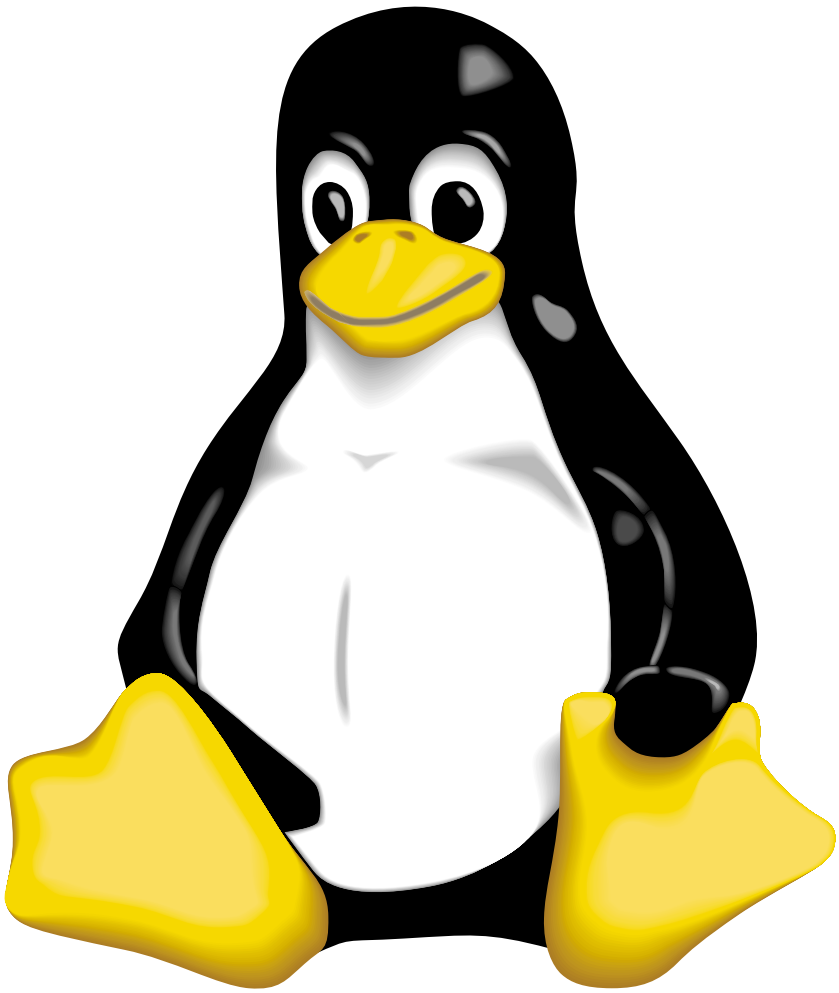
Oliver Böhm

Spaß an Linux & UNIX

Eine Einführung



Wenn du ein Schiff bauen willst,
so trommle nicht die Männer zusammen um Holz zu beschaffen und Werkzeuge vorzubereiten oder die Arbeit einzuteilen und Aufgaben zu vergeben,
sondern lehre die Männer die Sehnsucht nach dem endlosen, weiten Meer.

(Antoine de Saint-Exupery)

Spaß an Linux** & **UNIX
Ausgabe 6.2

Inhaltsübersicht

Das X-Window-System

Vernetzung

Anhang

Abbildungsverzeichnis

Tabellenverzeichnis

Vorwort

Das vorliegende Skript wurde für eine Vorlesung an der Berurfsakademie (BA) Stuttgart entwickelt. Es entstand aus meiner Tätigkeit bei Alcatel-SEL heraus, bei der ich Anfängern in die Geheimnisse von UNIX einweihte. Später kamen dann noch firmenweite Kurse hinzu, bei denen ich aber glücklicherweise das Skript von der Firma gestellt bekam.

Warum dann ein neues Skript? Aus Erfahrungen vor allem aus meinem C-Kurs weiß ich, daß so ein Skript eine sehr dynamische Sache sein kann. Erfahrungen aus vorhergehenden Kursen lassen sich einbringen, man kann neue Trends mit einfließen lassen bzw. die Unterlagen auf den neusten Stand bringen, und man ist flexibler, wenn sich die Schwerpunkte der Teilnehmer ändern. Noch wichtiger aber ist, daß durch die ständige Auseinandersetzung mit dem Skript die Qualität des Unterrichts steigt (hoffe ich zumindest).

Die verwendeten Unterlagen waren Teil meiner früheren UNIX-Einführungen, daneben werden aber auch einige wenige Teile aus den Kursunterlagen der Alcatel-SEL AG verwendet.

Danksagung

Hier nun die obligatorischen Danksagung an all jene Personen, dir mir dieses Skript ermöglicht haben: Danken möchte ich vor allem meiner Frau, daß sie so geduldig mit mir war und meinen Kindern, daß sie mir ab und zu etwas Zeit für andere Dinge (wie z.B. dieses Skript) gelassen haben. Stellvertretend für all die anderen Personen, die mir dieses Skript ermöglicht haben (und die ich aus Platzgründen nicht alle aufführen kann), möchte ich Herrn H. Kiesel vom Softwaretraining (ZPE/SWTR) für seine Unterstützung danken und dafür, daß er mir die Schulungsunterlagen überlassen hat.

September 1994 Oliver Böhm

Vorwort zur 2. und 3. Auflage

Hier wurde vor allem das Kapitel „Shell“ überarbeitet und ergänzt. Auch wird auf das Thema „Shellprogrammierung“ kurz eingegangen, da dieses Thema doch ab und zu in meinen Kursen hochkommt.

Auch im Kapitel 22 „Das Internet“ wurde eingiges noch nachgetragen, jedoch ist hier die Entwicklung schneller als meine Ergänzungen.

Was immer noch fehlt, ist die Beschreibung der graphischen Oberfläche. Dies liegt zum größten Teil daran, daß erst ab Anfang 95 eine einheitliche graphische Oberfläche für die diversen Unix-Ableger zur Verfügung steht, und ich mich erst noch damit anfreunden und einarbeiten muß. Vielleicht in der nächsten Ausgabe dann (als ich diese Zeilen schrieb, hatte ich selbst noch nicht diese Oberfläche zur Verfügung).

Februar 1996 Oliver Böhm

Vorwort zur 4. und 5. Auflage

Während in der 4. Auflage das Thema „Unix als Programmierumgebung“ hinzugekommen ist, habe ich in der 5. Auflage dem Linux-Trend Rechnung getragen und die Aufzucht und Hege eines Linux-Rechners mit in die Unterlagen übernommen.

Leider ist das Kapitel 22 „Das Internet“ immer noch nicht auf dem aktuellen Stand - allerdings braucht man heute den Begriff „Internet“ auch nicht mehr groß erklären

Ergänzungen, die hinzugekommen sind, wurden weitgehend nach der neuen Rechtschreibreform gemacht. Die älteren Teile des Manuskripts sind dadurch gekennzeichnet, dass sie sich noch nach der alten Rechtschreibung orientiert haben.

August 1999 Oliver Böhm

Vorwort zur 6. Auflage

Endlich kann ich wieder unter Unix die Dokumentation weiterpflegen, nachdem ich zwischenzeitlich unter Windows 95 ausgewichen bin (pssst, nicht weitersagen). Seit Januar 2000 gibt es FrameMaker 5.5 als Beta-Version für Linux.

Juni 2000 Oliver Böhm

Zielgruppe

Die Unterlagen wurden für BA-Studenten des ersten Semesters überarbeitet, richten sich aber auch an Berufsanfänger und Umsteiger, die sich in UNIX und Linux einarbeiten wollen (müssen).

Kursziele

Der Kursteilnehmer soll

- einen Überblick über UNIX bekommen
- sich unter UNIX an- und abmelden können
- Nachrichten empfangen und verschicken können
- Online-Hilfen abrufen können
- UNIX Kommandos absetzen können
- UNIX als Entwicklungsumgebung kennenlernen
- Arbeitsumgebung anpassen können

Konventionen

Als Suchhilfe sind Wörter, die im Index zu finden sind, in **halbfetter Schrift** gedruckt.

Beispiele von Dialogen mit dem Rechner werden in `Courier` gedruckt. Eingabe des Benutzers werden dabei durch **`halbfette Courier-Schrift`** hervorgehoben.

Wenn von UNIX die Rede ist, ist damit auch immer Linux mit eingeschlossen.

K&K (Korrekturen & Kritik)

Da dieses Skript laufend (na ja, eigentlich mehr sitzend) überarbeitet wird, schleicht sich doch der eine andere Fehler ein. Falls sie den einen oder anderen Fehler entdecken, behalten sie ihn nicht für sich, sondern melden sie ihn mir bitte. Am besten als Email an

Oliver.Boehm@rwg.de *bzw.*
boehm@ba-stuttgart.de

schicken (oder per Telefon/Fax: 0711-2012-2734/6734). Auch für Anregungen und Kritik bin ich dankbar. Und falls einer noch das eine oder andere Kapitel beitragen (oder ein bestehendes Kapitel überarbeiten) will, kann er das Angebot ebenfalls an die obige Adresse schicken.

Übersicht

Abbildung 1: Kurs-Übersicht

I

Einleitung

1 Wie alles begann...

A physician, a civil engineer, and a computer scientiest were arguing about what was the oldest profession in the world. The physican remarked, „Well, in the Bible, it says that God created Eve from a rib taken out of Adem. This clearly required surgery, and so I can rightly claim that mine is the oldest profession in the world.“
The civil engineer interrupted, and said, „But even earlier in the book of Genesis, it states that God created the order of the heavens and the spectacular application of civil engineering. Therefor, fair doctor, you´re wrong: mine is the oldest profession in the world.“
The computer scientist leaned back in her chair, smiled, and then said confidently, „Ah, but who dou you think created the chaos?“

(from G. Booch 1994)

1.1 UNIX - Ein Evolutionsprozeß

UNIX entstand nicht auf einem Reißbrett, es entstand auf vielen Rechnern unter Mitwirkung einer interessierten Öffentlichkeit. Letzterer ist zu verdanken, daß mit UNIX immer „Offene Systeme“ in Verbindung gebracht werden, und dies auf einer sehr prinzipiellen Ebene. UNIX ist traditionell ein Betriebssystem *von* Entwicklern *für* Entwickler. Es ist

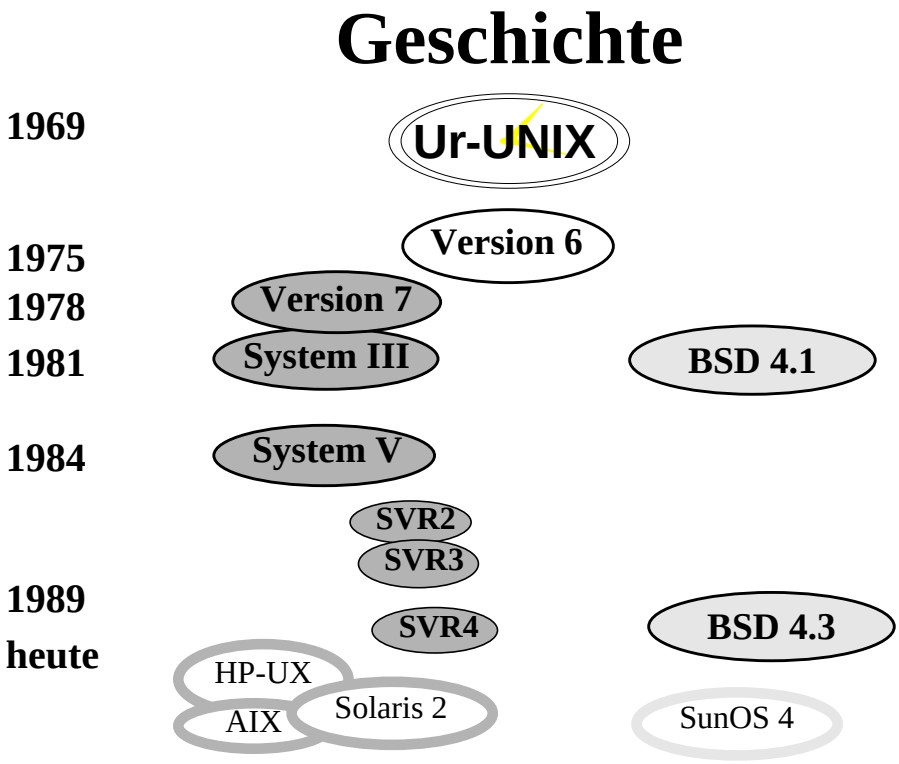


Abbildung 2: Die Geschichte von Unix

genauso einfach, UNIX zu erweitern, wie anzupassen. Verschiedenste Anbieter werben inzwischen auch und im Besonderen um die Gunst von Anwendern. Für letztere erschwert lediglich die ’Vielfalt’ der Kommandooptionen und deren willkürliche Vergabe den Einstieg. Durch graphische Oberflächen über UNIX-Grenzen hinaus verliert dies jedoch an Bedeutung.

1.2 Geschichte von UNIX

1965	Bell, General Electric und MIT wollen MULTICS entwickeln. Ziel ist der gleichzeitige Computerzugriff vieler Benutzer auf gegenseitige Daten
1969	Projekt ’Multics’ stirbt. Ken Thompson (Bell) entwickelt abgemagerte single-user Version auf PDP-7. Name: UNICS
1971	UNIX wird auf PDP-11 portiert.
1973	Dennis Ritchie schreibt UNIX in C um, um es leichter portierbar zu machen.
1977	UNIX wird auf Nicht-PDP-Maschinen portiert, z.B. von IBM, Cray, HP, ... Installationszahl: 500. Bell vergab viele Lizenzen an Universitäten, da AT&T laut Vertrag gewerblich keine Computer vermarkten durfte.
1981	Bell kombiniert einige AT&T-Varianten zu UNIX-System III.
1983	Bell bringt UNIX-System V heraus. University of California in Berkley gibt BSD-UNIX für Vax-Rechner heraus.
1984	UNIX System V Release 2.0 Installationszahl:

	100 000
1986	UNIX System V Release 3.0
1987	Installationszahl: 300 000
1989	UNIX System V Release 4.0 wird freigegeben. Gilt heute als Quasi UNIX Standard. Installationszahl: 1,5 Millionen (ca. 20% des Multiuser Markt
heute	Weltweit gibt es mehrere Millionen Unix-Installationen. Die bedeutestens sind: Linux, FreeBSD, SCO-Unix, SunOS, HP-UX, AIX

2 Die Linux-Story

...Im Prinzip hast du ja auch recht. In der Schule hab ich immer gerne unter NT programmiert. Jeder Absturz war eine Zigarettenpause. :-)
Aber wenn ich davon ausgehe, dass es auch Leute gibt, die nicht rauchen, dann hat Unix doch noch Existenzberechtigung.
(Georg.Datterl@fhs-hagenberg.ac.at in de.comp.advocacy)

2.1 Von Freax zu Linux

Begonnen hatte alles Anfang der 90er-Jahre, als ein finnischer Student, Linus Torvald, sich daran machte, die Grundlagen von Unix zu studieren und erste Experimente anstellte, wie wohl ein Task-Switch auf einem 80386 zu bewerkstelligen ist.

Nach und nach wurde das System ausgebaut und im Internet zur Verfügung gestellt, um andere Entwickler an seiner Entwicklung teilnehmen zu lassen.

Ursprünglich wollte Linux Torvald seinen Quellen unter dem Namen „FREAX“ ins Internet stellen, konnte aber glücklicherweise vom verantwortlichen Systemoperator dazu überredet werden, einen anderen Namen sich auszudenken.

2.2 Die Geschichte der Kernel

Es gibt zwei Arten von Kernel: die sogenannten „Hacker“-Kernel, die daran erkennbar sind, dass sie eine ungerade Nummer tragen (z.B. 2.3), und die „stabilen“-Kernel, die eine gerade Nummer tragen (z.B. 2.2).

2.2.1 Hacker-Kernel

Hacker-Kernel dienen dazu, neuere Entwicklung (z.B. Multi-Processing) und die Anbindung neuer Hardware (z.B. USB[1]) in die Entwicklung mit einfließen zu lassen.

Wer also immer auf dem neuesten Stand der Technik mitschwimmen will oder an der Entwicklung sich aktiv beteiligen will, ist mit einem Hacker-Kernel gut bedient. Allerdings muss man mit gelegentlichen Abstürzen des Kernels rechnen.

2.2.2 Stabile Kernel

Wenn die Entwicklung des Hacker-Kernel weitgehend abgeschlossen ist, wird der Zustand eingefroren und nur noch Bug-Fixing zugelassen, bis ein stabiler Zustand erreicht wird. Dieser Kernel bekommt dann eine gerade Nummer (z.Zt. 2.2).

Der Kernel zeichnet sich dadurch aus, dass er sehr stabil ist – eine Uptime[2] von mehreren Monaten ist hier durchaus die Regel – aber nicht auf dem neuesten Stand ist. Für den Otto-Normal-Verbraucher ist dieser Kernel in der Regel die erste Wahl, da er stabil und pflegeleicht ist.

---

1. Universal Serial Bus, manchmal auch als Useless Serial Bus bezeichnet ;-)

2. „uptime“ ist ein Programm unter Linux/Unix, das die Laufzeit seit dem letzten Booten angibt. Linux-Workstationen haben nicht selten eine „Uptime“ von mehrere Monaten. Zum Vergleich: die durchschnittliche „Uptime“ eines NT-Rechners beträgt 2 Tage.

Aug. 91	0.01	rudimentär benötigt noch MINIX
Okt. 91	0.02	„Hacker-System“ (bash, gcc)
	0.03	
	0.10	bereits mehrere Entwickler
Mär. 92	0.95	
Dez. 93	0.99pl14	

Abbildung 3: Die Geschichte von Linux

2.3 Vor- und Nachteile

2.3.1 Vorteile

Linux bietet gegenüber andere Betriebssysteme folgende Vorteile:

- stabiles und ausgereiftes System
- offenes System
- sicher (keine Viren)
- geht schonend mit HW-Resourcen um
- Unterstützung verschiedener Plattformen (x86, Merced, Alpha, Sparc, PowerPC, 680xx)
- viel Dokumentation vorhanden
- kein DLL-Terror
- beliebig konfigurierbar und anpassbar

2.3.2 Nachteile

Natürlich hat Linux auch Nachteile:

- aufwendigere Installation
- etwas mehr Administration
- kleineres (kommerzielles) SW-Angebot

2.4 Distributionen

Es gibt eine Reihe von Distributionen, die sich hinsichtlich der Zusammenstellung, Preis und Aktualität unterscheiden.

S.u.S.E. (www.suse.de)
: vor allem im deutschsprachigen Raum recht beliebt. Die S.u.S.E.-Distribution kennzeichnet sich durch eine leichte Installation aus und ist damit für den Einsteiger gut geeignet. Allerdings macht sie einige Dinge anders als andere Distributionen.

	Seit S.u.S.E. 5.0 wurde auf RPM (s. Red Hat) umgestellt.
Red Hat (www.redhat.com)	Beim Updaten von einer älteren auf eine neuere Version zeigt Red Hat seine Stärken: dank RPM[3] funktioniert es meist reibungslos.
DLD (Delix Linux Distribution, www.delix.de)	Eine rein deutsche Distribution
Debian	Bei der Debian ist vieles anders als bei anderen Distributionen. Dafür glänzt sie mit einem günstigen Preis (z.B. bei J.F.Lehmann für 19,80 DM)
Slackware	Der Dinosaurier unter den Distributionen. Für den Anfänger ungeeignet, bietet aber den Vorteil, dass sie sich auch auf schwachbrüstigen Rechnern (z.B. 386 mit 4 MB Hauptspeicher) betreiben läßt.

2.5 Informationsquellen

2.5.1 Diskussionsforen

Newsgruppen

Gerade rund um Linux gibt es eine Reihe von Newsgruppen, in die man Anfragen stellen kann, wenn man nicht mehr weiter weiß:

de.comp.os.linux

de.comp.os.linux.hardware

3. Red Hat Package Manager

Mailing-Listen

Wer kein Zugriff auf Newsgruppen hat oder will, kann auch über Mailing-Listen seine Erleuchtungen erlangen. Es gibt Mailing-Listen zu den unterschiedlichsten Aspekten von Linux. Auch viele Distributionen haben ihre eigene Mailing-Listen. Üblicherweise schickt man ein

subscribe mailing-liste

an den Mailing-Listen-Verwalter (oft Majordomo@irgendwo.de), und man nimmt am Nachrichtenaustausch teil. Aber Achtung: es gibt Mailing-Listen, die haben ein recht hohes Nachrichtenaufkommen (und meist leider auch ein hohes Rauschen).

RTFM und andere Weisheiten

Bevor man anfängt, eigene Anfragen in die Newsgruppen oder Mailing-Listen zu stellen, gilt es einige Regeln, die sogenannte „Netiquette“[4], zu beachten. Dazu gibt es ein ausgezeichnetes Buch, „Zen and the Art of Internet“[5], das hierzu eine Übersicht und Einführung bietet.

Ansonsten kann man sich sehr schnell sehr unbeliebt in den entsprechenden Foren machen, vor allem, wenn man typische Anfängerfragen zum tausendundersten Male stellt, die sich mit einem Blick in die zahllosen Dokumentation (s. dieses Kapitel) von selbst erledigt.

Sollte man statt einer gewünschten Antwort öfters ein *RTFM*[6] zu lesen bekommen, ist dies meist ein Anzeichen dafür, dass man vielleicht doch hätte erst die Dokumentation lesen sollen.

erstmal die Foren nur mitlesen und dann gezielt Fragen stellen. Und nicht vergessen, auch anderen zu helfen - die verschiedenen Diskussionsforen leben schließlich davon, dass

4. die leider kaum noch einer kennt.

5. Gibt es auch in einer deutschen Übersetzung: „Zen und die Kunst des Internets“

6. „Read that Fucking Manual“

es auch Leute gibt, die die Antworten liefern.

2.5.2 Online-Information

LDP (Linux Documentation Project):

ftp://sunsite.unc.edu/pub/Linux/docs/LDP

http://www.uni-paderborn.de/Linux/mdw/

2.5.3 Handbuch

Bei den meisten Distributionen ist auch ein Handbuch dabei, das die wichtigsten Schritte erklärt und als Einstieg dient.

Bei der S.u.S.E.-Distribution sind die Handbücher auch auf der ersten CD unter `buch` in verschiedenen Formaten (Text, PostScript (komprimiert), PDF) und Sprachen (deutsch, english) mit dabei. Wer also sein Handbuch nicht wiederfindet, kann z.B. die PostScript-Version[7] ausdrucken oder die PDF-Version am Bildschirm[8] anschauen.

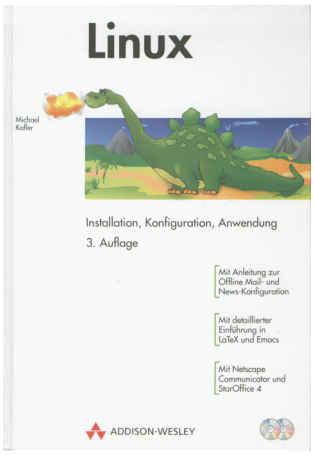
2.5.4 Bücher

Neben den Handbüchern, die den einzelnen Distributionen beiliegen, gibt es eine nahezu unüberschaubare Anzahl von Büchern im Linux-Bereich. Hier nur ein kleiner Auswahl von „Standard“-Büchern:

---

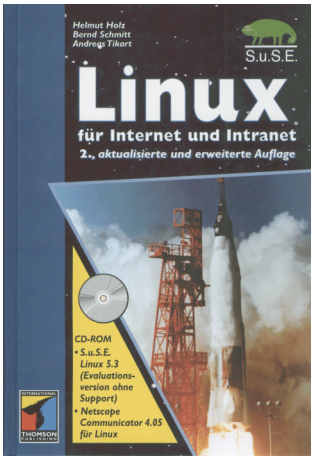
7. dies ist die Datei „buch.psz“ - sie muss allerdings mit ’`gunzip buch.psz`’ erst noch dekomprimiert werden, ehe sie auf einen PostScript-Drucker geschickt werden kann. Allerdings füllt die Postscript-Datei die DIN-A4-Seite nicht ganz aus. Wenn das stört, kann die Datei mit ’`pstops "0@1.25(0,-200)" buch.ps > buch_a4.ps`’ vergrößern und zurechtrücken.

8. üblicherweise nimmt man hierzu den Acrobat-Reader („acroread“ unter Unix) von Adobe



Michael Kofler
Linux
Installation, Konfiguration, Anwendung
Addison-Wesley
ISBN 3-8273-1304-X
DM 89,90

Der „Kofler“, wie dieses Buch auch manchmal kurz genannt wird, beschränkt sich auf die wesentliche Dinge und leistet damit als Einstiegsbuch, aber auch als Nachschlagewerk, gute Dienste.



Helmut Holz, Bernd Schmitt, Andreas Tikart
Linux für Internet und Intranet
International Thomson Publishing
ISBN 3-8266-0432-6
DM 69,00

Dieses Buch beschäftigt sich mehr mit der Netzwerk-Seite von Linux: wie man Internet-Dienste auf einem Standalone-Rechner einrichtet, um auf´s Internet zuzugreifen, aber auch, wie man einen Linux-Rechner als Server für´s Internet und für PC-Netze einsetzt. Es ist damit eher für den fortgeschritteneren Benutzer geeignet, der mehr mit Linux machen will, als ihn bloß als Windows-Ersatz zu betreiben.

Das Linux Anwenderhandbuch und Leitfaden für die Systemverwaltung

s. http://www1.lunetix.de/LHB/

Linux - Wegweiser zur Installation & Konfiguration
Matt Welsh & Lar Kaufmann
Deutsche Übersetzung von Matthias Kalle Dalheimer & Jörg Redmer
ISBN 3-930673-58-4
ca. 700 Seiten
DM 59,-

s. a. http://www.ora.de/german/freebooks/linux_install/inhalt.html

Von O´Reilly gibt es noch weitere lesenswerte Bücher (z.B. „Linux in a Nutshell“, „Linux - Wegweiser zur Programmierung & Entwicklung).

2.5.5 Zeitschriften

Es gibt inzwischen auch einige deutsch- und englischsprachigen Zeitschriften im Linux-Umfeld:

Linux-Magazin

Es erscheint monatlich. Einige Beiträge kann man auch unter

http://www.linux-magazin.de

online lesen.

LinuxFocus

http://www.linuxfocus.org/Deutsch/

LinuxFokus ist ein freies internationales Magazin für Linux, das in verschiedene Sprachen übersetzt wird.

LinuxFocus selbst ist eine gemeinnützige Organisation und das Magazin wird von freiwilligen Mitgliedern aus aller Welt getragen.

Linux BBS

http://www.linuxbbs.org/

Enthält Neuigkeiten rund um Linux.

Linux Journal

http://linuxjournal.com:82/cgi-bin/frames.pl/index.html

2.5.6 Linux-Links

Zu Linux gibt es tausende von Links. Hier eine sehr kleine Auswahl einiger wichtiger Links:

http://www.linuxhq.com (Linux Headquarter)

2.5.7 Online-Kurse

Natürlich gibt es auch Kurs, die man Online am Bildschirm lesen oder sich runterladen und ausdrucken kann:

http://www.uni-tuebingen.de/zdv/zriinfo/linux/kurse/kurse.html
Linux-Kurse am ZDV

http://www.mubo.saar.de/linux/crash.html
Crash-Kurs „Linux“, Version 3.5-ALPHA

2.6 Viren unter Linux

Viren unter Linux sind so gut wie unbekannt. Wenn man einige einfache Sicherheitshinweise beachtet, haben Viren auch kaum eine Chance, sich zu verbreiten. Eine größere Gefahr geht hierbei von Windows aus: hier

gibt es Viren, die z.B. die Partitionstabelle auf der Platte etwas verändern oder zufällig irgendwelche Sektoren auf der Platte löschen oder verändern. Wenn man Pech hat, kann so ein Virus unter Windows auch Linux-Partionen beschädigen. Abhilfe: Windows löschen ;-)

II

Installation & Konfiguration

3 Linux installieren

Nobody will ever need more than 640 K RAM
(Bill Gates 1981)

Windows 95 needs at least 8 MB RAM
(Bill Gates 1996)

Nobody will ever need Windows 95
(logical conclusion)

3.1 Hardware-Anforderungen

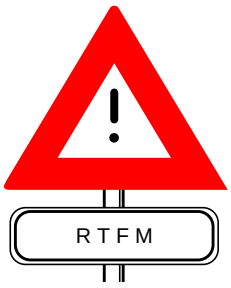
Die Mindestanforderungen für Linux sind recht bescheiden: ein 386 mit 4 MB Hauptspeicher mit einer 20 MB-Platte. Damit läßt sich ein ausgedienter Rechner vielleicht noch als Router oder Waschmaschinensteuerung reaktivieren, für ein sinnvolles Arbeiten unter einer graphischen

Benutzeroberfläche sollte doch ein leistungsstärkeres Gerät ausgesucht werden:

	Masochist	Minimalist	Otto Normal	Power-User
Prozessor	386	486	Pentium	Pentium II
Hauptspeicher	4 MB	16 MB	64 MB	128 MB
Plattenplatz	50 MB	500 MB	1 GB	4 GB
Monitor	12"	14"	17"	19"
Graphikkarte	-	1 MB	2 MB	4 MB

Tabelle 1: HW-Anforderungen

3.2 Benutzer-Anforderungen



Bevor man sich jetzt voller Erwartungen auf Linux stürzt, eine Warnung vorneweg: Linux setzt voraus, dass Sie sich mit dem System auseinandersetzen. Installationen dauern länger als unter Windows und es wird vorausgesetzt, dass man vor der Installation die Beschreibung liest.

Sollte etwas nicht auf Anhieb laufen (was durchaus der Normalfall sein kann), so gibt es für diesen Fall jede Menge von *HOWTOs* und Anleitungen, die man nur durchlesen muss. Es dauert länger, bis etwas läuft – dafür läuft es dann ewig.

3.2.1 Für wen geeignet?

Als Belohnung für all die Mühe winkt dann ein stabiles System, das für folgende Benutzer und Einsätze geeignet ist:

- Programmierer (keine System-Abstürze)

- Internet-Surfer (keine Angst vor Viren)
- Kommandozeilen-Fetischist (es gibt einfach Arbeiten, die sind mit der Tastatur schneller als mit der Maus zu erledigen)
- Informatik-Student (man hat die Sourcen, und kann damit neue Konzepte einbauen und ausprobieren)
- Server
- Netzwerk-Betreiber
- ...

3.2.2 Für wen ungeeignet?

Auch wenn es inzwischen Portierungen von *DOOM* und anderen bekannten PC-Spielen gibt, so ist Linux doch ungeeignet für

- Spielefreak (es gibt zu wenige Spiele unter Linux)
- HW-Freak (Treiber-Entwicklung hinkt etwas der PC-Entwicklung hinterher)
- Manager (können nicht mit der Tastatur umgehen)
- DAU[1] (für den gibt es kein geeignetes Betriebssystem)
- ...

3.3 Die einzelnen Schritte der Installation

Die Schritte, die nötig sind, um Linux auf den Rechner zu bringen, unterscheiden sich von Distribution zu Distribution kaum:

1. Platte vorbereiten
2. Booten eines „Ur-Linux“ (von einer Boot-Diskette oder bei neueren Rechnern von CD)
3. Anlegen der Linux-Partion(en) und einer Swap-Partition
4. Anlegen des Linux-Filesystems
5. Neustart
6. Installation der SW-Pakete

1. „Dümmster Anzunehmender User“

7. LILO einrichten
8. X-Windows einrichten und konfigurieren
9. Administration und Konfiguration

3.4 Festplatte vorbereiten

Bevor man sich daran macht, auf der Platte Platz für Linux zu schaffen und die Platte aufzuteilen, empfiehlt es sich, ein Backup zu machen. Hat man keine Partition oder Platte für Linux frei, sollte man erstmal auf der Platte aufräumen und unter Windows ein „defrag“ starten. Danach kann man mit entsprechenden Programmen (z.B. „Partition Magic“ oder „fips“[2]) eine zusätzliche Partionen anlegen oder bestehende Partitionen vergrößern oder verkleinern.

3.5 Installation starten

Die S.u.S.E.-Distribution kann von CD gebootet werden, falls es der Rechner zuläßt und das BIOS entsprechend eingestellt ist. Haben Sie einen älteren Rechner, booten Sie über die beiliegende Boot-Diskette.

Falls Sie keine Boot-Diskette haben, können Sie die Boot-Diskette mit Hilfe der CD erstellen. Allerdings müssen Sie dazu den Rechner unter DOS starten und Ihr CD-Laufwerk muss unter DOS bekannt sein. Notfalls muß man einen DOS-Treiber für das CD-Laufwerk nachinstallieren. Falls man keinen auftreiben kann, kann man alternativ den Inhalt der 1. CD auf Platte kopieren.

3.5.1 Zum ersten Mal Linux

Wenn Sie die ersten Hürden geschafft haben und den Rechner von CD oder Floppy booten, wird Linux gebootet und das Installationsprogramm gestartet.

---

2. „fips“ ist bei den meisten Distributionen mit dabei

3.5.2 linuxrc

Als erstes wird **linuxrc** gestartet. Hier gilt es, die Treiber für Ihren Rechner auszuwählen. Doch vorher sind noch andere Einstellungen dran (Sprache, Tastatureinstellung).

Unter dem Menü-Punkt „Kernel-Module“ können Sie dann die Treiber aussuchen, die Sie für Ihren Rechner brauchen, oder Sie lassen es den Rechner für erledigen („autoprobe“).

Danach wird mit „Installation / System starten“ die Installation fortgesetzt.

3.5.3 Die Aufteilung der Platte

Mittels YaST wird dann die weitere Installation vorgenommen. YaST ist eine Eigenentwicklung von S.u.S.E. und dient zur Installation, aber auch zur Administration (zumindestens teilweise) eines Linux-Rechners.

Wenn Sie „Linux neu installieren“ ausgewählt haben, geht es ans Partitionieren der Platte(n) und Einrichten der Swap-Partition. Über die Aufteilung der Platte und die Größe für die entsprechenden Dateisysteme gibt es unterschiedliche Ansichten.

Mein Tip: möglichst wenig Partitionen, am besten nur eine[3]. Dann braucht man sich auch keine Gedanken zu machen, wie groß man die einzelnen Partitionen macht. Denn egal, für was einen Wert man sich entscheidet, er ist entweder zu groß oder zu klein, aber selten genau richtig.

Sollte es sich herausstellen, dass eine Partition oder Platte zu klein ist, ist es oftmals das Einfachste (vor allem bei den heutigen Plattenpreisen), den Rechner um eine neue Platte zu ergänzen (und die alte 100 MB-Platte doch endlich rauszuschmeißen).

3. Es gibt hier unterschiedliche Auffassungen darüber, wie eine sinnvolle Aufteilung auszusehen hat. Für den Heimbereich ist eine oder zwei Partitionen ausreichend.

Für die Größe der Swap-Partition gilt als Faustregel: ca. doppelte Hauptspeichergröße, maximal aber 128 MB. Wenn Ihr Rechner schon mächtig viel Hauptspeicher mitbringt (128 MB oder größer), kann auch auf eine Swap-Partition verzichtet werden.

