

Kapitel 2

Auf Tuchfühlung mit UNIX/Linux

Nach so viel Theorie wird es Zeit, praktische Erfahrungen mit UNIX zu sammeln. Dieses Kapitel beschreibt die dazu notwendigen Schritte. In diesem Rahmen werden weitere wichtige Grundbegriffe wie z.B. das Verfahren des „Logins“ und der Begriff des „Home“-Verzeichnisses eingeführt. Ein wesentlicher Teil dieses Kapitels widmet sich der Beschreibung von Funktion und Möglichkeiten der Shell, dem UNIX-Kommandointerpreter. Im Anschluß an die Beschreibung der sogenannten Bourne-Shell werden in weiteren Abschnitten weitverbreitete Varianten dieser Standard-Shell beschrieben. Dazu zählt sowohl die Bourne-Again-Shell, die C-Shell als auch die Korn-Shell, die alle die Funktionalität der Bourne-Shell um für den Benutzer nützliche Eigenschaften erweitern.

Der Begriff UNIX steht im folgenden, falls nichts anderes gesagt wird, sowohl stellvertretend für kommerzielle UNIX-Systeme als auch für Linux.

2.1 Einführung

UNIX kann dem Benutzer in verschiedener Form, genauer gesagt, an verschiedenen Geräten begegnen. Da ist zum einen die *Konsole*, bestehend aus Tastatur und Bildschirm. Jedes UNIX-System verfügt über eine solche Konsole. Die Konsole hebt sich von anderen Terminals dadurch hervor, daß dort Meldungen des Systems erscheinen, beispielsweise beim Booten (Starten) des Rechners. Oftmals ist es so, daß man, um besondere Rechte im System erlangen zu können, an der Konsole arbeiten muß. Daher wird diese oft vom Systemadministrator verwendet.

In der Regel wird die Konsole kein reines Text-Terminal sein, sondern ein Graphik-Terminal mit einer graphischen Benutzeroberfläche. Für einen Benutzer, der an einem solchen System arbeiten will und sich zu diesem Zweck beim System anmelden möchte, ist

es kein prinzipieller Unterschied, ob er dies an einem Text-Terminal oder einem Graphik-Terminal mit einer darauf laufenden graphischen Benutzeroberfläche macht. Die unten beschriebene Prozedur der Anmeldung beim System bleibt immer gleich.

Neben einer Konsole können an einen UNIX-Rechner viele weitere Terminals angeschlossen sein, oder aber der Rechner ist Teil eines Netzwerks, in dem jeder Rechner von jedem anderen im Netz befindlichen erreicht werden kann. Dadurch wird es möglich, sich bei jedem beliebigen Rechner als Benutzer anzumelden. Über das Netzwerk kann man dann problemlos auf die eigenen Daten zugreifen, unabhängig davon, auf welchem Rechner sie wirklich gespeichert sind.

Häufig wird man nicht die Gelegenheit haben, sich direkt an einen UNIX-Rechner setzen zu können. In diesem Fall werden mit dem UNIX-Rechner verbundene PCs als Terminals verwendet. Diese Lösung wird oft aus Kostengründen gewählt, wenn man anstelle von mehreren kleineren (preiswerteren) UNIX-Rechnern nur einen leistungsfähigen (und teuren) angeschafft hat. Um dennoch mehreren Leuten die Nutzung des Rechners zu ermöglichen, werden in diesem Fall preiswerte PCs als Terminals zur Dateneingabe und Anzeige eingesetzt.

UNIX kennt einige Sonderzeichen. Neben dem /-Zeichen sind noch weitere Zeichen von besonderer Bedeutung. All diese Zeichen können nicht ohne weiteres z.B. für Datei- oder Verzeichnisnamen verwendet werden, da sie eine besondere Bedeutung haben, die in den folgenden Kapiteln beschrieben wird. Die wichtigsten Sonderzeichen sind:

/ | \ - < > , # . ~ ! \$ & () [] { } ` ' " ? ^

Man sollte daher beispielsweise nicht versuchen, eine Datei `kosten(96)` zu nennen, da in diesem Fall die Sonderbedeutung der runden Klammern zu Problemen führen würde.

Grundsätzlich werden in UNIX immer Klein- und Großbuchstaben unterschieden! Daher ist der Dateiname `Datei` verschieden vom Namen `datei`. Wer bisher mit einem System gearbeitet hat, das diesen Unterschied nicht kennt, wird eine Weile benötigen, bis er sich daran gewöhnt hat.

2.2 Die Login-Prozedur

Jeder, der an einem UNIX-Rechner arbeiten möchte, braucht eine sogenannte *Kennung*, auch *login* genannt. Die Kennung ist nichts anderes als ein Name, der den Benutzer eines UNIX-Rechners identifiziert. Das bedeutet, daß jeder Benutzer, der an einem bestimmten UNIX-System arbeiten will, einen „Namen“ erhält, der kein zweites Mal existieren darf. Der Name sollte sinnvoll gewählt werden, da er bei einer ganzen Reihe von Gelegenheiten wieder Verwendung findet. Ein Beispiel dafür ist die „E-Mail“-Adresse. Ein Teil dieser Adresse (beispielsweise `mueller@firma.com`) der elektronischen Post ist in aller Regel der Kennungsname des Benutzers. Darüber hinaus muß bei der Wahl des Kennungsnamens beachtet werden, daß bestimmte Namen wie `root`, `sys`, `daemon` als Systemkennungen bereits vergeben sind und dem Benutzer nicht mehr zur Verfügung stehen.

Kennungen können nur vom Systemadministrator, dem Verwalter eines UNIX-Systems, der über besondere Berechtigungen verfügt, vergeben oder entzogen werden. Oftmals werden die Nachnamen des Benutzers oder ein Teil einer Personalnummer als Kennungsname vergeben. Zu jeder Kennung gehört ein Paßwort, das nur dem Benutzer selbst bekannt sein darf. Das Paßwort soll sicherstellen, daß nur der Inhaber einer Kennung an dem Rechner arbeiten kann. Durch das Paar Kennungsname-Paßwort ist auch sichergestellt, daß niemand unberechtigt die Daten eines anderen Benutzers manipulieren kann. In der Regel hat nur der Benutzer selbst Zugriff auf die von ihm erzeugten Dateien. Wichtig ist daher, daß das Paßwort wirklich geheim bleibt!

Das erste, was ein Benutzer von einem UNIX-Rechner sieht, ist die Aufforderung zum „Einloggen“, also die Identifizierung des Benutzers durch Eingabe des Kennungsnamens (Login-Namens) und der anschließenden Eingabe des Paßworts. Der Name der Kennung wird direkt hinter dem Wörtchen

```
login:
```

einggegeben, woraufhin direkt die Frage nach dem Paßwort erscheint:

```
Password:
```

Das Schreiben des Paßwortes geschieht blind, d.h., man sieht nicht, was man tippt – andere, die einem dabei gerade über die Schulter schauen, ebenfalls nicht!

Hat man entweder einen falschen Kennungsnamen eingegeben oder sich beim Paßwort vertippt, so erscheint die Fehlermeldung `Login incorrect` und anschließend die erneute Aufforderung zum Einloggen. Macht man wiederholt fehlerhafte Eingaben, verzögert der Rechner das Erscheinen des nächsten Logins immer mehr, um dadurch zu verhindern, daß Paßwörter durch ausdauerndes Probieren (evtl. automatisiert mit einem Programm) herausgefunden werden können.

2.3 Meldungen nach dem Login

Was genau nach dem Login geschieht, hängt etwas davon ab, ob das System, an dem man sich gerade angemeldet hat, über eine grafische Benutzeroberfläche verfügt oder ob es sich um ein reines Text-Terminal handelt. Wer an einem System mit graphischer Oberfläche arbeitet, sieht natürlich zunächst, wie diese Oberfläche aufgebaut wird, was einige Sekunden dauern kann. Anschließend sollte manuell ein Programm wie `konsole` unter KDE oder ein entsprechendes Programm unter der verwendeten Oberfläche gestartet werden, um die folgende Beschreibung nachvollziehen zu können.

Nach dem Login an einem Text-basierten Terminal oder dem Starten einer neuen Shell oder einer Anwendung wie z.B. `konsole` können verschiedene Meldungen auf dem Bildschirm erscheinen. Sie dienen dazu, auf wichtige Dinge hinzuweisen, oder geben spezielle benutzerspezifische Informationen aus.

Zunächst erscheint während des Anmeldevorgangs die sogenannte *motd*, die „Message Of The Day“. Sie enthält Informationen, die für alle Benutzer von Interesse sind, etwa daß ein Rechner an einem bestimmten Tag wegen Installationsarbeiten nicht verfügbar ist,

oder vielleicht Informationen über die neuesten Versionen einer Software. Erstellt wird die „Message Of The Day“ immer vom Systemadministrator.

Als weitere Information wird in der gestarteten Shell die Zeit des *last login*, also Zeit und Datum des letzten Logins ausgegeben. Hier kann der Benutzer sehen, wann er sich zum letzten Mal bei dem System angemeldet hat. Diese Information wird vom UNIX-System automatisch verwaltet.

Falls für den Benutzer auf dem lokalen System Post vorliegt, erscheint zudem die Meldung *You have new mail*. Diese Meldung zeigt an, wenn elektronische Post (E-Mail) für den Benutzer eingetroffen ist, die er sich mit Hilfe eines Mail-Programms ansehen und beantworten kann. Diese Meldung sagt jedoch nur etwas darüber aus, ob auf dem System, an dem man sich angemeldet hat, Mail für den Benutzer vorhanden ist. Unter Verwendung eines zentralen Mailservers, der die Mail für alle Benutzer verwaltet, hängt es von der Art des Zugriffs auf diese Mail ab, ob die Meldung der Shell bzgl. neuer Mail stimmt. Mehr dazu in Kapitel 6.7.3 auf Seite 280.

