-Shell installiert sind. Dazu gibt es zu viele Unterschiede in den verschiedenen Implementierungen.

Beginnen wir mit einer ersten Bestandsaufnahme: Machen Sie mit dem Unix-tool *whereis* ausfindig, machen, welche Shells überhaupt installiert sind:

```
$ whereis sh
sh: /bin/sh
$ whereis ksh
ksh: /bin/ksh
$ whereis bash
bash:
$ whereis zsh
zsh: /usr/local/bin/zsh
```

Auf diesem System ist die Bash anscheinend nicht installiert.

Auch wenn wir nun wissen, welche Shelltypen zur Verfügung stehen, sind die Versionsnummern noch unklar. Da die Korn-Shell keine einfache Möglichkeit bietet, die Version ausfindig zu machen, müssen wir einige Tricks anwenden – mehr dazu jedoch erst in Kapitel 16.3.4 ab Seite 732.

Manchmal ist selbst die Unterscheidung zwischen Bourne-Shell und Korn-Shell schwierig. Beispielsweise besitzt IRIX keine eigene Bourne-Shell: */bin/sh* ist hier lediglich ein symbolischer Link auf */bin/ksh*. Zusätzliche Probleme treten häufig auf, wenn sogenannte Clones eingesetzt werden. Der prominenteste Clone ist die Public-Domain-Korn-Shell (*pdksh*). Hierbei handelt es sich um den Versuch, die Korn-Shell möglichst vollständig nachzuprogrammieren – leider ist die Umsetzung bisher noch nicht ganz vollständig, momentan wird immerhin die Korn-Shell88 fast komplett emuliert. Das grundsätzliche Problem der Public-Domain-Korn-Shell ist, daß die Weiterentwicklung in sehr kleinen Schritten erfolgt. Es kann also vorkommen, daß das Skript aus Kapitel 16.3.4 (siehe Seite 732 ff.) eine Korn-Shell93 meldet, obwohl nur einige wenige Features bisher implementiert wurden. Das wird vor allem BSD-Benutzer treffen, da hier die *pdksh* als Korn-Shell-Ersatz eingesetzt wird. Ein anderer bekannter Clone ist die Almquist Shell (*ash*). Diese Shell entspricht einer Bourne-Shell, auch wenn einige Erweiterung der Korn-Shell in ihr Einzug gefunden haben. NetBSD und FreeBSD benutzen eine eigene Variante der Almquist Shell als */bin/sh*.

Die folgende (unvollständige) Tabelle bietet eine Übersicht über die auf populären Unix-Systemen als Standard eingesetzten Shells:

Betriebssystem	Shells
Linux	bash (sh), pdksh (ksh)
OpenBSD	pdksh (sh, ksh)
FreeBSD	ash (sh), pdksh (ksh), tcsh
NetBSD	ash (sh), pdksh (ksh), tcsh
Solaris	ksh (sh)
IRIX	ksh (sh)
AIX	sh, ksh
HP-UX	sh, ksh
Mac OS X 10.3	bash (sh), zsh

Table 1.4

Bitte beachten Sie folgende Hinweise zu dieser Tabelle:

- ◆ In dieser Tabelle ist jeweils in Klammern angegeben, welche Shell die jeweilige Shell auf dem System ersetzt. Beispielsweise ist die Bourne-Shell (sh) von FreeBSD in Wirklichkeit die Almquist Shell.
- ◆ ksh steht jeweils für eine »echte« Korn-Shell, es handelt sich also entweder um eine Lizenzierung von David Korn oder um eine eigene Implementation. (Dies ist der Grund, warum es selbst unter den verschiedenen Korn-Shells Unterschiede gibt.) Da momentan kein System die Korn-Shell93 einsetzt, handelt es sich jeweils um eine Korn-Shell88. (Lizenzprobleme führten lange Zeit dazu, daß kein Softwarehersteller die neue Version einsetzte.) Da die Korn-Shell93 mittlerweile als Open Source freigegeben wurde, können Sie sich das Paket jedoch jederzeit von der Website <http://www.kornshell.com> downloaden und installieren.
- ◆ Die Z Shell und die Bash sind für jedes dieser Unix-Systeme verfügbar. Falls eine dieser Shells in der Tabelle nicht aufgeführt ist, heißt das nur, daß sie dem Standardsoftwarepaket nicht beigelegt ist.
- ◆ Da es kein »echtes« Linux gibt, enthält die Tabelle nur die Shells, die durch eine andere Shell emuliert werden. Die tatsächliche Standardinstallation hängt von der jeweiligen Distribution ab. Fester Bestandteil ist in der Regel lediglich die Bash.