3.5.4 SW-Pakete auswählen

Als nächstes dürfen Sie dann die SW-Pakete auswählen, die Sie installieren wollen. Wenn Sie sich nicht entscheiden können, bleiben Sie bei der Grundeinstellung. Sie können später immer noch Pakete hinzufügen oder löschen.

Nachdem Sie Ihre Pakete ausgewählt haben, wählen Sie „Pakete einspielen“ und holen sich etwas zum Lesen oder machen sich einen Kaffee. Jetzt nudelt der Rechner alleine vor sich hin und installiert die SW-Pakete auf Ihrem Rechner. Sie müssen nur ab und zu eine neue CD einlegen, wenn Sie der Rechner dazu auffordert.

3.5.5 Grundkonfiguration

Mit „Installation abschließen und YaST beenden“ wird der Rechner für den ersten Systemstart vorbereitet. Nehmen Sie am besten die Boot-CD bzw. -Floppy aus Ihrem Rechner.

Nach der Auswahl der Kernel-Version (stabile oder „Hacker“-Kernel) können Sie zwischen verschiedenen Kernel auswählen (EIDE-Kernel, SCSI-Kernel, mit oder ohne Netzwerk, ...).

3.5.6 LILO

Über LILO, dem „Linux Loader“ wird Linux später gebootet. Man kann über LILO aber auch andere Betriebssysteme booten bzw. verschiedene Betriebssysteme zur Auswahl stellen.

Die Angaben hierzu wird über die LILO Boot Konfiguration gemacht, die nach der Auswahl des Kernels folgt. Neben den Betriebssystemen bzw.

zu bootenden Partitionen kann man die Zeit einstellen, die auf eine Auswahl der Boot-Partition gewartet werden soll (Default = 10 Sekunden).

3.5.7 Zeitzone

Danach folgt die Auswahl der Zeitzone. Wählen Sie hier „MET“ (für „Middle European Time“)[4] ein - es sei denn, Sie wollen mit Ihrem Rechner auf die Bahamas auswandern, dann ist „EST“ (für „Eastern Standard Time“) die richtige Wahl.

3.5.8 Netzwerk

Danach werden Sie nach Rechnername und Domainname gefragt. Lassen Sie sich was Nettes einfallen, z.B. „lukas“ als Rechnername und „lummerland“ als Domainname.

Dann folgen einige Fragen zum Netzwerk, die wir aber noch ignorieren können. Lediglich das sogenannte „loopback“-Device wird später für die X-Windows benötigt.

3.5.9 Benutzer einrichten

Danach werden Sie aufgefordert, ein „root“-Passwort zu vergeben. Denken Sie sich eins aus.

Als nächstes schlägt Ihnen der Rechner vor, einen neuen Benutzer anzulegen. Tun Sie ihm den Gefallen. Als Benutzernamen können Sie z.B. Ihren Vornahmen angeben. Der Name sollte jedoch nicht länger als 8 Zeichen sein und üblicherweise in Kleinbuchstaben (z.B. „oliver“).

Dies ist nachher der Benutzer (auch „Account“ genannt), under dem Sie vorzugsweise arbeiten werden. Als „root“ sollten Sie nur in Ausnahmefällen (und dazu gehört z.B. die Installation und Konfiguration) arbeiten.

4. oder auch „CET“ für „Central European Time“

3.5.10 GPM

Danach werden Sie nach Modem (kann vorerst ignoriert) und Maus gefragt. Bei der Maus geben Sie den Typ an. Danach können Sie gleich ausprobieren, ob Sie den richtigen Typ verwischt haben. Falls nicht, probieren Sie den nächsten Typ aus oder lassen es bleiben.

Haben Sie dann erfolgreich die Maus konfiguriert, können Sie mit der Maus auf der Textkonsole kopieren und einfügen.

3.5.11 Installation beendet

Wenn Sie die Installation mittels YaST abgeschlossen haben, werden Sie vermutlich eine gewisse Plattenaktivität feststellen. Das System fängt jetzt nämlich an, diverse Programme und Skripte zu starten, um das System zu konfigurieren. Sie können dies auf der Konsole 9 (über „Alt 9“ erreichbar) mitverfolgen.

Ansonsten können Sie auf einer anderen Konsole (z.B. „Alt 1“) sich am System anmelden und Ihre erste Schritte mit Linux wagen.

3.6 Bootdiskette erstellen

Falls Sie es noch nicht getan haben, sollten Sie sich für den Fall der Fälle eine Bootdiskette anlegen. Dazu melden Sie sich als „root“ an und starten Sie YaST. Unter dem Menüpunkt „Adminstation → Boot-Konfiguration?“ kann man eine Bootdiskette anlegen.

Die Bootdiskette wird gebraucht, falls sich das System nicht mehr booten läßt.

3.7 Linux herunterfahren

Wenn man mit der Arbeit fertig ist, muß der Rechner ordnungsgemäß heruntergefahren werden. Dazu meldet man sich als „root“ an und fährt den Rechner über

```
halt
```

herunter. Alternativ kann man sich einen Benutzer „halt“ (ohne Passwort) einrichten, dem anstatt einer Shell das halt-Kommando eingetragen wird. Wenn man sich dann als Benutzer „halt“ am login-Prompt anmeldet, wird der Rechner ordnungsgemäß heruntergefahren.

Warum muss überhaupt der Rechner runtergefahren werden? Damit alle Dateien ordnungsgemäß abgeschlossen werden. Normalerweise werden Schreibzugriffe aus Performance-Gründen gepuffert, ehe sie tatsächlich auf der Platte landen. Schaltet man den Rechner einfach aus, kann es dadurch zu Dateiverlusten und Inkonsistenzen kommen. Um Inkonsistenzen zu vermeiden wird in einem solchen Fall (d.h. wenn der Rechner nicht ordnungsgemäß runtergefahren wurde) beim nächsten Booten ein **fsck** („File-System-Check“) gefahren, der die Platte auf Inkonsistenzen untersucht.[5] Ein solcher Boot-Vorgang mit File-System-Check dauert dann erheblich länger als ein normaler Boot-Vorgang.

Man kann einen Rechner aber auch einfach durchlaufen lassen. Dies ist sowieso der Normalfall bei Unix-Rechern[6]. Hier ist das Runterfahren eines Rechners eigentlich der Ausnahmefall. Andererseits, wieso sollte man den Rechner überhaupt laufen lassen, wenn man ihn nicht braucht?[7]

5. Ein File-System-Check wird auch automatisch nach einer gewissen Anzahl von Boot-Vorgängen (eigentlich: mount-Vorgängen) selbsttätig durchgeführt - also nicht erschrecken, wenn das System Ihnen beim Booten mitteilt, dass es jetzt einen File-System-Check durchführt.

6. Die durchschnittliche Laufzeit einer Unix-Workstation beträgt ca. 1 Jahr).

7. Ein durchschnittlicher PC benötigt immerhin ca. 500 Watt im Betrieb!

4 Konfiguration

When you say 'I wrote a program that crashed Windows', people just start at you blankly and say 'Hey, I got those with the system, ***for free****.'*
(Linus Torvalds)

4.1 X konfigurieren

Bevor wir uns näher mit dem Booten beschäftigen wollen, wenden wir uns erst der graphischen Oberfläche unter Linux zu. Es arbeitet sich meist einfacher, wenn man mehrere Fenster zur Eingabe zur Verfügung hat.

Aber auch mit der normalen Textkonsolen hat man bei den meisten Distributionen mehrere virtuelle Terminals (meist 6) zur Verfügung. Mit <Alt><2> kann man z.B. auf das Terminal 2 umschalten. Wem das ausreicht, der kann dieses Kapitel auch überspringen.

4.1.1 Welche Oberfläche?

Es gibt unter Linux nicht *die* graphische Oberfläche. Jeder kann sich seine Oberfläche aussuchen, die ihm am besten gefällt. Sehr beliebt ist hierbei die KDE-Oberfläche, die sich zunehmend als Standard etabliert

und im weiteren Verlauf dieses Kapitel verwendet wird.Allen Oberflächen ist die Basis gemeinsam: X-Windows. Genauer: X11R6.

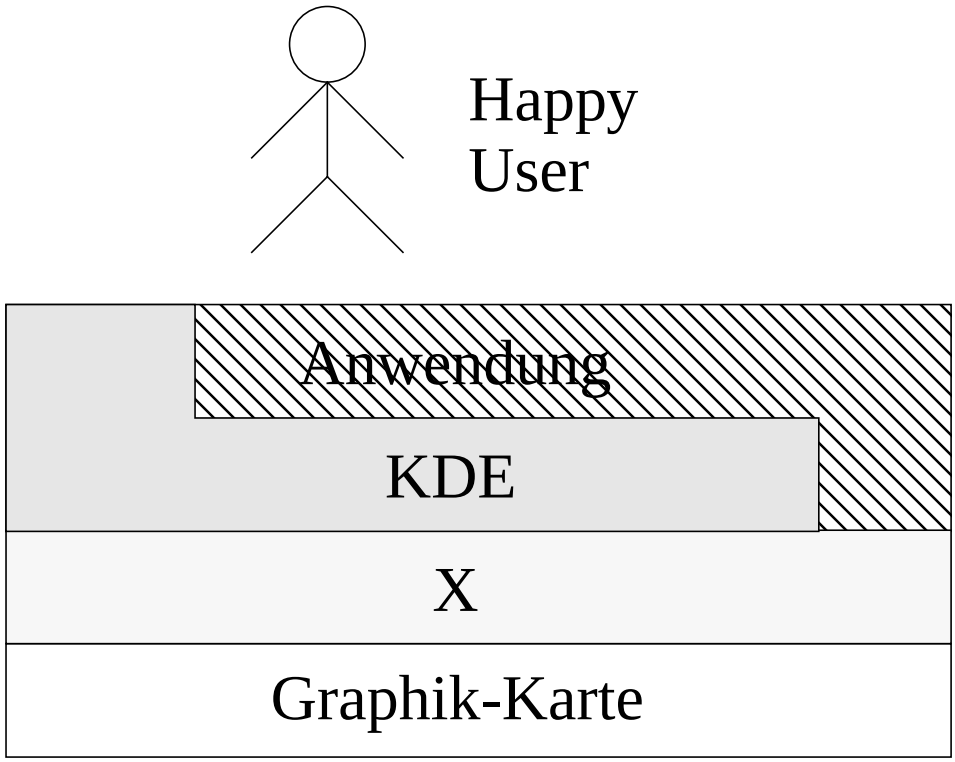


Abbildung 4: Schichten der graphischen Oberfläche

Statt KDE können hier auch andere graphische Oberflächen wie GNOME, fvwm, OpenWindows, ... stehen.

4.1.2 Welche X-Server?

Für Linux stehen mehrere X-Server zur Verfügung - kommerzielle (z.B. Accelerated X) und freie (XFree86). Der Unterschied liegt meist nur in der Anzahl der unterstützten Graphikkarten, manchmal auch in der Performance.

Im weiteren beziehe ich mich auf den XFree86-Server.

4.1.3 Voraussetzungen

Drei Dinge braucht der Mann/Frau, um X-Windows installieren zu können:

- ein Monitor (mit Beschreibung)
- eine Graphikkarte
- eine Maus (2 Tasten oder 3 Tasten)

Es gelten folgende Anhaltswerte für ein Arbeiten unter X-Windows::

	Masochist	Minimalist	Otto Normal	Power-User
Graphikkarte	500 KB	1 MB	2 - 4 MB	8 MB
Monitor	12 - 14"	15"	17"	19 - 21"
Auflösung	640 x 480	800 x 600	1024 x 768 1152 x 864	1024 x 1280 1600 x 1280
virtuelle Auflösung	800 x 600	1024 x 768 1152 x 864	-	-
Farbtiefe	8 Bit	8 Bit	16 - 24 Bit	24 Bit
Maus	2 Tasten	2 Tasten	2 - 3 Tasten	3 Tasten

Tabelle 2: X-Anforderungen

4.1.4 XFree86 konfigurieren

Es gibt mehrere Möglichkeiten, wie man XFree86 konfigurieren kann:

- von Hand (der unbequemste Weg)
- `xf86config` (auch nicht gerade komfortabel, aber zuverlässig)
- `XF86Setup` (graphisch - klappt leider nicht immer)
- `SaX` (nur S.u.S.E. - graphisch, mit automatischer Hardware-Erkennung)

Die Standardmethode ist XF86Setup, mit der die meisten Benutzer klarkommen dürften. Der Vorteil dieser Methode liegt darin, dass man unmittelbar nach der Konfiguration sieht, ob der X-Server läuft oder nicht.

Was man wissen sollte

Bevor man sich an die Konfiguration macht, sollte man folgende Dinge wissen (`SaX` versucht zwar einiges selber herauszubekommen, aber leider kann es nicht alles wissen):

- Art der Maus (braucht man, um mit `XF86Setup` sinnvoll arbeiten zu können)
- Tastatur (deutsch, amerikanisch)
- Graphikkarte
- Monitor

Maus

Bei der Maus sollte man wissen, um welche Art von Maus es sich handelt: PS/2-Maus, Logitech-Maus, ... Ist man sich nicht sicher, kann man es mit `gpm` ausprobieren bzw. nachschauen, was man dort eingestellt hat (YaST starten und den entsprechenden Menüpunkt auswählen).

Tastatur

Die Auswahl der Tastatur ist relativ unproblematisch. Ob Sie eine deutsche oder amerikanische Tastatur haben sehen Sie daran, ob Ihre Tastatur Umlaute (ä, ö, ü) besitzt oder nicht.

Graphikkarte

Die Auswahl der Graphikkarte ist inzwischen auch recht einfach geworden. Sie können sie bei der Konfiguration aus einer Liste von Karten aussuchen und erhalten dann die entsprechenden Hardware-Einstellungen. Natürlich müssen Sie dazu wissen, welche Graphikkarte Sie besitzen. Falls Sie es nicht mehr wissen, können Sie

- in der Dokumentation nachlesen (verflucht noch mal, wo liegt die jetzt wieder rum)
- hoffen, das die Graphikkarte auf der Rechnung aufgeführt ist
- beim Booten schauen, ob sich die Graphikkarte meldet
- den Rechner aufschrauben und nachschauen

Das Handbuch der Graphikkarte wird man auch dann brauchen, wenn die Graphikkarte nicht in dieser Liste auftaucht. Dann muss man wissen, welchen Chip-Satz die Karte besitzt, evtl. noch die Taktfrequenz und den Speicherausbau. Hat man diese Angaben nicht, kann man noch auf die Web-Seite des Herstellers gehen und versuchen, die Angaben von dort zu bekommen oder den Chipsatz auf der Graphikkarte suchen.

Auflösung und Farbtiefe

Die Auflösung und Farbtiefe, die man einstellen kann, hängt von der Graphikkarte ab - je mehr Speicher sie besitzt, desto größer kann die Auflösung und/oder Farbtiefe sein.

XFree86 bietet eine viruelle Auflösung an, d.h. die tatsächliche Auflösung kann höher als die momentan sichtbar Auflösung sein. Kommt man mit der Maus an den Bildschirmrand, scrollt das Bild einfach weiter. Man sieht also nur einen Ausschnitt der virtuellen Bildschirmgröße.

Eine Farbtiefe von 8 Bit bedeutet, dass nur maximal 256 Farben gleichzeitig dargestellt werden können. Dies kann dazu führen, dass die Farben ausgehen, da die meisten Programme die Standard-Farbtabelle benutzen und ihre benötigte Farben dort eintragen. Oder es kann bei Programmen, die eine eigene Farbtabelle benutzten, dazu führen, dass die Farben „umschlagen“, wenn man die Maus in dessen Programm-Fenster bewegt.

Monitor

Auch beim Monitor kann man aus einer Reihe von „Standard“-Monitoren (VGA-Monitor, SVGA-Monitor, ...) seinen Monitor auswählen oder man gibt die horizontalen und vertikalen Frequenzen des Monitors an, die i.a. im Handbuch (ja wo ist es denn?) aufgelistet sind.

Vorsicht bei Einstellungen zum Monitor: vor allem bei älteren Geräten können falsche Angaben zum Verlust des Monitors führen!

Neuere Monitore sind hier etwas toleranter - sie schalten sich einfach ab, wenn man Werte ausserhalb der Spezifikation angibt.

4.1.5 Die Konfigurationsdatei XF86Config

War der Konfigurationsvorgang erfolgreich, wird eine Datei `/etc/Xf86Config` erstellt, die folgende Abschitte („sections“) enthält:

Files	Pfade für Zeichensätze und Farbtabelle
ServerFlags	allgemeine Einstellungen (uninteressant)
Keyboard	Tastatur-Angaben
Pointer	Maus-Angaben
Monitor	Angaben zum Monitor
Device	Angaben zur Graphikkarte
Screen	Angaben der verschiedenen Auflösungen und Farbtiefen

Am interessantesten hierbei sind die Monitor- und Screen-Section.

Monitor-Section

```
Section "Monitor"
   Identifier      "MyMonitor"
   VendorName      "ViewSonic"
   ModelName       "P775"
   HorizSync       30-95
   VertRefresh     50-180
   Modeline  "1152x864"  137.65 1152 1184 1312 1536
                         864 866 885 902 -hsync -vsync
   Modeline  "1024x768"  115.50 1024 1056 1248 1440
                         768 771 781 802 -hsync -vsync
   Modeline  "800x600"    69.65 800 864 928 1088
                         600 604 610 640 -hsync -vsync
```

```
   Modeline  "640x480"    45.80 640 672 768 864
                         480 488 494 530 -hsync -vsync
EndSection
```

Neben allgemeinen Angaben sind hier vor allem die „Modelines“ interessant: sie beschreiben die verschiedenen Auflösungen mit den verschiedenen Timing-Parameter. Sie wird von XF86Setup erstellt, kann aber auch von Hand oder mittels `xvidtune` erstellt bzw. optimiert werden. Nährere Informationen dazu findet man im XFree86-Video-Timings-HOWTO.

Screen-Section

```
Section "Screen"
   Driver          "SVGA"
   Device          "MyVideoCard"
   Monitor         "MyMonitor"
   DefaultColorDepth 32
   BlankTime       2
   SuspendTime     3
   OffTime         5
   SubSection "Display"
      Depth        8
      Modes        "1152x864" "1024x768" "800x600"
                   "640x480"
   EndSubSection
   SubSection "Display"
      Depth        15
      Modes        "1152x864" "1024x768" "800x600"
                   "640x480"
   EndSubSection
   SubSection "Display"
      Depth        16
      Modes        "1152x864" "1024x768" "800x600"
                   "640x480"
   EndSubSection
   SubSection "Display"
      Depth        24
      Modes        "1152x864" "1024x768" "800x600"
                   "640x480"
   EndSubSection
   SubSection "Display"
      Depth        32
      Modes        "1152x864" "1024x768" "800x600"
```

```
                "640x480"
   EndSubSection
EndSection
```

Hier erfolgt die Zuordnung der Graphikkarte zum Monitor und die Wahl der entsprechenden Auflösungen und Farbtiefen.

4.1.6 X starten

Über `startx` wird der X-Server gestartet. Unter `.X.err` kann man üblicherweise nachlesen, ob das Starten erfolgreich war bzw. welche Einstellung aus XF86Config verwendet wurden.

Will man den X-Server mit einer anderen Farbtiefe starten, kann man dies über `startx -- -bpp 24` (für 24 Bit Farbtiefe) versuchen.

4.1.7 Informationsquellen

Ein ausführliche Beschreibung von XFree86 befindet sich im *Linux-Anwenderhandbuch* von Dirk Hohndel. Dirk Hohndel ist Mitglied des XFree86-Entwicklerteams.

Aber auch mit XFree86 werden einige hilfreiche Dokumentation mit ausgeliefert:

- `/usr/X11R6/lib/X11/Cards`
  eine Liste mit Graphikkarten mit ihrem Chipsatz mit dem dazu passenden X-Server
- `/usr/X11R6/lib/X11/doc/Monitors`
  Modelines für zahlreiche Monitore
- `/usr/X11R6/lib/X11/doc/README.`*xxx*
  Wissenswertes zum X-Server *xxx*

4.2 Der Windowmanager

Auch wenn es möglich ist, direkt mit X-Windows zu arbeiten, so ist dies doch recht unbequem. So können z.B. keine Fenstern verschoben, vergrö-

ßert oder verkleinert werden. Ein Windowmanager muss also her, der diese Aufgabe übernimmt. Da hätten wir z.B. den

4.2.1 KDE

Vor einigen Jahren war die (Linux-) Welt grau und eintönig. Es gab zwar einige Windowmanager, aber die sahen im Vergleich zur Windows-Welt doch recht altbacken aus. Um diesem Zustand ein Ende zu bereiten, setzte sich eine kleine Entwicklerschar zusammen, um eine neue, zeitgemäße Oberfläche, das K Desktop Environmet (KDE), auf die Beine zu stellen.

KDE entwickelt sich immer mehr zur Standard-Oberfläche unter Linux und Unix, auch wenn in letzter Zeit Konkurrenz durch GNOME hinzugekommen ist.

4.2.2 Einbindung von KDE

Wenn X-Windows auf der Textkonsole über `startx` gestartet wird, wird von *startx* der Windowmanager, der in der Variablen `WINDOWMANAGER` steht, gestartet. Folgendes müssen Sie daher tun, um KDE als *Ihren* Windowmanager zu starten:

- Setzen der Environment-Variablen `WINDOWMANAGER`
- Erweitern des Suchpfades um `/opt/kde/bin` (unter `/opt/kde` wird bei den meisten Distributionen KDE abgelegt)
- X-Windows starten (`startx`)

Mit der C-Shell sehen diese Befehle folgendermaßen aus:

```
setenv WINDOWMANAGER kde
set path=($path /opt/kde/bin)
startx
```

Bei der Bourne-Shell oder bash gibt man folgendes ein:

```
WINDOWMANAGER=kde
export WINDOWMANAGER
PATH=($PATH:/opt/kde/bin)
startx
```

4.3 Bootkonzepte

4.3.1 Der Bootvorgang

Was passiert, wenn der Rechner eingeschaltet wird?

- Als erstes übernimmt das **BIOS** (Basic Input Output System) die Kontrolle über den Rechner, versucht die Hardware zu erkennen (Hauptspeicher, Festplatte, Graphikkarte) und zu testen, und stellt danach einige Geh-Hilfen zur Verfügung, um MS-DOS und Windows auf die Beine zu helfen.
- Als nächstes gibt das BIOS die Kontrolle ab und startet den **Bootloader** (auch „Primary Bootloader“ genannt), den er vom ersten Sektor der ersten Platte, dem MBR („Master Boot Record“) einliest.
- Der Bootloader lädt das Betriebstystem von der aktiven Partition und startet es. Alternativ kann auch ein zweiter Bootloader („Secondary Bootloader“) gestartet werden und über diesen zweiten Bootloader dann erst das Betriebssystem.

4.3.2 LILO

Bei den meisten Linux-Distributionen wird LILO, der „Linux Loader“, mitgeliefert, der sowohl als primärer Bootloader, als auch als sekundärer Bootloader eingesetzt werden kann.

LILO startet automatisch nach einer gewissen Wartenzeit (default = 10 Sek.) das voreingestellte Betriebssystem (das nicht unbedingt Linux sein muss). Während der Wartezeit hat man die Chance, auch ein anderes Betriebssystem auszuwählen. Falls man nicht mehr weiß, welche Betriebssysteme zur Auswahl stehen, kann man dies über die Tabulator-Taste erfahren.

Konfiguriert wird LILO über die Datei `/etc/lilo.conf`, die man bei S.u.S.E. auch über Yast anpassen kann. Hier trägt man die Platten-Partition ein, von der gebootet werden soll, und die Zeichenkette, die man bei

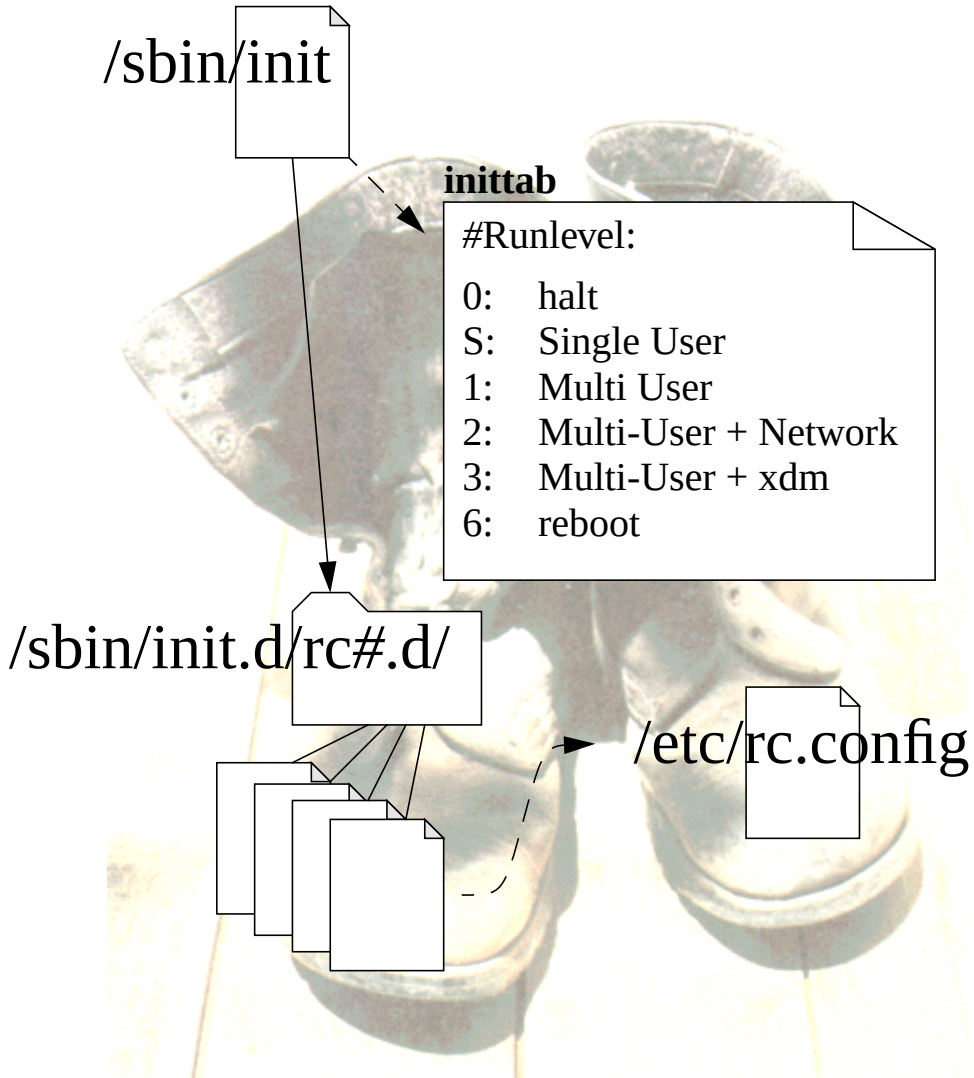


Abbildung 5: Boot-Folge ("SysV-Init")

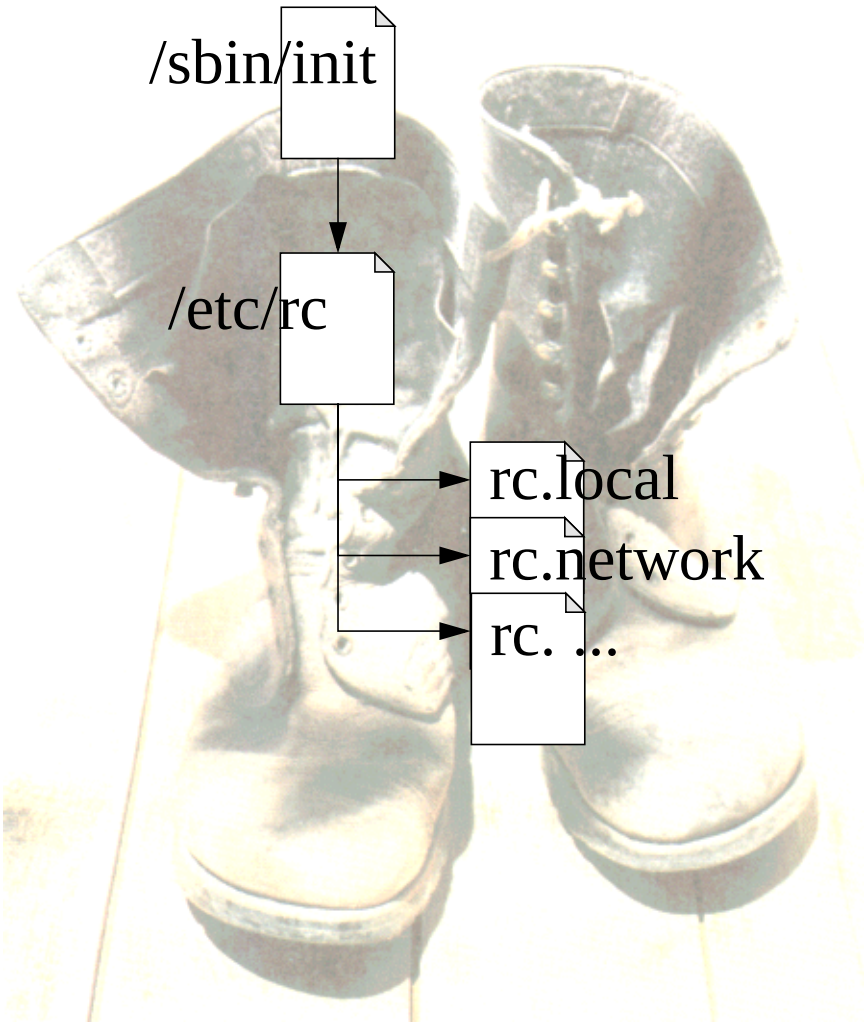
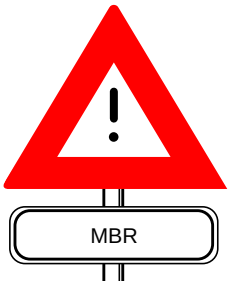


Abbildung 6: Boot-Folge ("Simple-Init")

LILO auswählen kann. Danach muss man noch `lilo` starten, um LILO im MBR fest zu installieren.[8]



Da sich LILO im MBR einnistet, kann es vorkommen, dass Ihr Virenscanner der Meinung ist, dass sich da bei Ihnen möglicherweise ein Virus im MBR eingenistet hat. Ignorieren Sie einfach diese Meldung.[9]

Wird nachträglich DOS oder Windows installiert, so wird in aller Regel der MBR überschrieben[10]. Dann muss LILO nochmal neu installiert werden.[11]

4.3.3 Loadlin

Es gibt auch die Möglichkeit, Linux von DOS aus zu starten. Dazu existiert `loadlin.exe`, das als Parameter noch die zu bootende Linux-Partition übergeben bekommt. Damit kann Linux über

- ein DOS-Start-Menü ausgewählt werden,
- von der Windows-Oberfläche gestartet werden,
- über LILO gebootet werden oder
- von einer Boot-Diskette oder Boot-CD gestartet werden.

Wer sich näher zum Starten über DOS oder Windows interessiert, kann sich über das *Loadlin+Windows95 mini-HOWTO* näher informieren.

---

8. bei S.u.S.E. wird dies automatisch von YaST erledigt.

9. Auf gar keinen Fall die Frage nach dem Entfernen des Virus aus dem MBR bejahen.

10. frei nach dem Motto „...du sollst keine andere Betriebssysteme neben mir haben!“

11. Hoffentlich haben Sie sich dafür eine Bootdiskette erstellt, wie im vorigen Kapitel empfohlen.

4.3.4 Bootdiskette

Natürlich besteht auch die Möglichkeit, Linux über Floppy zu booten (sofern im BIOS das Diskettenlaufwerk A: als erstes Bootlaufwerk eingestellt ist). Die Bootdiskette kann über YaST erstellt werden.

Eine Bootdiskette sollte man sich für alle Fälle anlegen, falls irgend etwas schief gegangen ist, um wieder an seine Daten heranzukommen.

4.4 Backe, backe, Kernel...

4.4.1 Kernelgenerierung - wozu?

Man kann mit den Standard-Kerneln, die den meisten Distributionen beiliegen, i.d.R. ganz gut leben. Trotzdem wird irgendwann mal der Zeitpunkt kommen, wo man um eine Kernelgenierierung nicht umhin kommt. Gründe hierfür können sein:

- Sie wollen Ihre Soundkarte einbinden
- Sie wollen sonstige Hardware einbinden, die vom Standardkernel nicht unterstützt wird
- Sie wollen einen Patch einspielen
- Sie wollen ein Sicherheitsloch stopfen
- Sie wollen nur die Komponenten in Ihren Kernel aufnehmen, die Sie auch wirklich brauchen

4.4.2 Vorraussetzungen

Um einen neuen Kernel generieren zu können,

- müssen die Kernel-Sourcen vorhanden sein[12]
- muss der GNU-C-Compiler installiert sein.

12. werden bei allen Distributionen mitgeliefert

4.4.3 Einbindung neuer Hardware

Die Einbindung von neuer Hardware wird der häufigste Fall sein, wenn man einen neuen Kernel übersetzt. Allerdings kann es bei brandneuer Hardware passieren, dass sie von Linux noch nicht unterstützt wird, da die Hersteller leider nur Windows-Treiber beilegen. Daher empfiehlt es sich, vorher nachzuforschen, ob die Hardware, mit der man liebäugelt, schon unterstützt wird, z.B. unter

http://cdb.suse.de

4.4.4 Das Modul-Konzept

Es gibt zwei Möglichkeiten, einen Treiber im Kernel zu verankern:

1. im Kernel fest eingebunden
2. als dynamisches ladbares Modul

Der Vorteil, wenn man einen Treiber als dynamisch ladbaren Code einbindet, liegt auf der Hand: der Treiber wird erst dann nachgeladen, wenn er auch wirklich gebraucht wird. Dadurch wird der eigentliche Kernel kleiner.

Nachteil: das Nachladen kostet natürlich Zeit. Daher wird man es nur bei den Treiber einsetzen, die man relativ selten braucht und bei denen es auf ein schnelles Ausführen nicht ankommt – ein Festplattentreiber zum Ansprechen der Linux-Partition wird man daher nie als Modul einbinden.

4.4.5 Die Kernel-Konfiguration

Bevor man den Kernel-Konfiguration beginnt, muss man sich zuerst in das Verzeichnis begeben, in der die Kernel-Sourcen liegen (üblicherweise `/usr/src/linux`):

```
cd /usr/src/linux
```

Die Konfiguration des Kernel, welche Treiber im Kernel enthalten sein sollen, gestaltet sich inzwischen recht einfach. Prinzipiell gibt es folgende Möglichkeiten:

- Editieren der `.config`-Datei
- `make menuconfig`
- `make xconfig`

Am einfachsten geht es mit `make xconfig` (unter X-Windows). Es erscheint danach ein graphisches Menü, über den sich die einzelnen Treiber per Maus einfach auswählen lassen.