Zuletzt erscheint der Shell-Prompt. Er zeigt an, daß die Shell auf Eingaben des Benutzers wartet. Zu diesem Zweck zeigt die Shell am Zeilenanfang ein spezielles Zeichen, den *Shell-Prompt* an. Der Shell-Prompt kann ein einfaches Zeichen wie z.B. `$` sein oder aber auch eine informative Zeichenkette enthalten, wie beispielsweise den Rechnernamen, den Kennungsnamen und das aktuelle Verzeichnis:

```
fred@tux:/home/fred >
```

Hinter dem Shell-Prompt kann der Benutzer Befehle eingeben, die von der Shell ausgeführt werden sollen. Die Eingabe wird jedoch erst bearbeitet, wenn der Benutzer die Zeile mit der *Return* (oft auch *Enter*)-Taste abgeschlossen hat. Wenn die Shell bereit ist, den nächsten Befehl auszuführen, erscheint ein neuer Prompt, um die Eingabebereitschaft anzuzeigen.

2.4 Dateien und Verzeichnisse

Dateien spielen in UNIX eine zentrale Rolle. Dateien sind Mittel, um Informationen zu speichern. Konzeptuell sind Dateien wie Papierblätter, auf denen Informationen untergebracht sind. Sie können grob nach ihrem Inhalt bzw. ihrer Verwendung eingeteilt werden.

So gibt es Dateien, die Programme enthalten, also ausgeführt werden können, die sogenannten *executables*. Andere Dateien sind einfache Textdateien, die man sich direkt ansehen kann. Der Inhalt einer solchen Datei ist eine Folge von Zeichen, die nach dem ASCII-Code (**A**merican **S**tandards **C**ode for **I**nformation **I**nterchange) codiert sind. Der ASCII-Code ist ein Internationaler Standard, in dem festgelegt wurde, welchem Zeichen welcher Code zugeordnet ist. Tabelle 2.1 auf Seite 44 zeigt diese Zuordnung.

Man sieht, daß der Standard alle Codes von „Dezimal 0“ bis „Dezimal 127“ umfaßt. Sonderzeichen, wie z.B. Umlaute, sind hier nicht enthalten. Dennoch tauchen diese Zeichen in Textdateien auf. Das Problem mit diesen Zeichen besteht jedoch darin, daß auf einem anderen System plötzlich andere Zeichen an der gleichen Stelle auftauchen können. Das

geschieht beispielsweise, wenn man unter UNIX Umlaute verwendet und sich diese Datei auf einem MS-DOS-Rechner ansieht und umgekehrt. Für diese Fälle braucht man Konvertierungsprogramme, die die nicht standardisierten Zeichen umsetzen.

Ein weiterer Dateityp in UNIX sind die Gerätedateien. In UNIX werden alle Geräte (Bandlaufwerk, Drucker . . .) als Dateien dargestellt, die sich auf den ersten Blick nicht von „normalen“ Dateien unterscheiden. Auch sonst kann man in gewohnter Weise auf diese Gerätedateien zugreifen, auf diese schreiben oder von ihnen lesen. Der Unterschied zu „normalen“ Dateien besteht darin, daß die Daten, die man auf diese Dateien schreibt oder von ihnen liest, zu dem Gerät gesendet, bzw. von dem Gerät gelesen werden, das der Datei zugeordnet ist. So kann man auf diesem Weg beispielsweise Daten zu einem Bandlaufwerk schicken, indem man die entsprechenden Daten auf die Datei schreibt, die das Bandlaufwerk repräsentiert.

Damit eine Datei ansprechbar ist, ist jede Datei durch einen Namen eindeutig gekennzeichnet. Dieser Name darf aus Buchstaben, Zahlen und einigen Sonderzeichen oder auch Leerzeichen bestehen. Allerdings sollte man sich bei der Verwendung bestimmter Sonderzeichen darüber im klaren sein, daß diese Zeichen unter Umständen zu Problemen führen können, da die Shell, die die Eingaben des Benutzers verarbeitet, bestimmten Zeichen eine besondere Bedeutung zuweist. In Kapitel 2.7.3 wird darauf näher eingegangen.

Um die mit der Zeit wachsende Flut von Dateien organisieren zu können, gibt es die Möglichkeit, Dateien zu Gruppen in sogenannten Verzeichnissen zusammenzufassen. Jedes Verzeichnis erhält ebenfalls einen Namen, unter dem es ansprechbar ist. Dadurch hat man die Möglichkeit, seine Daten sinnvoll aufzuteilen.

Beispielsweise kann ein Verzeichnis *a* die Dokumentation zu einem Projekt *A* aufnehmen, während alle Dokumente zu einem weiteren Projekt *B* in einem anderen Verzeichnis mit dem Namen *b* liegen. Dabei werden für jedes Projekt verschiedene Gruppen von Informationen gespeichert (z.B. Zeitplan, Kostenplan). Da sowohl das Verzeichnis *a* als auch *b* Projekte beschreiben, wäre es sinnvoll, beide Verzeichnisse in einem weiteren übergeordneten Verzeichnis unterzubringen (nennen wir es `projekte`). Dadurch werden diese zusammengehörigen Informationen von anderen Daten (z.B. Adressen von Kunden) abgegrenzt. Dem Prinzip nach entspricht ein Verzeichnis einem Aktenordner, der viele Texte, geordnet nach bestimmten Kriterien, beinhaltet.

Will man auf eine bestimmte Information zugreifen, muß man erst den Aktenordner mit der entsprechenden Bezeichnung suchen und kann dann darin das gesuchte Dokument finden. Entsprechend muß man in UNIX, um auf eine bestimmte Datei zuzugreifen, sowohl den Namen des Verzeichnisses angeben, in dem sich die Datei befindet, als auch den Namen der Datei selbst.

In UNIX wird dies dadurch erreicht, daß man den Namen des Verzeichnisses vor den Namen der Datei schreibt und beide mit einem `/` verbindet. Da ein Verzeichnis nicht nur Dateien enthalten darf, sondern auch weitere Verzeichnisse, in denen wiederum Dateien oder Verzeichnisse enthalten sein können, entsteht hierdurch eine baumartige Struktur, die einen Anfang (die Wurzel oder `root`) hat, sozusagen das alleroberste Verzeichnis, das in keinem anderen Verzeichnis enthalten ist. Von der Wurzel aus kann man jede Datei erreichen, indem man den Weg dorthin von der Wurzel aus angibt, also die Kette der Verzeichnisnamen

Tabelle 2.1: Der ASCII-Code

Oktal:	Dezimal:	Hexadezimal:	Zeichen:	Oktal:	Dezimal:	Hexadezimal:	Zeichen:
000	0	00	NUL '\0'	100	64	40	@
001	1	01	SOH	101	65	41	A
002	2	02	STX	102	66	42	B
003	3	03	ETX	103	67	43	C
004	4	04	EOT	104	68	44	D
005	5	05	ENQ	105	69	45	E
006	6	06	ACK	106	70	46	F
007	7	07	BEL '\a'	107	71	47	G
010	8	08	BS '\b'	110	72	48	H
011	9	09	HT '\t'	111	73	49	I
012	10	0A	LF '\n'	112	74	4A	J
013	11	0B	VT '\v'	113	75	4B	K
014	12	0C	FF '\f'	114	76	4C	L
015	13	0D	CR '\r'	115	77	4D	M
016	14	0E	SO	116	78	4E	N
017	15	0F	SI	117	79	4F	O
020	16	10	DLE	120	80	50	P
021	17	11	DC1	121	81	51	Q
022	18	12	DC2	122	82	52	R
023	19	13	DC3	123	83	53	S
024	20	14	DC4	124	84	54	T
025	21	15	NAK	125	85	55	U
026	22	16	SYN	126	86	56	V
027	23	17	ETB	127	87	57	W
030	24	18	CAN	130	88	58	X
031	25	19	EM	131	89	59	Y
032	26	1A	SUB	132	90	5A	Z
033	27	1B	ESC	133	91	5B	[
034	28	1C	FS	134	92	5C	\
035	29	1D	GS	135	93	5D]
036	30	1E	RS	136	94	5E	^
037	31	1F	US	137	95	5F	_
040	32	20	SPACE	140	96	60	
041	33	21	!	141	97	61	a
042	34	22	"	142	98	62	b
043	35	23	#	143	99	63	c
044	36	24	\$	144	100	64	d
045	37	25	%	145	101	65	e
046	38	26	&	146	102	66	f
047	39	27	'	147	103	67	g
050	40	28	(150	104	68	h
051	41	29)	151	105	69	i
052	42	2A	*	152	106	6A	j
053	43	2B	+	153	107	6B	k
054	44	2C	,	154	108	6C	l
055	45	2D	-	155	109	6D	m
056	46	2E	.	156	110	6E	n
057	47	2F	/	157	111	6F	o
060	48	30	0	160	112	70	p
061	49	31	1	161	113	71	q
062	50	32	2	162	114	72	r
063	51	33	3	163	115	73	s
064	52	34	4	164	116	74	t
065	53	35	5	165	117	75	u
066	54	36	6	166	118	76	v
067	55	37	7	167	119	77	w
070	56	38	8	170	120	78	x
071	57	39	9	171	121	79	y
072	58	3A	:	172	122	7A	z
073	59	3B	;	173	123	7B	{
074	60	3C	=	174	124	7C	
075	61	3D	>	175	125	7D	}
076	62	3E	?>	176	126	7E	~
077	63	3F	?	177	127	7F	DEL