4.4.6 Kernel backen

Wenn man seine Kernel-Konfiguration beendet und gesichert hat, kann der neue Kernel genieriert („gebacken“) werden. Dazu werden folgende Befehle eingegeben:

make dep	erstellt und überprüft die Abhängigkeiten der Dateien (z.B. Include-Dateien)
make clean	räumt erstmal auf
make bzImage[13]	übersetzt die Kernel-Sourcen und erzeugt einen komprimierten Kernel, der sich beim Booten selbst auspackt.
make modules	die Module werden übersetzt
make modules_install	die übersetzten Module werden installiert
make bzlilo	LILO und der generierte Kernel wird unter `/boot/vmlinuz` installiert; der alte Kernel wird zur Sicherheit nach `vmlinuz.old` umbenannt.

Man kann all diese Schritte alle auf einmal ausführen, indem man

```
make dep clean bzImage modules modules_install bzlilo
```

13. wenn der Kernel nicht zu groß wird, kann man auch „make zImage“ angegeben - der Unterschied liegt nur in der Art der Komprimierung.

auf der Kommandozeile eingibt. Je nach System dauert es dann mehrere Stunden bis wenigen Minuten, bis der neue Kernel neu übersetzt und installiert ist. Hier einige Erfahrungswerte (Kernel 2.2):

Rechner	Hauptspeicher	Dauer
386	8 MB	mehrere Stunden
Pentium 133 MHz	64 MB	30 - 40 Minuten
schneller PentiumPro	128 MB	5 Minuten

Tabelle 3: Übersetzungszeiten

Danach empfiehlt es sich, den Rechner herunterzufahren und den neuen Kernel auszuprobieren. Sollte er wider Erwarten nicht funktionieren, so hat man die Möglichkeit, den alten Kernel zu booten. Danach kopiert man den alten Kernel auf den neuen Kernel drauf, und beginnt wieder von vorne mit der Konfiguration.

Hat man dann einen funktionierende Kernel erzeugt, empfiehlt es sich, mit diesem Kernel eine Bootdiskette für Notfälle anzulegen. Diese wird mit

```
make bzdisk[14]
```

erstellt.

4.5 Benutzerverwaltung

Linux und Unix sind Mehrbenutzersysteme („Multi-User“). Das bedeutet aber auch, dass Benutzer eingerichtet werden müssen. Bis hierher haben Sie vorwiegend unter der Benutzerkennung „root“ gearbeitet, um Ihr System zu konfigurieren und einige Dinge einzurichten.

14. oder `make zdisk`

Jetzt ist es an der Zeit. für sich (und evtl. für andere) einen eingenen „Account“ einzurichten. Unter S.u.S.E. startet man dazu wieder YaST und ruft den Menüpunkt „Administration des Systems → Benutzerverwaltung“ auf. Folgende Angabe werden für das Anlegen benötigt:

Benutzername	der Name, unter dem Sie sich später am System anmelden (max. 8 Zeichen, z.B. „oliver“)
Numerische User-ID	wird vom System schon vorgeschlagen und kann übernommen werden
Gruppe	kann numerisch oder per Name angegeben werden. Eine Liste der bekannten Gruppen findet man unter `/etc/groups`
Home-Verzeichnis	Verzeichnis für die Benutzer-Dateien, üblicherweise `/home/`*`benutzername`*
Login-Shell	der Kommandozeilen-Interpreter, mit dem Sie gerne arbeiten möchten (üblicherweise `/bin/bash` oder `/bin/tcsh`)
Passwort	das Passwort, das Sie bei der Anmeldung angeben müssen. Das Passwort darf beliebig lang sein[15] und beliebige Zeichen[16] enthalten.
Beschreibung des Benutzers	Üblicherweise gibt man hier seinen vollen Namen, gefolgt von einige weiteren wichtigen Angaben wie Telefon-Nummer, Schuhgröße und Lieblingsgruppe an.

15. bei mehr als 8 Zeichen wird der Rest allerdings ignoriert

16. Seien Sie vorsichtig mit Umlauten - nicht alle Tastaturen haben sie. Bei der Anmeldung über einen anderen Recher mit amerikanischer Tastatur steht man dann vor einem verschlossenem System, weil das Passwort nicht eingegeben werden kann.

Hat man dann seinen Account erfolgreich angelegt, probiert man ihm am besten gleich auf einem anderen Fenster aus. Wenn es geklappt hat, meldet man sich als „root“ ab und meldet sich unter seiner eigenen Benutzerkennung an. Den *root*-Account benutzt man dann nur noch in Notfällen oder zu Administrationszwecken.

4.6 Die Verbindung nach draußen

Um mit der großen weiten Welt zu kommunizieren, muss dem Rechner noch beigebracht werden, wie er mit dieser großen weiten Welt kommunizieren soll.

4.6.1 Modemanschluß

Wenn man das Modem über YaST eingetragen hat, kann man mit

```
minicom -s
```

das Modem konfigurieren (als *root*) und unter `/etc/minicom.users` die Benutzer eintragen, die das Modem benutzen dürfen.

4.6.2 ISDN

4.6.3 PPP

4.6.4 Netzwerk-Karten

4.7 Linux macht Druck

4.8 Sicherheitshinweise

Linux ist ein sehr sicheres System, wenn man einige Grundregeln beherzigt:

1. als „root“ nur dann arbeiten, wenn es wirklich notwendig ist
2. un„crack“bare Passwörter
3. ...

III

UNIX

5 Die ersten Schritte

5.1 Benutzer-Profil

Unter Unix gibt es nur zwei Arten von Benutzer:

Root	Ihre Majestät, der Systemadministrator (darf alles)
User	normaler Benutzer (darf fast alles)

Zur Unix-Philosophie gehört es auch, daß (fast) alles erlaubt ist, auch als normaler Benutzer. Das was nicht erlaubt sein soll, muß meist explizit vom Systemadministrator unterbunden werden[1].

1. Bei anderen Betriebssystemen (z.B. auf der IBM Mainframe) ist es z.T. genau anders rum: Von Haus aus ist erst mal (fast) nichts erlaubt, und was dem Benutzer noch zusätzlich erlaubt werden soll, muß vom Administrator freigegeben werden.

5.2 An- und Abmelden

5.2.1 Login

Bei Einschalten Ihrer Workstation erscheint zunächst der Login-Prompt:

```
sel login:
```

SunOS erwartet die Eingabe Ihrer Benutzerkennung, die

- vom Systemverwalter vergeben wird
- durch <Return> abzuschließen ist

Danach wird nach Ihrem Paßwort gefragt, das:

- ebenfalls durch <Return> abzuschließen ist
- nicht am Bildschirm sichtbar ist

```
Password:
```

Wurde vom Systemverwalter kein **Paßwort** vergeben, fahren Sie mit <Return> fort.

Ihre Eingaben werden überprüft und bei Richtigkeit bekommen Sie Zugang zum System. Im Laufe Ihrer Sitzung arbeiten Sie nun unter der eingegebenen Kennung, genauer gesagt unter einer User-ID, die Ihrer Kennung entspricht. Aus dieser User-ID leiten sich im folgenden all Ihre Rechte ab.

5.2.2 Paßwortvergabe

Nach erfolgreichem Einloggen können Sie Ihr Paßwort ändern bzw. ein Paßwort einrichten. Dazu dient das SunOS-Kommando ’passwd’.

Nach Absetzen des Kommandos:

1. werden Sie nach Ihrem alten Paßwort gefragt
2. müssen Sie Ihr neues Paßwort eingeben
3. müssen Sie das neue Paßwort verifizieren

Login/Logout

login	Anmelden
logout/exit/^D	Abmelden
passwd	Ändern des Passworts

Abbildung 7: An-/Abmelden am Rechner

```
sel% passwd
Changing password for kurs3 on sel
Old password:
New password:
Retype new password:
sel%
```

Bei der Wahl des Paßwortes zu berücksichtigen ist, daß:

- es mindestens 6 Zeichen lang sein muß[2]
- zwar mehr als 8 Zeichen erlaubt sind, aber ab dem 9. Zeichen nicht mehr relevant sind
- alle Arten von Zeichen erlaubt sind
- Steuerzeichen nicht ratsam sind
- Groß- und Kleinbuchstaben unterschieden werden

Ein so gesetztes Paßwort:

- ist nun auf Ihrem lokalen Rechner bekannt,
- aber nicht unbedingt netzwerkweit bekannt

Sollten Sie das Pech haben, daß ihr Passwort nur lokal geändert wurde, sie aber das Paßwort netzwerkweit ändern wollen,

- muß in Ihrem Netzwerk eine NIS-Datenbank existieren
- müssen Sie anstelle von ’passwd’ ’nispasswd’ eingeben

Glücklicherweise tritt dieser Fall immer seltener auf, so daß sie den letzten Abschnit ganz schnell wieder vergessen können (außerdem beruhen diese Erfahrungen auf SunOS 4.x und früher).

5.2.3 Logout

Am Ende einer Sitzung steht das Ausloggen. Sie erreichen es durch:

- das Kommando „logout“
- drücken der Tastenkombination Ctrl-d (geht nicht immer)

---

2. Die meisten Systeme meckern zwar, wenn man Passworte angibt, die kürzer als 6 Zeichen sind, aber wenn man hartnäckig genug ist, kann man die meisten Systeme dazu überreden, das Passwort dennoch zu akzeptieren.

```
sel% logout
```

6 Eigenschaften

Das UNIX-Betriebssystem ist durch folgende Eigenschaften charakterisierbar:

- Multitaskingfähigkeit
- Multiuserfähigkeit
- Timesharing
- virtuelles Speichermanagement

6.1 Multitaskingfähigkeit

Multitaskingfähigkeit bedeutet, daß ein Benutzer gleichzeitig mehrere Programme laufen lassen kann. Eine **Task** wird unter UNIX **Prozeß** genannt.

Ein Prozeß ist ein Programm, das sich gerade in Ausführung befindet. Beispiele hierfür sind:

- die Login-Shell
- Kommandos, die im Laufe einer Sitzung aufgerufen werden
- ein beliebiges Anwenderprogramm
- ein Fenster

Der Anwender kann Prozesse:

- erzeugen
- kontrollieren
- beenden

Die CPU kann zu einem Zeitpunkt nur einen Prozeß bearbeiten. Mehrere Prozesse werden bedient, indem:

- die CPU den einzelnen Prozessen jeweils nur für ein kurzes Zeitintervall zur Verfügung gestellt wird
- ständig die Prioritäten der einzelnen Prozesse neu berechnet werden

6.2 Multiuserfähigkeit

Multiuserfähigkeit bedeutet, daß mehrere Benutzer *gleichzeitig* am Rechner arbeiten können. Voraussetzung dafür ist die Multitasking-Eigenschaft.

Dies hat zur Konsequenz, daß durch UNIX:

- Benutzer in irgend einer Form verwaltet werden müssen
- bei einer Login-Anfrage eine Benutzervalidierung durchgeführt werden muß
- bei jedem Dateizugriff die Rechte des Zugreifers mit denen der Datei verglichen werden müssen

6.3 Time Sharing

Ein Betriebssystem hat **Time-Sharing**-Eigenschaften, wenn die Rechenleistung der CPU nach einem durch Prioritäten festgelegten Zeitverwaltungsmechanismus auf die geforderten Aktionen (Tasks) verteilt wird.

Was heißt das? Das heißt nichts anderes, als daß die CPU nicht einem Programm alleine zur Verfügung steht, sondern unter allen laufenden Programmen zeitlich aufgeteilt wird. Die Priorität eines Programms beeinflußt dabei die Länge der ihm zugeteilten „Zeitscheiben“ - je höher die Priorität, desto länger ist seine Zeitscheibe.

UNIX-Time-Sharing

- Rechenleistung wird nach internen Algorithmen auf die Anzahl der aktiven Tasks verteilt
- Rechenleistung wird in Zeitscheiben aufgeteilt (→ Task-Scheduler)
- Prioritäten bei der Zeitscheibenverwaltung lassen sich (begrenzt) beeinflussen
- Tasks werden »quasi« gleichzeitig bearbeitet (TIME-SHARING-Betrieb)

Abbildung 8: Unix-Time-Sharing

Klassische Abteilungs- und Großrechner sind typische Beispiele für Time-Sharing-Systeme.

6.4 Virtuelles Speichermanagement

Speichermanagement meint die Verwaltung der Speicherressourcen.

Speicherressourcen sind:

- der Hauptspeicher
- der lokale Sekundärspeicher
- der Sekundärspeicher von File Servern im LAN

Ziel des Speichermanagements ist es, die Anzahl der lauffähigen Prozesse im Hauptspeicher zu optimieren.

UNIX unterstützt ein virtuelles Speichermanagement. Hierbei wird unterschieden zwischen:

- virtuellen Adressen
- physikalischen Adressen

Virtuelle Adressen

- sind Adressen, die vom Programm verwendet werden
- werden in Verweise auf physikalische Adressen übersetzt

Physikalische Adressen

- sind Adressen, die vom Prozessor verwendet werden
- beziehen sich auf Adressen tatsächlich vorhandener Speicherstellen

Bei Unterstützung von virtuellem Speichermanagement:

- wird der Sekundärspeicher als Erweiterung des Hauptspeichers benutzt
- muß nicht der gesamte Inhalt des Adressraums während der gesamten Prozeßausführung resident im Hauptspeicher geladen sein
- wird der virtuelle Prozeßraum in Einheiten gleicher Größe aufgeteilt, die sogenannten Pages

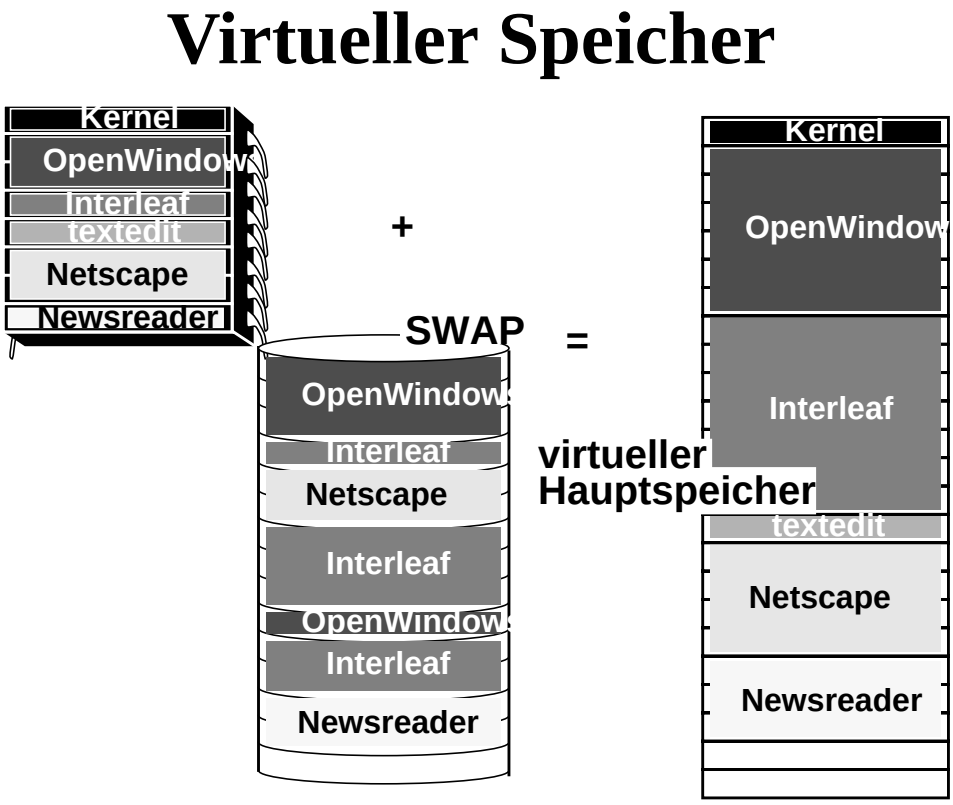


Abbildung 9: Virtuelles Speichermanagement

- wird bei Bezugnahme auf eine nicht geladenen Adresse die entsprechende Page nachgeladen (Demand-Paging)
- werden nicht mehr benötigte Pages auf den Sekundärspeicher ausgelagert

Swapping:

- ist eine Alternative oder Ergänzung zu Paging
- bedeutet die Ein- oder Auslagerung ganzer Prozesse
- ist notwendig, wenn große Teile des Hauptspeichers kurzfristig benötigt werden

Vorteile des virtuellen Speichermanagements:

- Anwenderprogramme sind in ihrer Größe nicht durch den physikalischen Speicher begrenzt
- größere Programme können schneller gestartet werden, da nur ein kleiner Abschnitt geladen werden muß
- Eine effektivere Speichernutzung, da auf Instruktionen und/oder Daten von verschiedenen Benutzern gemeinsam zugegriffen werden kann

6.5 Schichtenmodell

Der **Kernel**:

- bildet den Kern des Betriebssystems
- wird beim Booten resident geladen
- verwaltet den Speicher
- verwaltet das Dateisystem
- verwaltet Prozesse
- verwaltet die Benutzer
- steuert die gesamte Ein-/Ausgabe über angepaßte Gerätetreiber
- wird über System Calls angesprochen

Die **Shell**:

- stellt den klassischen Kommandointerpreter dar

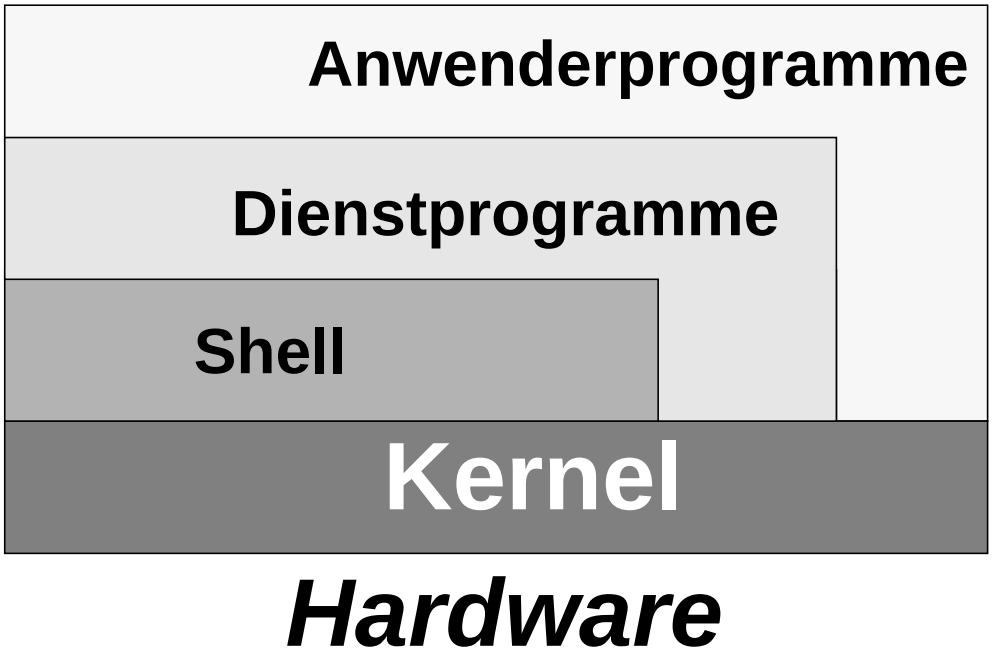


Abbildung 10: Unix-Schichtenmodell

- erlaubt dem Benutzer die Kommunikation mit dem Betriebssystem
- bietet alle Elemente einer Programmiersprache der dritten Generation
- ist Kommandos und Anwendungen logisch gleichgestellt
- bietet zusätzlich eigene Leistungen
- ist austauschbar

Utilities:

- sind die klassischen Dienstprogramme (Kommandos)
- sind der Systemsoftware zugeordnet, aber nicht Teil des Betriebssystems
- sind austauschbar

6.6 Allgemeines Kommandoformat

Eine Kommandozeile:

- besteht aus einem oder mehreren Wörtern, welche durch die Separatoren <SPACE> oder <TAB> getrennt werden
- wird dem System durch <Return> übergeben

Die Shell interpretiert:

- das erste Wort als Kommando- oder Programmname
- die weiteren Worte als Parameter die an das Kommando oder Programm zu übergeben sind

Zu unterscheiden sind zwei Arten von Parametern:

- Optionen
- Argumente

Optionen:

- ermöglichen eine Feineinstellung des Kommandos
- werden i.d.R. durch ein „-“ Zeichen eingeleitet
- stehen i.d.R. vor den Argumenten

Argumente:

- geben dem Kommando an, worauf sich die Aktion beziehen soll
- sind z.B. Dateinamen

Leider halten sich nicht alle Kommandos an dieses Format!

6.7 Benutzerinformationen

Im folgenden werden eine Reihe von Kommandos vorgestellt, die Ihnen verschiedene Informationen liefern.

date

% date [+format]

Ohne weitere Angaben liefert *date* das aktuelle Datum und die Uhrzeit. Als Option kann eine Formatangabe übergeben werden

Beispiele:

```
% date
 Mon Aug 24 11:16:15 GMT 1992

% date +%a%j                # mit Formatangabe
 Mon237

% date +'heute ist der %j.Tag des Jahres'
 heute ist der 237.Tag des Jahres
```

who

who gibt alle momentan am System angemeldeten Benutzer aus.

Beispiel:

```
% who
alf tty01 Aug 24 09:49
erich tty02 Aug 24 09:51

% whoami
alf
```

Die einzelnen Spalten bedeuten:

- Login-Name des Benutzers
- Name des benutzten Terminals
- Zeitpunkt der Anmeldung

Falls man gerade sein Gedächtnis verloren hat und nicht mehr weiß, wer man ist, kann mit **whoami** seinem Gedächtnis wieder auf die Sprünge helfen. Auch **who am i** führt dabei zum Zeil.

cal

% cal [[monat] jahr]

cal ohne weitere Angaben gibt den Kalender des laufenden Monates aus Mit „Jahr“ als Argument gibt den Kalender des gewünschten Jahres aus. Mit „Monat“ und „Jahr“ als Argumente gibt *cal* den gewünschten Monat des angegebenen Jahres aus

Beispiel:

```
% cal
August 1992
 S  M Tu  W Th  F  S
 1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31
```

Bei der Eingabe des Jahres ist zu beachten, daß die Jahreszahl 4-stellig(!) eingetippt werden muß. *cal 94* gibt nicht den Kalender von 1994 an, sondern den Kalender vom Jahre 94 n.Chr. (bzw. das, was das Programm für das Jahr 94 n. Chr. hält)

6.8 Kontrollzeichen

Im folgenden werden eine Reihe von Tastenkombinationen vorgestellt, die u.a. eine Korrektur der aktuellen Kommandozeile oder einen Programmabbruch ermöglichen.

kill	Ctrl-u	löscht die gesamte eingegebene Kommandozeile
word erase	Ctrl-w	löscht das letzte Wort der aktuellen Kommandozeile
erase	Ctrl-h	löscht das letzte Zeichen der aktuellen Kommandozeile, entspricht Backspace
eof	Ctrl-d	generiert ein EOF, beendet die Eingabe bei einer Reihe von interaktiven Kommandos
interrupt	Ctrl-c	bewirkt den Abbruch des laufenden Prozesses
quit	Ctrl-\	bewirkt den Abbruch des laufenden Prozesses und erzeugt einen Speicherauszug (core dump)
suspend	Ctrl-z	hält den laufenden Prozess an (kein Abbruch!)
stop	Ctrl-s	hält die Bildschirmausgabe an
start	Ctrl-q	setzt die Bildschirmausgabe fort

Tabelle 4: Kontrollzeichen

7 Das UNIX-Dateisystem

7.1 Eigenschaften des UNIX Dateisystems

Das UNIX Dateisystem:

- besteht aus einer Vielzahl von Dateien
- ist hierarchisch strukturiert

Dateien werden logisch gruppiert, indem sie in Verzeichnissen zusammengefaßt werden. Ein **Verzeichnis** kann seinerseits wieder Verzeichnisse beinhalten.

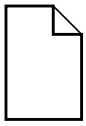
Sinn dieser Strukturierung ist es, Informationen beliebiger Art:

- unter einem eindeutigen Namen abzuspeichern
- schnell wiederzufinden

Dateien:

- sind als Folge von Bytes zu verstehen, d.h. ihnen werden von Seiten des Betriebssystems keine interne Strukturierung, z.B. in Datensätze, auferlegt
- können teilweise vom Benutzer gelesen werden (ASCII-Dateien)
- können teilweise nur vom Rechner gelesen werden (Binärdateien)

Was ist eine Datei?



Sequentielle Folge von Bytes

- ASCII-Format
- Binäres Format

Im Dateibaum beliebig adressierbar

Mehrfachverweis (*link*) möglich

Dateiende wird über EOF (end of file) definiert

Ablage erfolgt in Blöcken (meist 512 oder 1024 Bytes)

Abbildung 11: Was ist eine Datei?

- werden in Blöcken von Bytes gespeichert und sie brauchen nur den Speicherplatz in Blöcken, der notwendig ist, ihren aktuellen Inhalt abzuspeichern
- können wachsen und schrumpfen
- können vor unerwünschtem Zugriff geschützt werden

Verzeichnisse:

- sind das strukturierende Element des UNIX Dateisystems
- sind eine besondere Ausprägung von Dateien
- haben im Gegensatz zu gewöhnlichen Dateien eine vom System vorgegebene interne Struktur
- enthalten die Namen der in ihnen zusammengefaßten Dateien und Verzeichnisse, den eigenen Namen, und den Namen des ihnen jeweils übergeordneten Verzeichnisses
- können von Anwenderprogrammen nicht beschrieben werden

7.2 Dateitypen

Es werden 3 Typen von Dateien unterschieden:

- gewöhnliche Dateien
- Verzeichnisse
- Spezialdateien

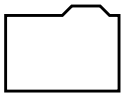
Gewöhnliche Dateien können sein:

- Texte
- Quelldateien
- Objektdateien
- Programmdateien
- Shellprozeduren
- weitere Dateien wie Bild-, Sound-, ... Dateien

Verzeichnisse können andere Dateien (auch Verzeichnisse) enthalten und dienen zur Gliederung des Dateisystems.

Spezialdateien sind:

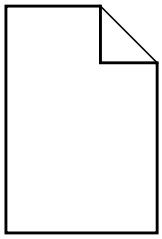
Dateiverzeichnis



- Zusammenfassen von Dateien
- hierarchische Anordnung (Dateibaum)
- keine Mehrfachverweise möglich

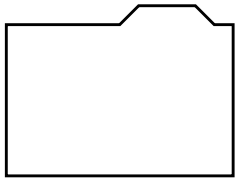
Abbildung 12: Dateiverzeichnis

Datei-Typen:



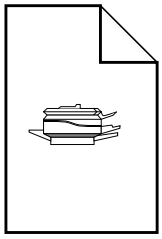
gewöhnliche Datei

- Folge von Bytes



Verzeichnisse

- Zusammenfassen von Datein



Spezial-Dateien

- Devices
- Fifo-Dateien

Abbildung 13: Datei-Typen

- blockorientierte Gerätedateien
- zeichenorientierte Gerätedateien
- FIFO Dateien

Geräte werden in UNIX über sogenannte Gerätedateien angesprochen. Dadurch wird eine weitgehende Geräteunabhängigkeit sichergestellt. Das Interface besteht lediglich aus Datei-Lese- oder Schreib-Anweisungen.

7.2.1 Versteckte Dateien

Versteckte Dateien („*hidden files*“) beginnen immer mit einem Punkt im Dateinamen. Im Gegensatz zu den **sichtbaren Dateien** bekommt sie der Benutzer beim Auflisten der Dateien nicht zu Gesicht.

7.3 Das UNIX Dateisystem

Leider ist Unix nicht gleich Unix, d.h. es gibt Unterschiede zwischen den einzelnen Unixe, die sich in der Namensgebung der einzelnen Verzeichnisse, aber auch deren Struktur unterscheiden können. Die folgenden Angaben beziehen sich im wesentlichen auf SunOS, lassen sich aber weitgehend auch auf die anderen Unixe übertragen.

7.3.1 /

Das **Root**-Verzeichnis ´/´:

- ist die Wurzel des UNIX-Dateisystems
- enthält den UNIX-Kernel, das Boot-Programm und eine Reihe systemspezifischer Verzeichnisse

7.3.2 /var

Das /var-Verzeichnis enthält Verzeichnisse mit variabler Größe

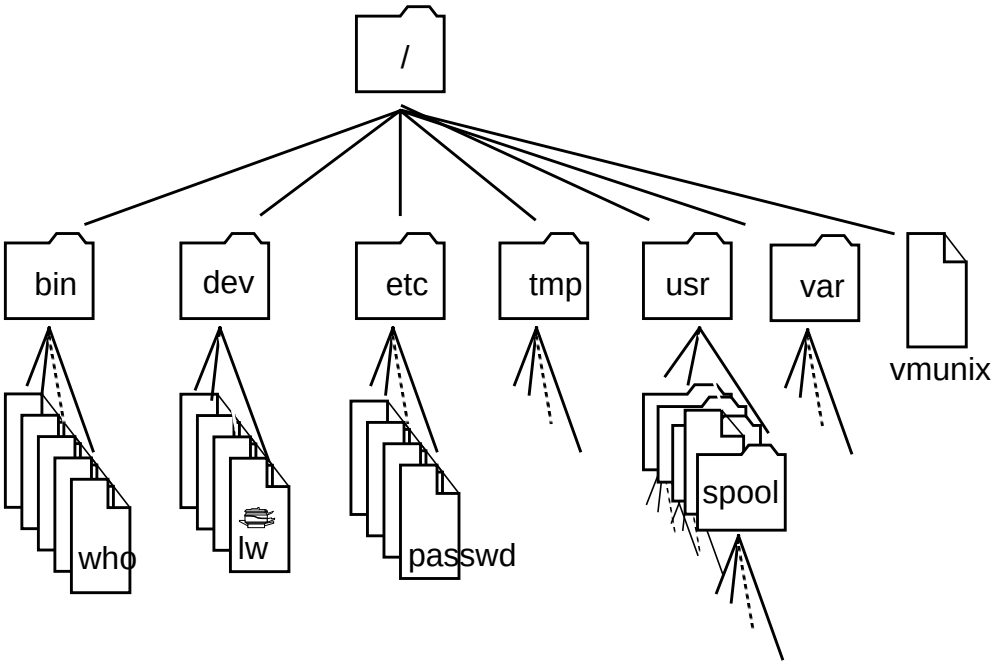


Abbildung 14: Das *root*-Verzeichnis

7.3.3 /export

Das /export-Verzeichnis:

- ist nur auf dem NFS-Server vorhanden
- enthält Verzeichnisse für die Clients

7.3.4 /lost+found

Das /lost+found-Verzeichnis:

- ist zunächst immer leer
- wird vom Programm '*fsck* benötigt, um verlorengegangene Dateien einzusammeln

7.3.5 /home[1]

Das /home-Verzeichnis enthält die Heimat-Verzeichnisse der einzelnen Benutzer.

7.3.6 /dev

Das /dev-Verzeichnis enthält alle Gerätedateien z.B. für Festplatten, Terminals und Bandlaufwerke.

7.3.7 /etc

Das /etc-Verzeichnis enthält eine Reihe von Konfigurationsdateien und Shellprozeduren für den Systemstart; u.a. steht steht hier auch die Passwortdatei (passwd) drin.

---

1. Unter älteren Unixe oder auch unter HP-UX findet man dieses Verzeichnis unter */users* vor. Auch kann vom Systemverwalter ein anderer Name hierfür vergeben worden sein.

7.3.8 /mnt

Das /mnt-Verzeichnis ist für das temporäre Einhängen eines Dateisystems vorgesehen

7.3.9 /bin

Hier stehen die UNIX-Kommandos drin. Oftmals ist es mit dem Verzeichnis →**/usr/bin** identisch.

7.3.10 /sbin

Das /sbin-Verzeichnis enthält Kommandos, die für die Initialisierungsphase des Betriebssystems notwendig sind.

7.3.11 /tmp

Das /tmp-Verzeichnis:

- dient einer Reihe von Programmen als Ablage für temporäre Dateien
- wird bei jedem Systemstart gelöscht

7.3.12 /usr

Das /usr-Verzeichnis enthält eine Reihe klassischer Unterverzeichnisse. Dazu gehören:

- /usr/bin für alle Anwenderkommandos
- /usr/etc für Systemverwalterkommandos
- /usr/lib für Bibliotheksarchive
- /usr/include für sogenannte Headerdateien

Die letzten beiden Verzeichnisse (/usr/lib und /usr/include) sind vor allem für den (C-)Programmierer von Bedeutung: hier kann er nach den Headerdateien und nach den verfügbaren Bibliotheken schauen.

7.4 Namenskonventionen

Der Namen einer Datei bzw. eines Verzeichnisses darf je nach System 14 Zeichen (POSIX) bzw. 256 Zeichen (BSD-UNIX) nicht überschreiten.

Dabei dürfen Datei- und Verzeichnisnamen alle Zeichen außer ’/’ enthalten.

Pfadnamen dürfen maximal 1024 Zeichen lang sein.

Das **Root-Verzeichnis**:

- ist die Spitze des Dateisystems
- hat kein übergeordnetes Verzeichnis
- wird auch Wurzel-Verzeichnis genannt

Das aktuelle Verzeichnis ist immer das Verzeichnis, in dem Sie sich gerade befinden.

Das **Home-Verzeichnis**:

- wird jedem Benutzer vom Systemverwalter zugewiesen
- ist das oberste Verzeichnis des benutzereigenen Arbeitsbereiches
- steht mit seinem gesamten Pfad in der Umgebungsvariablen HOME
- ist das Verzeichnis, indem Sie sich nach dem Anmelden befinden

7.5 Pfadnamen

Ein **Pfadname** ist die Beschreibung des Pfades zu einer Datei oder einem Verzeichnis innerhalb des Dateisystems.

Eine Pfadangabe beginnt immer mit:

/	Root-Verzeichnis
datei/datei	Unterverzeichnis
.	aktuelles Verzeichnis
..	übergeordnetes Verzeichnis

In der **C-Shell** kann eine Pfadangabe zusätzlich mit „~“ beginnen:

~	eigenes Home-Verzeichnis
~*user*	Home-Verzeichnis von *user*

Pfadangaben enden immer mit:

- einem Dateinamen
- oder einem Verzeichnisnamen

Zu unterscheiden sind:

- absolute Pfadnamen
- relative Pfadnamen

Bei Bezugnahme auf eine Datei oder ein Programm können immer absolute oder relative Pfadnamen verwendet werden.

7.5.1 Absoluter Pfadname

Der absolute Pfadname:

- umfaßt den Namen und die Position der Datei im Dateisystem
- hat das Root-Verzeichnis oder das Home-Verzeichnis als Bezugspunkt
- beginnt immer mit einem ’/’ oder einem ’~’

Ist „tarzan“ ihr Home-Verzeichnis, lautet der absolute Pfadname der Datei:

adressen	/home/tarzan/info/adressen oder ~/info/adressen
prg2.c	/home/tarzan/entwickl/src/prg2.c oder ~/entwickl/src/prg2.c

7.5.2 Relativer Pfadname

Der relative Pfadname:

- umfaßt den Namen und die Position der Datei im Dateisystem
- hat Ihr aktuelles Verzeichnis als Bezugspunkt
- beginnt nicht mit einem ’/’ oder ’~’

Befinden Sie sich im Verzeichnis „info“, ist der relative Pfadname der Datei:

prg1.c	../entwickl/src/prog1.c
prg2	../entwickl/prg/prg2

Das Kommando ’**pwd**’:

- steht für „**p**rint **w**orking **d**irectory“
- dient der Ermittlung Ihres aktuellen Verzeichnisses
- gibt immer den vollständigen absoluten Pfadnamen auf dem Bildschirm aus

7.6 Bewegen im Dateisystem

Um Operationen auf Dateien durchzuführen, können Sie:

- entweder in Ihrem aktuellen Verzeichnis verbleiben und die Dateien immer über ihren Pfadnamen ansprechen
- oder einmalig in das Verzeichnis wechseln, in dem sich die Dateien befinden, und diese lediglich über ihren Dateinamen ansprechen

Der Wechsel in ein anderes Verzeichnis erfolgt mit dem Kommando „cd“.

% **cd** [Verzeichnis]

Wird „cd“:

- mit einer relativen oder absoluten Pfadangabe aufgerufen, ist das angegebene Verzeichnis anschließend Ihr aktuelles Verzeichnis
- ohne Parameter aufgerufen, erfolgt ein Wechsel in Ihr Home-Verzeichnis

Beispiel:

```
% pwd
/home/tarzan/info
% cd ../entwickl/src
% pwd
/home/tarzan/entwickl/src
```

Bewegen im Dateibaum

cd *path* Wechseln ins Dateiverzeichnis *path*

Reservierte Namen für *path*:

- / Wurzel-Verzeichnis (root directory)
- .. drüberliegendes Dateiverzeichnis (parent directory)
- . aktuelles Verzeichnis
- ~ Heimatverzeichnis (home directory)

cd Wechseln ins Heimatverzeichnis

pwd Anzeige des Dateiverzeichnisses (**p**rint **w**orking **d**irectory)

Abbildung 15: Bewegen im Dateibaum

7.7 Anzeigen von Verzeichnisinhalten

Der Inhalt von Verzeichnissen kann mit dem Kommando ’ls’ angezeigt werden.

% **ls** [Optionen] [Verzeichnis(se)|Datei(en)]

Wird „ls“:

- ohne Verzeichnis- oder Dateiangabe aufgerufen, werden alle Dateien und Unterverzeichnisse des aktuellen Verzeichnisses namentlich aufgelistet
- mit einer Verzeichnisangabe aufgerufen, wird der Inhalt des benannten Verzeichnisses aufgelistet
- mit einer Dateiangabe aufgerufen, wird der Eintrag für diese Datei im aktuellen Verzeichnis gesucht und ggf. angezeigt

Beispiele:

```
% pwd
/home/tarzan/entwickl/src
% ls
prg1.c
prg2.c
% ls prg1.c
prg1.c
% ls ../prg
prg1
prg2
```

Eine Vielzahl von Optionen ermöglichen die Ausgabe zusätzlicher Informationen. Die wichtigsten Optionen sind:

-a (all) listet zusätzlich die versteckten Dateien auf (das sind die, die mit einem Punkt beginnen)

-F kennzeichnet die einzelnen Dateien mit ihrem Dateityp.

Der Dateiname wird dabei ggf. mit einem besonderen Zeichen abgeschlossen:

kein Zeichen	gewöhnliche Textdatei
*	ausführbare Datei

/ Verzeichnis

@ symbolischer Link

-l (*l*ong listing) gibt ausführliche Informationen zu einzelnen Einträgen

-R (*R*ecursive listing) listet auch die Inhalte aller Unterverzeichnisse auf

Die einzelnen Optionen können natürlich kombiniert werden!

Beispiele:

```
% pwd
/home/tarzan/entwickl/src
% ls -al
drwxr-x--- 2 tarzan 120 May 10 07:24 .
drwxr-x--- 2 tarzan 110 May 05 11:18 ..
-rw-r--r-- 1 tarzan 1284 May 12 12:13 prg1.c
-rw-r--r-- 1 tarzan 2566 May 23 09:24 prg2.c
% ls -F ../prg
prg1*
prg2*
```

7.8 Anzeigen von Dateiinhalten

Dem Anzeigen von Dateiinhalten dienen die Kommandos:

- more
- cat
- head
- tail
- od

7.8.1 more

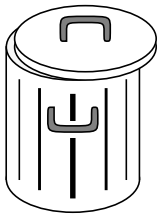
„more“:

- ist ein interaktives Kommando
- zeigt Dateien seitenweise an

% more [+Zeilennummer] Datei(en)

„more“ zeigt die erste Bildschirmseite der als Argument übergebenen ersten Datei an und erwartet die Eingabe eines Kommandos. Folgende Kommandos stehen u.a. zur Wahl:

<Leerzeichen>	Anzeigen der nächsten Seite
b	Anzeigen der vorigen Seite
nf	’n’ Seiten vorwärts blättern
nb	’n’ Seiten rückwärts blättern
q	Beenden von ’more’
?	Hilfe - Anzeige einer Liste gültiger Kommandos
/string	Suche nach dem nächsten Vorkommen von String
:n	Laden der nächsten Datei
:p	Laden der vorigen Datei



Ein kleiner Bug/Feature am Rande: unter SunOS 4.1.3 (und sicherlich auch auf anderen Unix*en*) führt das Anschauen des „Mülleimers“ mit

```
more /dev/null
```

zu recht sonderlichem Verhalten.

7.8.2 cat (*con*cat*enate)*[2]

„cat“ dient der Ausgabe von Dateiinhalten.

% cat [Optionen] [Datei(en)]

2. Der Name „cat“ kommt von „con*cat*enate“; mit dem *cat*-Kommando können mehrere Dateien aneinandergehängt ausgegeben werden

Bei Aufruf von „cat“:

- mit Dateien als Argumente werden deren Inhalte sukzessive auf den Bildschirm geschrieben
- mit der Option ’-n’ werden neben dem Dateiinhalt noch Zeilennummern ausgegeben[3]
- ohne Argumente wird die Eingabe von der Tastatur wieder auf den Bildschirm geschrieben. Das Ende der Eingabe ist mit Ctrl D zu signalisieren.

Die Ausgabe kann mit Ctrl S gestoppt und mit Ctrl Q wieder fortgesetzt werden.

Beispiele:

```
% cat /etc/termcap
% cat -n /etc/printcap
% cat
Erste Zeile von der Tastatur[4]
Erste Zeile von der Tastatur
Zweite Zeile von der Tastatur
Zweite Zeile von der Tastatur
Ctrl D
%
```

7.8.3 head

„head“ schreibt die ersten Zeilen einer Datei auf den Bildschirm.

% head [Option] Datei(en)

Wird „head“:

- ohne Option aufgerufen, werden die ersten 10 Zeilen der als Argument angegebenen Datei auf den Bildschirm geschrieben

3. leider nicht unter HP-UX

4. Eigentlich würde man ja erwarten, daß jedes Zeichen gleich wieder ausgegeben wird. Aber unter Unix wird die Eingabe gepuffert. Erst wenn die Return-Taste betätigt wird, bekommt das Programm die Eingabe zu sehen.

- mit der Option ’-n’ aufgerufen, werden die ersten ’n’ Zeilen der Datei ausgegeben

Beispiel:

```
% head -5 /etc/printcap
```

7.8.4 tail

’tail’ schreibt die letzten Zeilen einer Datei auf den Bildschirm.

% tail [Optionen] Datei(en)

Wird ’tail’:

- ohne Option aufgerufen, werden die letzten 10 Zeilen der Datei auf den Bildschirm geschrieben
- mit der Option ’-n’ aufgerufen, werden die letzten ’n’ Zeilen der Datei ausgegeben
- mit den Option ’+n’ aufgerufen, wird der Dateiinhalt ab Zeile ’n’ ausgegeben

Beispiele:

```
% tail -3 /etc/passwd
% tail +2100 /etc/termcap
```

7.8.5 od (o*ctal* d*ump*)[5]

„od“ zeigt auch nicht druckbare Zeichen einer Datei an.

% od [Optionen] datei(en)

„od“ zeigt den Inhalt der Datei, abhängig von der verwendeten Option:

- oktal (default)
- hexadezimal

---

5. Der Name *od* ist historisch bedingt: UNIX wurde ursprünglich auf einer PDP-7 entwickelt, und das bevorzugte Dump-Format war oktal. Dies ist im übrigen auch der Grund dafür, daß sich die Oktal-Schreibweise auch in der Sprache C wiederfindet. Heute hat die Oktalschreibweise an eine Bedeutung verloren.

- zeichenweise

Mögliche Optionen sind:

-b	jedes Byte wird in oktaler Schreibweise angezeigt
-c	jedes Byte wird, wenn möglich, als ASCII-Zeichen dargestellt, sonst in C-Notation: \00 \nLinefeed \rReturn \tTabulator
-x	jedes Wort wird hexadezimal angezeigt

Beispiel:

```
% od -bc prog.c
```

7.9 Suchen nach Dateiinhalten

Sollen Dateien auf einen bestimmten Inhalt hin untersucht werden, bietet UNIX das mächtige Kommando „grep“.

% **grep** [Optionen] Suchmuster Datei(en)
% **egrep** [Optionen] Suchmuster Datei(en)
% **fgrep** [Optionen] String Datei(en)

Die Kommandos der „grep“-Familie

- durchsuchen die angegebene Datei(en) nach einem Muster (falls keine Datei angeben ist, wird von der Standardeingabe gelesen)
- schreiben jede Zeile mit gefundenem Muster auf den Bildschirm
- schreiben bei mehr als einer zu durchsuchenden Datei auch deren Dateinamen auf den Bildschirm

„*f*grep“ unterstützt nur „*f*ixed strings“, d.h. es können keine Suchmuster angegeben werden.

„*e*grep“ steht für „*extended* grep“. Es unterscheidet sich vom normalen *grep* durch die Unterstützung der vollen regulären Ausdrücke (*grep* bietet nur eingeschränkte Suchmuster). Im Allgemeinen ist *egrep* das schnellste der drei Programme, aber leider nicht auf jedem Unixsystem vorhand.

Das Suchmuster

- kann ein(e) gewöhnliche(s) Zeichen(folge) sein
- kan regulärer Ausdrücke enthalten

Folgende Optionen sind möglich:

-n	Neben der Zeile wird auch die Zeilennummer ausgegeben
-v	Es wird nach Zeilen gesucht, in denen das Muster nicht enthalten ist
-i	Groß-/Kleinschreibung wird bei Suchmuster nicht berücksichtigt
-c	Nur die Anzahl der Zeilen, die das Suchmuster beinhalten, wird ausgeben
-l	Nur die Namen der Dateien, die das Suchmuster beinhalten, wird ausgegeben

Beispiele:

```
% grep tarzan /etc/passwd
% grep -l tarzan /etc/*
```

7.10 Bestimmung des Dateityps

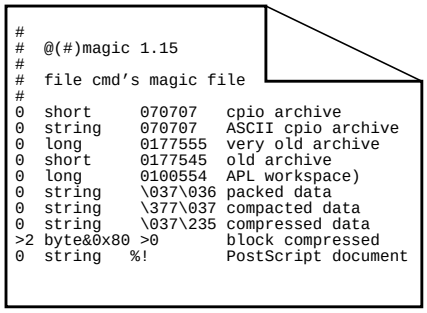
Der Inhalt einer Datei kann mit dem **file**-Kommando ermittelt werden. Im Gegensatz zu *ls -F* analysiert *file* den Dateiinhalt und unterscheidet u.a. folgende Dateiarten:

- ascii text
- c-shell commands
- English text

- executable
- directory
- symbolic link
- weitere... (vgl. Datei /etc/magic)

Beispiel:

```
% file /usr/include/stdio.h
```



```
#
#  @(#)magic 1.15
#
#  file cmd's magic file
#
0  short     070707   cpio archive
0  string    070707   ASCII cpio archive
0  long      0177555  very old archive
0  short     0177545  old archive
0  long      0100554  APL workspace)
0  string    \037\036 packed data
0  string    \377\037 compacted data
0  string    \037\235 compressed data
>2 byte&0x80 >0       block compressed
0  string   %!        PostScript document
```

7.11 Suchen von Dateien im Dateisystem

Das Kommando 'find' durchsucht Verzeichnisse nach Dateien, die bestimmte Kriterien erfüllen. Die Liste gefundener Dateien kann auf verschiedene Arten weiterverarbeitet werden.

% **find** Pfadname(n) Bedingung(en) Aktion

„Pfadname“:

- ist das Einstiegsverzeichnis für die Suche, d.h. ausgehend von 'Pfadname' werden auch sämtliche Unterverzeichnisse durchsucht
- kann mehrfach angegeben werden

„Bedingung“:

- ist als einschränkendes Kriterium für die Suche zu verstehen, d.h. jede gefundene Datei muß dieses Kriterium erfüllen
- muß nicht angegeben werden - alle Dateien von 'Pfadname' werden dann selektiert
- kann mehrfach angegeben werden, wobei gefundene Dateien dann alle angegebenen Bedingungen erfüllen müssen

„Aktion“:

- wird auf die Liste der gefundenen Dateien angewendet
- muß angegeben werden, sonst findet keine weitere Verarbeitung statt

Beispiele:

```
% find /home/tarzan -print
% find /home/tarzan /home/jane -print
```

```
% find /home/tarzan -name prg1.c -print
```

Von der Vielzahl möglicher Bedingungen sind die wichtigsten:

-name dateiname	Name der zu suchenden Datei(en)

Dateien mit einer Größe von:

-size +n	mehr als ’n’ Blöcken
-size -n	weniger als ’n’ Blöcken

Dateien, die vor:

-mtime n	’n’ Tagen modifiziert wurden
-mtime +n	mehr als ’n’ Tagen modifiziert wurden
-mtime -n	weniger als ’n’ Tagen modifiziert wurden
-type dateityp	Dateien eines bestimmten Typs. Folgende Typen sind zu unterscheiden: ’f’ einfache Dateien ’d’ Verzeichnisse ’c’ zeichenorientierte Gerätedateien ’b’ blockorientierte Gerätedateien
-user username	Dateien des Benutzers ’username’

Die wichtigsten Aktionen sind:

-print	Namentliche Ausgabe der gefundenen Dateien auf dem Bildschirm.
-exec cmd	’cmd’ wird auf die Liste der gefundenen Dateien angewendet.

Das Ende der Kommandozeile muß mit einem ´\;´ gekennzeichnet werden. ’{}’ steht für die Liste gefundener Dateien.

Beispiele:

```
% find /home -type d -print
% find / -size +200 -print
% find / -name core -exec rm {} \;
```

7.12 Drucken von Dateien

Die Druckerverwaltung ist eines der Dinge, in der sich BSD-UNIXe und System-V-UNIXe wesentlich unterscheiden. Aus der Sicht des Benutzers sind lediglich die Kommandos etwas anders, während es für den Systemadministrator zwei Welten bedeutet.

7.12.1 BSD-UNIX

lpr

Die Erzeugung eines Druckauftrages erfolgt mittels „lpr“.

% lpr [Option(en)] Datei(en)

Wird „lpr“:

- ohne Option aufgerufen, werden die angegebenen Dateien auf dem Standarddrucker gedruckt
- mit der Option ’-Pdruckername’ aufgerufen, geht der Druckauftrag an ’druckername’
- mit der Option ’-#n’ aufgerufen, werden ’n’ Kopien jeder angegebenen Datei gedruckt

Beispiele:

```
% lpr -#3 prog1.c
% lpr -Plaser prog2.c
```

lpq

Das Kommando „lpq“ zeigt Informationen zu Druckaufträgen an, die in der Warteschlange stehen.

% lpq [Option(en)]

„lpq“ zeigt dabei:

- den Status des Druckauftrages
- den Benutzernamen des Auftraggebers
- die Nummer des Druckauftrages

- die Datei(en), die gedruckt werden sollen
- die Dateigröße in Byte

Mit der Option „-Pdruckername“ können Informationen zu der Warteschlange eines anderen Druckers ermittelt werden.

Beispiel:

```
% lpq -Pmatrix
matrix is ready and printing
RankOwnerJob FilesTotal Size
activetarzan126 prog2.c1284
```

lprm

Das Kommando „lprm“ dient dem Entfernen von Druckaufträgen aus der Warteschlange.

% lprm [Option(en)] Auftragsnummer(n)

Wird „lprm“:

- ohne Argumente aufgerufen, wird der aktuelle Druckauftrag terminiert, vorausgesetzt er gehört dem Aufrufer
- mit der Auftragsnummer als Argument aufgerufen, entfernt das Spoolersystem den entsprechenden Job

Die Option *-Pdruckername* spezifiziert den Drucker „druckername“

Beispiel:

```
% lpq -Pmatrix
matrix is ready and printing
RankOwnerJob FilesTotal Size
activetarzan126 prog2.c1284
waitingtarzan127 prog1.c266
% lprm -Pmatrix 127
```

7.12.2 System V[6]

Eigentlich sollte hier jetzt die Beschreibung von System V folgen. Doch leider - der Geist war willig, doch das Fleisch war müd. Nur soviel sei verraten: Die entsprechenden Kommandos heißen

- lp -d*Druckername*
- cancel
- lpstat

7.13 Aufgaben

Aufgabe 1: Welche Kommandos benötigt man für folgende Aufgaben? Machen Sie eine Liste für die verschiedenen Betriebssysteme, die sie kennen:

Wechseln des Verzeichnisses
Inhalt eines Verzeichnisses anzeigen
Daiteiinhalt anzeigen (Text-/Binär-Datei)
Nach Muster suchen
Dateien suchen
Dateien ausdrucken
Dateien/Verzeichnisse kopieren
Dateien umbenennen
Dateien löschen
Mehrfachverweise („links“) anlegen
Zugriffsrechte abändern
Dateien vergleichen
Verzeichnis anlegen
Sonstige Datei-Kommandos

6. Unter Solaris 2 und vielen anderen SV-Unixe funktionieren auch die BSD-Druckkommandos.

8 Dateikommandos

8.1 Dateien kopieren, umbenennen und löschen

8.1.1 Dateien kopieren

Das Kopieren von Dateien erfolgt mittels „cp“ („**c**o**p**y“). Zu unterscheiden sind 2 Formen des Kommandoaufrufes:

% **cp** [Option] Datei1 Datei2

% **cp** [Option] Datei(en) Verzeichnis

Generell gilt, daß:

- die Zieldatei erzeugt wird, falls sie noch nicht existiert
- der Inhalt einer bereits existierenden Zieldatei verloren geht, falls die Zugriffsrechte dies zulassen
- derjenige Besitzer der Kopie wird, der das Kommando ausführt
- die Zugriffsrechte mitkopiert werden, aber ggf. auf den neuen Dateibesitzer anzuwenden sind

Die 1. Form: % cp Datei1 Datei2

- dient dem Kopieren eines Dateiinhaltes in eine andere Datei

Dateien...	**Unix**	**DOS**	**Vax/VMS**
...anschauen	cat od	type	type dump
...seitenweise blättern	more, less	more <	type /page
...Kopf/Fußteil	head/tail		
...kopieren	cp	copy	copy
..durchsuchen	(e,f)grep	find	search
...vergleichen	(s)diff, cmp	comp	diff
...umbenennen	mv	ren[ame]	rename
...löschen	rm	del, erase	delete
...drucken	lp, lpr	print	print
...(de)komprimieren	(un)compress, (un)pack		(un)pack

Tabelle 5: Datei-Kommandos (Übersicht)

- erzeugt die Kopie im aktuellen Verzeichnis

Die 2. Form: % cp Datei(en) Verzeichnis

- dient dem Kopieren einer oder mehrerer Dateien in ein anderes Verzeichnis
- erzeugt namensgleiche Kopien im angegebenen Verzeichnis

Beispiele:

```
% cp prg1.c prg1.c.bak
% cp hello.c hello.h /tmp
```

Mit der Option „-i“ (interaktiv) können Sie das Überschreiben bereits existierender Zieldateien vermeiden.

Im Gegensatz zu DOS werden Wildcards von der Shell (s. Kap. *Wildcards* auf Seite 170) ersetzt und nicht vom Kommando interpretiert; d.h. Kommadoszeilen wie

```
cp *.c *.c.bak
```

funktioneren unter Unix anders!

8.1.2 Dateien umbenennen

Das Umbenennen von Dateien erfolgt mittels „mv“ („**m**o**v**e“):

% **mv** [Option] Datei1 Datei2

Datei2 muß dabei nicht im selben Verzeichnis liegen, d.h. mit diesem Kommando können auch Dateien in ein anderes Verzeichnis „verschoben“ werden, allerdings nur innerhalb einer Partition. Als Ziel für das *mv*-Kommando kann auch ein Verzeichnis angegeben werden:

% **mv** [Option] Datei(en) Verzeichnis

Beispiele:

```
% mv hello.c hello.c.bak
% mv hello ../bin
```

Mit der Option „-i“ (interaktiv) können Sie das Überschreiben bereits existierender Zieldateien vermeiden.

8.1.3 Dateien löschen

Dateien werden mit dem Kommando „rm“ („**r**e**m**ove“) gelöscht:

% **rm** [Option] Datei(en)

Als Voraussetzung für die erfolgreiche Anwendung benötigt man ein Schreibrecht im entsprechenden Verzeichnis, nicht jedoch Schreib- oder Leserecht der Datei, die gelöscht werden soll.

Beispiele:

```
% rm prg2.c
% rm ../prg/prg1 ../prg/prg2
```

Aus Sicherheitsgründen empfehlenswert ist die Option ’-i’ („**i**nteraktiv“), die eine Bestätigung jeder zu löschenden Datei verlangt.

8.2 Verzeichnisse erzeugen, löschen und umbenennen

8.2.1 Erzeugen von Verzeichnissen

Verzeichnisse werden mit dem Kommando „mkdir“ („**m**a**k**e **dir**ectory“) angelegt.

% **mkdir** Verzeichnis(se)

mkdir:

- erfordert Schreibrecht im aktuellen Verzeichnis
- legt automatisch die Einträge ’.’ und ’..’ im neuen Verzeichnis an

Beispiele:

```
% mkdir test
% mkdir ~/faxen ~/privat
```

8.2.2 Löschen von Verzeichnissen

Verzeichnisse werden mit dem Kommando „rmdir“ („**r**e**m**ove **dir**ectory“) gelöscht.

Verzeichnisse...	**Unix**	**DOS**	**Vax/VMS**
...auflisten	ls ls -l	dir /w dir	directory directory /full
...erzeugen	mkdir [-p]	mkdir, md	create /dir
...kopieren	cp -r		
...umbenennen	mv		
...löschen	rmdir, rm -r	rmdir	delete
...wechseln	cd	cd	set default
...anzeigen	pwd		show default

Tabelle 6: Verzeichnis-Kommandos (Übersicht)

% **rmdir** Verzeichnis(se)

Die als Argument angegebenen Verzeichnisse werden gelöscht, falls:

- sie leer sind
- im übergeordneten Verzeichnis Schreibrecht existiert

Beispiele:

```
% rmdir test
% rmdir ~/faxen ~/privat
```

Hat das zu löschende Verzeichnis Datei- und/oder Unterverzeichniseinträge, ist es mit dem Kommando 'rm -r' („**re**m**o**ve **r**ecursive") zu löschen. Der gesamte Baum unter dem angegebenen Verzeichnis wird dabei gelöscht!

% **rm** -r Verzeichnis(se)

Das *rm*-Kommando ist mit Vorsicht zu genießen - was weg ist, ist weg!!! Anders als bei DOS gibt es keine Möglichkeit mehr, an die gelöschten Dateien zu kommen. Man kann dann nur darauf hoffen, daß von der Administration regelmäßig ein Backup gemacht wurde, daß das Backup nicht zu alt ist und Mr. Root bereit ist, das Backup demnächst einzuspielen.

Beispiel:

```
% pwd
/home/tarzan/entwickl
% rm -r prg
```

8.2.3 Umbenennen und Kopieren von Verzeichnissen

Das Umbenennen und Kopieren von Verzeichnissen ist mit den bereits bekannten Kommandos „mv" und „cp" zu realisieren.

Verzeichnisse können umbenannt werden mit:

% **mv** Verzeichnis1 Verzeichnis2

Rekursives Kopieren eines Verzeichnisses ist ebenfalls möglich:

% **cp -r** Verzeichnis1 Verzeichnis2

8.3 Links

Ein **Link** ist ein zusätzlicher Name für eine bereits existierende Datei oder ein existierendes Verzeichnis. Gegenüber einer Kopie hat ein Link den Vorteil, daß:

- die zur Speicherung des Datei- bzw. Verzeichnisinhaltes notwendigen Datenblöcke nur einmal physikalisch vorhanden sind
- es keine Inkonsistenzen zwischen Original und Link geben kann

Zu unterscheiden sind:

- **Hard Links**
- **Soft Links** bzw. **Symbolic Links**

8.3.1 Hard Links

Bei Aufruf des Kommandos „ls -l“ sehen Sie rechts der Zugriffsrechte den sogenannten Linkcount, die Anzahl Links, die auf diese Datei existieren.

```
% ls -l
-rw-r--r-- 1 tarzan  1284 May 12 12:13 prg1.c
-rw-r--r-- 1 tarzan  2566 May 23 09:24 prg2.c
```

Informationen, die durch „ls“ ausgegeben werden, sind in einer sogenannten Inode gespeichert. Dies sind u.a.:

- der Dateityp
- die Zugriffsrechte
- der Linkcount
- der Dateibesitzer
- die Dateigröße
- das Modifikationsdatum

Dateien/Verzeichnisse...	**Unix**	**DOS**	**Vax/VMS**
...suchen	find		dir
Mehrfachverweise	ln		
...Zugriffsrechte ändern	chmod	attrib	set file /prot
...Eigentümer ändern	chown	-	set file /owner

Tabelle 7: Datei/Verzeichniss-Kommandos (Übersicht)

Eine **Inode**:

- beschreibt die Datei
- wird bei Anlegen einer neuen Datei automatisch erzeugt
- enthält neben der Verwaltungsinformation auch Verweise auf die Datenblöcke der Datei

Die Verbindung von Dateiname zu Inode erfolgt über den Verzeichniseintrag. Dieser besteht aus:

- dem Dateinamen
- einer Inumber, der Nummer der entsprechenden Inode

Erzeugt wird ein Hard Link durch das Kommando „ln“.

% **ln** Originaldatei Linkdatei

Zu beachten ist, daß hierbei:

- ein neuer Eintrag im aktuellen Verzeichnis erzeugt wird
- keine neue Datei erzeugt wird
- Original und Link gleichwertig sind
- die Datei erst dann physikalisch gelöscht wird, wenn der Linkcount Null ist

Beispiel:

```
% ln prg1.c prglink.c
```

Nicht möglich ist:

- das Erzeugen von Hard Links über Partitionsgrenzen hinweg, da Inodes nur innerhalb einer Partition eindeutig sind
- die Anwendung von Hard Links auf Verzeichnisse

8.3.2 Soft Links (Symbolic Links)

Ein symbolischer Link:

- ist eine Datei mit eigener Inode, die in ihren Datenblöcken den Pfadnamen einer anderen Datei enthält
- ist ein eigener Dateityp

- hebt obige Beschränkungen für Hard Links auf, d.h. symbolische Links lassen sich auch über Partitionsgrenzen und auf Verzeichnisse einrichten.

Erzeugt wird ein symbolischer Link durch „ln -s“.

% **ln -s** Originaldatei Linkdatei

Ein so eingerichteter symbolischer Link wirkt bei Standardkommandos transparent, d.h. bei Zugriff auf den symbolischen Link, wird tatsächlich auf die damit verbundene Datei zugegriffen.

Verwendung finden symbolische Links vorwiegend bei Verzeichnissen, die bei den beiden großen UNIX-Linien traditionell an anderen Stellen im Dateisystem stehen, um so ein bestimmtes Maß an Übereinstimmung zu gewährleisten.

8.4 Zugriffsschutz

Zugriffsrechte:

- legen fest, welche Rechte ein Benutzer hinsichtlich einer Datei hat
- sind für drei Kategorien festgelegt
- für eine bestimmte Datei lassen sich mit „ls -l“ ermitteln

```
% ls -l
-rw-r--r-- 1 tarzan  1284 May 12 12:13 prg1.c
-rw-r--r-- 1 tarzan  2566 May 23 09:24 prg2.c
```

Die drei Kategorien sind:

`rwx------`	Besitzer der Datei
`---rwx---`	Gruppe von Benutzern mit besonderen Rechten
`------rwx`	alle anderen Benutzer

Es werden drei Rechte unterschieden:

`r--`	Leserecht (*r*ead)
`-w-`	Schreibrecht (*w*rite)
`--x`	Ausführungsrecht (e*x*ecute)

8.4.1 Zugriffsrechte für Dateien

Das Leserecht auf eine Datei erlaubt das:

- Anschauen des Dateiinhaltes
- Kopieren der Datei in das eigene Verzeichnis

Das Schreibrecht auf eine Datei ermöglicht deren Modifikation.

Das Ausführungsrecht auf eine Datei:

- erlaubt diese zu starten
- setzt voraus, daß es sich um eine Programmdatei oder um ein Shell-Skript handelt

8.4.2 Zugriffsrechte für ein Verzeichnis

Das Leserecht auf ein Verzeichnis erlaubt die Auflistung des Inhaltes mit „ls“.

Das Schreibrecht auf ein Verzeichnis ermöglicht das:

- Anlegen von Dateien in diesem Verzeichnis
- Ändern von Dateien in diesem Verzeichnis
- Löschen von Dateien in diesem Verzeichnis

Das Ausführungsrecht auf ein Verzeichnis ist Voraussetzung:

- um es mit ’cd’ zum aktuellen Verzeichnis zu machen
- für das Anlegen und Löschen von Dateien
- um daraus Dateien zu kopieren

8.5 Zugriffsrechte ändern

Zugriffsrechte werden mit ’chmod’ geändert. Zu unterscheiden sind zwei Formen des Kommandoaufrufes:

% **chmod** WerWieWas Datei(en)

% **chmod** Modus Datei(en)

chmod:

- verändert die Zugriffsrechte der angegeben Datei(en) oder Verzeichnis(se)
- setzt voraus, daß der Benutzer Eigentümer der im 2. Parameter angegebenen Datei ist, oder Superuser

Die symbolische Form „*chmod WerWieWas ...*“ wird i.d.R. verwendet, wenn ein Rechte hinzugefügt oder weggenommen werden sollen. Welche Rechte für wen hinzugefügt oder weggenommen werden sollen, wird suexh das erste Argument „*WerWieWas*“ bestimmt:

Wer Wie Was	Der erste Buchstabe gibt an, *wer* Rechte geschenkt oder entzogen bekommt:
u	**u**ser (Benutzer)
g	**g**roup (Gruppe)
o	**o**thers (alle anderen)
a	**a**lle drei Kategorien

Wer **Wie** Was	Der zweite Buchstabe gibt an, *wie* die Rechte vergeben werden:
+	Recht zu bestehenden hinzufügen
-	Recht von bestehenden wegnehmen
=	Recht absolut setzen

Wer Wie **Was**	Der dritte Buchstabe gibt an, *was* gesetzt oder entzogen wird:
r	**r**ead (Leserecht)
w	**w**rite (Schreibrecht)
x	e**x**ecute (Ausführungsrecht=

Es ist möglich, mehrere Modusangaben durch Kommata zu trennen.

Beispiele:

```
% ls -l prg1.c
-rw-r--r-- 1 tarzan  1284 May 12 12:13 prg1.c
% chmod a+w prg1.c
% ls -l prg1.c
```

```
-rw-rw-rw- 1 tarzan  1284 May 12 12:13 prg1.c
% chmod go-w,o-r prg1.c
% ls -l prg1.c
-rw-r----- 1 tarzan  1284 May 12 12:13 prg1.c
```

Die absolute Form:

- setzt alle Zugriffsrechte neu, unabhängig, wie sie zuvor gesetzt waren
- gibt die Zugriffsrechte für jede Kategorie in Form einer Oktalzahl an

Die Zugriffsrechte ergeben sich dann aus der Addition der Einzelberechtigungen:

User	Group	Others
rw-	rw-	r--
421	421	421
6	6	4

Tabelle 8: Zugriffsrechte

Beispiele:

```
% ls -l prg1.c
-rw-r--r-- 1 tarzan1284 May 12 12:13 prg1.c
% chmod 666 prg1.c
% ls -l prg1.c
-rw-rw-rw- 1 tarzan1284 May 12 12:13 prg1.c

% chmod 640 prg1.c
% ls -l prg1.c
-rw-r----- 1 tarzan1284 May 12 12:13 prg1.c
```

8.5.1 Default Zugriffsrechte

Für neu zu erzeugende Dateien und Verzeichnisse werden Standardzugriffsrechte vergeben. Dazu dient das Kommando ’umask’.

% **umask** [Rechte]

Wird ’umask’:

- ohne Argumente aufgerufen, wird die aktuelle Dateierzeugungsmaske ausgegeben
- mit Argument aufgerufen, wird die Dateierzeugungsmaske gesetzt / modifiziert

Die Dateierzeugungsmake:

- ist ein 9 Bit Wert
- muß bei 'umask' immer als 3-stellige Oktalzahl angegeben werden
- legt die Rechte fest, die beim Erzeugen einer neuen Datei, oder eines neuen Verzeichnisses, auf keinen Fall gewährt werden dürfen
- hat nur Einfluß auf neu zu erzeugende Dateien
- hat keine Auswirkungen auf „mv“, da dieses Kommando das Zugriffsrecht unverändert läßt

Das Argument von „umask“ setzt sich aus „ugo“ (user, group, others) zusammen.

Beispiele:

```
% umask 022
% umask 077
```

Das Ausführungsrecht wird für reguläre Dateien generell nicht gesetzt. Die resultierenden Zugriffsrechte ergeben sich wie folgt:

	Verzeichnisse	Dateien
	777	666
umask	- nnn	- nnn
Zugriffsrechte	yyy	yyy

Tabelle 9: umask-Verknüpfung

Für die beiden Beispiele ergeben sich also folgende tatsächliche Zugriffsrechte:

umask 022 Verzeichnisse: 777 - 022 = 755 Dateien: 666 - 022 = 644
umask 077 Verzeichnisse: 777 - 077 = 700 Dateien: 666 - 077 = 600

Der langen Rede kurzer Sinn: mit *umask* legt man die Rechte fest, die auf gar keinen Fall vergeben werden sollen.

8.6 Aufgaben

Aufgabe 2: Welche Größe hat die Datei „/etc/magic“?

Aufgabe 3: Wieveile Zeilen enthält die Datei „/usr/include/stdio.h“?

Aufgabe 4: Was macht das „sort“-Kommando?

Aufgabe 5: Unter dem Verzeichnis „/usr/include“ gibt es irgenwo die Datei „types.h“. Aber wo?

Aufgabe 6: Was machen die Kommando *uuencode* und *uudecode*? Probieren Sie doch einach mal

```
uuencode /usr/ucb/yes jajaja > blabla
```

aus (statt „/usr/ucb/yes“ können sie auch jede andere Datei Ihrer Wahl verwenden).