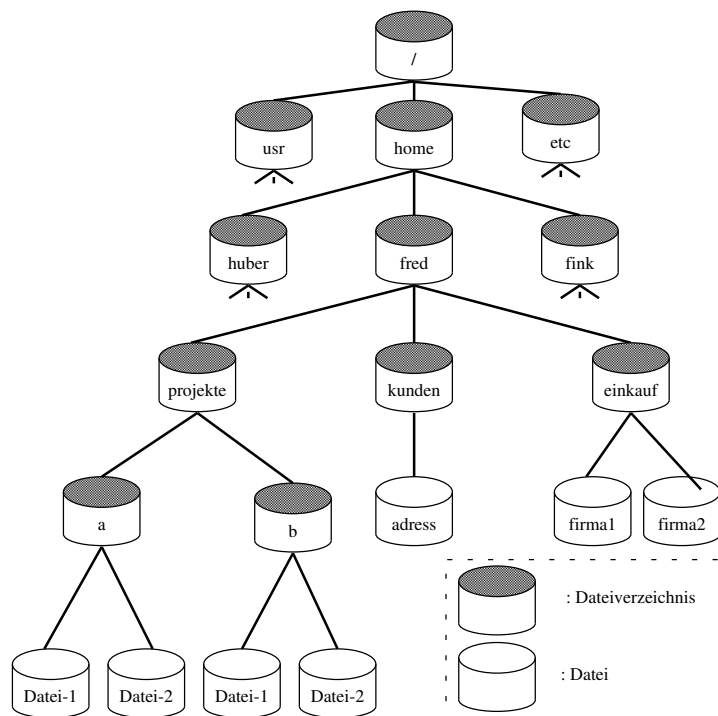


Abbildung 2.1: Beispiel eines UNIX-Verzeichnisbaums

zu der Datei auflistet. Eine solche Angabe nennt man *Pfad*. In obigem Beispiel ist der Pfad zu der Datei `Datei-1` aus dem Projekt A beispielsweise:

```
/home/fred/projekte/a/Datei-1
```

Das sich ergebende Bild ist ein sogenanntes hierarchisches Dateisystem mit einer Wurzel, der *root*, durch den `/` gekennzeichnet. Verzeichnisse, die in einem anderen Verzeichnis liegen, nennt man Unterverzeichnisse oder *subdirectories*. Das Verzeichnis, das andere enthält, ist das Vaterverzeichnis. So ist in obigen Beispielen `a` ein Unterverzeichnis von `projekte`, und `fred` ist das Vaterverzeichnis von `projekte`, `kunden` und `einkauf`.

Wenn ein Benutzer die Daten des Projekts `a` aktualisieren möchte, muß er auf die Dateien, des Verzeichnisses `/home/fred/projekte/a` zugreifen. Da es umständlich wäre, bei jedem Zugriff auf eine der Dateien den gesamten Pfad angeben zu müssen, gibt es den Begriff des *aktuellen Verzeichnisses*. Unser Benutzer könnte beispielsweise das Verzeichnis des Projekts `A` zu seinem aktuellen machen. Das hat zur Folge, daß er die in diesem Verzeichnis enthaltenen Dateien direkt durch ihren Namen erreichen kann, ohne den gesamten Pfad von der Wurzel aus angeben zu müssen. Das aktuelle Verzeichnis ist also eine Art „aktueller Standort“ im Verzeichnisbaum, von dem aus man auf Dateien und Verzeichnisse zugreifen kann.

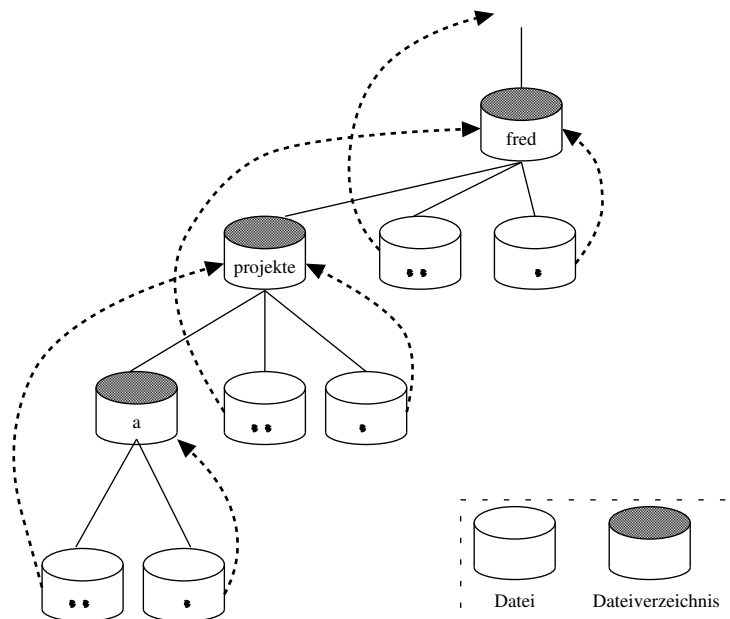


Abbildung 2.2: Die Bedeutung von `.` und `..`

Zwei Dateien, die in jedem Verzeichnis enthalten sind (nicht in Bild 2.1 auf der vorigen Seite dargestellt), haben eine besondere Bedeutung: `.` und `..`. Befindet man sich in einem Verzeichnis, etwa `projekte`, so steht `.` für genau dieses, also das aktuelle Verzeichnis. `..` steht für das nächsthöher liegende Verzeichnis. In unserem Beispiel also `fred`. Bild 2.2 veranschaulicht diesen Sachverhalt.

Es zeigt nur einen Ausschnitt der Verzeichnisse aus Bild 2.1 auf der vorigen Seite ohne die dort dargestellten Dateien. Dafür sind hier die `.` und `..` Dateien in ihrer Bedeutung dargestellt. Die gestrichelten Linien deuten an, für welches Verzeichnis eine `.` oder `..`-Datei steht.

Diese speziellen Dateien können dazu verwendet werden, in dem Verzeichnisbaum nicht von der Wurzel (`/`) aus, sondern relativ zu dem Verzeichnis, in dem man sich gerade befindet, auf Dateien zuzugreifen.

Will man in dem Beispiel von Bild 2.1 auf der vorigen Seite auf die Datei `Datei-1` im Verzeichnis `a` zugreifen, so kann man zum einen vom aktuellen Verzeichnis ausgehen. Nehmen wir an, `/home/fred/projekte/b` wäre unser aktuelles Verzeichnis. `Datei-1` wäre dann erreichbar über `../a/Datei-1`. Diese Art eines Pfades nennt man relativen Pfad, da er nicht von der Wurzel ausgeht, sondern von einem beliebigen Verzeichnis im Dateibaum. Dabei kann der erste Punkt inklusive des Pfadtrennzeichens `/` auch weggelassen werden. UNIX nimmt dann an, daß man vom aktuellen Verzeichnis ausgeht.

Die zweite Möglichkeit ist, von der Wurzel aus auf `Datei-1` des Projekts *A* zuzugreifen. Der Pfad lautet hier `/home/fred/projekte/a/Datei-1`. Dieser Pfad wird absolut genannt, da er von der Wurzel `/` ausgeht.

2.5 Das Home-Verzeichnis

Jeder Benutzer hat ein Home-Verzeichnis, in dem seine eigenen Daten liegen. Dieses Verzeichnis wurde vom Systemverwalter eingerichtet und liegt entweder auf einer der im Rechner eingebauten Festplatten oder aber auf einem über ein Netzwerk verbundenen Rechner. Für den Benutzer macht dies keinen Unterschied. Er sieht nicht, wo seine Daten tatsächlich gespeichert werden, und muß sich auch nicht damit befassen.

Im Gegensatz zu MS-DOS, wo jeder Benutzer auf alle Daten des Systems zugreifen kann, hat in UNIX jeder Benutzer seinen eigenen Datenbereich, auf den nur er selbst zugreifen kann. Dieser Datenbereich ist sein Home-Verzeichnis. Hier kann jeder Benutzer seine eigenen Dateien und Verzeichnisse anlegen, ohne daß diese Dateien von einem anderen Benutzer gelesen oder gelöscht werden können. Weil jeder Benutzer sein eigenes Home-Verzeichnis hat, besteht auch keine Gefahr, daß der eine dem anderen versehentlich (oder auch absichtlich) Daten zerstört oder, was mindestens ebenso schlimm ist, diese Daten verändert.

Nachdem der Benutzer sich bei dem System erfolgreich angemeldet hat, befindet er sich automatisch in seinem Home-Verzeichnis, also einem ganz bestimmten Punkt innerhalb des Dateisystems.

Das Verzeichnis, in dem ein Benutzer sich befindet, wird aktuelles Verzeichnis genannt. Nach dem Login-Vorgang ist das aktuelle Verzeichnis gleich dem Home-Verzeichnis. Wechselt der Benutzer später in ein anderes Verzeichnis, dann wird dieses zu seinem aktuellen.

Um herauszufinden, in welchem Verzeichnis man sich gerade befindet, was also das aktuelle Verzeichnis ist, gibt es den Befehl `pwd`. Nach dem Einloggen des Benutzers `fred` könnte die Eingabe von `pwd` zu der Ausgabe:

```
fred@tux:/home/fred > pwd
/home/fred
```

führen. Das heißt, `/home/fred` ist direkt nach dem Login sowohl das aktuelle als auch das Home-Verzeichnis des Benutzers `fred`. Wechselt der Benutzer `fred` in ein Unterverzeichnis mit dem Namen `tmp`, so würde bei der Ausführung von `pwd` jetzt das Verzeichnis

```
fred@tux:/home/fred > cd tmp
fred@tux:/home/fred/tmp > pwd
/home/fred/tmp
```

ausgegeben werden. Sein neues aktuelles Verzeichnis! Für den Wechsel von einem Verzeichnis zu einem anderen dient der Befehl `cd`, der als Parameter den Pfad zu dem Verzeichnis braucht, das der Benutzer zu seinem aktuellen machen will. Um also von dem

Verzeichnis `/home/fred` in das Verzeichnis `/home/fred/tmp` zu wechseln, gibt der Benutzer den Befehl `cd tmp` an. Anschließend ist `/home/fred/tmp` sein aktuelles Verzeichnis.