Aufgabe 7: Was passiert, wenn ich das Kommando

```
split -20 textfile
```

absetze? (einfach ausprobieren und schauen, was angelegt wird).

9 Schlag nach bei UNIX

9.1 Dokumentation

9.1.1 Handbücher

Werden zum System Handbücher mit ausgeliefert (was immer seltener der Fall ist), sind die Handbücher oft nach folgendem Schema aufgeteilt:

- UNIX Release & Install
- System Administration Manual
- User's Guide
- Reference Manual
- Programmer's Guide

Als Einstieg bietet sich am ehesten das *User´s Guide* an. Falls man Glück hat, ist es sogar irgendwo greifbar in der Nähe. Falls nicht, macht auch nichts - Sie haben ja diese Kursunterlagen (na ja).

9.1.2 CD-ROM

Dieses Medium verdrängt zunehmend die konventionelle Handbücher, da sie billiger herzustellen sind und einfacher ausgeliefert werden können. Handbücher gibt es oft nur auf ausdrücklichen Wunsch und gegen Bares.

Zur Bedienung der CD-ROM liefert der Hersteller die entsprechende Software mit aus, womit man dann in den elektronischen Handbücher nachschlagen bzw. ausdrucken kann. Für den Anwender hat es den Vorteil, daß die Suche nach den Handbücher entfällt („User´s Guide? Ach ja, das hat im Moment die Frau Müller von der Buchhaltung. Ach nein, das liegt gerade beim Koch - der wollt´ was über irgendwelche Menüs nachlesen...“). Außerdem kann man sich nur die Seiten ausdrucken lassen, die man gerade benötigt.

9.2 Online Hilfe

Jedes UNIX-System wird standardmäßig mit Online-Manuals ausgeliefert, die mit dem **man**-Kommando angeschaut werden können. Die Manuals sind in verschiedenen „Sections“ aufgeteilt:

9.2.1 Benutzerkommandos(1)

Dieses Kapitel enthält alle wichtigen UNIX-Kommandos, die der gewöhnliche Benutzer im täglichen Umgang mit dem System benötigt.

9.2.2 Systemaufrufe(2)

Hier findet der Systemprogrammierer die Beschreibungen aller unter UNIX verfügbaren System Calls.

9.2.3 Bibliotheksfunktionen(3)

Hier findet der Anwendungsprogrammierer die Beschreibungen aller Funktionen der Standard C-Bibliothek.

9.2.4 Geräte und Netzwerk(4)

Hier werden sowohl Gerätetreiber und -charakteristika, als auch die Schnittstellen zur Netzwerkprogrammierung erläutert.

UNIX-Manual

aufgeteilt in *Sections*:

1 Kommandos
2 C-Library (System-Aufruf)
3 C-Library
4 Devices und Netzwerk
5 Files
6 Games
7 Verschiedenes
8 Administration
l local

Abbildung 16: UNIX-Manual

9.2.5 Dateiformate(5)

Diese Sektion enthält die Beschreibung der Formate und des Aufbaus einzelner Systemdateien.

9.2.6 Spiele und Demos(6)

Im Lieferumfang von UNIX befinden sich auch Spiele und Demos. Zwar werden sie meistens vom Systemadminstrator gelöscht, zum Trost verbleiben die Manual-Seiten im System.

9.2.7 Verschiedenes(7)

Diese Sektion enthält all das, was keiner anderen Sektion zugeordnet werden kann. Dazu gehören eine Reihe von Tabellen, Makros zur Textformatierung und Umgebungsvariablen.

9.2.8 Administration(8)

Die Sektion enthält u.a. die Beschreibung von Kommandos zur Sicherung von Datenbeständen, zur Fehlersuche und zum Hoch- und Herunterfahren des Systems.

9.2.9 lokal(l)

Diese Sektion ist für lokale Kommandos reserviert.

Informationen zu einem bestimmten Kommando erhalten Sie durch das man-Kommando.

9.2.10 man

% man [-k Schlüsselwort] [-s[1] Sektion] [Name]

man:

---

1. Auf manchen UNIXen funktioniert das ganze auch nur durch Angabe der Sektion ohne „-s“ - am besten einfach ausprobieren oder mit *man man* nachschlagen.

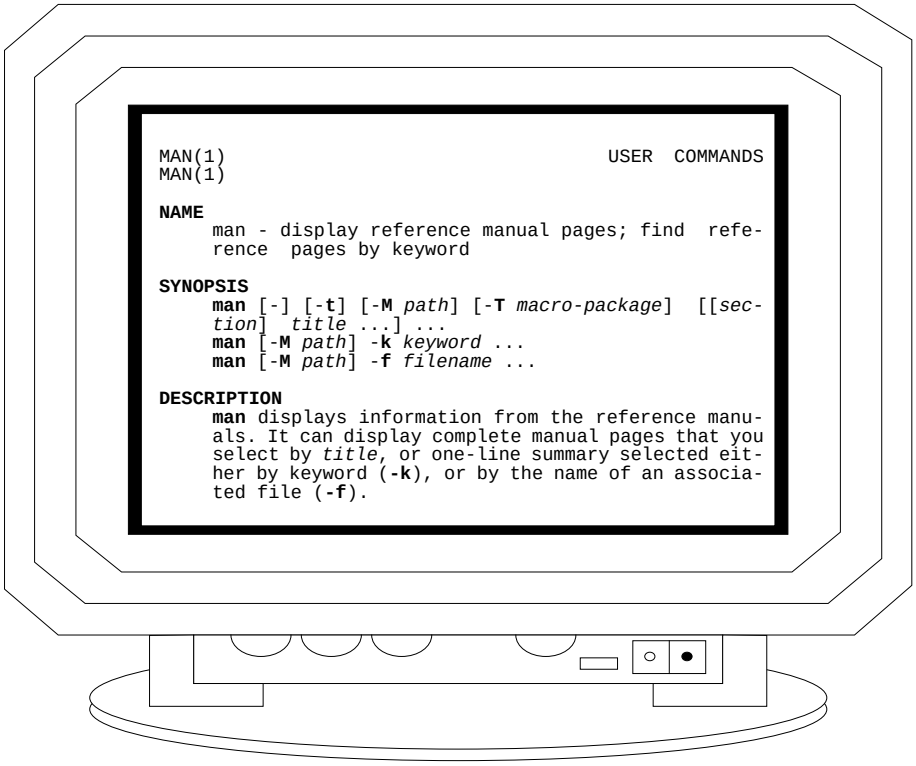


Abbildung 17: das *man*-Kommando

- schreibt die angeforderten Manual-Seiten auf den Bildschirm zur seitenweisen Ansicht
- wird zumeist mit dem Namen eines Kommandos als Argument aufgerufen
- kann optional die Sektion des Namens mitgegeben werden

Beispiele:

```
% man man
% man date
% man -s 1 passwd
% man -s 4 passwd
```

Wissen Sie gerade das Kommando nicht, was zur Durchführung Ihrer Aufgabenstellung notwendig ist, können Sie *man* auch mit der Option „—k“ und einem Schlüsselwort aufrufen. Dies führt zur Auflistung all der Zeilen des Inhaltsverzeichnisses, die das angegebene Schlüsselwort enthalten.

Beispiele:

```
% man -k password
% man -k “working directory“
```

9.2.11 Aufbau

Eine typische Beschreibung enthält folgende Abschnitte (s.a. Anhang B: „man-Beispiel” auf Seite 337):

Name	Kommandoname und Kurzbeschreibung
Synopsis	Kurzüberblick über die Syntax nebst Optionen und Argumenten
Description	Ausführliche Beschreibung des Kommandos / der Datei
Option	Beschreibung der Optionen
Note	Zusatzinfo, wie „nicht standardisiert“ ...
See Also	Querverweise auf verwandte Kommandos
Files	Dateien, die von Kommando benutzt bzw.

KOMMANDO(#) **SECTION** **KOMMANDO(#)**

NAME
Name des Kommandos - kurze Erklärung
SYNOPSIS
knappe Zusammenfassung
DESCRIPTION
Beschreibung
OPTION
Beschreibung der Optionen
FILES
Dateien, auf die das Kommando zugreift
SEE ALSO
Referenzen auf anderen Manuals zu verwandten Kommandos
DIAGNOSTICS
besondere Hinweise und evtl. auftretende Fehlermeldungen
BUGS
bekannte Schwächen oder Fehler

SVR4 **Last change: 31.08.2000** **1**

Abbildung 18: Manual-Abschnitte

	modifiziert werden
Diagnostics	Beschreibung möglicher Fehlermeldungen und Fehlerückgabewerte
Bugs	Bedingungen, unter denen das Kommando nicht richtig funktioniert
Warnings	Hinweise auf mögliche Fehlerquellen
Author	Autor

9.3 Aufgaben

Sie sollten jetzt in der Lage sein, sich selbst helfen zu können, falls sie ein bestimmtes Kommando suchen oder eine bestimmte Aufgabe lösen wollen.

Aufgabe 8: Mit welchen Kommandos kann man die Plattenbelegung und den Plattenbedarf eines Verzeichnisbaumes feststellen?

Aufgabe 9: Was ist der „vi“? Welches sind (Ihrer Meinung nach) die 10 wichtigsten „vi“-Kommandos?

Aufgabe 10: Wie drucke ich aus? Wie kann ich die Drucker-Queue anschauen bzw. manipulieren?

Aufgabe 11: Mit welchen Kommandos kann man feststellen, ob zwei Dateien unterschiedlich sind?
Wie kann man sich die Unterschiede ausgeben lassen?

Aufgabe 12: Das „diff“-Kommando kann Unterschiede auch im „ed“-Format ausgeben. Was bedeutet das bzw. was verbirgt sich hinter „ed“? Was bedeuten die einzelnen Anweisungen, die bei der „diff“-Ausgabe erscheinen?

Aufgabe 13: Was machen die einzelnen Befehle im UNIX-ABC (s. Abbildung 19:: „Das Unix-ABC” auf Seite 128)?[2]

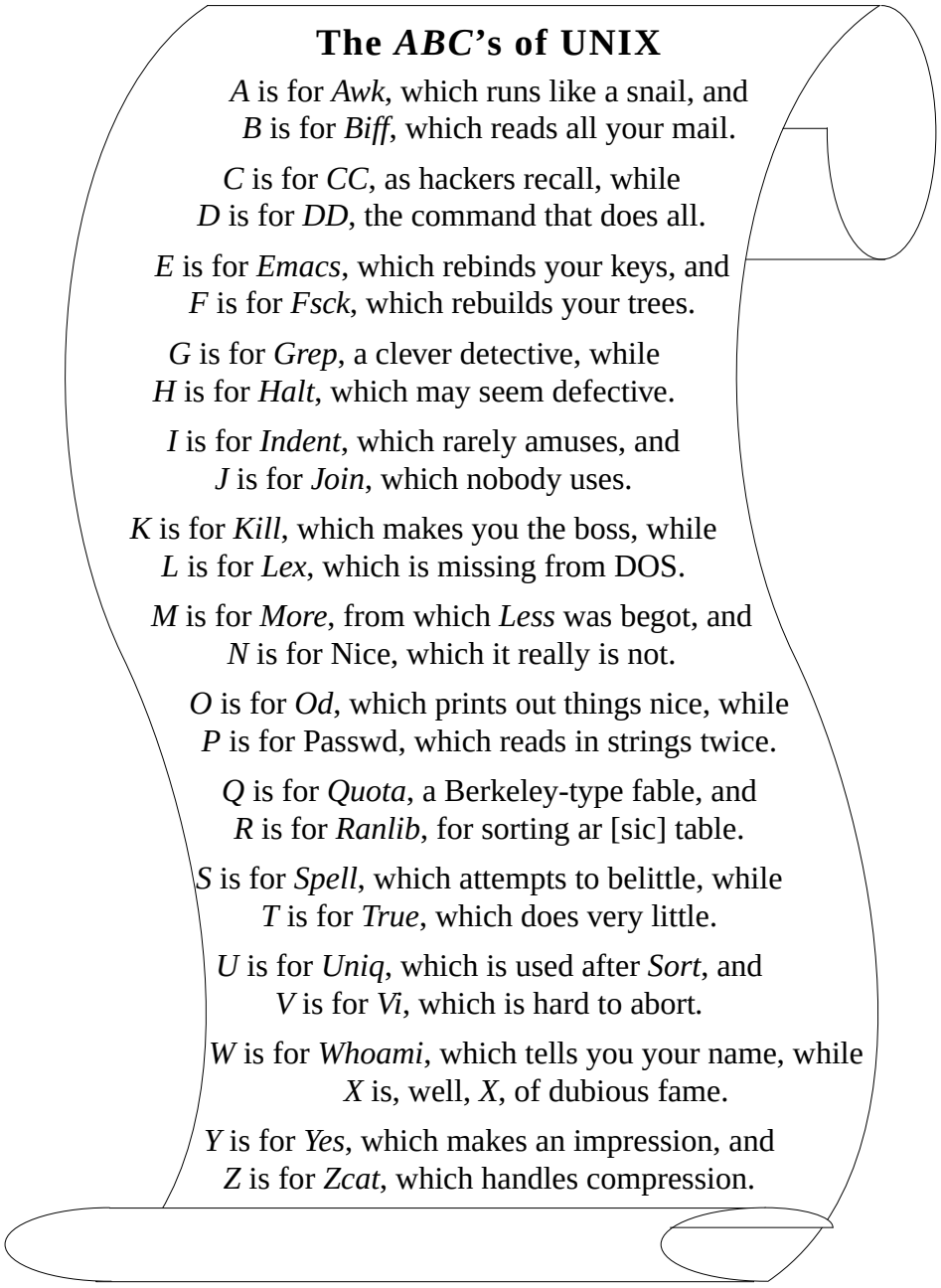
Aufgabe 14: Welches sind Ihrer Meinung nach die 10 wichtigsten Unix-Kommandos?

Aufgabe 15: Wie eröffnet man eine Verbindung via „FTP“ zu einem anderen Rechner (z.B. „slsa02t“). Welches sind die (10) wichtigsten FTP-Kommandos?

Aufgabe 16: Stellen Sie fest, ob „rsync“ auf Ihrem System vorhanden ist.

---

2. Nicht alle Befehle im Unix-ABC sind auf allen Unix*en* vorhanden. Stellen Sie fest, welche von den erwähnten Kommandos auf Ihrem System vorhanden sind.



The *ABC*'s of UNIX

A is for *Awk*, which runs like a snail, and
B is for *Biff*, which reads all your mail.

C is for *CC*, as hackers recall, while
D is for *DD*, the command that does all.

E is for *Emacs*, which rebinds your keys, and
F is for *Fsck*, which rebuilds your trees.

G is for *Grep*, a clever detective, while
H is for *Halt*, which may seem defective.

I is for *Indent*, which rarely amuses, and
J is for *Join*, which nobody uses.

K is for *Kill*, which makes you the boss, while
L is for *Lex*, which is missing from DOS.

M is for *More*, from which *Less* was begot, and
N is for Nice, which it really is not.

O is for *Od*, which prints out things nice, while
P is for Passwd, which reads in strings twice.

Q is for *Quota*, a Berkeley-type fable, and
R is for *Ranlib*, for sorting ar [sic] table.

S is for *Spell*, which attempts to belittle, while
T is for *True*, which does very little.

U is for *Uniq*, which is used after *Sort*, and
V is for *Vi*, which is hard to abort.

W is for *Whoami*, which tells you your name, while
X is, well, *X*, of dubious fame.

Y is for *Yes*, which makes an impression, and
Z is for *Zcat*, which handles compression.

Abbildung 19: Das Unix-ABC

10 Noch mehr Kommandos

Dieses Kapitel befindet sich noch in Vorbereitung. Anregungen und Beiträge werden dankend gern entgegengenommen.

10.1 Die 10 wichtigsten Kommandos

login
logout
man
cd
more
cp
rm
mail
who
lpr

Kommando	**Unix**	**DOS**	**Vax/VMS**
Hilfe!	man	help	help
Passwort ändern	passwd	-	set password
Wer da?	who, w	-	show users
„Small Talk“	talk	-	phone
Prozesse auflisten	ps	-	show process
Prozesse anhalten	kill		stop
Version	uname	ver	?
Druckerqueue	lpq / lpstat	print	show queue
Terminal einstellen	stty		set terminal
Archivieren	tar, cpio, dd	archive, restore	
Datum anzeigen	date	date, time	
Plattenbelegung	df	chkdsk	

Tabelle 10: Noch mehr Kommandos...

10.2 Programmierumgebung

10.2.1 Compiler

cc

fc / f77

pc

gcc

10.2.2 Debugger

adb, kdb

sdb

dbx

gdb

10.2.3 Tuning

prof

gprof

10.2.4 Code-Generatoren

lex

yacc

flex

bison

10.2.5 Hilfsmittel

make

lint

SCCS, RCS
indent
cb
cxref
ctags

10.2.6 Dump-Utilities

od
xd

11 Der Editor „vi“

Der „**vi**“:

- ist ein bildschirmorientierter Editor
- kann auf jeder Art von Terminal betrieben werden
- arbeitet immer an einer Kopie, nicht am Original
- ist der Standardeditor auf UNIX-basierten Systemen

Der *vi* kann aufgerufen werden:

- mit der Angabe einer existierenden Datei
- mit Angabe einer nicht existierenden Datei, diese wird dann angelegt
- ohne Angabe einer Datei, ein Dateiname muß dann bei Verlassen des Editors angegeben werden
- mit der Angabe mehrerer Dateien, wobei diese dann sukzessive abgearbeitet werden.

Wird der *vi* aus einem Command-Tool heraus gestartet, darf die Fenstergröße nachträglich nicht mehr verändert werden, da dies zu Fehldarstellungen führt.

Geschichte

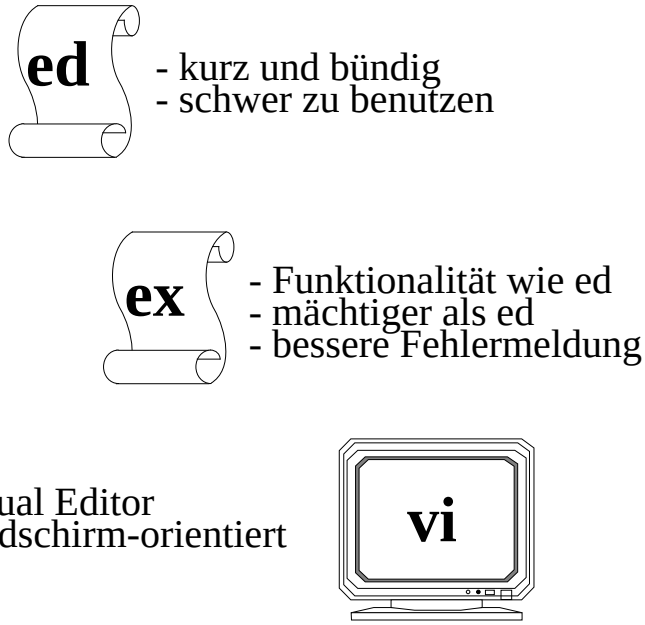


Abbildung 20: Geschichte des *vi*

„view“:

- ist die „read only“ Version des vi-Editors
- kennt die vi-Kommandos
- schreibt kein Veränderungen in die Datei

Beispiele:

```
% vi Dat_exist
% vi Dat_nonexist
% vi
% vi Dat1 Dat2
```

11.1 Modi des vi - Editors

Der vi kennt drei Modi:

- Kommando-Modus
- Last-Line- oder Ex-Modus
- Eingabe-Modus

Wurde eine Editorsitzung ohne Speichern abgebrochen, so kann in der Regel über eine Recovery-Datei (Option ’-r’) die letzte Editorsitzung vom Rechner nochmals nachvollzogen werden.

Aufruf:

```
% vi -r text.dat
```

11.1.1 Kommandomodus

Der Kommandomodus:

- ist der aktuelle Modus, wenn ’vi’ aufgerufen wird
- dient der Eingabe von Editierkommandos. Diese sind am Bildschirm nicht sichtbar!

Kommandos bestehen i.d.R. aus einzelnen Buchstaben und haben häufig folgende Syntax:

[Anzahl] Kommando [Object]

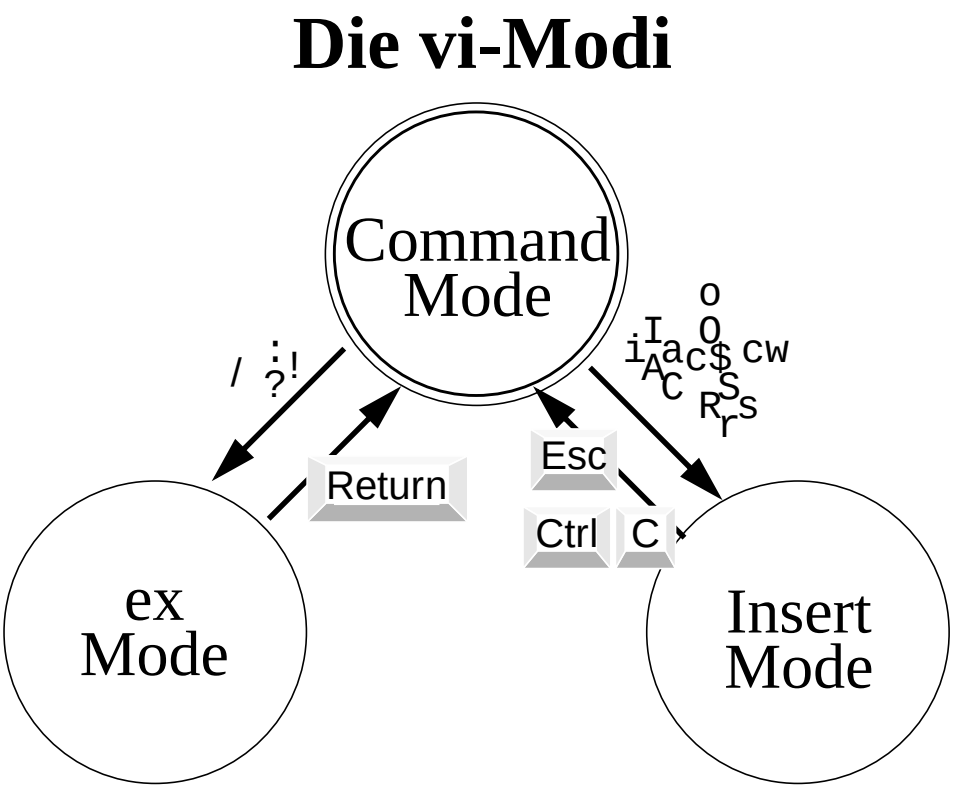


Abbildung 21: vi-Modi

„Anzahl“ bedeutet hierbei, wie oft das Kommando ausgeführt werden soll.

„Objekt“ spezifiziert, worauf operiert werden soll.

11.1.2 Eingabemodus

Wollen Sie Text eingeben, müssen Sie zunächst in den Eingabemodus wechseln (s. Kap. *Texteingabe* auf Seite 140).

11.1.3 Last Line Mode

Der Last Line Mode:

- erreicht man durch Eingabe des Zeichens ’:’, ’/’, ’?’ oder ’!’
- wird mit <Esc> oder <Return> beendet und mit <Ctrl>-C abgebrochen
- entspricht dem zeilenorientierten Editor ’ex’
- dient in erster Linie dem Absetzen von Kommandos für Dateioperationen wie Laden, oder Speichern einer Datei
- erlaubt zusätzlich das Absetzen von Shell-Kommandos, ohne ’vi’ zu verlassen

Verlassen wird ’vi’ im ’Last Line Mode’ durch Eingabe von:

:q	ohne den Inhalt abzuspeichern, da er seit dem letzten Speichern nicht mehr verändert wurde
:q!	ohne den Inhalt abzuspeichern, obwohl er von Ihnen modifiziert wurde
:wq	mit Abspeichern des Inhaltes (auch mit ZZ möglich)

Zwischenzeitliches Speichern ohne Verlassen des ’vi’ erfolgt durch „:w“ im ’Last Line Mode’.

Ein kleines Beispiel:

```
% vi beispiel.dat
i
```

```
Dies ist ein Test.
<ESC>
:wq
% cat beispiel.dat
Dies ist ein Test.
%
```

Weitere Kommandos des ’vi’ im ’Last Line Mode’:

:w ’filename’	Abspeichern des Inhaltes unter der Datei ’filename’
:r ’filename’	Einfügen des Inhaltes der Datei ’filename’ unterhalb der aktuellen Zeile
:e ’filename’	wechseln in die Datei ’filename’

Beispiel:

```
:1,5 w teil.doc
```

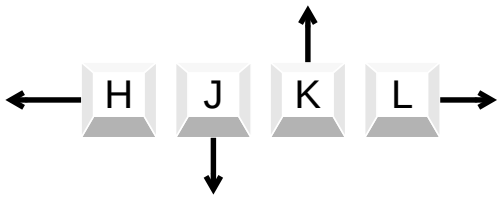
11.2 Bewegen des Cursors

Kommandos zur Cursorsteuerung:

- müssen im Kommandomodus ausgeführt werden
- funktionieren nur über bereits existierendem Text

Zum Bewegen des Cursors dienen grundsätzlich:

- die Pfeiltasten[1]
- Carriage Return
- Backspace
- die Leertaste
- die Tasten H, J, K, L



1. leider nicht immer

11.2.1 Cursorpositionierung innerhalb der aktuellen Zeile

Die wichtigsten Kommandos zur Positionierung des Cursors innerhalb einer Zeile sind:

w	Cursor auf den Anfang des nächsten Wortes
b	Cursor auf den Anfang des letzten Wortes
e	Cursor auf das Ende des nächsten Wortes
f'char'	Cursor auf das nächsten Auftreten des Zeichens 'char' in der aktuellen Zeile setzen

Diesen Kommandos kann eine 'Anzahl' vorangestellt werden, was bewirkt, daß der Cursor um 'Anzahl' Worte bewegt wird.

Weitere Kommandos sind:

$	Cursor an das Zeilenende
^	Cursor auf erstes sichtbares Zeichen der Zeile
0	Cursor auf die erste Spalte der Zeile

11.2.2 Zeilenweise Cursorpositionierung

Die wichtigsten Kommandos zur vertikalen Cursorpositionierung sind:

G	Cursor auf die letzte Zeile der Datei
nG	Cursor auf die n. Zeile der Datei
Ctrl Y	eine Zeile zurück
Ctrl E	eine Zeile vorwärts
H	Cursor auf die erste momentan sichtbare Zeile
M	Cursor auf die mittlere momentan sichtbare Zeile
L	Cursor auf die letzte momentan sichtbare Zeile

Um den Cursor auf die n. Zeile positionieren zu können ist die Zeilennummer zu ermittlen. Diese erhalten Sie durch Ctrl G.

Für das Scrollen des Bildschirmes stehen zur Verfügung:

Ctrl F	eine Seite vorwärts (*forward*)
Ctrl B	eine Seite zurück (*backward*)

11.3 Texteingabe

Die Möglichkeiten, in den Eingabemodus zu gelangen, hängen von der Position ab, an der neuer Text eingefuegt werden soll. DerText kann beliebig lang sein. Mit Esc kommen Sie wieder in den Kommandomodus.

Mit den folgenden Kommandos gelangen Sie in den Eingabemodus. Der im Anschluß einzugebende Text wird:

i	vor der aktuellen Cursorposition eingefügt
I	am Anfang der aktuellen Zeile eingefügt
a	nach der aktuellen Cursorposition eingefügt
A	am Ende der aktuellen Zeile eingefügt
o	in einer neuen Zeile unter der aktuellen Zeile eingefügt
O	in einer neuen Zeile oberhalb der aktuellen Zeile eingefügt

11.4 Löschen und überschreiben von Text

11.4.1 Löschen von Text

Die Löschkommandos sind ein gutes Beispiel für die bereits erwähnte allgemeine Kommandosyntax „Kommando[Objekt]“. Das grundlegende Kommando ist ’d’, die folgenden Variationen dienen dem Löschen:

d<SPACE>	des aktuellen Zeichens (Alternative: x)
dw	des aktuellen Wortes

d0	von der aktuellen Cursorposition bis zum Zeilenanfang
d$	von der aktuellen Cursorposition bis zum Zeilenende
dd	der aktuellen Zeile

Bezieht sich das Kommando auf die aktuelle Zeile, ist einfach das Kommandozeichen zu wiederholen.

Den Kommandos ’dw’ und ’dd’ kann eine ’Anzahl’ vorangestellt werden.

11.4.2 Überschreiben von Text

Dasselbe gilt für das überschreiben bzw. Auswechseln von Text. Das grundlegende Kommando ist ’c’, die folgenden Variationen dienen dem überschreiben:

c<Space>	des aktuellen Zeichens (Alternative: r)
cw	des aktuellen Wortes
c0	vom Zeilenanfang bis zur aktuellen Cursorposition
c$	von der aktuellen Cursorposition bis zum Zeilenende
cc	der aktuellen Zeile
s’string’	das aktuelle Zeichen durch einen beliebigen ’string’ ersetzen
~	Groß-/Kleinschreibung des aktuellen Zeichens ändern

Soll generell in den Überschreibmodus geschaltet werden, geben Sie ’R’ ein. Durch <ESC> gelangen Sie wieder in den Kommandomodus.

11.5 Arbeiten mit Textbereichen

Folgenden Kommandos dienen dazu, Textbereiche zu:

- kopieren
- verschieben

Das zugrundeliegende Prinzip orientiert sich an „Copy und Paste“, d.h. ’vi’ stellt einen Puffer zur Verfügung:

- in den zuerst ein Textbereich kopiert wird
- der anschließend an einer anderen Position wieder eingefügt wird

11.5.1 Textbereich in Puffer kopieren

Das Kopieren eines Textbereiches in den Puffer erfolgt mit dem Kommando ’y’. Folgende Variationen dienen dem Kopieren:

y<Space>	des aktuellen Zeichens
yw	des aktuellen Wortes
y0	von der aktuellen Cursorposition bis zum Zeilenanfang
y$	von der aktuellen Cursorposition bis zum Zeilenende
yy	der aktuellen Zeile

Besonders häufig verwendet wird „[Anzahl]yy“, um ’Anzahl’ Zeilen in den Puffer zu kopieren. Das Zählen der Zeilen kann bei einem großen Absatz sehr lästig sein. Eine Alternative bietet das Setzen einer Marke. Hierzu:

- positionieren Sie den Cursor an das Ende des zu kopierenden Abschnittes
- und markieren die Position mit ’ma’, wobei ’a’ hier nur eine Bezeichnung der Position darstellt - jedes andere Zeichen wäre ebenso zulässig

Um den Textbereich in den Puffer zu kopieren:

- positionieren Sie den Cursor wieder auf den Beginn des Absatzes

- und kopieren ihn mit y'a in den Puffer. Das Hochkomma ist hier notwendig, um die Sonderbedeutung von 'a' auszuschalten

11.5.2 Pufferinhalt in Text einfügen

Der Pufferinhalt wird mit:

p	hinter der aktuellen Cursorposition eingefügt
P	vor der aktuellen Cursorposition eingefügt

Generell ist zum Kopieren von Textbereichen anzumerken, daß

- eine Reihe von vi-Kommandos den Puffer ebenfalls benutzen
- das Einfügen des gepufferten Textbereiches unmittelbar nach dem Füllen des Puffers erfolgen sollte, will man nicht Gefahr laufen, daß der Pufferinhalt zwischenzeitlich ungewollt modifiziert wird

11.5.3 Textbereich verschieben

Im Unterschied zum Kopieren ist ein verschobender Textbereich nach der Operation an der Originalposition nicht mehr vorhanden.

Man macht sich hierzu die Tatsache zunutze, daß auch die Löschkommandos den Puffer benutzen. Löschen Sie z.B. 5 Zeilen mit „5dd", können Sie diese an einer anderen Position mit „p" wieder einfügen.

11.6 Hilfskommandos

In diesem Abschnitt sind all die wichtigen Kommandos zusammengefaßt, die in keinen der anderen Abschnitte passen. Dies sind:

.	für die Wiederholung des letzten Editierkommandos
u	für das Rückgängigmachen des letzten Editierkommandos
U	für das Rückgängigmachen aller Editierkommandos, welche zuletzt auf die aktuelle

	Zeile ausgeführt wurden
J	für das Zusammenfügen zweier Zeilen

11.7 Stringsuche

Die Suche einer Zeichenfolge erfolgt analog zum Kommando ’more’. Sie geht immer von der aktuellen Position aus und wird im Kommandomodus eingeleitet durch:

/muster	zur Suche nach ’muster’ in Richtung Dateiende
?muster	zur Suche nach ’muster’ in Richtung Dateianfang

Die Suche geht in beiden Fällen bis zur Ausgangsposition.

Wird ’muster’:

- nicht gefunden, erscheint die Meldung „pattern not found“ und der Cursor kehrt zur aktuellen Position zurück
- gefunden, springt der Cursor an die entsprechende Stelle

Im letzteren Fall kann die Suche wiederholt werden durch:

n	in derselben Richtung
N	in umgekehrter Richtung

11.8 Textsubstitution

Eine Textsubstitution erfolgt im „Last Line Mode“. Sie erfolgt mit ’s’ für „substitute“. Das Kommandoformat ist:

:[Bereich]s/muster/ersatz[/option]

’Bereich’ ist optional. Wird er nicht angegeben, wird das erste Vorkommen von ’muster’ in der aktuellen Zeile durch ’ersatz’ ersetzt. Eine Bereichsangabe erfolgt nach dem Muster:

Anfangszeile,Endzeile

Für Anfangs- und Endzeile können absolute Zahlen angegeben werden, oder die Symbole:

.	für die aktuelle Zeile
$	für die letzte Zeile der Datei

Beispiele:

```
24,51s/funk1/f1
1,$s/funk2/f2
.,$s/funk3/f2
```

Zulässige Optionen sind:

g	für „global“, d.h. es wird nicht nur das erste Vorkommen von ’muster’ je Zeile ersetzt, sondern alle
c	für „confirm“. Wird ’muster’ gefunden, werden Sie jeweils gefragt, ob eine Substitution stattfinden soll (y), oder nicht

11.9 Konfiguration des „vi“

Der *vi* verfügt über eine Vielzahl von Optionen, von denen die maßgeblichen hier angesprochen werden.

Das Setzen von Optionen:

- dient der Anpassung des Editors an die eigenen Erfordernisse
- erfolgt mit dem Kommando ’set’ im ’Last Line Mode’
- ist flüchtig, d.h. bei Verlassen des Editors gehen die Einstellungen verloren

Um die gewünschten Optionen permanent zu machen, können sie in eine Datei namens „.exrc“ in Ihrem Heimat-Verzeichnis eingetragen werden. Bei jedem Start des *vi* wird sie gelesen.

Mit `:set all`

- wird eine Liste aller verfügbaren Optionen ausgegeben

- können die momentanen Einstellungen ermittelt werden

Eine Vielzahl von Optionen haben den Charakter von Schaltern. Durch Voranstellen der Vorsilbe ’no’ werden sie ausgeschaltet, durch deren Entfernen eingeschalten.

Die wichtigsten Optionen sind:

(no)nu	schaltet die Zeilennummerierung ein (aus)
(no)ai	sorgt für automatisches Einrücken nach einem Zeilenumbruch
(no)ic	unterscheidet bei Suchvorgängen nicht zwischen Groß- und Kleinschreibung
(no)sm	zeigt während Schreibens von schließenden Klammern immer kurz die entsprechende öffnende Klammer
(no)smd	zeigt Ihnen an, ob Sie sich im ’Insert’ oder ’Override Mode’ befinden
ts=	legt die Tabulatorpositionen fest
sw=	legt die shift-Größe fest

11.10 Aufgaben

Aufgabe 17: Laden Sie die Datei „/kursall/unix-faq/“ in den *vi*, löschen sie den Kopf und die letzten 3 Zeilen und speichern sie das ganze unter dem Namen „~/kurs/unix.faq“ ab.

12 UNIX als Kommunikationshilfe

Einer der Stärken von Linux und UNIX ist die Kommunikation zwischen den Benutzern, sowohl innerhalb einer lokalen Vernetzung, als auch weltweit mit anderen Rechnern in der großen, weiten Welt des Internets.

12.1 Mail

Dem Austausch von Nachrichten dient das Dienstprogramm „mail“.

mail [User@Rechnername]

Bei Aufruf von „mail“:

- mit Argument können Nachrichten gesendet werden
- ohne Argument kommen Sie in den interaktiven Modus und können Nachrichten lesen und eine Reihe weiterer Operationen durchführen

12.1.1 Senden von Nachrichten

Rufen Sie „mail“ mit einem Argument auf:

- werden Sie zunächst nach einem Betreff gefragt, der die Nachricht später im Inhaltsverzeichnis des Adressaten spezifiziert
- dann geben Sie den Nachrichtentext ein
- und schließen diesen mit <Ctrl-D> am Beginn einer Zeile ab (<Ctrl-D> kennzeichnet das Ende der Nachricht und schickt diese ab)

Beispiel:

```
% mail jane@pluto
Subject: Husten
Liebe Jane,
das Lianenschwingen heute abend muß ausfallen.
Es tut mir sehr leid, aber ein schlimmer Husten hat
mich darniedergeworfen. Eine Chance sehe ich jedoch
noch - gehe zu unserem Medizinmann und verlange “Wick
Blau“. Soll mächtig gut wirken, habe ich mir sagen
lassen.
Melde Dich bitte!
Ciao
Tarzan
<Ctrl-D>
```

12.1.2 Lesen von Nachrichten und weitere Operationen

Wird „mail“ ohne Argument aufgerufen:

- werden die Nachrichten der Mailbox in Form eines Inhaltsverzeichnisses aufgelistet
- gelangen Sie in den interaktiven Modus, d.h. Sie müssen von nun an Kommandos eingeben

Eingegangene Nachrichten befinden sich in einer Datei mit Ihrem Benutzernamen im Verzeichnis /var/spool/mail.

Einzugebende Kommandos unterliegen folgender Syntax:

& [Kommando] [Nachrichtenliste]

Geben Sie kein(e):

- Kommando an, wird „print“ ausgeführt
- Nachrichtenliste an, bezieht sich das Kommando auf die “aktuelle Nachricht“, die im Inhaltsverzeichnis mit „>“ gekennzeichnet ist

Für „Nachrichtenliste“ kann angegeben werden:

n	Nachricht Nummer „n“
.	aktuelle Nachricht
n-m	Nachrichten mit den Nummern „n“ bis „m“

Die wichtigsten mail-Kommandos sind:

?	Beschreibung möglicher Kommandos
h	Inhaltsverzeichnis anzeigen
p	Inhalt der spezifizierten Nachricht anzeigen
d	aktuelle Nachricht als gelöscht markieren
dn	Nachricht „n“ als gelöscht markieren
dn-m	Nachrichten „n“ bis „m“ als gelöscht markieren
q	„mail“ verlassen
!cmd	Shell-Kommandos „cmd“ absetzen, ohne „mail“ zu verlassen

Als gelöscht markierte Nachrichten werden:

- wirklich gelöscht

gelesene Nachrichten werden:

- an das Ende der Datei „mbox“ in Ihrem Home-Verzeichnis angefügt

Hinweis:
Neben dem hier auf Kommandozeilen-Ebene beschriebenen Dientsprogramm stellt OpenWindows die Anwendung „Mail Tool“ für genau denselben Zweck zur Verfügung.

12.2 elm (Electronic Mail)

Auf vielen Rechnern steht statt dem „mail“-Programm „elm“ zur Verfügung. Zwei Modi kennt elm:

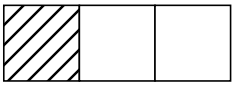
- interaktiver Modus („command mode“)
- Batch-Modus

12.2.1 Interaktiver Modus

Ruft man „elm“ ohne Optionen und ohne Parameter auf, so befindet man sich im „command mode“. Das abgesetzte Kommando bezieht sich dabei auf die Nachricht, die entweder invers oder durch einen Pfeil gekennzeichnet ist.

12.2.2 Message Status

Im Kopf von „elm“ werden die eingegangenen Nachrichten aufgelistet. Vor jeder Nachricht steht ein 3-stelliges Feld, die den Status der Nachricht kennzeichnen.

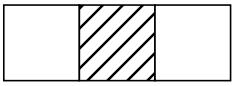


Das erste Zeichen kennzeichnet einen temporären Status:

D „**D**eleted message“ - gelöschte Nachricht

E „**E**xpired message“ - abgelaufene Nachricht[1]

N „**N**ew message“ - neue Nachricht



Das zweite Zeichen kennzeichnet einen permanenten Status (nach absteigender Priorität aufgelistet):

C „**C**onfidential mail“ - definiert durch den ISO X.400 Standard (erscheint, wenn im Mail-Header ein „Sensitivity“-Feld mit dem Wert „3“ steht)

U „**U**rgent mail“ - dringende Nachricht (Mail-

1. Es gibt die Möglichkeit, über ein „Expires“-Feld im Mail-Header ein Datum anzugeben, wie lange die Nachricht gültig sein soll. Folgende Formate werden dabei als Datum erkannt:
   - Sun, 25 Sep 94
   - Sep 25, 94
   - 25 Sep, 94
   - 940925*HHMMZ* (ISO X.400 Format)

	Header enthält ein „Priority“-Feld)
P	„**P**rivate mail“ - private Nachricht (Mail-Header enthält „Sensitivity: 2)
A	„**A**ction“ - Nachricht enthält ein „Action“-Feld
F	„**F**orm Letter“

Das dritte Zeichen kann ein ’+’ Zeichen handelt. Es handelt sich dann um eine *tagged messag*e (markierte Nachricht).

12.2.3 Nachrichten verschicken

Mit ’m’ kann ein Nachricht verschickt werden. Um die Nachricht einzugeben, ruft *elm* Ihren Lieblingseditor auf. Damit können Sie dann Ihre Nachricht eingeben. Nachdem Sie dann Ihre Nachricht abgespeichert und den Editor dann verlassen haben, können Sie Ihre Nachricht verschicken.

Woher weiß *elm*, welches Ihr Lieblingseditor ist? Ganz einfach, nachdem in tausenden von UNIX-Kursen und -Bücher immer wieder vom *vi* die Rede ist, kommt nur der *vi* in Frage. Es gibt jedoch Zeitgenossen, die da anderer Meinung sind (dürfte wohl die überwiegende Mehrheit sein). Diese können über die Environment-Variable „EDITOR“ ihren eigenen Lieblingseditor angeben - doch dazu später mehr (s. Kap. 14.7 *Umgebungsvariable* auf Seite 197).

12.2.4 Kommandos

	Kommando		Kommando		Kommando
?	Hilfe	|	Aktuelle Nachricht per Pipe an ...	$	Erzwingt Synchronisierung der aktuellen Mailbox.

Tabelle 11: elm-Kommandos

Die wichtigsten *elm*-Kommandos

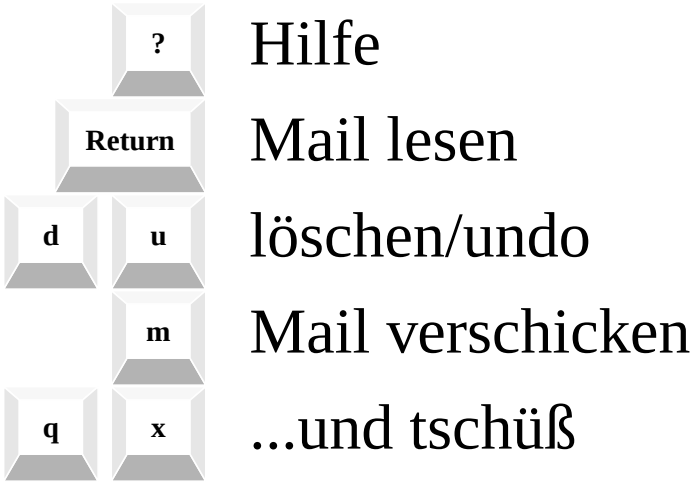


Abbildung 22: die wichtigsten *elm* Kommandos

	Kommando		Kommando		Kommando
!	Shell-Escape	?	Hilfe anzeigen	+	Nächste Kopfseite
-	Vorherige Kopfseite	=	Setzt aktuelle Nachricht auf 1	*	Wählt letzte Nachricht als aktuelle Nachricht
<n>	wählt n als aktuelle Nachricht	/	Sucht Von/Betreff nach Muster ab	//	Sucht gesamten Nachrichteninhalt nach Muster ab
<	Sucht aktuelle Nachricht nach Kalendereinträgen ab	a	Alias, Wechsel in 'Alias'-Modus	b	Weitersendung (Bounce) der aktuellen Nachricht
c	Wechselt die aktuelle Mailbox	d	Löscht die aktuelle Nachricht	^D	Löscht Nachrichten mit angegebenem Muster
f	Leitet Nachricht an angegebene Benutzer weiter	g	Gruppenantwort (alle Empfänger) auf Nachricht	h	Nachricht wird mit vollem Kopf angezeigt
j	Weiter zur nächsten Nachricht	k	Zurück zur vorherigen Nachricht	l	Schränkt Nachrichten durch angegebene Kriterien ein
m	Senden an beliebige Benutzer	n	Nächste Nachricht lesen	o	Ändern der Elm-Optionen

Tabelle 11: elm-Kommandos

	Kommando		Kommando		Kommando
p	Drucken der Nachricht	q	Verlassen (Quit) - Post gelöscht, in mbox oder left gesichert	r	Antwort auf aktuelle Nachricht
s	Speichert Nachricht in angegebener Datei	t	Markiert/löscht Markierung einer Nachricht für weitere Bearbeitung	^T	Markiert Nachrichten mit angegebenem Muster
u	Stellt aktuelle Nachricht wieder her	^U	Stellt Nachricht mit angegebenem Muster wieder her	x	Verlassen (Exit), Lesen nicht registriert, nicht sichern...
^L	Baut Bildschirminhalt neu auf	return	Lesen der aktuellen Nachricht	^Q, DEL	Verlassen (Exit), Lesen nicht registriert, nicht sichern...

Tabelle 11: elm-Kommandos

12.2.5 Formular-Modus

Im Formular-Modus können Formulare verschickt werden. Voraussetzung dafür sind, daß in der Datei .elm/elmrc der Formular-Modus eingeschaltet ist (forms=ON), desweiteren der *userlevel* auf *intermediate* (Level 1) oder *advanced* (Level 2) steht.

Jedes Feld, das mit einem Doppelpunkt (:) abgeschlossen wird, wird als Eingabefeld aufgefaßt. Nach dem Doppelpunkt kann eine Angabe der Feldlänge folgen oder ein Zeilenumbruch (newline). Falls sich Eingabe über mehrere Zeilen erfolgen sollen, werden die Folgezeilen mit einem alleinstehenden Doppelpunkt (:) gekennzeichnet.

Beispiel:

```
...
Willkommen zum UNIX-Einführungskurs.
Bitte füllen Sie folgende Angaben aus:

                    Persönliche Angaben
              Name:
        Geburtsort:

                    Fachliche Angaben
           Studium:
Programmiersprache:
   Betriebssysteme:
```

12.2.6 *elm* für Fortgeschrittene

Eine Reihe von Einstellungen lassen sich über die Datei **.elm/elmrc** vornehmen. So lassen sich z.B. folgende Variablen in dieser Datei setzen:

editor	Einstellung des Lieblings-Editors
mailbox	Datei, in der Mails abgespeichert werden
signature	Signatur, automatischer Abspann
userlevel	0 = Anfänger 1 = Fortgeschrittener 2 = Experte

Falls die Datei **.elm/elmheaders** existiert, wird der Inhalt dieser Datei automatisch an den Mail-Header angehängt. Damit läßt sich ein Briefkopf realisieren, der allen abgehenden Nachrichten vorangestellt wird. Steht dabei ein Ausdruck in Backquotes (`), wird der Ausdruck als Kommando aufgefaßt und durch das Ergebnis des Kommandos ersetzt.

Bsp.: `Datum: ‘date +DD.MM.YY‘`

wird ersetzt durch

```
Datum: 25.09.94
```

12.3 Mailtool

Auf der Sun gibt es glücklicherweise ein Programm namens „mailtool“, das den Umgang mit *mails* erheblich vereinfacht. Man kann dort schon anhand des Icons erkennen, ob neue Mail eingetroffen ist, oder ob sonst noch neue Mail im Postkorb liegt (s. Abb. 23).

Mail verschicken kann man über den (Compose)-Button. Nach Betätigen des (Compose)-Buttons öffnet sich ein Fenster, in dem man den Adressaten („To:“), den Betreff („Subject“) und eventuell noch zusätzliche Adressaten im *cc*-Feld eintragen kann. Hat man dann seine Nachricht eingetippt, kann man es mit (Deliver) abschicken.

Faule Zeitgenossen können sich zum Erstellen von Nachrichten die Drag&Drop-Fähigkeiten des Mailtools zu Nutze machen: z.B. Dateimanager aufmachen, eine fertige Datei anwählen und mittels Drag&Drop in´s Attachment-Feld des Mailtools „rüberziehen“. Auch Grafiken und Geräusche lassen sich auf diese Weise verschicken („Multimedia“ läßt grüßen“). Aber Achtung: die verschickten Datenmengen können auf diese Weise leicht die Megabyte-Grenze überschreiten, und das kann ins Geld gehen. Außerdem haben so manche Mailer, die die Mails außerhalb weiterleiten sollen, so ihre liebe Mühe mit Nachrichten, die eine bestimmte Größe überschreiten (Mails, die kleiner als 64 KByte sind, sollten allerdings problemlos weitergeleitet werden).

Beim Verschicken von Bildern oder Geräuschen ist zu beachten, daß der Empfänger auch an einer Sun sitzen sollte. Ansonsten ist das Anschauen oder Anhören der Multimedia-Dateien etwas problematisch (keine Angst, es geht auch, man muß nur wissen, wie).

Apropos Anschauen/Anhören: Hat man eine Mail bekommen, in dem sich im unteren Attachment-Fenster verschiedene Multimedia- oder WAI[2]-Dateien tummeln, so genügt ein Doppelklick auf die entsprechende Datei, und - Simmsalabimm - das entsprechende Programm wird

2. **W**as-**A**uch-**I**mmer

Mailtool-Icons

keine Mail

neue Mail
ist angekommen

noch Mail
im Postkorb

Abbildung 23: Mailtool-Icons

gestartet, um das Bild anzuzeigen oder die Sounddatei abzuspielen (dauert leider manchmal ´ne ganze Weile, bis es soweit ist).

12.4 Small-Talk

„**talk**“ ermöglicht die (tastaturgesteuerte) Online-Kommunikation mit anderen Benutzern, die:

- sich an einem anderen Terminal desselben Rechners befinden
- sich auf einem anderen Rechner im selben lokalen Netz befinden

Nach Absetzen des Kommandos *talk User@Rechnername*:

- erscheint ein interaktiver Bildschirm
- versucht „talk“ mit der Zielmaschine eine Verbindung aufzubauen

Während des Verbindungsaufbaues erscheint die Meldung „No connection yet“. Gelingt der Verbindungsaufbau, erscheint die Meldung „Waiting for your party to respond“. Ist der angerufene Benutzer nicht eingeloggt, erscheint die Meldung „Your party is not logged on“

Beim Adressaten wird folgende Meldung angezeigt:

```
Message from Talk_Daemon@venus at ..
talk: connection requested by tarzan@venus
talk: respond with: talk tarzan@venus
```

Der angerufene Benutzer:

- muß das Kommando „talk tarzan@venus“ absetzen, wenn er kommunizieren möchte
- reagiert überhaupt nicht, wenn er keine Lust auf Smalltalk hat

’talk’ kann von beiden Teilnehmern mit Ctrl-c beendet werden.

Der interaktive Bildschirm:

- ist in 2 Abschnitte unterteilt
- erlaubt gleichzeitiges Schreiben für beide Teilnehmer

12.5 Sag´s mit Gefühl!

12.5.1 Smileys

Die besprochenen Kommunikations-Mittel unter Unix haben einen entscheidenden Nachteil: sie können das persönliche Gespräch, die Gestik, die Stimme und Stimmung nicht ersetzen. Manchmal sagt ein Grinsen mehr aus, als der eigentliche Wortlaut. Dies wird versucht, durch die „Smileys[3]“ nachzubilden :-)

:-)	grins	**:-(**	grumble (griesgrämiges Gesicht)
;-)	Augenzwinkern	**[:-)**	Walkman-Träger
:-o	Oh	**=:-)**	Punker
0:-)	Engel	**>:-)**	Teufel
:-{	Bartträger	**=|:-)=**	Onkel Sam

Tabelle 12: Smileys

Es gibt noch tausende von weitere Smileys, die etwas von dem Gefühl des Schreibers in den Text rüberretten sollen. Man muß sich die Smileys als ein um 90˚ gekipptes Gesicht vorstellen :-o (noch so ein Smiley, das Erstaunen ausdrücken soll; erkennbar am aufgerissenen Mund :-)

3. manmal auch als „*Emoticon*“ bezeichnet

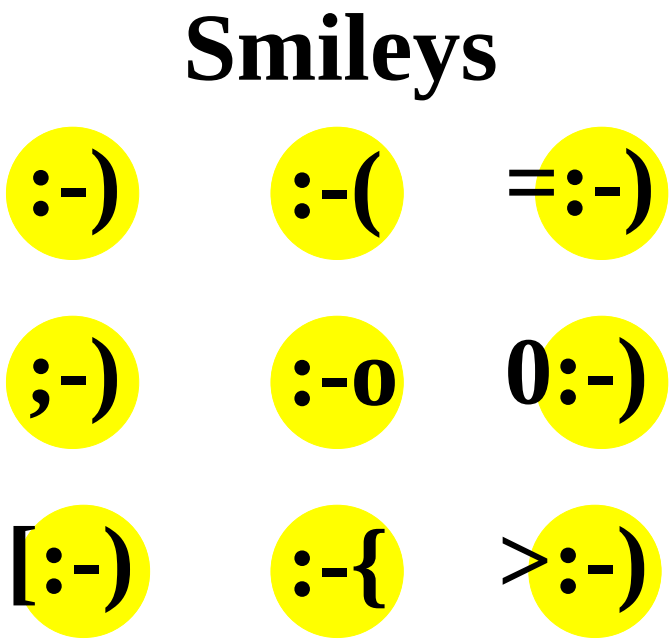


Abbildung 24: Smileys

12.5.2 Akronyme

Es gibt eine Reihe von Abkürzungen bzw. Akronyme, die sich im Laufe der Zeit gebildet haben und zum Sprachschatz eines jeden „Netzteilnehmers“ gehören

Akronym	Beschreibung
AFAIK	As Far As I Know (soweit ich weiß)
BTW	By The Way (übrigens...)
CU	See You (man sieht sich)
CUL8R	See You Later (bis später)
FYI	For Your Information (zur Information)
IMHO	In My Humble Opinion (meiner bescheidenen Meinung nach)
ROFL	Rolling On Floor, Laughing (sich vor Lachen auf dem Boden wälzen)
RTFM	Read That Fucking Manual (lies das verdammte Handbuch)
TGIF	Thank God Its Friday (Gottseidank ist Freitag)
WIMP	Windows, Icons, Mouse, Pointing (damit werden Fenstersysteme und ähnliche Interface beschrieben)
2B|^2B	To Be or Not To Be (Sein oder nicht sein)

Tabelle 13: Akronyme

Darüberhinaus gibt es noch jede Menge weitere Abkürzungen, aber die werden seltener (oder gar nicht) verwendet. Wenn man eine Abkürzung nicht versteht - einfach Nachfragen.

12.6 Aufgaben

Aufgabe 18: Schicken Sie eine Mail an *boehm* mit dem „Subject“ *Anmeldung*. Schreiben Sie in den Rumpf der Mail Ihre Wünsche und Anregungen zum Kurs.

Aufgabe 19: Glauben Sie an den Weihnachtsmann? Selbst wenn man an ihn glaubt, so hat man immer noch das Problem, an welche Adresse man seinen Wunschzettel schicken soll. Probieren Sie’s doch mal mit *santa@northpole.net*!

Aufgabe 20: Viele Firmen sind an das Internet über **email** erreichbar. Eine typische email-Adresse ist z.B. *OBoehm@stgl.sel.alcatel.de*. Schikken Sie an diese Adresse eine Nachricht mit *subject: Schnitzeljagd* (bitte Groß-/Kleinschreibung beachten). Dort erhalten Sie (hoffentlich) weitere Informationen, was Sie zu tun haben.

IV

What Shells?

13 Die Shell

Um überhaupt mit dem UNIX-Kernel in Kontakt treten zu können, ist eine Benutzeroberfläche nötig. Diese Aufgabe übernimmt in UNIX die Shell. Die Shell legt sich wie eine „Schale“ oder eine Muschel um die kostbare Perle names „UNIX-Kernel“ und schützt sie vor den Unbilden des Benutzers. Wenn man ganz behutsam lauscht, kann man vielleicht sogar das Rauschen der CPU hören...

Na ja, ganz so ist es nicht, aber als normaler Benutzer tritt man mit dem UNIX-Kernel i.a. über die Shell in Kontakt. Die Shell nimmt dabei die Kommandos des Benutzers entgegen und leitet sie an den Kernel weiter. Auf der anderen Seite gibt die Shell Nachrichten des Kernels an den Benutzer weiter. Die Shell ist damit Schnittstelle zwischen Kernel und Benutzer.

Folgende Eigenschaften kennzeichnen die Shell:

- Schnittstelle zwischen Kernel und Benutzer
- Kommandointerpreter
- programmierbar
- eigenständiges Programm
- austauschbar

Man kann also die Shell auch durch ein anderen Kommando-Interpreter oder Programm ersetzen. Zum Beispiel wäre statt der Shell eine graphische Oberfläche denkbar, was in der Praxis zunehmend an Bedeutung gewinnt.

13.1 Skripts

Die Shell entält eine Reihe von Kontroll-Anweisungen, mit der sich Programme entwickeln lassen. Diese werden Shell-Skripts oder auch nur Skripts genannt. Sie sind vergleichbar mit *.bat*-Dateien unter DOS.

Skripts werden von der Shell interpretiert. Gegenüber ausführbaren Programmen laufen Skripts zwangsläufig langsamer ab, dafür können sie auf anderen UNIX-Plattformen ohne Änderung übernommen werden (soweit die Theorie; in der Praxis sieht es leider manchmal etwas anders aus).

13.2 Verfügbare Shells

Dem Anwender stehen unter UNIX verschiedene Shells zur Verfügung. Jede hat ihre eigenen Geschichte, ihre eigene Vorzüge und Nachteile.

Fangen wir mit der ältesten Shell an, der

13.2.1 Bourne-Shell

Sie ist nach ihrem Erfinder Steve Bourne benannt und heißt kurz und bündig „**sh**“. Sie dient oft als Standard-Shell für den Systemadministrator.

Gegenüber der C-Shell hat sie den Vorteil, daß Skripts, die mit der Bourne-Shell programmiert wurden, i.d.R. schneller ablaufen als C-Shell-Skripts.

Leistungsmerkmal	Bourne-Shell-Familie				C-Shell-Fam.	
	sh	**ksh**	**zsh**	**bash**	**csh**	**tcsh**
History-Mechanismus		o	+	+	o	+
Tippfehlerkorrektur			+			+
Wildcards (Jokerzeichen)	+	+	+	+	+	+
Namensvervollständigung		o	+	+	o	+
Ein-/Ausgabeumleitung	+	+	+	+	o	o
Überschreibschutz		+	+	+	+	+
Jobkontrolle		+	+	+	+	+
Login-Dateien	+	+	+	+	+	+
.rc-Dateien		o	+	o	o	+
Logout-Dateien			+		+	+
Aliase		o	o	o	+	+
Signalbehandlung	+	+	+	+	o	o

Tabelle 14: Shell-Übersicht

13.2.2 C-Shell

Die C-Shell hört auf den Namen „**csh**“ und wurde von Bill Joy an der UCB entwickelt. Sie gilt als die Standard-Shell für die BSD-Linie und hat eine an die Programmiersprache C angelehnte Syntax - daher der Name.

Seit 1978 gehört die C-Shell zum Lieferumfang des BSD-Unix. Gegenüber der Bourne-Shell liegen ihre Vorzüge in der besseren interaktiven Benutzbarkeit. Dagegen ist sie aufgrund verschiedener Bugs für die Skripterstellung nicht zu empfehlen.

13.2.3 Korn Shell

Die Korn-Shell ist eine neuere Entwicklung und wurde, wie die Bourne-Shell, in den Bell Laboratories von AT&T entwickelt. Analog zur Bourne-Shell wurde sie nach ihrem Entwickler David Korn benannt und heißt unter UNIX „**ksh**“.

Die Korn-Shell versucht, die Vorteile der Bourne-Shell mit denen der C-Shell zu verbinden, dabei aber aufwärtskompatibel mit der Bourne-Shell zu bleiben. Das heißt, Skripts, die für die Bourne-Shell entwickelt wurden, laufen auch unter der Korn-Shell (von einigen wenigen Ausnahmen mal abgesehen).

Die Korn-Shell entwickelt sich zunehmend zu *der* Standard-Shell und wird vor allem bei neueren Unix-(System V)-Releases standardmäßig mit ausgeliefert.

13.2.4 TC-Shell

Im Gegensatz zu den vorigen drei Shells ist die „**tcsh**“ keine „offizelle“ Shell, die zum Lieferumfang von Unix dazugehört. Sie ist eine Weiterentwicklung der C-Shell, die im wesentlichen von William Joy (genau, dem Erfinder der C-Shell) vorangetrieben wurde.

Für die Programmierung ist sie der C-Shell vorzuziehen, da auch einige Bugs bereinigt wurden. Aber auch der Anfänger tut sich gegenüber der C-Shell leichter, da die TC-Shell vor allem deutliche Vorteile bezüglich

des History-Mechanismus bietet. Hier kann der Benutzer mit den Cursor-Tasten hoch- und runterblättern, so wie er es von anderen Systemen (PC oder VAX) gewohnt ist[1].

Trotz ihrer Vorzüge gegenüber der C-Shell ist sie dennoch nicht so verbreitet, da sich viele Systemadministratoren davor scheuen, „Public-Domain"-Software zu installieren.

13.2.5 Bourne-Again-Shell

Die Bourne-Again-Shell oder auch „**bash**" setzt auch auf der Bourne-Shell auf. Sie ist die Shell der Free Software Foundation (GNU Software) und damit frei verfügbar. Die „bash" ist vergleichbar mit der Korn-Shell, hat aber Vorzüge beim History-Mechanismus[2].

13.2.6 Z-Shell

Die „**zsh**" ist eine Weiterentwicklung der Korn-Shell (so eine Art „eierlegende-Wollmilchsau"-Shell).

13.2.7 Weitere Shells

Daneben gibt es noch Dutzende von weiteren Shells, die entweder aus den oberen abgeleitet wurden (z.B. die *restricted* Shell „rsh"), oder Eigenentwicklungen sind. Je nachdem, aus welcher Shell sie hervorgegangen sind bzw. welche Shell als Vorbild diente, ordnet man sie der Bourne-Shell-Familie oder der C-Shell-Familie zu.

Am verbreitesten sind die Bourne- und die C-Shell und zusehends auch die Korn-Shell, da sie standardmäßig mit jedem UNIX ausgeliefert werden.

---

1. Auf einer Sun funktioniert das Blättern mit den Cursor-Tasten leider nicht im Command-Tool („cmdtool"). In einer Shell-Tool dagegen funktioniert es.

2. Auch hier kann man wie bei der TC-Shell mit den Cursor-Tasten durch die letzten Kommandos durchblättern und editieren, wie man es vom PC her gewöhnt ist.

13.3 Aufgaben der Shell

Die Aufgaben der Shell lassen sich unterteilen in:

- die interaktive Ausführung von Kommandos
- die Bereitstellung der Arbeitsumgebung
- die Interpretation und Ausführung von Shell-Skripts
- Ein-/Ausgabe und Pipe-Verarbeitung
- Ersetzung von Sonderzeichen und „Wildcards“

13.3.1 Ein-/Ausgabe

Die Shell stellt drei Ein-/Ausgabe-„Kanäle“ zur Verfügung

- Standard-Eingabe (Kanal 0 oder *stdin*)
- Standard-Ausgabe (Kanal 1 oder *stdout*)
- Standard-Fehlerausgabe (Kanal 2 oder *stderr*)

Normalerweise erfolgt die Standard-Eingabe über die Tastatur, die Standard-Ausgabe und die Standard-Fehlermeldungen erscheinen auf dem Bildschirm. Diese Einstellung läßt sich aber vom Benutzer auf Dateien *umleiten*.

13.3.2 Wildcards

Wildcards dienen der Expandierung von Dateinamen, d.h sie:

- ermöglichen die Angabe von Kurzformen für Dateinamen
- ermöglichen das Ansprechen mehrerer Dateien
- bezieht sich nur auf Dateinamen

Die Shell zerlegt dazu die Kommandozeile zunächst in einzelne Worte, durchsucht diese Worte dann nach „Wildcards und ersetzt ein Wort, das „Wildcards“ enthält, durch eine alphabetisch geordnete Liste von Dateinamen

Folgende „Wildcards“ sind zu unterscheiden:

*	spezifiziert 0 bis n Zeichen an der

	entsprechenden Stelle im Dateinamen
?	spezifiziert ein beliebiges einzelnes Zeichen an der entsprechenden Stelle im Dateinamen
[ ]	spezifiziert ein Zeichen aus einer Menge von Zeichen. Die Zeichen der Menge können aufgezählt, oder als Bereich angegeben werden.

Beispiele:

```
% ls *.tmp
% more rest*
% ls *list*
% ls *
% ls .*
% more 1.?
% ls ????
% ls [A-z]
% ls [ABCDa-z]
% ls file.[1-50]
```

Wildcards werden von der Shell und nicht vom Kommando aufgelöst. Dazu ein Beispiel: Angenommen, folgende Dateien befinden sich im Directory:

```
a.c       a.c.bak   a.o
hello.c   world.c   zorro.p
```

Dann sieht beim Aufruf des Kommandos

```
cp *.c *.bak
```

das *cp*-Kommando nicht die Wildcards, sondern die bereits aufgelöste Zeile

```
cp a.c hello.c world.c a.c.bak
```

Dies mag zwar für den einen oder anderen als Nachteil erscheinen, daß es sich anders als z.B. in DOS enthält, hat aber den Vorteil, daß sich alle Programme in Bezug auf Wildcards gleich verhalten (was man von DOS-Programmen nicht immer behauten kann) und zum anderen bedeutet es

eine Entlastung für den Programmierer (er braucht sich über die Wildcards keine Gedanken mehr machen).

13.3.3 Sonderzeichen

Folgende Sonderzeichen haben in den verschiedenen Shells meist dieselbe Bedeutung:

~[3]	Abkürzung für das Heimatverzeichnis
~*user*[4]	Heimatverzeichnis von *user*
;	Trennen von Kommando-Eingaben innerhalb einer Kommandozeile
(...)	Kommandos gruppieren und starten in einer Subshell

13.3.4 Quoting-Mechanismus

Der eine oder andere wird sich vielleicht gefragt habe, wie kann ich die Shell daran hindern, daß sie Sonderzeichen oder Wildcards auf ihre Weise interpretiert. Dazu gibt es den **Quoting-Mechanismus**, mit der mnan die Bedeutung von Sonderzeichen aufheben kann:

\	das auf \ folgende Zeichen verliert seine Sonderbedeutung
´*String*´	alle Zeichen in *String* verlieren ihre Sonderbedeutung
“*String*“	alle Zeichen außer *$* verlieren ihre Sonderbedeutung
‘*Kommando*‘	*Kommando* wird ersetzt durch die Ausgabe von *Kommando*

3. nicht in der Bourne-Shell

4. nicht in der Bourne-Shell

Beispiel:

```
% echo alles klar\?
alles klar?
% echo ´Haeh?´
Haeh?
% echo “Wo ist $HOME?“
Wo ist /home/boehm?
% echo anno ‘date +%y‘
anno 94
```

13.3.5 Pipes

Eine „**Pipe**“:

- ist ein temporärer Puffer
- wird in der Kommandozeile durch ein ’|’ zum Ausdruck gebracht
- verbindet die Standardausgabe eines Kommandos mit der Standardeingabe eines anderen Kommandos.

Beispiel:

```
% ls -l | more
```

In diesem Beispiel wird die Ausgabe von *ls -l* zu *more* geleitet und als Eingabe für dieses Kommando verwendet. Damit läßt sich die Ausgabe vom *ls*-Kommando über das *more*-Kommando seitenweise durchblättern.

Voraussetzung für den Einsatz von Pipes ist, daß

- das Kommando links der Pipe auf die Standardausgabe schreibt,
- das Kommando rechts der Pipe von der Standardeingabe liest

Werden mehrere Kommandos über Pipes verbunden, spricht man von einer Pipeline. Pipelines verlaufen immer von links nach rechts.

Beispiel:

```
% grep MAX *.h | grep define | sort
```

Damit wird in sämtlichen Dateien mit der Endung *.h* (in C sind dies die sogenannen Include- oder Header-Dateien) nach sämtlichen MAX-Definitionen gesucht und sortiert ausgegeben.