Gibt `fred` bei der Eingabe von Kommandos Dateinamen an, ohne einen Pfad voranzustellen, nimmt UNIX an, daß die Dateien in seinem aktuellen Verzeichnis gemeint sind. Möchte der Benutzer eine Datei bearbeiten, die außerhalb seines aktuellen Verzeichnisses liegt, muß er einen Pfad zu dieser Datei angeben. Dabei hat er grundsätzlich die Wahl, diesen Pfad relativ zu seinem aktuellen Verzeichnis anzugeben (relativer Pfad) oder den Pfad von der Wurzel (`/`) aus anzugeben (absoluter Pfad).

In allen folgenden Beispielen wurde der Shell-Prompt so definiert, daß mit ihm immer der Rechnername und das aktuelle Verzeichnis angezeigt werden. Ganz links steht der Benutzername, dann das `@`-Zeichen, dem der Rechnername folgt. Rechts davon steht das aktuelle Verzeichnis des Benutzers.¹ Ist `/home/fred/tmp` das aktuelle Verzeichnis des Benutzers `fred`, sieht der Shell-Prompt, der den Benutzer zur Eingabe von Kommandos auffordert, daher wie folgt aus:

```
fred@tux:/home/fred/tmp >
```

In der dem Shell-Prompt folgenden Zeile erscheinen die Ausgaben, die das vom Benutzer eingegebene Kommando erzeugt – in dem Beispiel unten eine Liste von Datei und Verzeichnisnamen.

Das Zeichen `#` dient dazu, Kommentare einfügen zu können. Alles, was in einer Zeile rechts davon steht, wird von der Shell als Kommentar betrachtet und somit einfach überlesen. Von dieser Möglichkeit wird normalerweise nur dann Gebrauch gemacht, wenn Kommandos in eine Datei geschrieben werden, die später ausgeführt werden sollen (sogenannte Shell-Skripte, siehe auch Abschnitt 5 auf Seite 239). In den folgenden Beispielen werden mit Hilfe dieses Zeichens kurze Erläuterungen zu den Anweisungen des jeweiligen Beispiels gegeben, die beim Ausprobieren natürlich nicht mit eingegeben werden müssen.

Hat sich unser Benutzer `fred` soeben eingeloggt, so könnte er z.B. daran interessiert sein, zu erfahren, welche Dateien in seinem aktuellen Verzeichnis (in diesem Fall seinem Home-Verzeichnis), welche Dateien in `tmp` enthalten sind, welche Dateien in `/home` stehen und welche Dateien im Wurzelverzeichnis `/` liegen. Zu diesem Zweck könnte er die unten stehenden Kommandos ausführen. Dabei wird angenommen, daß im Home-Verzeichnis von `fred` bereits einige Dateien und zumindest das Verzeichnis `tmp` existieren! Zunächst wird der Inhalt des aktuellen Verzeichnisses aufgelistet:

```
fred@tux:/home/fred > ls # auch ls . oder ls /home/fred  
tmp Mail
```

Das Besondere am aktuellen Verzeichnis ist, daß es durch die Datei `.` repräsentiert wird. Das führt hier sowohl die Anweisung `ls .` als auch einfach `ls` zum Ziel. Als nächstes sollen die weiteren genannten Verzeichnisse aufgelistet werden:

¹ Wie man den hier beschriebenen Prompt definiert, wird in Kapitel 2.7.6 auf Seite 73 erläutert.

```
fred@tux:/home/fred > ls tmp # auch ls ./tmp oder
test.txt text.bak # ls /home/fred/tmp
fred@tux:/home/fred > ls .. # auch ls /home ls ../..
bernd freak fred suse zuse
fred@tux:/home/fred > ls ../.. # auch ls /
etc bin usr var home
```

Natürlich sehen die Ausgaben, die das Kommando `ls` erzeugt, auf jedem System anders aus, da sowohl die Verzeichnisstruktur als auch der Inhalt auf jedem System unterschiedlich sein werden. Die oben dargestellten Ausgaben dienen nur als Beispiel.

Man kann erkennen, daß man immer verschiedene Möglichkeiten hat, um auf ein bestimmtes Verzeichnis zuzugreifen. Zum einen kann der absolute Pfad für das Zielverzeichnis angegeben werden, zum anderen besteht die Möglichkeit, ausgehend von dem aktuellen Verzeichnis einen relativen Pfad anzugeben. Der Benutzer hat die freie Wahl, welchen Weg er wählt. Da man als Benutzer generell „schreibträge“ ist, wird man natürlich die Variante wählen, die am wenigsten Arbeit macht. Darüber hinaus können besondere Umstände die eine oder andere Wahl als sinnvoller erscheinen lassen.

In seinem Home-Verzeichnis (und allen darin enthaltenen Unterverzeichnissen) darf der Benutzer Dateien anlegen, löschen und beliebig verändern. Er darf auch neue Verzeichnisse einrichten und alte Verzeichnisse löschen. Diese Rechte hat der Benutzer in anderen Verzeichnissen in der Regel nicht. Jeder Benutzer kann selbst für seine eigenen Dateien bestimmen, ob ein anderer Benutzer diese lesen, schreiben oder im Fall von Programmdateien, ausführen darf. Diese Thematik wird in Kapitel 2.13.4 auf Seite 152 genau besprochen.

2.6 Logout

Wenn man seine Arbeit am System abgeschlossen hat, kann man nicht wie etwa bei einem MS-DOS-System den Rechner einfach ausschalten, sondern man muß sich ausloggen. Man sollte also zunächst alle Programme, die man während des Arbeitens gestartet hat, beenden. Ist das geschehen, gilt es lediglich noch die Shell zu beenden, die beim Login gestartet wurde. Dazu gibt man einfach das Kommando `exit` ein, das die Shell beendet und den Benutzer aus dem System ausloggt. Anstelle von `exit` kann man die Shell auch durch das Dateiende-Zeichen `Ctrl-d` (gleichzeitiges Drücken der Taste `Control` oder `Strg` und `d`) beenden, sofern die Shell nicht konfiguriert wurde, ein `Ctrl-d` Zeichen zu ignorieren, um ein versehentliches Ausloggen zu vermeiden. Eine kurze Weile nach dem Ausloggen erscheint wieder der Login-Bildschirm mit der Aufforderung, die Benutzerkennung und das Paßwort einzugeben.

Wird eine graphische Oberfläche verwendet, wie z.B. KDE, so erfolgt die Abmeldung vom System durch einen entsprechenden Menüeintrag, bei KDE z.B. den Menüpunkt „Abmelden“ im K-Menü.

Man sollte nicht vergessen, sich auszuloggen, da sonst jeder, der an das Terminal herankommt, die eigenen Dateien lesen oder, wenn er böse Absichten hat, auch löschen kann.

Wenn man ausgelogged ist, kann dies nicht vorkommen, da der Bösewicht in diesem Fall den Kennungsnamen und im besonderen auch das geheime Paßwort des Benutzers kennen müßte (das natürlich niemand außer dem Benutzer selbst kennen sollte).

2.7 Die Bourne-Shell

Die Shell ist ein wesentlicher Bestandteil von UNIX. Ohne die Shell wäre es nicht möglich, UNIX-Kommandos auszuführen. Somit wäre das gesamte System nutzlos. Die Shell ist die Schnittstelle zwischen dem Innenleben von UNIX und dem Benutzer. Dabei unterscheidet sie sich nicht grundsätzlich von anderen UNIX-Tools. Sie ist ein ganz normales Programm, wie alle anderen UNIX-Tools. Der wichtigste Unterschied besteht darin, daß sie automatisch bei einem Login gestartet wird. Abbildung 2.3 zeigt die Stellung der Shell zwischen UNIX und dem Benutzer.

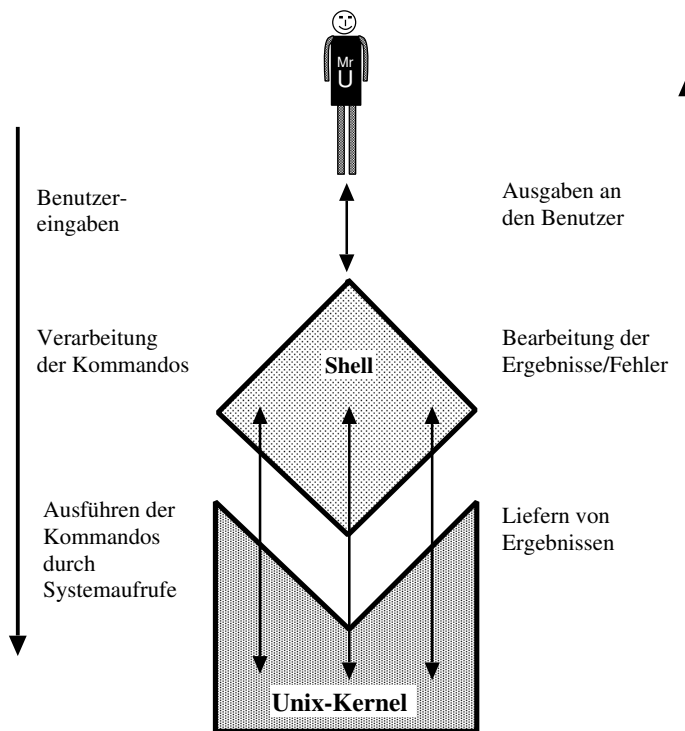


Abbildung 2.3: Die Shell als „Vermittler“

Alle Eingaben des Benutzers werden von der durch das Login gestarteten Shell interpretiert. Wie schon in Abschnitt 1.2 auf Seite 20 dargestellt, fungiert sie als Schale um den UNIX-Systemkern und bildet die Basis für die Mensch-Maschine-Kommunikation. Wie

in Abbildung 2.3 auf der vorigen Seite zu erkennen ist, dient die Shell dem Benutzer als Mittel, um Kommandos auszuführen. Die Shell „übersetzt“ diese Kommandos in Aufrufe von Systemfunktionen an den UNIX-Kernel. Der Kernel führt die entsprechenden Anweisungen aus und liefert als Ergebnis Statusmeldungen zurück, die von der Shell ausgewertet werden.