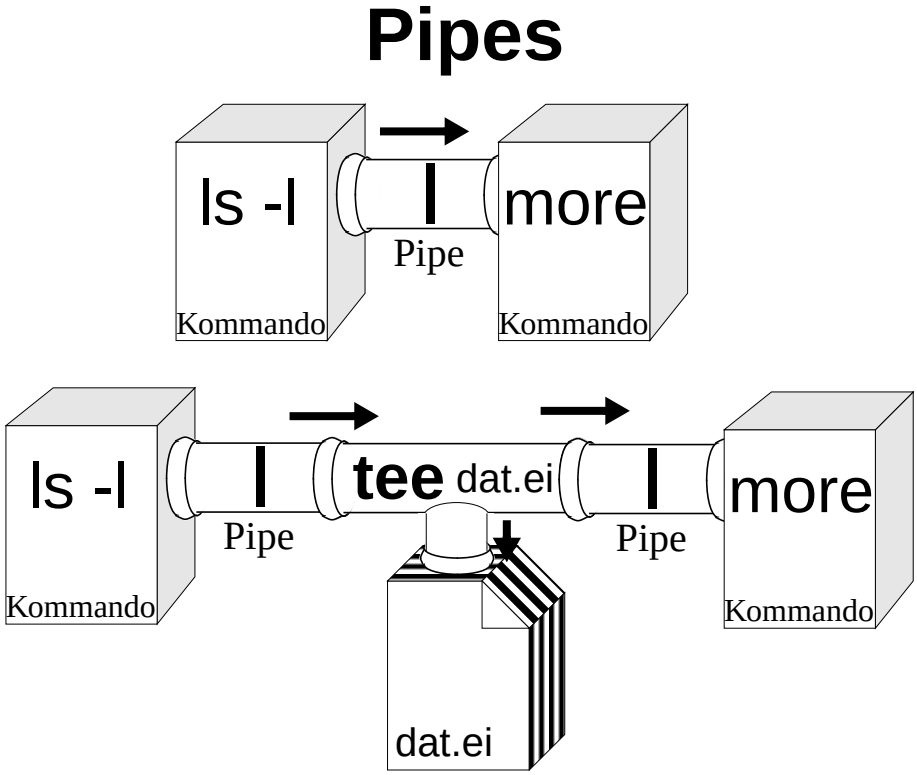


Abbildung 25: Pipes

Pipes können beliebig lang sein. Sämtliche Kommandos in der Pipeline laufen dabei parallel ab, wobei die Pufferung und Synchronisation vom System geregelt wird.

13.3.6 Filter

Als **Filter** werden Kommandos bezeichnet, die:

- von der Standardeingabe lesen
- den gelesenen Text analysieren oder modifizieren
- das Resultat auf die Standardausgabe schreiben

Kommandos wie ’cat’, ’wc’ und ’grep’ sind Beispiele für Filter.

13.3.7 tee

Der Speicherung von Zwischenergebnissen einer Pipeline in einer Datei dient das Kommando ’tee’. Dabei ist *tee* einem T-Stück vergleichbar - daher auch der Name.

tee Datei

Beispiel:

```
% who | tee who.dat
tarzan tty01 Aug 24 09:49
jane   tty02 Aug 24 09:51
% cat who.dat
tarzan tty01 Aug 24 09:49
jane   tty02 Aug 24 09:51
%
```

13.3.8 Kommandoausführung in einer Subshell

Steht ein Kommando oder eine Kommandofolge in runden Klammern, werden die Kommandos in einer Subshell ausgeführt.

Die Kommandoausführung in einer Subshell ist dann notwendig wenn die Umgebung der aktuellen Shell durch die Kommandofolge nicht verändert werden soll.

Beispiele:

```
% pwd
/home/tarzan/entwickl
% ( cd prg; ls )
prg1   prg2
% pwd
/home/tarzan/entwickl
%
```

Das Beispiel soll verdeutlichen, daß nur innerhalb der Subshell in das Verzeichnis „prg“ gewechselt wird. Das aktuelle Verzeichnis der Mutter-Shell bleibt dasselbe.

13.3.9 Environment

Der Environment-Variablensatz ist allen gängigen Shells gemeinsam und wird im Fall des gegenseitigen Aufrufs bzw. Aufruf einer Subshell weitervererbt. Was heißt das? Das heißt nichts anderes, daß Environment-Variablen, die in einer Shell gesetzt werden, allen weiteren Shells, die von dieser „Mutter“-Shell aufgerufen werden, auch zur Verfügung stehen. Leider geschieht das Anlegen und Setzen von Environment-Variablen in den einzelnen Shells unterschiedlich.

Zu was braucht man überhaupt Environment-Variablen? Environment-Variablen speichern Informationen zur Benutzerumgebung und werden zur Steuerung der Shell und von Programmen eingesetzt. Hier eine kleine Übersicht über einige gebräuchliche Environment-Variablen:

Variable	Beispiel	Bedeutung
HOME	/home/boehm	Heimat-Verzeichnis
PATH	/bin:/usr/bin:/usr/ucb:/etc	entält den Suchpfad für Kommandos und Programme
SHELL	/bin/csh	Name der Shell; wird von einigen Kommandos (z.B. *vi*) benutzt, wenn sie eine Subshell erzeugen

Variable	Beispiel	Bedeutung
EDITOR	vi	Standard-Editor
MAIL, MAILPATH	/var/spool/mail/boehm	wird von der Shell benutzt, um unter diesem Pfad nach neuer Post zu schauen und den Benutzer davon zu informieren
PRINTER LPDEST	laser_1	Name des Standard-Druckers (BSD bzw System-V UNIX
MANPATH	/usr/man:/usr/local/man	Suchpfad für die Online-Manuals
TERM	vt100	enthält den Terminal-Typ; wird z.B. vom *vi* benutzt

Tabelle 15: Environment-Variablen

Wie man aus dieser Tabelle sieht, werden Environment-Variablen groß geschrieben. Das muß nicht so sein, hat sich aber so eingebürgert.

13.4 Aufgaben

Aufgabe 21: Gegeben seien folgende Dateien im aktuellen Verzeichnis:

```
u370     u3b5     units      uucp      uuname  uuto
u3b      unadv    unix2dos   uudecode  uupick  uux
u3b15    uname    unpack     uuencode  uusend
u3b2     uniq     unwhiteout uulog     uustat
```

Welche Kommandos werden von der Shell abgesetzt, wenn der Benutzer folgende Eingaben macht:

d) ls uu*
e) cp un???? /tmp
f) cp uus*
g) wc u[a-z]*
h) file u?[a-ex]*

Aufgabe 22: Irgendein böser Kollege hat Ihnen die Dateien

* .*

untergejubelt. Wie bekommen sie diese Dateien wieder weg?

Aufgabe 23: Geben sie von sämtlichen Dateien unter */usr/include* mit der Endung „.h“ diejenigen Zeilen aus, in denen ein „*#define*“ drin vorkommt. Geben Sie diese Zeilen sortiert aus. Sortierkriterium soll dabei das Wort sein, das in der 2. Spalte steht.

14 C-Shell

14.1 Ein- und Ausgabeumlenkung

14.1.1 Standarddateien

Beim Laden wird ein Programm automatisch mit 3 Dateien verbunden:

- der Standardeingabe (**stdin**)
- der Standardausgabe (**stdout**)
- der Standardfehlerausgabe (**stderr**)

Dabei ist die Standardeingabe mit der Tastatur verbunden, die Standardausgabe und die Standardfehlerausgabe sind mit dem Bildschirm verbunden.

Die Unterscheidung von Standardausgabe und Standardfehlerausgabe entspricht der Trennung von:

- auszugebenden Nutzdaten und
- auszugebenden Diagnose- bzw. Fehlermeldungen

Die Shell ist nun in der Lage:

- diese Voreinstellungen zu verändern
- die Standardkanäle mit Dateien Ihrer Wahl zu verbinden

14.1.2 Umlenkung der Standardeingabe

Die Umlenkung der Standardeingabe:

- erfolgt durch Voranstellen von '<' vor den Dateinamen
- ist bei allen Kommandos möglich, die von der Standardeingabe lesen

Beispiel:

Wie Sie sich erinnern, schreibt 'cat' den Inhalt der Datei auf den Bildschirm, die als Argument übergeben wird. Wird kein Argument übergeben, liest 'cat' von der Standardeingabe.

```
% cat file1
Inhalt der Datei file1
% cat
Diesen Text list cat von stdin
^d
Diesen Text list cat von stdin
% cat <file1
Inhalt der Datei file1
%
```

14.1.3 Umlenkung der Standardausgabe

Die Umlenkung der Standardausgabe:

- erfolgt durch Voranstellen von '>' vor den Dateinamen
- ist bei allen Kommandos möglich, die auf die Standardausgabe schreiben

Beispiel:

```
% cat file1
Inhalt der Datei file1
% cat file1 > filecopy
% cat filecopy
Inhalt der Datei file1
%
```

Bei Umlenkung der Standardausgabe:

- erfolgt keine Ausgabe auf dem Bildschirm (nur Fehlermeldungen)

- wird die Datei, in die umgelenkt werden soll, angelegt, bevor das Kommando ausgeführt wird
- geht der Inhalt der Datei, in die umgelenkt werden soll, ggf. verloren

Beispiel:

```
% date > whoson
% cat whoson
Mon Aug 24 11:16:15 GMT 1992

% who > whoson
% cat whoson
tarzan tty01 Aug 24 09:49
jane tty02 Aug 24 09:51
%
```

Das Risiko des Dateiverlustes kann umgangen werden, indem die Standardausgabe eines Kommandos an das Ende einer Datei angehängt wird.

Die Umlenkung der Standardausgabe und das Anhängen der Ausgabean das Ende einer Datei erfolgt durch Voranstellen von '>>' vor den Dateinamen.

Beispiel:

```
% date > whoson
% cat whoson
Mon Aug 24 11:16:15 GMT 1992
% who >> whoson
% cat whoson
Mon Aug 24 11:16:15 GMT 1992
tarzan tty01 Aug 24 09:49
jane tty02 Aug 24 09:51

% ( date; who ) > whoson
% cat whoson
Mon Aug 24 11:16:15 GMT 1992
tarzan tty01 Aug 24 09:49
jane tty02 Aug 24 09:51
```

Dasselbe Ergebnis ist auch durch „Subshelling“ zu erreichen.

14.1.4 Umlenkung der Standardfehlerausgabe

Die Umlenkung der Standardfehlerausgabe:

- erfolgt durch Voranstellen von '>&' vor den Dateinamen
- hat immer auch die Umlenkung der Standardausgabe zur Folge

Soll die Standardausgabe und die Standardfehlerausgabe getrennt werden, ist dies durch Kommandoausführung in einer Subshell möglich.

Beispiel:

```
% find / -name prg1.c -print >& errlog
% ( find / -name prg1.c -print > gefunden ) >& errlog
```

Beim ersten Aufruf werden sowohl die Fehlerausgaben, als auch die Pfade der Datei 'prg1.c' in die Datei 'errlog' geschrieben.

Beim zweiten Aufruf:

- verbindet die aktuelle Shell sowohl 'stdout', als auch 'stderr' mit der Datei 'errlog'
- startet die aktuelle Shell eine Subshell und vererbt die Dateiverbindungen an diesen Sohnprozeß
- verbindet die Subshell 'stdout' mit der Datei 'gefunden' und startet das find-Kommando

14.1.5 Sicherheit bei der Ausgabeumlenkung

Wird die Ausgabe eines Kommandos:

- in eine existierende Datei umgelenkt, wird diese überschrieben
- an das Ende einer nicht existierenden Datei angehängt, wird diese erzeugt

Ist die Shell-Variable '**noclobber**' gesetzt, wird beides verhindert.

Beispiel:

ohne noclobber	mit noclobber
%**unset nocobber**	% **set noclobber**
% **ls**	% **ls**
dat	dat

ohne noclobber	mit noclobber
dat_exist	dat_exist
% **cat dat > dat_exist**	% **cat dat > dat_exist**
% **cat dat >> dat_notexist**	dat_exist: file exists
% **ls**	% **cat dat >> dat_notexist**
dat	dat_notexist: no such file
dat_exist	% **ls**
dat_notexist	dat
	dat_exist

Soll „noclobber“ lediglich für das aktuelle Kommando aufgehoben werden, ist an das Umlenkungszeichen ein ’!’ anzuhängen.

Beispiel:

```
% cat dat >! dat_exist
% cat dat >>! dat_notexist
```

Neben der im letzten Kapitel besprochenen Basisfunktionalität bietet die C Shell eine Reihe von Features, die durch built-in Kommandos realisiert sind. Im einzelnen sind dies:

- Alias-Mechanismus
- History-Mechanismus
- Job-Kontrolle
- Directory-Stacks

14.2 History-Mechanismus

Der History-Mechanismus ermöglicht die wiederholte Ausführung von zuvor abgesetzten Kommandos:

- ohne daß diese erneut getippt werden müssen
- mit kleinen Modifikationen

Voraussetzung hierfür ist das Setzen der Shell-Variablen ’history’.

% **set history =** *n*

Mit diesem Kommando:

- wird ein History-Puffer der Größe ’n’ initialisiert

- werden die jeweils ’n’ letzten Kommandozeilen gespeichert

Mit

% **set history = 0**

kann der History-Mechanismus wieder abgeschalten werden.

Will man die History über das Ende der Shell hinwegretten, kann mit

% **set savehist =** *n*

die letzten *n* Kommandos abgespeichert werden. Diese Kommandos stehen dann beim nächsten Einloggen wieder in der History zur Verfügung.

14.2.1 Anzeigen der History-Liste

Die einzelnen Kommandozeilen werden fortlaufend nummeriert und können mit dem Kommando ’history’ angezeigt werden.

% **history**

14.2.2 Wiederholung ganzer Kommandos

Der Zugriff auf ein zuvor abgesetztes Kommando erfolgt mittels Metazeichen ’!’.

Folgende Variationen sind zu unterscheiden:

!!	Wiederholung des letzten Kommandos
!n	Wiederholung des Kommandos ’n’ der History-Liste
!-n	Wiederholung des n-letzten Kommandos
!string	Wiederholung der letzen Kommandozeile, die mit ’string’ beginnt
!?string?	Wiederholung der letzten Kommandozeile, die ’string’ enthält

Die selektierte Kommandozeile wird dabei vor der Ausführung nochmals angezeigt.

Beispiele:

```
% !23
% !cc
```

14.2.3 Zugriff auf Argumente des letzten Kommandos

Als Argumente gelten alle Worte der Kommandozeile außer dem Kommandonamen selbst. Der Zugriff auf folgende Argumente ist möglich:

!^	Erstes Argument der letzten Kommandozeile
!$	Letztes Argument der letzten Kommandozeile
!*	Alle Argumente der letzten Kommandozeile

Beispiel:

```
% ls
file1 file2 datei sub
% mv file* sub
% mv !$/!^ .
```

14.2.4 Editieren des letzten Kommandos

Ist beim letzten Kommando ein Schreibfehler aufgetreten, oder wurde am Kommandozeilenende etwas weggelassen, stehen folgende Editiermöglichkeiten zur Verfügung:

^alt^neu	Ersetzt den Text ’alt’ der letzten Kommandozeile durch ’neu’
!!erg	Hängt an das Ende der letzten Kommandozeile ’erg’ an

Beispiele:

```
% cc -o prog1 src
% ^og^g
cc -o prg1 src
% !!/prg1.c
cc -o prg1 src/prg1.c
```

14.2.5 Editieren beliebiger Kommandos

Soll ein beliebiges Kommando editiert werden, ist zunächst das Kommando zu spezifizieren, dann die Änderung.

cmdspez:[Modifier]s/alt/neu/

Für ’cmdspez’ sind alle im Abschnitt ’Wiederholung ganzer Kommandos’ genannten Möglichkeiten zulässig.

’Modifier’ kann sein:

g	für globale Substitution, d.h. nicht nur das erste Auftreten von ’alt’, sondern alle ’alt’ werden durch ’neu’ ersetzt

Wird die Zeile mit ’:p’ (für ’print’), abgeschlossen, wird das modifizierte Kommando nur angezeigt, aber nicht ausgeführt

Beispiele:

```
% !cc:gs/prg1/prg2/
% !cc:gs/prg1/prg2/:p
```

14.3 Alias-Mechanismus

Der Alias-Mechanismus erlaubt die Vergabe von Kurznamen für Kommandos.

% **alias** [name [wortliste]]

Wird „alias“:

- ohne Argumente aufgerufen, werden alle momentan definierten Aliase angezeigt
- mit „name“ als Argument aufgerufen, wird der Wert des Aliases „name“ angezeigt
- mit den Argumenten „name“ und „wortliste“ aufgerufen, wird das Alias „name“ neu definiert

14.3.1 Aliase definieren

Die Definition von Aliases ist sinnvoll für:

- die Abkürzungen von Kommandos
- das Umbenennen von Kommandos
- die Vorbelegung von Kommandos mit Optionen bzw. Argumenten
- die Abkürzungen von Kommandosequenzen

„wortliste“:

- kann eine beliebige Kommandozeile sein
- sollte in Hochkommas gesetzt werden, um die Interpretationvon Metazeichen (*?[]|...) durch die C Shell bereits bei der Alias-Definition zu unterdrücken
- kann „name“ enthalten, jedoch nur am Anfang

Beispiele:

```
% alias h history
% alias dir 'ls -l'
% alias ll 'ls -CF'
% alias stat 'pwd;date;who'
% alias rm 'rm -i'
```

An Aliase können auch Kommandozeilenargumente übergeben werden. Hierbei steht:

!*	für alle Kommandozeilenargumente
!^	für das erste Kommandozeilenargument
!$	für das letze Kommandozeilenargument

Zu beachten ist lediglich die Maskierung des Aufrufezeichens, da es sonst als History-Kommando interpretiert wird.

Beispiel:

```
% alias pfad 'find ~ -name \!^ -print'
```

14.3.2 Aliase löschen

Dem Löschen von Aliases dient das build-in Kommando 'unalias'

% **unalias** name(n)

14.4 Job-Kontrolle

Jedes Kommando und jede Pipeline die sich in Ausführung befindet wird als **Job** bezeichnet. Build-in Kommandos der C Shell erlauben die Kontrolle dieser Jobs.

Ein Job kann:

- im Vordergrund aktiv sein
- im Hintergrund aktiv sein
- suspendiert, d.h. angehalten sein

Im Vordergrund kann zu einem Zeitpunkt nur ein Job aktiv sein, während beliebig viele Jobs im Hintergrund laufen bzw. suspendiert sein können.

Mit der C Shell lassen sich:

- Vordergrund Jobs anhalten und wieder fortsetzen
- Jobs vom Vorder- in den Hintergrund verlagern und umgekehrt
- Hintergrund Jobs anhalten und wieder fortsetzen
- Jobs abbrechen

Ein Job wird über seine Jobnummer angesprochen. Wird keine Jobnummer angegeben, beziehen sich Operationen immer auf den aktuellen Job; dieses ist der Job, der zuletzt gestartet oder angesprochen wurde.

Soll ein Job im Hintergrund ausgeführt werden, lenkt man am besten die Ausgabe in eine Datei um. Ansonsten erscheint die Ausgabe auf dem Bildschirm und vermischt sich evtl. mit der Ausgabe von anderen gestarteten Kommandos.

14.4.1 Informationen zu aktuellen Jobs anzeigen

Informationen zu allen vorhandenen Hintergrund Jobs und angehaltenen Jobs lassen sich mit dem built-in Kommando ’jobs’ anzeigen.

% **jobs**

Beispiel:

```
% jobs
[1] - Stopped find /home -name prg1.c -print >&
prg1.lis
[2] + Stopped cat /etc/termcap > term.dsc
[3] Running shelltool
%
```

Die Informationen sind folgendermaßen zu interpretieren. Die

- 1. Spalte zeigt die **Jobnummer** an
- 2. Spalte zeigt den aktuellen Job (+) und den vorigen Job (-) an
- 3. Spalte zeigt den **Jobstatus** an
- 4. Spalte zeigt die zugehörige Kommandozeile an

Folgende Jobstati sind zu unterscheiden:

Running	Job befindet sich in Ausführung
Stopped	Job ist momentan angehalten
Done	Job wurde normal beendet
Terminated	Job wurde abgebrochen

Endet die Ausführung eines Hintergrundjobs, wird dies dem Anwender immer vor Ausgabe des nächsten Prompts angezeigt.

14.4.2 Vordergrund Jobs anhalten und wieder fortsetzen

Das Anhalten eines Vordergrund Jobs:

- erfolgt mit der Suspend-Taste, i.d.R. Ctrl-Z
- ist Voraussetzung, um einen Job in den Hintergrund zu verlagern

Ein Job wird durch das Kommando „fg“ (**f**ore**g**round) im Vordergrund fortgesetzt.

% **fg** [%Jobnummer(n)]

Wird „fg“:

- kein Argument übergeben, bezieht sich das Kommando auf den aktuellen Job

- mit mehreren Jobnummern aufgerufen, werden diese Jobs sukzessive im Vordergrund fortgesetzt

Beispiel:

```
% find /home -name prg1.c -print >& prg1.lis
ctrl-z
% cat /etc/termcap > term.dsc
ctrl-z
% jobs
[1] - Stopped find /home -name prg1.c -print >&
prg1.lis
[2] + Stopped cat /etc/termcap > term.dsc
% fg %1
```

14.4.3 Jobs vom Vorder- in den Hintergrund verlagern und umgekehrt

Ein Job wird durch das Kommando „bg“ (**b**ack**g**ound) im Hintergrund fortgesetzt.

% **bg** [%Jobnummer(n)]

Voraussetzung hierfür ist, daß der Job im Vordergrund zunächst angehalten wurde.

Beispiel:

```
% find /home -name prg1.c -print >& prg1.lis
ctrl-z
% cat /etc/termcap > term.dsc
ctrl-z
% jobs
[1] - Stopped find /home -name prg1.c -print >&
prg1.lis
[2] + Stopped cat /etc/termcap > term.dsc
% bg %1 %2
% jobs
[1] - Running find /home -name prg1.c -print >&
prg1.lis
[2] + Running cat /etc/termcap > term.dsc
% fg %1
```

14.4.4 Hintergrund Jobs anhalten und wieder fortsetzen

Ein Hintergrund Job wird durch das Kommando ’stop’ angehalten.

% **stop** %Jobnummer(n)

Beispiel:

```
% jobs
[1] - Stopped find /home -name prg1.c -print >&
prg1.lis
[2] + Stopped cat /etc/termcap > term.dsc
% bg %1 %2
% jobs
[1] - Running find /home -name prg1.c -print >&
prg1.lis
[2] + Running cat /etc/termcap > term.dsc
% stop %1
% jobs
[1] + Stopped find /home -name prg1.c -print >&
prg1.lis
[2] - Running cat /etc/termcap > term.dsc
% bg %1
```

14.4.5 Jobs abbrechen

Ein Job kann mit dem Kommando „kill“ abgebrochen werden.

% **kill** Jobnummer(n)

„kill“ bewirkt, daß:

- dem Job ’Jobnummer’ das Signal SIGTERM gesendet wird
- die C Shell die Beendigung des Jobs meldet

Beispiel:

```
% jobs
[1] + Stopped find /home -name prg1.c -print >&
prg1.lis
[2] - Running cat /etc/termcap > term.dsc
% kill %1
[1] Terminated find /home -name prg1.c -print >&
prg1.lis
%
```

14.5 Directory-Stack

Der ’Directory Stack’ ist ein Stack, in dem Pfadnamen von Verzeichnissen gespeichert werden. Sinnvoll ist dessen Verwendung, wenn häufig zwischen denselben Verzeichnissen gewechselt wird.

14.5.1 Erzeugen eines Directory-Stacks

% **pushd** [verzeichnis | +n]

Mit dem Kommando ’pushd’ wird:

- ’verzeichnis’ zum aktuellen Verzeichnis (vergl. ’cd’)
- ’verzeichnis’ das oberste Element des Stacks, während das vorige Verzeichnis zum zweiten Element des Stacks wird

Wird ’pushd’:

- ohne Argument aufgerufen, werden die beiden obersten Elemente des Stacks vertauscht und entsprechend zum letzten aktuellen Verzeichnis gewechselt
- mit ’+n’ aufgerufen, rotiert der Stack dergestalt, daß das n-te Element das oberste Element und das aktuelle Verzeichnis wird. Die Stackelemente sind mit ’0’ (oberstes Element) beginnend durchnummeriert.

Die Beispiele gehen von folgender Verzeichnishierarchie aus: (fehlt ⇒ versuchen Sie´s mal selber rauszubekommen)

Beispiel:

```
% pwd
/usr/karl
% pushd work
/usr/karl/work /usr/karl
% pushd test
/usr/karl/work/test /usr/karl/work /usr/karl
% pushd ../doc
/usr/karl/work/doc /usr/karl/work/test /usr/karl/work
/usr/karl
% pushd
```

```
/usr/karl/work/test /usr/karl/work/doc /usr/karl/work
/usr/karl
% pushd
/usr/karl/work/doc /usr/karl/work/test /usr/karl/work
/usr/karl
% pushd +2
/usr/karl/work /usr/karl /usr/karl/work/doc
/usr/karl/work/test
% pushd +2
/usr/karl/work/doc /usr/karl/work/test /usr/karl/work
/usr/karl
```

14.5.2 Anzeigen des Directory-Stacks

Mit dem Kommando ’dirs’ kann der aktuelle Inhalt des Directory-Stacks angezeigt werden.

% **dirs**

14.5.3 Entfernen eines Eintrages

Mit dem Kommando ’popd’ wird ein Eintrag aus dem Directory-Stack entfernt.

% **popd** [+n]

Wird ’popd’:

- ohne Argument aufgerufen, wird das oberste Element des Directory-Stacks entfernt und das nun oberste Element zum aktuellen Verzeichnis
- mit ’+n’ aufgerufen, wird das n-te Element entfernt

Beispiel:

```
% dirs
/usr/karl/work/doc /usr/karl/work/test usr/karl/work
/usr/karl
% popd
/usr/karl/work/test /usr/karl/work /usr/karl
% popd
/usr/karl/work /usr/karl
```

```
% dirs
/usr/karl/work/doc /usr/karl/work/test /usr/karl/work
/usr/karl
% popd +1
/usr/karl/work/doc usr/karl/work /usr/karl
```

14.6 Shell Variable

Shell Variable:

- sind auf die lokale C Shell begrenzt
- werden üblicherweise klein geschrieben
- sind teilweise schon von der ’csh’ vordefiniert

Shell Variable werden verwendet:

- um Shell-Dienstleistungen zu konfigurieren
- in Shell-Prozeduren

Einige Shell Variable sind bereits vordefiniert. Zu unterscheiden sind solche, die:

- vom Benutzer sinnvoll veränderbar sind
- ständig von der Shell auf den richtigen Wert gesetzt werden

Zu letzteren gehören:

argv	Vektor, der Aufrufparameter für Shell-Skripte enthält
cwd	enthält immer das aktuelle Verzeichnis
home	enthält den Pfad des Home-Verzeichnisses
status	enthält Exit-Status des zuletzt ausgeführten Kommandos

Vom Benutzer modifizierbar sind:

history	um die Länge des History-Buffers festzulegen
ignoreeof	um die Sonderbedeutung von Ctrl-d auszuschalten

noclobber	um das überschreiben von Dateien bei der Ausgabeumlenkung zu verhindern
notify	um eine sofortige Meldung zu bekommen, wenn ein Hintergrund Job beendet ist
noglob	um die Dateinamenexpandierung auszuschalten
path	um Kommandos und Programme im Verzeichnisbaum zu finden
prompt	zur Definition des aktuellen Promptes
shell	enthält Pfadnamen der Login Shell
term	enthält Bezeichnung des Terminals

Die aktuellen Werte von Variablen können mit dem Kommando ’set’ ohne Angabe von Argumenten angezeigt werden.

Beispiel:

```
% set
argv ()
cwd /home/tarzan/entwickl
home /home/tarzan
path (/bin /usr/bin /usr/ucb /usr/5bin .)
prompt %
shell /bin/csh
status 0
term sun
```

14.6.1 Definition lokaler Variable

Die Werte von Variablen werden als Strings gespeichert. Die Definition einer Shell Variablen erfolgt mit dem Kommando ’set’.

% set variable [= wert]

Wird der Variablen kein Wert zugewiesen, entspricht dies der Zuweisung eines Leerstrings.

Sind Leerzeichen Bestandteil des Variablenwertes, muß dieser in Anführungszeichen gesetzt werden.

Beispiele:

```
% set vname
% set vname = Natascha
% set namen = „Natascha Neven Tessa Oliver“
```

Darüber hinaus können eindimensionale Arrays, auch Wortlisten genannt, definiert werden. Hierbei ist der Variablen eine Liste von Werten in runden Klammern zuzuweisen.

Beispiel:

```
% set namen = ( natascha neven tessa oliver )
% mail $namen <message
% set path = ( $path /home/tarzan/entwickl/prg )
```

14.6.2 Zugriff auf lokale Variable

Zur Ausgabe von Argumenten auf die Standardausgabe dient das Kommando ’echo’.

% **echo** [Argument(e)]

Die einzelnen ’Argumente’ werden, jeweils durch ein Leerzeichen getrennt, auf die Standardausgabe geschrieben.

Beispiele:

```
% echo eins zwei drei
eins zwei drei
% echo „eins zwei“ drei
eins zwei drei
%
```

Der Zugriff auf den Wert einer Variablen, eine sogenannte Variablensubstitution, erfolgt durch Voranstellen eines ’$’ vor den Variablennamen.

Der Zugriff auf einzelne Worte einer Wortliste erfolgt mittels ’$variable[index]’. Die Indizierung beginnt mit 1.

Beispiele:

```
% echo Hallo $vname
Hallo Natascha
% echo Hallo $namen[4]
Hallo oliver
```

14.6.3 Zurücksetzen lokaler Variable

Lokale Variable werden mit dem Kommando ’unset’ zurückgesetzt.

% **unset** variable

Beispiele:

```
% unset noclobber
% unset namen
```

14.7 Umgebungsvariable

Wird der Wert einer Umgebungsvariablen, wie z.B. PATH geändert, modifiziert die C Shell automatisch die Variable ’path’. Dasselbe gilt auch in umgekehrter Richtung.

Die aktuellen Werte der Umgebungsvariablen können mit dem Kommando ’printenv’ angezeigt werden.

Beispiel:

```
% printenv
HOME=/home/tarzan
SHELL=/bin/csh
PATH=:/bin:/usr/bin:/usr/ucb:/usr/5bin:.
TERM=sun
```

14.7.1 Definition von Umgebungsvariablen

Die Definition einer Umgebungsvariablen erfolgt mit dem Kommando ’setenv’.

% **setenv** VARIABLE [wert]

Wird der Variablen kein Wert zugewiesen, entspricht dies der Zuweisung eines Leerstrings. Dies ist aber nicht auf allen Systemen einheitlich. So gibt z.B. HP-UX eine Fehlermeldung aus, wenn man *wert* wegläßt. Will man dort eine Variable ohne Inhalt anlegen, so muß man explizit einen Leerstring (““) angeben.

Beispiele:

```
% setenv VNAME
% setenv VNAME Natascha
% echo Hallo $VNAME
Hallo Natascha
% set a = 10
% echo $a
10

% csh
% echo $a
a: undefined
% exit
% setenv A 10
% echo $A
10
% csh
% echo $A
10
% exit
```

Was will uns dieses Beispiel zeigen? Eigentlich nur, daß die Environmentvariable „A“ weitervererbt wird, die Shellvariable „a“ nicht. Daneben zeigt es aber auch, daß zwischen Groß- und Kleinschreibung unterschieden wird.

14.7.2 Zurücksetzen von Umgebungsvariablen

Umgebungsvariablen werden mit dem Kommando ’unsetenv’ zurückgesetzt.

% **unsetenv** VARIABLE

Beispiele:

```
% unsetenv VNAME
% unsetenv A
```

14.8 Anpassen der Arbeitsumgebung

Beim Aufsetzen der Arbeitsumgebung scheinen selbst Systemadministratoren nicht immer eine glückliche Hand zu haben. Um wieviel schwerer tut sich dann ein einfacher Benutzer, der die C-Shell seinen Bedüfnissen

anpassen will? Dieses Kapitel will eine kleine Hilfestellung sowohl für den geplagten Systemadministrator als auch den Otto-Normal-Benutzer geben, wie man die Arbeitsumgebung sinnvoll aufsetzen kann.

14.8.1 Welche Dateien gibt es für die C-Shell?

Folgende Dateien im Heimatverzeichnis des Benutzers beeinflussen das Verhalten der C-Shell:

.login	Diese Datei wird nur von der login-Shell ausgeführt (daher der Name *.login*), d.h. sie wird nur einmal von der C-Shell interpretiert (nach dem Einloggen)
.cshrc	Jedesmal, wenn eine neue Shell aufgerufen wird, wird diese Datei interpretiert. Hierüber können Einstellung für die C-Shell vorgenommen werden. Da die *.cshrc*-Datei jedesmal interpretiert werden muß, wenn eine neue Shell aufgerufen wird, sollte diese Datei nicht zu umfangreich werden, da ansonsten der Aufruf einer neuen C-Shell unnötig verlangsamt wird.
.logout	Wird beim Beenden Login-Shell (d.h. vor dem Ausloggen) ausgeführt. Dies kann z.B. dazu benutzt werden, um temporäre Dateien, die nicht mehr benötigt werden, wieder zu löschen.

14.8.2 Wann wird eine Shell gestartet?

- Beim Einloggen (dies ist die sogenannte Login-Shell)
- expliziter Aufruf einer neuen Shell (/bin/csh)
- Aufruf eines Shell-Skripts

Auch bei Fenstersystemen wird beim Öffnen von bestimmten Fenster (z.B. *shelltool*, *cmdtool* oder *xterm* unter OpenWindows oder OSF/Motif) eine neue Shell aufgerufen.

Umfangreiche *.cshrc*-Dateien können dazu führen, daß

- C-Shell-Skripts länger brauchen
- bei Fenstersystemen (z.B. OpenWindows oder OSF/Motif) der Aufruf eines „Shell“-Fensters länger dauert

14.8.3 Aufteilung

Da die .login-Datei nur nach dem Login ausgeführt wird, schreibt man üblicherweise folgende Dinge in diese Datei:

- Initialisierung des Terminals (z.B. stty -tabs)
- Anzeige von Message-of-the-Day (oder ähnlichem)
- Setzen von Environment-Variablen
- Setzen des Suchpfads
- generelle Kommandos (z.B. umask)
- Starten des Fenstersystems auf einer Workstation

Environment-Variablen und der Suchpfad werden weitervererbt, so daß es ausreicht, sie in die .login-Datei zu schreiben. Für die .cshrc-Datei verbleibt damit:

- Setzen von Shell-spezifischen Variablen (z.B. set ignoreeof)
- Aufsetzen des Prompts
- Aliases

14.8.4 Aufsetzen des Suchpfads

Vor allem das Aufsetzen des Suchpfades (`set path = ...`) stellt eine Belastung für das System dar und kann sich über mehrere Sekunden (!) erstrecken (intern wird von der Shell eine Hash-Tabelle aufgebaut, die mit jedem „*set path*“-Befehl wieder verworfen wird). Dies ist vor allem für Fenstersystem und anderen interaktiven Systemen unbefriedigend, da hier aus Software-ergonomischer Sicht Antwortzeiten unter 2 Sekunden

anzustreben sind. Aus diesem Grund sollte der Suchpfad möglichst nicht in der *.cshrc*-Datei aufgesetzt werden, sondern in *.login*.

Ein weiterer Grund, der für das Aufsetzen des Suchpfades in der *.login*-Datei spricht, ergibt sich beim Aufruf eines C-Shell-Skripts. Auch hier führt das Aufsetzen des Suchpfades in der *.cshrc*-Datei zu einem vermeidbaren Overhead. Zwar läßt sich das Lesen und Interpretieren der .cshrc-Datei bei der Skript- Programmierung durch das '-f'-Flag unterbinden, aber nicht jedes C-Shell-Skript macht davon Gebrauch.