Neben der Möglichkeit, als Kommandointerpreter zu dienen, kann man in der Shell-Sprache auch Programme schreiben und in einer Shell ablaufen lassen. Diese interessante Möglichkeit wird in Kapitel 5 auf Seite 239 genauer beschrieben.

Die Shell kann Eingaben (entsprechend der UNIX-Philosophie) vom Terminal oder auch von einer Datei lesen und die Ausgaben entsprechend in eine Datei oder auf ein Terminal (Default) schreiben.

Es existieren verschiedene Shell-Varianten. Diese Varianten, die verschiedene Namen tragen, unterscheiden sich im wesentlichen hinsichtlich des Komforts für den Benutzer. Der Systemadministrator kann auf Wunsch eines Benutzers eine bestimmte Shell einrichten, die beim Einloggen des Benutzers automatisch gestartet werden soll.

Oftmals wird bei einem Login die sogenannte Bourne-Shell gestartet. Das Programm selbst trägt den Namen `sh`. Diese Variante ist am meisten verbreitet, hat allerdings den Nachteil, daß sie nicht sehr benutzerfreundlich ist. So bietet die Bourne-Shell beispielsweise keine Unterstützung, um auf einfache Weise Kommandos zu wiederholen oder die Kommandozeile zu editieren, bevor ein Kommando gestartet wird. Bei der Bourne-Shell muß die Kommandozeile immer neu eingegeben werden.

Andere Shells wie die Korn-Shell (`ksh`) oder die C-Shell (`csh`), die TC-Shell (`tcsh`) oder die Bourne-Again-Shell (`bash`) bieten wesentlich mehr Unterstützung für den Benutzer und werden daher in der Regel der Bourne-Shell vorgezogen.

Der Benutzer kann selbst herausfinden, mit welcher Shell er arbeitet. Dazu muß er sich einloggen und anschließend folgendes Kommando eingeben:

```
fred@tux:/home/fred > echo $SHELL
/bin/csh
```

Die Ausgabe `/bin/csh` bedeutet, daß die Shell des Benutzers eine C-Shell ist. Neben den oben genannten existieren eine Reihe anderer Shells, so daß auch hier nicht aufgeführte Shells als Ausgabe des oben beschriebenen Tests erscheinen können.

In diesem Kapitel wird zunächst die *Bourne-Shell* genutzt. Sie bildet die Ausgangsbasis für einige andere Shells. Daher können die in diesem Kapitel beschriebenen Techniken in gleicher oder nur leicht veränderter Art und Weise ebenfalls bei den in den folgenden Kapiteln dargestellten Shells angewendet werden. Wer zur Zeit nicht mit einer Bourne-Shell arbeitet, sollte, um die folgenden Beispiele unverändert übernehmen zu können, für dieses Kapitel die Bourne-Shell starten. Dazu muß man einfach nur in seiner aktuellen Shell die Bourne-Shell durch Eingabe des Kommandos

```
fred@tux:/home/fred > sh
```

starten. Alle Eingaben werden jetzt nicht mehr von der beim Einloggen gestarteten Shell interpretiert, sondern von der neu gestarteten Bourne-Shell. Um diese Shell wieder zu verlassen und zur Login-Shell zu gelangen, gibt man das Kommando `exit` ein.

Manchmal kann es vorkommen, daß anstelle des gewohnten Prompts ein anderes Zeichen, z.B. `>`, erscheint. Dies bedeutet, daß die Shell weitere Eingaben erwartet und die Kommandoeingabe noch nicht abgeschlossen ist. Passieren kann dies, wenn man

- mit Kommandos für Shell-Skripte (siehe Kapitel 5 auf Seite 239) arbeitet,
- eine Zeichenkette der Kommandozeile mit einem Apostroph (`'` oder `"`) beginnt und der zugehörige Apostroph am Ende noch nicht eingegeben wurde,
- eine Kommandozeile mit einem `\` abschließt, wodurch nach Eingabe von `Return` weitere Eingaben in der nächsten Eingabezeile erwartet werden.

In einem solchen Fall kann man einfach einen schließenden Apostroph eingeben oder aber das Kommando ganz abbrechen. Dies geschieht durch Drücken einer speziellen Taste oder Tastenkombination.

Je nach System ist der Abbruch eines Kommandos oder der Abbruch der Eingabe eines Kommandos durch gleichzeitiges Drücken der Tasten `Ctrl` und `c` (kurz `Ctrl-c`) oder durch eine Taste mit der Bezeichnung `Stop` oder `Unterbrechung` möglich. Die Taste `Ctrl` hat bei deutschen Tastaturen übrigens einen anderen Namen, nämlich `Strg`. War der Abbruch des Kommandos erfolgreich, erscheint ein neuer Prompt.

2.7.1 Kommandos

In den letzten Abschnitten wurden bereits verschiedene Kommandos vorgestellt. UNIX verfügt über eine sehr große Zahl solcher Kommandos. Am Anfang des Buchs war die Rede vom „Werkzeugkasten UNIX“ mit seinen Tools. Genau diese Tools sind hier gemeint. Jedes Tool (Kommando) erfüllt nur eine in sich abgeschlossene Aufgabe, die vom Auflisten eines Verzeichnisses bis hin zum Sortieren und automatisierten Editieren einer Datei reichen kann. Im folgenden Abschnitt soll der Begriff Kommando näher beschrieben werden. Darüber hinaus werden die wichtigsten, sozusagen „lebenswichtigen“ Kommandos vorgestellt, die man beherrschen sollte.

Allgemeiner Aufbau und Syntax

Bevor einzelne Kommandos besprochen werden, soll hier die allgemeine Syntax beschrieben werden. Zwar erfüllen die Kommandos unterschiedlichste Aufgaben, jedoch sind sie bezüglich des Aufrufs alle einheitlich.

Der grundsätzliche Aufbau eines Kommandos ist:

```
kommando optionen argumente
```

Hierbei steht `kommando` für den Namen des Kommandos. Es muß auf Klein-/Großschreibung geachtet werden. Normalerweise werden alle Kommandos klein geschrieben. Hinter dem Kommandonamen muß mindestens ein Leerzeichen stehen, falls Optionen oder Argumente folgen. Mehrere Leerzeichen sind bedeutungslos, aber zulässig.

Das Wort `optionen` repräsentiert die möglichen Option(en) für ein Kommando. Optionen beeinflussen das Verhalten eines Kommandos. Optionen beginnen in UNIX mit dem Optionsstrich `-`. Direkt dahinter steht die Option selbst. Oft haben auch die Optionen noch Argumente. Diese Argumente stehen entweder direkt hinter der Option oder aber hinter der letzten Option. Will man bei einem Kommando mehrere Optionen verwenden, so werden diese einfach hintereinander geschrieben.

Das Wort `argumente` schließlich steht für weitere Argumente, die an das Kommando übergeben werden können. Argumente werden durch Leerzeichen voneinander getrennt hinter den Optionen und deren Argumenten geschrieben.

Oftmals benötigt man weder Optionen noch Argumente. In diesem Fall braucht man nur das Kommando selbst einzugeben, bevor die Eingabezeile mit `Return` abgeschlossen wird, wie im folgenden Beispiel von `ls` zu sehen ist, das zum Auflisten von Dateien und Verzeichnissen dient:

Kommando:	Optionen:	Argumente:
<code>ls</code>	<code>-la</code>	<code>/etc /home</code>

In diesem Beispiel wird ein Kommando `ls` mit den Optionen `l` und `a` und den Argumenten `/etc` und `/home` aufgerufen. Bei der Ausführung dieses Kommandos würden die Verzeichnisse `/etc` und `/home` in einem ausführlichen Listing angezeigt werden (Option `-l`). Dabei würden auch Dateien, die mit einem `.` beginnen, dargestellt, die normalerweise nicht aufgelistet werden (Option `-a`).

Um genauer beschreiben zu können, wie Kommandos aufgebaut sind, muß man sich einer genaueren Schreibweise bedienen, um ausdrücken zu können, ob an einer bestimmten Stelle ein Argument oder eine Option stehen darf oder vielleicht sogar stehen muß. Bei dem Kommando `ls` beispielsweise kann man Optionen und Dateinamen angeben, muß es aber nicht. Ohne Optionen und Argumente würden einfach alle Dateien und Verzeichnisse des aktuellen Verzeichnisses aufgelistet.

Um zu verdeutlichen, daß man bei diesem Kommando Optionen und Argumente angeben darf, es aber nicht muß, schreibt man alle optionalen Angaben in `[]`. Den Aufruf des Kommandos `ls` würde man dann wie folgt beschreiben:

```
ls [optionen] [verzeichnis|datei] ...
```

Diese Schreibweise bedeutet, daß am Zeilenanfang die Zeichenkette `ls` gefolgt von einem Leerzeichen stehen muß. Hinter dem Kommandonamen dürfen Optionen wie z.B. `-l` stehen und schließlich Datei- oder Verzeichnisnamen. Das *oder* wird in obiger Schreibweise durch einen `|` dargestellt. Die sich direkt anschließenden Punkte bedeuten, daß mehrere Datei-Verzeichnisnamen angegeben werden dürfen, die durch Leerzeichen getrennt werden müssen. Grundsätzlich könnte man auch bei den Optionen die Zeichenkette `...` verwenden, um anzuzeigen, ob eine oder mehrere Optionen hinter einem Kommando stehen sollen. Da aber in UNIX bei allen Kommandos, die Optionen kennen, eine oder mehre-

re Optionen angegeben werden dürfen, würde es die Beschreibung eines Kommandos nur unnötig komplizieren, müßte man an dieser Stelle immer . . . schreiben. Daher verwendet man einfach den Begriff `optionen`, der intuitiv ausdrückt, daß an dieser Stelle eine oder auch mehrere Optionen stehen dürfen.

In unklaren Fällen kann das Leerzeichen besonders kenntlich gemacht werden. Zu diesem Zweck wird das Zeichen `_` eingesetzt. Dies geschieht aber wie gesagt nur, wenn nicht klar ist, wo ein Leerzeichen stehen muß.

Zum besseren Verständnis noch ein Kommando, das weiter unten beschrieben wird. Im Moment geht es nicht darum, was das Kommando bewirkt, sondern nur darum, wie es aufgebaut ist:

```
cp [optionen] quelldatei zieldatei |
cp [optionen] quelldatei... verzeichnis
```

Diese Beschreibung besagt, daß das Kommando `cp` entweder zwei Argumente haben muß, die beide eine Datei darstellen, oder daß die ersten Argumente (beliebig viele) Dateien sein müssen und das letzte Argument ein Verzeichnisname. In beiden Fällen dürfen eine oder mehrere Optionen verwendet werden, die natürlich noch genauer beschrieben werden müßten.

Alles, was nicht in `[]` geschrieben ist, muß beim Aufruf eines Kommandos angegeben werden. Dies ist zum einen natürlich immer der Kommandoname, der wortwörtlich übernommen werden muß. Darüber hinaus können aber auch andere Argumente notwendig sein, die nicht weggelassen werden dürfen. Oben muß beispielsweise immer der Parameter `quelldatei` angegeben werden. Dabei steht `quelldatei` für einen beliebigen Dateinamen. Das gleiche gilt für den Parameter `zieldatei` oder `verzeichnis`. Auch diese Parameter sollten nicht wörtlich übernommen werden, sondern stehen stellvertretend für bestimmte Datei- oder Verzeichnisnamen.

Neben der Vorgabe, welche Parameter angegeben werden müssen, bestimmt die Schreibweise auch, wie man das Kommando *nicht* verwenden kann. Entsprechend dem obigen Beispiel des Kommandos `cp` ist es nicht erlaubt, zusätzlich zum Kommandonamen drei Argumente zu verwenden, die alle Dateien sind! Auch ist es nicht erlaubt, nur ein einziges Argument oder kein Argument zu übergeben!

Grundlegende Kommandos

UNIX verfügt über mehrere hundert Kommandos, die dem Benutzer zur Verfügung stehen. Eine so große Zahl an Befehlen, wobei jedes der Kommandos wiederum verschiedene Optionen hat, kann sich natürlich kein Mensch merken. Glücklicherweise muß man es auch nicht. Zum einen gibt es nur eine relativ kleine Zahl an Kommandos, die man für den täglichen Gebrauch kennen sollte, zum anderen verfügt UNIX über ein leistungsfähiges Online-Dokumentationssystem, in dem man im Zweifelsfall nachschlagen oder suchen kann.

In diesem Abschnitt sollen zunächst die Kommandos dargestellt werden, die man im täglichen Umgang mit dem System immer wieder benötigt:

`echo` `variable|zeichenkette` Ausgeben von Variablen und Text;
meistens wird `echo` für die Ausgabe von Variablenwerten verwendet. Hierzu muß der auszugebenden Variablen das `$`-Zeichen vorangestellt werden (z.B. `echo $HOME`). Wenn Text ausgegeben werden soll, kann er einfach hinter dem Kommando stehen, wie z.B. `echo "Hallo ich bin es"`.

`cd` [`verzeichnis`] Wechsel des aktuellen Verzeichnisses;
mit diesem Kommando kann man in ein anderes Verzeichnis wechseln, das neue Verzeichnis also zum aktuellen machen. Als `verzeichnis` können auch `..` und `.` mit ihrer normalen Bedeutung verwendet werden. Wird `cd` ohne Parameter aufgerufen, so wird der Wert der Shell-Variablen `HOME` als Zielverzeichnis verwendet, das heißt, man wechselt in sein Home-Verzeichnis. Das Kommando `cd -` führt dazu, daß in das jeweils letzte Verzeichnis gewechselt wird, also das Verzeichnis, aus dem heraus zuletzt ein `cd`-Kommando in das aktuelle Verzeichnis ausgeführt wurde.

`cat` [`datei`]... Ausgabe von Dateien;
`cat` gibt den Inhalt der Dateien, die als Argumente übergeben wurden, auf dem Terminal aus. Diese Ausgabe kann, wie noch beschrieben wird, mit Hilfe des `>` Zeichens in eine Datei umgelenkt werden. So kann man beispielsweise den Inhalt zweier Dateien mit `cat` aneinanderhängen (konkatenerieren). Da wir bis jetzt noch keinen Befehl kennen, um Dateien zu erzeugen, nehmen wir zum Probieren zwei Systemdateien, die immer existieren. Die eine heißt `/etc/passwd`, die andere `/etc/group`. Versuchen Sie zunächst einfach, die Dateien einzeln auszugeben mit `cat /etc/passwd` und anschließend `cat /etc/group`. Die jeweilige Datei wird einfach auf dem Bildschirm ausgegeben. Versuchen sie jetzt folgenden Befehl: `cat /etc/passwd /etc/group`. Sie sehen, daß beide Dateien direkt hintereinander ausgegeben werden, man die Dateien also aneinandergehängt hat. Will man die Ausgabe nicht auf dem Bildschirm sehen, sondern in eine Datei (z.B. die Datei `ausgabe`) lenken, so geschieht dies durch Anwendung des `>`-Zeichens. Die Ausgabe-datei wird automatisch erstellt, sofern sie nicht bereits existiert:

```
fred@tux:/home/fred > cat /etc/passwd /etc/group > ausgabe
```

Wird keine Datei angegeben, der Befehl also ohne Argumente aufgerufen, liest `cat` Eingaben von der Tastatur und schreibt die Ausgaben auf den Bildschirm. Daher kann man `cat` auch als Minieditor verwenden. Möchte man Text in eine Datei schreiben, kann dies durch `cat > datei` geschehen. Dabei wird `datei` zunächst gelöscht. Anschließend werden die Eingaben des Benutzers in die Datei geschrieben. Beenden kann man die Eingabe mit `Ctrl-d`, dem Dateiendezeichen.

`ls` [`optionen`] [`datei|verzeichnis`]... Auflisten von Dateien und Verzeichnissen;
die angegebenen Dateien oder Verzeichnisse werden aufgelistet. Wird kein Argument angegeben, listet `ls` das aktuelle Verzeichnis auf.
Normalerweise werden keine Dateien und Verzeichnisse aufgelistet, die mit einem Punkt beginnen, z.B. `.geheim`. Diese Dateien sind gewissermaßen versteckte Datei-

en, die oft Konfigurationsinformationen für bestimmte Programme enthalten. Solche Dateien/Verzeichnisse können nur mit der Option `-a` sichtbar gemacht werden.

`more [datei]...`

`less [datei]...` seitenweises Anzeigen von Dateien;

die Kommandos `more` und `less` ermöglichen ein seitenweises Anzeigen von Dateien. Darüber hinaus kann mit diesen Programmen in einer Datei vor- und zurückgeblättert werden. Beispielsweise kann man sich mit `more /etc/passwd` die Datei `passwd` seitenweise ansehen. Um vorwärtszublättern, drückt man die Tasten `Ctrl-f`, zum Rückwärtsblättern die Tasten `Ctrl-b`. Eine Hilfe erhält man durch Drücken von `h`. Verlassen kann man das Programm mit `q`.

Eine weiteres nützliches Feature dieser Programme ist die Möglichkeit, in der angezeigten Datei nach Zeichenketten zu suchen. Dazu drückt man die `/` Taste, woraufhin in der untersten Bildschirmzeile der Suchbegriff eingegeben werden kann. Nach dem Drücken von `Return` wird die Textstelle angezeigt, wo der Begriff gefunden wurde. Eine erneute Suche des gleichen Suchbegriffs ist durch einfaches Drücken der Taste `n` (Next) möglich.

Auf manchen Systemen existiert nur eines der beiden oder anstatt der beiden ein drittes Programm mit dem Namen `pg`. Auch dieses Programm erfüllt den gleichen Zweck. Die Programme unterscheiden sich alle leicht bezüglich der Bedienung und des Komforts, den sie dem Benutzer anbieten. `less` bietet die meisten Möglichkeiten.

`cp [optionen] quelle ziel`

`cp [optionen] datei... verzeichnis` Kopieren von Dateien;

in der ersten obenstehenden Form wird eine einzige Datei in eine neue Datei kopiert. Beispielsweise kopiert der Befehl `cp /etc/group g` die in dem Verzeichnis `/etc` stehende Datei `group` in die Datei `g` im aktuellen Verzeichnis. In der zweiten Form können mehrere Dateien durch einen Aufruf des Kommandos in ein Verzeichnis kopiert werden. So kopiert der Befehl `cp /etc/passwd /etc/group tmp` die Dateien `/etc/passwd` und `/etc/group` unter ihrem alten Namen in das Verzeichnis `tmp`, wenn dieses existiert. Wenn nicht, oder wenn `tmp` eine Datei ist, meldet `cp` einen Fehler.

`pwd` Arbeitsverzeichnis bestimmen;

Durch die Eingabe dieses Befehls wird der Pfad zu dem aktuellen Verzeichnis ausgegeben.

`touch [optionen] datei...` Leere Dateien erzeugen oder das Datum der Dateien aktualisieren. Das Datum für eine Datei kann mit `ls -l` ausgegeben werden.

`rm [-ir] name...` Löschen von Dateien;

`rm` dient zum Löschen von Dateien. Dazu wird einfach der Name der zu löschenden Datei(en) angegeben. Sollten auch Verzeichnisse gelöscht werden, kann die Option `-r` verwendet werden. Dadurch wird das angegebene Verzeichnis mit *allen* Dateien und *allen* Unterverzeichnissen gelöscht! Daher sollte man mit dieser Option vorsichtig umgehen. `-i` bietet einen gewissen Schutz. Gibt man diese Option an, so wird

vor jedem Löschen einer Datei gefragt, ob diese Datei wirklich gelöscht werden soll. Will man mehrere Dateien auf einmal löschen, ohne alle Namen eingeben zu müssen, kann man Wildcards benutzen. Eine Wildcard ist ein besonderes Zeichen, das auf bestimmte Buchstabenkombinationen „paßt“. Die wichtigste Wildcard ist der `*`. Er steht für beliebige Zeichen, so daß das Kommando `rm *` alle Dateien im aktuellen Verzeichnis löscht, ein `rm -r *` sogar alle Dateien und Verzeichnisse, die im aktuellen Verzeichnis und allen Unterverzeichnissen liegen! Dagegen löscht `rm *.bak` nur die Dateien, die die Endung `.bak` haben. Im Gegensatz zu MS-DOS steht in UNIX der `*` auch für das Zeichen `.` innerhalb eines Dateinamens, so daß der Befehl `rm xyz*` alle Dateien löscht, die mit `xyz` beginnen, also auch `xyz.txt`. In MS-DOS hätte man für den gleichen Zweck `del xyz*. *` schreiben müssen.

`mkdir verzeichnis...` Dateiverzeichnisse erstellen;
mit `mkdir` können ein oder mehrere Verzeichnisse erstellt werden.

`rmdir verzeichnis...` Dateiverzeichnisse löschen;
mit `rmdir` können nur leere Verzeichnisse, die keine Dateien oder andere Verzeichnisse enthalten, gelöscht werden.

`wc [optionen] [datei]` Zeilen, Wörter, Zeichen einer Datei zählen;
`wc` dient zum Zählen der Zeilen, Wörter und Zeichen einer Datei. Die Ausgabe des Kommandos besteht aus drei Spalten, von der die erste die Zahl der Zeilen, die zweite die Zahl der Wörter und die dritte Spalte die Zahl der Zeichen in der Datei enthält. Wird kein Argument angegeben, zählt `wc` die Zeilen, Wörter, Zeichen, die auf der Tastatur eingegeben werden, bis zu einem `Ctrl-d`.

`grep [optionen] muster [datei]` nach Mustern in Dateien suchen;
oftmals möchte man in einer Datei nur solche Zeilen ausgeben, die einen bestimmten Text enthalten. Diesem Zweck dient `grep`. Mit `grep` wird eine Datei nach einer Zeichenkette `muster` durchsucht. Alle Zeilen der Datei, die diese Zeichenkette enthalten, werden ausgegeben. Alle anderen Zeilen werden nicht ausgegeben. Wenn Sie bei `grep name /etc/passwd` anstelle von `name` den Namen Ihrer Kennung eintragen, erhalten Sie genau eine Zeile als Ausgabe. Das liegt darin begründet, daß in der Datei `/etc/passwd` für jeden Benutzer, den das System kennt, genau eine Zeile vom Systemadministrator eingetragen wurde.²

`date [optionen] [datum]` Ausgeben und Setzen des Datums.

Jedes Kommando liefert einen Ergebniswert in Abhängigkeit davon, ob es seine Aufgabe erfüllen konnte oder nicht. In UNIX gilt ein Ergebniswert von Null als Erfolg und ein von Null verschiedener Wert als Fehlerzustand. Diese Rückmeldung ist sehr nützlich für die Verkettung von Kommandos.

²Unter Umständen erhalten Sie keine Ausgabe. Falls der Rechner, an dem Sie arbeiten, Teil eines Netzwerkes ist, kann die Datei `/etc/passwd` zentral auf einem einzigen Rechner gespeichert sein. Nur auf diesem Rechner könnte `grep` erfolgreich sein, da die `passwd`-Dateien aller anderen Rechner in diesem Fall keine Einträge für Benutzer enthalten.

Der Ergebniswert des letzten Kommandos kann mit `echo $?` abgefragt werden. Führen Sie beispielsweise das Kommando `cd` aus. Anschließend befinden Sie sich in Ihrem Home-Verzeichnis. Da das `cd` erfolgreich war, sollte der Rückgabewert des Kommandos Null sein. Testen Sie es, indem Sie direkt anschließend `echo $?` eingeben. Für den Gegenteil, ob auch wirklich ein von Null verschiedener Wert einen Fehler anzeigt, geben Sie beispielsweise `cd /xyz` ein. Da dieses Verzeichnis nicht existiert, meldet `cd` zunächst einen Fehler auf dem Bildschirm. Darüber hinaus wird der Fehlerzustand aber auch erkenntlich, wenn Sie erneut `echo $?` eingeben, also den Rückgabewert des zuletzt ausgeführten Kommandos ermitteln. Das Ergebnis sollte von Null verschieden sein.

Jedem Kommando, das als Argument einen Dateinamen benötigt, kann auch ein Pfad zu einer Datei geliefert werden. Auch hier können die zwei besonderen Zeichenketten `..` und `.` verwendet werden. Da wir inzwischen die notwendigen Kommandos kennen, kann die Wirkung der beiden Dateien mit Hilfe der Kommandos `cd` und `pwd` direkt beobachtet werden:

```
fred@tux:/home/fred > cd .
fred@tux:/home/fred > pwd
/home/fred
fred@tux:/home/fred > cd ..
fred@tux:/home > pwd
/home
fred@tux:/home > cd
fred@tux:/home/fred > ls .
tmp torte
fred@tux:/home/fred > ls ../..
freak fred zuse
```

2.7.2 Eingabe, Ausgabe

Die meisten Kommandos erfordern Eingaben, und fast alle erzeugen Ausgaben. Daher muß festgelegt sein, woher die Eingaben entgegengenommen und wohin Ausgaben geschickt werden sollen. Zu diesem Zweck existieren drei vordefinierte Datenwege, die ständig verfügbar sind. Für diese Datenwege existieren feststehende Bezeichnungen: *Standardin* (`stdin`) für Eingabe, normalerweise die Tastatur des Terminals, an dem man sitzt, *Standardout* (`stdout`) für die Ausgabe von Daten (normalerweise der Bildschirm) und schließlich *Standarderror* (`stderr`) für Fehlerausgaben, normalerweise ebenfalls der Bildschirm. Programme, die während des Ablaufs Daten vom Benutzer einlesen, verwenden `stdin` zu diesem Zweck. Um Ausgaben zu schreiben, können die Programme im Normalfall `stdout` und für Fehlermeldungen `stderr` verwenden. Diese Unterscheidung ist sinnvoll, da man auf diese Weise normale Ausgaben in eine Datei lenken kann, um sie sich später anzusehen und Fehlermeldungen in eine zweite Datei oder auf den Bildschirm schreiben kann. Diese Möglichkeit der Trennung erhöht die Übersichtlichkeit, wenn ein Kommando viele Ausgaben erzeugt, unter denen auch einmal eine Fehlermeldung ist, auf die es ankommt.

Die drei Datenwege sind von Null an durchnummeriert: `stdin=0`, `stdout=1` und `stderr=2`. Diese Numerierung wird im Zusammenhang mit der Shell noch einmal wichtig werden.

2.7.3 Spezielle Zeichen, Quoting

Wie schon früher bemerkt, haben manche Zeichen für die Shell eine besondere Bedeutung. Als Beispiel war schon der `*` genannt worden, der für eine beliebige Zeichenkette (außer einer solchen, die mit einem `.` beginnt) stehen kann. Dadurch war eine Schreibvereinfachung möglich, indem man z.B. bei `ls` nicht alle Dateinamen einzeln angeben mußte, sondern an deren Stelle einfach einen `*` schreiben konnte. Der Nachteil dieser Vereinfachung ist beispielsweise, daß man Probleme bekommt, wenn ein Dateiname einen `*` enthalten soll. In einem solchen Fall muß man dem `*` seine besondere Bedeutung nehmen, so daß er zu einem ganz normalen Zeichen wird. Diesen Vorgang nennt man *Quoten*. Ein einfaches Beispiel soll diesen Unterschied verdeutlichen.

```
fred@tux:/home/fred > ls *
tmp text

fred@tux:/home/fred > ls \*
/bin/ls: *: No such file or directory
```

Im ersten Fall wurde der Stern als Sonderzeichen interpretiert, das für beliebige Zeichen in einem Dateinamen steht. Daher wurden alle im aktuellen Verzeichnis vorhandenen Dateien und Unterverzeichnisse aufgelistet. In diesem Beispiel das Verzeichnis `tmp` und die Datei `text`.

Im zweiten Fall wurde der `*` seiner besonderen Bedeutung enthoben. Dies geschieht z.B. durch Voranstellen eines `\`-Zeichens. Das Kommando `ls` hat jetzt nach einer Datei mit dem Namen `*` gesucht. Diese existiert aber nicht, daher die Fehlermeldung `no such file or directory`.

Bei der Auswertung der eingegebenen Kommandos wird die Eingabezeile immer zunächst auf das Auftreten von solchen Sonderzeichen hin untersucht. Die Sonderzeichen werden in Abhängigkeit der existierenden Dateinamen durch eine bestimmte Zeichenkette ersetzt. Nachdem die Shell alle Sonderzeichen gemäß ihrer Bedeutung ersetzt hat, wird das Kommando mit den von der Shell veränderten Argumenten aufgerufen. Bei einem Kommando wie `ls *` ersetzt die Shell zunächst den `*` durch alle Dateien des aktuellen Verzeichnisses etwa zu `ls adressen tmp torte`. Erst nach dieser Ersetzung wird das Kommando gestartet.

Im einzelnen kennt die Shell folgende Sonderzeichen:

- * wird zu beliebigen Zeichen (auch der leeren Zeichenkette) für Dateinamen, ausgenommen Dateien mit einem `.` am Beginn, expandiert. Die sogenannten versteckten Dateien und Verzeichnisse mit einem Punkt am Anfang werden daher von dieser Wildcard nicht erfaßt.

- ? wird zu *einem* beliebigen Zeichen eines Dateinamens expandiert.
- [s] wird mit einem Zeichen aus der Zeichenkette *s* entsprechend dem Dateinamen ersetzt. So würde beispielsweise anstelle von [abc] *ein* Zeichen der erlaubten Zeichen *a*, *b* und *c* entsprechend einem Dateinamen eingesetzt.
- [!s] wird mit einem Zeichen ersetzt, das nicht in *s* enthalten ist.
- [c1 – c2] wird durch genau ein Zeichen ersetzt, das sich in dem Bereich von *c1–c2* befindet (z.B. *a–z*).
- [!c1 – cn] wird durch ein Zeichen ersetzt, das nicht im Bereich von *c1–cn* liegt.
- \$ Dieses Zeichen beschreibt den Beginn einer Variablen, deren Wert hier eingesetzt werden soll. Der Name der Variablen muß direkt hinter dem *\$*-Zeichen stehen.
- `kommando` Die Ausgaben von *kommando*, das automatisch ausgeführt wird, werden an der Stelle von *`kommando`* eingesetzt. *kommando* darf ein beliebiges Programm sein, das seine Ausgaben auf *stdout* schreibt.
- Space, Tab Dienen als Trennzeichen. Space ist das Leerzeichen und Tab steht für ein Tabulatorzeichen. Mehrere Trennzeichen werden von der Shell zu einem einzigen verkürzt.
- Ctrl-s Üblicherweise das Anhalten der Textausgabe auf einem Terminal.
- Ctrl-q Üblicherweise das Fortsetzen der Textausgabe auf einem Terminal, die zuvor mit Ctrl-s angehalten wurde.
- Ctrl-d Das EOF-Zeichen (End of file, Dateiende) .

Einige Beispiele sollen die Anwendung der Sonderzeichen verdeutlichen. Als Voraussetzung sollen im aktuellen Verzeichnis folgende Dateien existieren:

```
.cshrc tmp titel torte changes adressen.txt italy
```

Unter dieser Voraussetzung liefern die untenstehenden Kommandos die dargestellten Ergebnisse:

```
fred@tux:/home/fred > ls *es
changes

fred@tux:/home/fred > ls *es?*
adressen.txt
```

Im ersten Fall werden alle Dateien aufgelistet, die am Ende des Namens die Zeichen „es“ beinhalten. Nur eine Datei erfüllt diese Bedingung: *changes*. Das zweite Beispiel listet alle Dateien auf, die in der Mitte ein „es“ beinhalten, dem ein weiteres Zeichen und anschließend entweder beliebige oder auch keine Zeichen mehr folgen. Die Datei *changes* paßt nicht mehr zu diesem Muster, da ihr Name hinter dem „es“ *kein* Zeichen mehr enthält. Es paßt nur noch die Datei *adressen.txt*.

```
fred@tux:/home/fred > ls t[im][pt]*
titel tmp

fred@tux:/home/fred > ls t[a-zA-Z]*
titel torte tmp
```

Diese Beispiele verdeutlichen die Verwendung von Zeichenklassen in []. Im ersten Beispiel werden all jene Dateien angezeigt, die mit einem `t` beginnen und anschließend *eines* der Zeichen `i` oder `m` haben und daran anschließend *entweder* ein `p` oder ein `t`. Im Anschluß daran dürfen beliebige weitere Zeichen stehen. Die Zeichen innerhalb [] stehen also immer für *ein* Zeichen aus einer Menge von Zeichen. Nur die Dateien `titel` und `tmp` erfüllen die genannten Bedingungen.

Im zweiten Beispiel wird eine ganze Reihe von Zeichen, die hinter einem `t` stehen sollen, durch eine Zeichenklasse beschrieben. Neben der Möglichkeit, eine Auswahl an Zeichen in [] angeben zu können, darf man auch einen Bereich *von-bis* angeben. In diesem Beispiel darf ein Zeichen aus dem Bereich `a-z` oder `A-Z` hinter einem `t` stehen. Dadurch werden nicht nur die Dateien `t-i-tel` und `t-m-p` ausgegeben, sondern auch die Datei `t-o-rte`.

```
fred@tux:/home/fred > ls $HOME/i*
italy
```

Hier sieht man die Möglichkeit, Shell-Variablen einzusetzen. Wird dem Namen der Variablen ein `$` vorangestellt, so wird an dieser Stelle der Wert der Variablen durch den Namen ersetzt. `$HOME` wird also durch `/home/fred` ersetzt. Nach diesem ersten Schritt hat sich die Kommandozeile also zu `ls /home/fred/i*` verändert. Als nächstes wertet die Shell das `*`-Zeichen aus. In `/home/fred` befindet sich nur eine Datei, die mit einem „i“ beginnt: `italy`. Diese Datei wird aufgelistet.

```
fred@tux:/home/fred > cp `ls t[io]*` /tmp
```

Dieses Beispiel zeigt die interessante Möglichkeit, Ausgaben eines Kommandos (hier `ls t[io]*`) als Argumente für ein anderes Kommando zu verwenden. Zunächst wird der Befehl `ls t[io]*` ausgeführt, da er in `` gefaßt ist. Die Ausgaben des Kommandos ersetzen das Kommando selbst. Das Ergebnis ist eine Liste von Dateinamen, die mit `t[io]` beginnen (`t-i-tel`, `t-o-rte`). Die Kommandozeile lautet jetzt: `cp titel torte /tmp`. Anstatt diese Namen auszugeben, werden sie direkt an das Kommando `cp` übergeben. Anschließend startet `cp` und kopiert diese beiden Dateien in das Verzeichnis `/tmp`.

Den gleichen Effekt hätte man in diesem Beispiel natürlich auch dadurch erreichen können, daß man das Muster `t[io]*` direkt in der Kommandozeile von `cp` angibt. Daher sollte dieses Beispiel eher als Demonstration dieser Möglichkeit angesehen werden.

Wie bereits gesagt, erleichtern diese speziellen Schreibweisen das Aufsuchen von Dateien. Es muß nicht immer der gesamte Name einer Datei angegeben werden, und man kann mit einem Muster viele Dateien auf einmal ansprechen (z.B. `ls *.txt`). Daraus ergeben sich jedoch auch Probleme.

Was beispielsweise, wenn man in einer Datei alle Zeilen finden möchte, in denen sich das Zeichen `*` befindet? Der erste Einfall `grep * datei` führt zu einem unerwarteten Ergebnis. Der Grund liegt in der besonderen Bedeutung des `*` für die Shell, die daraus eine Liste von Dateinamen erzeugt (`*` steht ja für beliebig viele beliebige Zeichen eines Dateinamens). Da `grep` als ersten Parameter ein Suchmuster erwartet und anschließend die Dateien, die durchsucht werden sollen – jetzt aber nur eine Liste von Dateien (gegen die der `*` ersetzt wurde) erhält, verwendet `grep` den ersten hinter dem Kommandonamen stehenden Dateinamen als Suchmuster und kann nicht das tun, was wir erwartet hatten. Hier muß dem Sonderzeichen `*` seine besondere Bedeutung genommen werden. Der `*` muß „gequotet“ werden. Die Shell bietet verschiedene Möglichkeiten, um dieses Ziel zu erreichen:

- "..." Innerhalb dieser Anführungszeichen werden nur noch Shell- und Environment-Variablen ersetzt (Shell- und Environment-Variablen werden in Kapitel 2.7.6 auf Seite 67 besprochen). Sonderzeichen wie der `*` sind hier normale Zeichen. Die einzelnen Worte des Kommandos, die durch Leerzeichen getrennt sind, werden als einzelne Worte an das Kommando übergeben.
- '...'' Innerhalb dieser Zeichen wird von der Shell keine Ersetzung mehr vorgenommen. Das aufgerufene Kommando erhält nur ein Argument, nämlich die Zeichenkette zwischen den beiden Apostrophen.
- \ Nimmt dem unmittelbar folgenden Zeichen seine besondere Bedeutung. Dies gilt, auch wenn dieses Zeichen am Zeilenende steht. Damit wird dem Newline-Zeichen seine spezielle Bedeutung genommen. Dadurch kann sich eine Kommandozeile über mehrere Eingabezeilen erstrecken, d.h. die Eingabe eines Kommandos kann sich über mehrere Eingabezeilen erstrecken.

Auch hier sollen einige Beispiele helfen, das Gesagte besser zu verstehen. Das aktuelle Verzeichnis enthalte die gleichen Dateien wie oben:

```
fred@tux:/home/fred > ls \ $HOME
/bin/ls: $HOME: No such file or directory
```

Hier wurde dem `$`-Zeichen seine besondere Bedeutung geraubt, indem man einen `\` vorangestellt hat. Dadurch erkennt die Shell die Variable `HOME` nicht mehr als Shell-Variable, und `HOME` wird nicht durch den Wert der Variablen `/home/fred` ersetzt. `ls` versucht daraufhin, eine Datei mit dem Namen `$HOME` zu finden. Diese gibt es nicht, und `ls` bricht mit einer Fehlermeldung ab.

```
fred@tux:/home/fred > ls "$HOME"
tmp  titel  torte  changes  adressen.txt  italy
fred@tux:/home/fred > ls ' $HOME '
/bin/ls: $HOME: No such file or directory
fred@tux:/home/fred > ls "$HOME/*"
/bin/ls: /home/fred/*: No such file or directory
```

Das erste dieser drei Beispiele listet alle Dateien des Home-Verzeichnisses auf. Auch innerhalb der doppelten Anführungszeichen ("`"` `"`) werden Shell-Variablen durch ihren Wert