Zu beachten ist, daß beim Login die *.cshrc*-Datei vor der *.login*-Datei durchlaufen wird, d. h. wenn der Pfad in .login gesetzt wird, steht er für die *.cshrc*-Datei bei der Login-Shell noch nicht zur Verfügung. Will man daher in der .cshrc-Datei Kommandos aufrufen, die nicht im Standard-Pfad (. /usr/ucb /usr/bin) stehen, muß man sie mit voller Pfadangabe aufrufen.

14.8.5 Remote Shell

Mit der Remote Shell *rsh* (auch *remsh* oder *rcmd*, s.a. Kap. 21.5 »Remote Shell«) läßt sich ein Kommando oder mehrere Kommandos auf einem anderen Rechner starten (sofern dies von der Administration eingerichtet wurde). Dazu wird auf dem Remote Rechner eine Shell gestartet, die das angegebene Kommando absetzt. Bedauerlicherweise werden weder der Suchpfad, noch Environment-Variablen an die Remote Shell weitervererbt, d.h. diese Daten stehen der Remote Shell nicht zur Verfügung, wenn sie in der *.login*-Datei plaziert wurden.

Also doch in die .cshrc-Datei schreiben? - Zwei Gründe sprechen dagegen: Zum einen wird der *rsh*-Befehl relativ selten benötigt und *rsh* ohne Kommando startet eine interaktive login-Shell auf dem Remote Rechner (d.h. *.login* wird durchlaufen). Zum andern ist man eher an eine schnelle Ausführung des Befehls interessiert und nimmt dafür in Kauf, daß das benötigte Environment notfalls beim Aufruf mit aufgesetzt werden muß[5].

Beispiel:

```
rsh luna “setenv PRINTER laser_1; lpq“
rsh venus “source .login; lpq“
```

14.8.6 Tuning

Es kann auch Gründe geben, den Suchpfad in die *.cshrc*-Datei zu schreiben (s.o.). Da der Suchpfad und die Environment-Variablen weitervererbt werden, kann man in *.cshrc* das Setzen des Suchpfads und der Environment-Variablen mit folgender Konstruktion auf das 1. Mal beschränken:

```
...
if ($?ENV_SET == 0) then

    #   set up search path
    set path = ...
    ...

    #   set up environment
    setenv ...
    ...

    setenv ENV_SET
endif
...
```

Durch diese Konstruktion wird der Suchpfad und das Environment nur dann aufgesetzt, wenn er noch nicht gesetzt wurde. Damit läßt sich die Interprationszeit der .cshrc-Datei von mehreren Sekunden auf unter einer Sekunde herabsenken[6].

Ein weiterer, wenn auch geringer, Performance-Gewinn erreicht man für den Aufruf von C-Shell-Skripts dadurch, indem man vor der Definition der verschiedenen *alias*e (die man zweckmässigerweise am Ende der .cshrc-Datei vornimmt), folgende Zeilen

```
# ignore rest for non interactive csh
```

---

5. Notfalls kann man immer noch mit

```
rsh remote_host “source .login; ...“
```

das gesamte Environment und Suchpfad aufsetzen.

6. Basis: SunIPX, Dateisystem über NFS gemountet

```
    if (! $?prompt ) exit
```

einfügt oder durch

```
    if ($?prompt) then
        ...
        alias ...
        ...
    endif
```

ausklammert.

Damit werden die folgenden Zeilen nur dann ausgeführt, wenn es sich um eine interaktive Shell handelt. Für ein C-Shell-Skript werden diese Zeilen nicht ausgeführt. Als angenehmen Nebeneffekt sind die Aliases dem Skript nicht bekannt und können so auch nicht zu Problemen führen.

Bei der Programmierung von C-Shell-Skripts kann man die Interpretation der .cshrc-Datei durch das Flag '-f' unterbinden. Die erste Zeile des Skripts lautet damit:

```
#! /bin/csh -f
```

14.8.7 Performance-Gewinn

Es ist schwierig, die Zeitersparnis und den Performance-Gewinn in Zahlen auszudrücken, da dies vom jeweiligen Benutzerprofil abhängt. Subjektiv läßt sich sagen, daß das System wesentlich agiler wirkt. Damit steigt die Motivation des Benutzers.

14.8.8 Administration

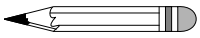
Für die Administration ergibt sich oft das Problem, daß man neben dem Standard-Suchpfad und dem Aufsetzen des Environments eine Reihe von Tools noch zu berücksichtigen hat, die in der *.login*- oder *.cshrc*-Datei aufgesetzt werden müssen. Dies führt oft zu langen Suchpfaden, die den Aufruf eines Kommandos unnötig verlängern - und nicht jeder Benutzer benötigt jedes Tool.

Wünschenswert wäre es, wenn jeder Benutzer sich seine Tools, die er benutzen will, selber raussuchen könnte, der Support dafür aber weiterhin

bei der Systemadministration bleibt. Dies kann dadurch realisiert werden, daß tool-spezifische *.login*- und *.cshrc*-Dateien angelegt werden, die sich dann der Benutzer selbst über den **source**-Befehl in sein *.login* bzw. *.cshrc* einbinden kann.

Beispiel: Ein Benutzer möchte gerne den C++-Compiler und den Emacs-Editor benutzen. Seine .login- und .cshrc-Dateien könnten dabei so aussehen:

```
#########################################################
#
#   .login file
#
#   Read in after the .cshrc file when you log in.
#   Not read in for subsequent shells.  For setting
#   up terminal and global environment
#   characteristics.
#
#   See Also:       /usr/lib/Login
#
#########################################################

echo „WELCOME TO THE ‘hostname‘ MACHINE“
date

source /supportfiles/.login
source /supportfiles/.login.c++
source /supportfiles/.login.emacs

# if possible, start the window system.
if ( „$term“ == „sun“ && ‘tty‘ == „/dev/console“ )
then
    exec openwin
endif
```



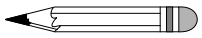
```
#########################################################
#
#   .cshrc file
#
#   initial setup file for both interactive and
#   noninteractive C-Shells
#
```

```
#   See Also:       /usr/lib/Cshrc     #
#########################################################

set history=40   # number of lines safed in history
set noclobber    # restrict output redirection

source /supportfiles/.cshrc
source /supportfiles/.cshrc.c++
source /supportfiles/.cshrc.emacs

# ignore rest for non interactive csh
if (! $?prompt ) exit
alias del     'chmod og-wxr \!*; mv \!* /tmp'
alias deldir  'chmod -R og-wxr \!*; /bin/cp -rp \!*
/tmp; /bin/rm -r -f \!*'
```

Mit „*source /supportfiles/.login*“ wird die Standard-.*login*-Datei eingebunden, die in diesem Beispiel unter dem Verzeichnis „/supportfiles“ zu finden ist, mit „*source /supportfiles/.login.c++*“ und „*source /supportfiles/.login.emacs*“ werden die entsprechenden .login-Dateien für C++ und den Emacs-Editor eingebunden. Entsprechendes gilt für die .*cshrc*-Datei.

Die .login-/.cshrc-Dateien für C++ könnten z.B. folgendermaßen aussehen:

```
#########################################################
#
#   .login.c++ file
#
#   .login file for setting up the C++
#   environment
#
#########################################################

set path = ($path /tools/c++)
setenv MANPATH      ${MANPATH}:/tools/c++/man
```



```
#########################################################
#
#   .cshrc.c++ file
#
#   Some useful aliases for C++
```

```
#
#####################################################
```

```
alias c++licens 'lmstat -a -c /tools/c++/license.dat'
```

Sollte sich z.B. der Suchpfad für C++ ändern, so kann dies an zentraler Stelle geschehen, und beim nächsten Einloggen bekommt der Benutzer automatisch den neuen Suchpfad aufgesetzt.

14.8.9 Andere Shells

Bei der Bourne-Shell (/bin/sh; s. nächstes Kapitel) gibt es nur eine .profile-Datei, die der Benutzer aufsetzen kann. Diese Shell wird allerdings weniger für interaktive Sessions benutzt, sondern mehr zur Erstellung von Shell-Skripts.

Andere Shells orientieren sich meist an der C-Shell oder Bourne-Shell, so daß sich das dort beschriebene meist übertragen läßt.

Nachdem Sie sich einloggen wird das Programm gemäß der Datei */etc/passwd* gestartet, i.d.R. die C Shell. Diese Login Shell sucht der Reihe nach folgende Dateien und führt die darin beschriebenen Kommandos aus:

1. ~/.cshrc
2. ~/.login

14.8.10 Modifikation und Test der Spezialdateien

Wenn Sie die Datein .login, .cshrc oder .logout Ihren Bedürfnissen angepaßt haben und Sie nun diese Änderung ausprobieren wollen, können Sie dazu das Kommando *source* benutzen:

% **source** Kommandoprozedur

source bewirkt, daß die Kommandos der Kommandoprozedur von der *aktuellen* Shell[7] ausgeführt werden. Neue Definitionen sind der aktuellen Shell danach bekannt.

14.9 Aufgaben

Aufgabe 24: Defineren Sie einen alias *lpn*, der eine Datei mit Zeilennummern auf dem Drucker ausgibt.

Aufgabe 25: Welche DOS- (oder auch VAX/VMS-, ...) Kommandos können über *Alias*e nachgebildet werden?

```
Bsp:      alias dir   "ls -l"
```

Aufgabe 26: Listen Sie alle Dateien unter `/bin` auf, die nicht mit „uu“ anfangen.

---

7. Es gibt auch Kommandoprozeduren (auch Skript oder Shell-Skript genannt), die nur durch den Name der Kommandoprozedur gestartet werden. Hierbei wird aber eine Subshell aufgerufen, die die Kommandoprozedur abarbeitet. Dies bedeutet, daß (Environment-) Variablen, die in dieser Kommandoprozedur gesetzt werden, nicht weitervererbt werden (es wird immer von oben nach unten vererbt).

15 Die Bourne-Shell

15.1 Ein- und Ausgabeumlenkung

Im Gegensatz zur C-Shell lassen sich in der Bourne-Shell alle drei Ein-/Ausgabekanäle umlenken:

Kanal 0:	Standardeingabe (**stdin**)
Kanal 1:	Standardausgabe (**stdout**)
Kanal 2:	Standardfehlerausgabe (**stderr**)

15.1.1 Umlenkung der Standard-Ein/Ausgabe

Will man die Standard-Eingabe oder -Ausgabe umleiten, braucht man die Kanalnummer nicht mit anzugeben. Damit verhält sie sich genauso wie die C-Shell: mit <*file* wird die Standardeingabe von *file* gelesen, mit >*file* wird die Standardausgabe nach *file* umgeleitet.

Vorraussetzung für eine erfolgreiche Umleitung ist natürlich, daß das Kommando auch von der Standardeingabe liest bzw. auf die Standardausgabe ausgibt.

Beispiel:

```
$ echo hallo > gruss
```

```
$ cat gruss
hallo
```

15.1.2 Umlenkung der Standardfehlerausgabe

Will man nur die Standardfehler umlenken, schreibt man vor ´>´noch die Kanalnummer, 2.

Beispiel:

```
% cc hello.c 2> hello.err
```

15.1.3 Weitere Umleitungen

Will man sowohl die Ausgabe als auch die Fehlermeldungen umleiten , kann man dies auf folgende Arten bewerkstelligen:

... 1> *file.ok* 2> *file.err*	Umleitung der Standardausgabe nach *file.ok* und der Fehlermeldungen nach *file.err*
... 2>&1 > *file.out*	Kanal 2 (Standard-Fehlerausgabe) mit Kanal 1 (Standard-Ausgabe) verbinden und nach *file.out* umleiten

15.1.4 Logdateien

Gibt man bei der Ausgabe-Umlenkung nicht ´>´, sondern ´>>´ an, wird eine bestehende Datei nicht überschrieben, sondern die Ausgabe wird an die angegebene Datei angehängt. Falls die Datei nicht exisitert, wird sie angelegt.

Beispiel:

```
$ who >> logged
```

15.1.5 Eingabeumleitung, die Zweite

Auch bei der Eingabeumleitung ist es möglich, zwei ´<<´ anzugeben.

Beispiel:

```
$ cat << tschuess
Was wird wohl passieren?
... und tschuess
Was wird wohl passieren?
... und
$
```

Es wird solange von der Eingabe gelesen, bis das Wort erscheint, das hinter ´<<´steht (also *tschuess* in unserem Beispiel) - oder bis das EOF-Zeichen eigelesen wird.

15.2 Variablen

Variablen spielen beim Aufsetzen des Environments (s. 15.3 »Das Environment«) und bei der Programmierung von Shell-Skripts eine Rolle. Wir werden später nochmals darauf eingehen.

Im Gegensatz zu Programmiersprachen wie z.B. C oder Pascal müssen in der Bourne-Shell Variablen nicht deklariert werden. Es gibt auch keine verschiedene Typen - Variablen in der Bourne-Shell sind immer vom Typ String.

Das Anlegen von Variablen geschieht mit der Zuweisung:

var=wert

Vor und nach dem Gleichheitszeichen darf *kein* Leerzeichen stehen!

So, nun wissen wir, wie wir eine Variable anlegen. Aber wie kann ich darauf zugreifen? Dazu muß man vor dem Variablennamen ein ´$´ davor schreiben, ohne Leerzeichen zwischen ´$´ und Variablennamen.

Beispiel:

```
$ sound_path=/usr/demo/sound
$ echo $sound_path
/usr/demo/sound
$
```

Man kann auch den Variablennamen klammern - entweder mit runden Klammern () oder mit geschweiften Klammern {}. Dies ist dann nötig,

wenn kein Leerzeichen oder anderes Trennzeichen hinter dem Variablennamen steht.

Beispiel:

```
$ $(sound_path)/play ${sound_path}/SOUNDS/rooster.au
```

Greift man auf Variablen zu, die die Shell nicht kennt, gibt es keine Fehlermeldung, sondern die Shell liefert eine leere Zeichenkette zurück.

15.2.1 Der Prompt

Die Bourne-Shell benutzt im interaktiven Betrieb den Inhalt von zwei Variablen für den Prompt:

PS1	primärer Prompt: der primäre Prompt erscheint, wenn die Bourne-Shell auf Kommando-Eingaben wartet. Voreinstellung: $
PS2	sekundärer Prompt: dieser Prompt erscheint, wenn ein Kommando noch nicht abgeschlossen wurde und die Shell weitere Eingaben zu diesem Kommando erwartet. Voreinstellung: >

Beispiel:

```
$ PS1="ready> "
ready> PS2="    ---> "
ready> for x in (eins zwei)
    ---> do
    ---> echo und $x
    ---> done
und eins
und zwei
ready>
```

Wenn man diese Variable exportiert, stehen diese beiden Prompts auch weiteren Shells zur Verfügung, die von dieser Shell aus aufgerufen werden. Dies führt uns zum nächsten Kapitel:

15.3 Das Environment

Aus einer Variablen wird eine Environment-Variable, indem man sie „exportiert“:

export [var1 ...]

Werden keine Variablen nach *export* aufgelistet, so gibt die Bourne-Shell eine Aufstellung über sämtlichen exportierten Variablen aus:

```
$ EDITOR=vi
$ PRINTER=laser_1
$ export EDITOR PRINTER
$ export
EDITOR=vi
PRINTER=laser_1
$
```

Im Gegensatz zu normalen Variablen werden Environment-Variablen weitervererbt, d.h. wird aus der Shell eine weitere Shell (oder auch ein Shell-Skript) aufgerufen, so steht in der neuen Shell (bzw. Shell-Skript) die Environment-Variable mit dem zuletzt zugewiesenen Wert zur Verfügung.

15.3.1 Einrichten der Benutzerumgebung

Damit man nicht jedesmal sein Environment von Hand aufsetzen muß, gibt es die Datei **.profile**. Darin kann man die gewünschten Kommandos aufrufen und Voreinstellungen vornehmen, die für die **login-Shell** gelten sollen (s. Abbildung 26: ».profile«).

Was heißt nun *login-Shell*? Das ist die Shell, in der man sich unmittelbar nach dem Einloggen befindet. Dabei werden zwei *Profile*s durchlaufen:

/etc/profile	dies ist das *Profile*, das vom System bzw. Administrator vorgegeben wird. Es enthält Einstellungen, die auf das jeweilige System abgestimmt sind (oder zumindestens sein sollten)

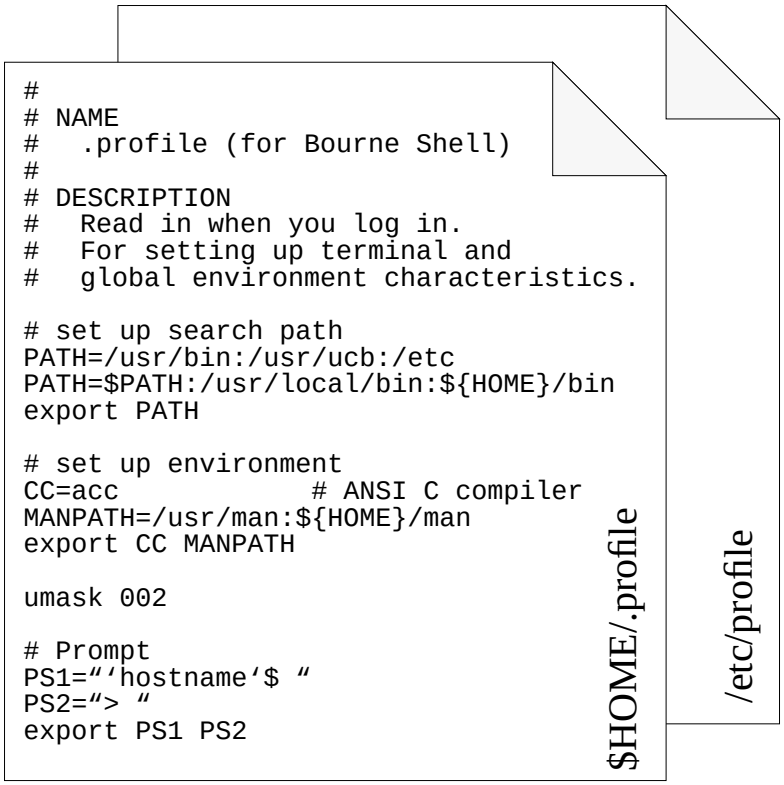


Abbildung 26: .profile

~/.profile	das ist Ihr ganz persönliches *Profile*, das sie sich selbst nach Lust und Laune einrichten können.

Dabei wird zuerst */etc/profile* durchlaufen, dann das *.profile* im Heimatverzeichnis. Fehlt eines der beiden Dateien, wird es auch nicht durchlaufen (eigentlich logisch, oder?).

Was bringt man am besten in seiner *.profile* Datei unter? Welche Einstellungen kann oder sollte man hier vornehmen? Hier eine kleine Orientierung:

- Aufsetzen des Suchpfads (*PATH*=...)
- Ausgabe von irgendwelchen Begrüßungsmeldungen oder sogenannten *Messages-of-the-Day*
- Setzen der Terminal-Charakteristik (*stty* ...)
- Setzen der Datei-Maske (*umask*...)
- Setzen des Prompts
- Setzen von Environment-Variablen
- Sonstiges (z.B. Starten des Fenstersystems, ...)

Vergessen Sie nicht, Ihre Environment-Variablen zu *export*ieren, sonst stehen sie in Subshells nicht zur Verfügung.

16 Die Korn-Shell

Die Korn-Shell ist eine Weiterentwicklung der Bourne-Shell. Gegenüber der Bourne-Shell bietet sie folgende Erweiterungen:

- Befehlszeileneditor (*vi*- oder *emacs*-orientiert)
- History-Mechanismus
- Alias-Definitionen
- verbesserte Jobsteuerung
- eingebaute Integer-Arithmetik
- Arrays

16.1 Vergangenheitsbewältigung

Gegenüber der Bourne-Shell besitzt die Korn-Shell einen eingebauten **History-Mechanismus**. Dieser wird über folgende zwei Variablen gesteuert:

HISTSIZE	damit wird das Erinnerungsvermögen der Korn-Shell festgelegt; standardmäßig erinnert sich die Korn-Shell an die letzten 128 Kommandos.

HISTFILE	Festlegung der History-Datei; vor dem Ausloggen wird hier die History abgespeichert und steht beim nächsten Einloggen wieder zur Verfügung. Die Standardeinstellung zeigt auf die Datei *~/.sh_history.*

Mit dem *history*-Kommando kann man sich die letzten 16 Befehle auflisten lassen. Will man es etwas genauer spezifizieren, kann man sich des *fc*-Befehls bedienen.

16.1.1 Editier-Modus

Der Editier-Modus profitiert ebenfalls vom History-Mechanismus. Aktiviert wird er über die *Esc*-Taste[8]. Je nach Voreinstellung kann man sich dann mit den entsprechenden Cursor-Kommandos in der Geschichte hoch und runter bewegen und mit den entsprechenden Editorkommandos editieren und manipulieren. Mit *Return* wird das editierte Kommando abgeschickt.

Zwei Editier-Modi stehen zur Verfügung: der *vi*-Modus und der *emacs*-Modus. Mehrere Möglichkeiten gibt es, den Editier-Modus festzulegen:

- über die EDITOR-Variable
- über die VISUAL-Variable
- über das Kommando *set -o*

Als gültige Werte werden nur *vi* oder *emacs* akzeptiert. *set -o* überschreibt die anderen Einstellungen, während die *VISUAL*-Variable Priorität vor der *EDITOR*-Variablen hat. Falls gar nichts angegeben wird, wird der *vi* als Standard-Einstellung angenommen.

8. Unter SunOS 4.x darf man sich dazu nicht in einem *cmdtool* befinden.

16.2 Noch mehr Suchmuster

Gegenüber den üblichen Suchmustern (neudeutsch: Pattern) ´*´, ´?´ und ´[...]´ sind die Suchmuster erweitert worden:

*(*pattern*)	*pattern* kann beliebig oft vorkommen
?(*pattern*)	*pattern* kann 0 oder 1 mal vorkommen
+(*pattern*)	*pattern* kann 1 oder mehrmal vorkommen
@(*pattern*)	*pattern* darf genau 1 mal vorkommen
!(*pattern*)	Negation

Beispiel:

```
$ cd /bin
$ ls u*
u370     u3b5     units      uucp      uuname  uuto
u3b      unadv    unix2dos   uudecode  uupick  uux
u3b15    uname    unpack     uuencode  uusend
u3b2     uniq     unwhiteout uulog     uustat
$ ls *(u)name
uname   uuname
$ ls ?(u)name
uname
$ ls +(u)name
uname   uuname
$ ls @(u)name
uname
$ ls !(u)name
basename    dname       hostname    uuname
dirname     domainname  logname
$
```

16.3 Alias-Mechanismus

Der Alias-Mechanismus erlaubt die Vergabe von Kurznamen für Kommandos.

alias [name[=wortliste]]

Wird ’alias’:

- ohne Argumente aufgerufen, werden alle momentan definierten Aliase angezeigt
- mit ’name’ als Argument aufgerufen, wird der Wert des Aliases ’name’ angezeigt
- mit den Argumenten ’name’ und ’wortliste’ aufgerufen, wird das Alias ’name’ neu definiert

16.3.1 Aliase definieren

Die Definition von Aliases ist sinnvoll für:

- die Abkürzungen von Kommandos
- das Umbenennen von Kommandos
- die Vorbelegung von Kommandos mit Optionen bzw. Argumenten
- die Abkürzungen von Kommandosequenzen

’wortliste’:

- kann eine beliebige Kommandozeile sein
- sollte in Hochkommas gesetzt werden, um die Interpretationvon Metazeichen (*?[]|...) durch die Shell bereits bei der Alias-Definition zu unterdrücken
- kann ’name’ enthalten, jedoch nur am Anfang

Beispiele:

```
$ alias h=history
$ alias dir=’ls -l’
$ alias ll=’ls -CF’
$ alias stat=’pwd;date;who’
$ alias rm=’rm -i’
```

16.3.2 Aliase löschen

Dem Löschen von Aliases dient das build-in Kommando ’unalias’

unalias name(n)

16.3.3 Aliase exportieren

Aliase stehen nur in der Shell zur Verfügung, in der sie definiert wurden. Im Gegensatz zur C-Shell kann man sie aber auch exportieren. Dazu dient die Option -x:

alias -x name[=wortliste]

16.3.4 Tracking

Tracking läßt sich über

set -o trackall

aktivieren. Was heißt *Tracking*? Ist Tracking eingestellt, legt die Korn-Shell für jedes Kommando, das sie ausgeführt hat, intern einen Alias an. Beim nächsten Aufruf eines ge*track*ten Kommandos muß sie das Kommando nicht mehr im Suchpfad suchen, sondern kann ihn mit dem absoluten Pfad aufrufen, den sie sich beim ersten Aufruf im alias vermerkt hat.[9]

Beispiel:

```
$ alias wc
alias not found
$ wc /etc/hosts
...
$ alias wc
wc=/bin/wc
$
```

Mit *alias -t* lassen sich alle ge*track*ten Kommandos auflisten.

Der Alias bleibt solange gültig, bis er mit *unalias* aufgehoben wird oder bis der Suchpfad geändert wird.

9. Die Bourne-Shell kennt hier ein anderes Verfahren: den Hashing-Algorithmus. Dabei wird beim Aufsetzen des Suchpfades intern eine Hash-Tabelle aufgebaut, um den späteren Suchvorgang zu beschleunigen.

16.4 Alles unter Kontrolle?

Jedes Kommando und jede Pipeline, die sich in Ausführung befinden, wird als **Job** bezeichnet. Build-in Kommandos der Korn-Shell erlauben die Kontrolle dieser Jobs. Ist die **Job-Kontrolle** nicht gesetzt, kann sie über *set -o monitor* oder *set -m* aktiviert werden.

Ein Job kann:

- im Vordergrund aktiv sein
- im Hintergrund aktiv sein
- suspendiert, d.h. angehalten sein

Im Vordergrund kann zu einem Zeitpunkt nur ein Job aktiv sein, während beliebig viele Jobs im Hintergrund laufen bzw. suspendiert sein können.

Mit der Korn-Shell lassen sich:

- Vordergrund Jobs anhalten und wieder fortsetzen
- Jobs vom Vorder- in den Hintergrund verlagern und umgekehrt
- Hintergrund Jobs anhalten und wieder fortsetzen
- Jobs abbrechen

Ein Job wird über seine Jobnummer angesprochen. Wird keine Jobnummer angegeben, beziehen sich Operationen immer auf den aktuellen Job; das ist der Job, auf den zuletzt operiert wurde.

Soll ein Job im Hintergrund ausgeführt werden, sollte seine Ausgabe umgelenkt werden. Ansonsten kann sich seine Ausgabe mit der Ausgabe von anderen Kommandos vermischen und dies führt im allgemeinen zur Verwirrung („von welchem Job kommt jetzt plötzlich die Ausgabe her?“).

16.4.1 Informationen zu aktuellen Jobs anzeigen

Informationen zu allen vorhandenen Hintergrund Jobs und angehaltenen Jobs lassen sich mit dem built-in Kommando ’jobs’ anzeigen.

jobs

Beispiel:

```
$ jobs
[1] - Stopped find /home -name prg1.c -print >
prg1.lis
[2] + Stopped cat /etc/termcap > term.dsc
[3] Running shelltool
```

Die Informationen sind folgendermaßen zu interpretieren. Die

- 1.Spalte zeigt die **Jobnummer** an
- 2.Spalte zeigt den aktuellen Job (+) und den vorigen Job (-) an
- 3.Spalte zeigt den **Jobstatus** an
- 4.Spalte zeigt die zugehörige Kommandozeile an

Folgende Jobstati sind zu unterscheiden:

Running	Job befindet sich in Ausführung
Stopped	Job ist momentan angehalten
Done	Job wurde normal beendet
Terminated	Job wurde abgebrochen

Endet die Ausführung eines Hintergrundjobs, wird dies dem Anwender immer vor Ausgabe des nächsten Prompts angezeigt.

16.4.2 Vordergrund Jobs anhalten und wieder fortsetzen

Das Anhalten eines Vordergrund Jobs:

- erfolgt mit der Supend-Taste, i.d.R. Ctrl-z
- ist Voraussetzung, um einen Job in den Hintergrund zu verlagern

Ein Job wird durch das Kommando ’fg’ im Vordergrund fortgesetzt.

fg [%Jobnummer(n)]

Wird ’fg’:

- kein Argument übergeben, bezieht sich das Kommando auf den aktuellen Job
- mit mehreren Jobnummern aufgerufen, werden diese Jobs sukzessive im Vordergrund fortgesetzt

Beispiel:

```
$ find /home -name prg1.c -print > prg1.lis
ctrl-z
$ cat /etc/termcap > term.dsc
ctrl-z
$ jobs
[1] - Stopped find /home -name prg1.c -print >
prg1.lis
[2] + Stopped cat /etc/termcap > term.dsc
$ fg %1
```

16.4.3 Jobs verlagern

Ein Job wird durch das Kommando ’bg’ im Hintergrund fortgesetzt.

bg [%Jobnummer(n)]

Voraussetzung hierfür ist, daß der Job im Vordergrund zunächst angehalten wurde.

Beispiel:

```
$ find /home -name prg1.c -print > prg1.lis
ctrl-z
$ cat /etc/termcap > term.dsc
ctrl-z
$ jobs
[1] - Stopped find /home -name prg1.c -print >
prg1.lis
[2] + Stopped cat /etc/termcap > term.dsc
$ bg %1 %2
$ jobs
[1] - Running find /home -name prg1.c -print >
prg1.lis
[2] + Running cat /etc/termcap > term.dsc
$ fg %1
```

16.4.4 Jobs abbrechen

Ein Job kann mit dem Kommando ’kill’ abgebrochen werden.

kill Jobnummer(n)

’kill’ bewirkt, daß:

- dem Job ’Jobnummer’ das Signal SIGTERM gesendet wird

- die Korn-Shell die Beendigung des Jobs meldet

Beispiel:

```
$ jobs
[1] + Stopped find /home -name prg1.c -print >
prg1.lis
[2] - Running cat /etc/termcap > term.dsc
$ kill %1
[1] Terminated find /home -name prg1.c -print >
prg1.lis
$
```

17 Shell-Skripts

Dieses Kapitel soll keinen Programmierkurs in die Shell-Skript-Programmierung darstellen. Dazu sei auf weiterführende Literatur bzw. Kurse verwiesen. Vielmehr soll er nur einen groben Überblick bieten, was sich hinter den verschiedenen Shell-Skripts verbirgt, um am Ende dieses Kapitel in der Lage zu sein, selber bestehende Skripts zu verstehen, evtl. zu verändern oder gar selbst einfache Skripts selber erstellen zu können.

17.1 Was sind Shell-Skripts?

Wie schon in Kap. 13.2 »Verfügbare Shells« erwähnt, sind Shell-Skripts mit den Batch-Dateien unter DOS vergleichbar. Es handelt sich um Programme, die von der Shell eingelesen und interpretiert werden. Dazu stehen die üblichen Kontrollstrukturen (Verzweigung, Schleifen, Sprünge) zur Verfügung, mit denen sich verschiedenen andere Unix-Kommandos verknüpfen lassen.

Woran erkennt nun die Shell, ob es sich bei einer Datei um ein Skript handelt? Jedenfalls nicht an der Endung, die spielt für die Shell keine Rolle. Nein, es muß sich um eine ausführbare Datei handeln. Wenn man also aus einer normalen Textdatei ein Skript machen will, braucht man nur das *x*-Bit (Ausführungs-Bit) zu setzen, z.B. mit „chmod +x ...“.

Was sind Skripts?

- Folge von Unix-Kommandos
- Verknüpfung der Kommandos mit Hilfe von Kontrollstrukturen
- in ausführbarer Datein abgelegt
- Entstehung eines neuen Kommandos

Abbildung 27: Was sind Skripts?

Worin unterscheiden sich Skripts von normalen Programmen? Bei normalen Programmen handelt es sich um eine Binärdatei, die am Anfang eine digitale Kennung besitzt. Aber auch Skripte haben am Anfang eine Kennung. Schauen wir uns dazu Abbildung 28: »Beispiel für ein Skript« an: In der ersten Zeile steht die Kennung, um was für ein Skript es sich handelt:

```
#! /bin/csh
```

Es handelt sich um ein Skript der C-Shell, erkennbar an */bin/csh* - dies ist der Aufrufpfad für die C-Shell. Andere mögliche Shells sind

Pfad	Name der Shell
/bin/sh	Bourne-Shell
/bin/csh	C-Shell
/bin/ksh	Korn-Shell
/bin/bash	GNU-Shell (Bourne-Again-Shell)

Tabelle 16: verschiedene Shells

Wichtig dabei sind die ersten beiden Zeichen: „**#!**“. Fehlen diese beiden Zeichen, wird als Default-Einstellung die *Bourne-Shell* genommen. Von dieser einen Ausnahme abgesehen, leitet das #-Zeichen einen Kommentar bis zum Zeilenende ein.

17.2 Wozu braucht man Skripts?

```
#! /bin/csh
#
#   Change Words
#
#   The 1. argument will be replaced by the 2. argument. The
#   3. argument is the file, in which the words should be
#   replaced. The output  will be saved in <filename>.bak.
#
#   created: 28-APR-88 by O. Boehm (SEL - PS/ECST, ph.262)
#

#   checking the correct syntax
@ arguments = ${#argv}
if ($arguments < 3) then
    echo Usage: chwords word1 word2 files...
    exit (1)
endif

#    replacing the expressions
@ i = 3
while ($i <= $arguments)
    set filename=$argv[$i]
    mv $filename $filename.bak
    cat $filename.bak|sed -e s/$argv[1]/$argv[2]/g        \
        > $filename
    @ i = $i + 1
end
```

Abbildung 28: Beispiel für ein Skript

Wozu braucht man Skripts?

- um mehrere Kommandos zu einem Kommando zusammenzufassen
- um die Arbeit mit UNIX zu erleichtern
- schnelle Erstellung
- leicht erlernbar

Abbildung 29: Wozu Skripts?

V

Unix als Programmier-Umgebung

18 make

18.1 Einführung

Wie schön wäre es doch, wenn man zum Rechner einfach „mach mal“ sagen könnte, und der Rechner würde das Gewünschte tun? Genau zu diesem Zweck gibt es das *make*-Kommando. Doch ganz so einfach ist es doch nicht. Ein

```
make love
```

wird vom Rechner leider mit

```
don´t know how to make ´love´
```

beantwortet[1]. Vor einem erfolgreichen Abschluß dieser Aufforderung ist erstmal Aufklärungsarbeit angesagt.

18.1.1 Ein einfaches „Makefile“

Bevor Unix „love“ machen kann, muß man es ihm erstmal beibringen. Basis für unsere Aufklärungsbemühungen ist das „Makefile“[2], in dem

1. Leider geben nicht alle *make*-Variante diese schöne Fehlermeldung aus. GNU-make z.B. meint dazu:

```
make: *** No rule to make target 'love'. Stop.
```

drinsteht, von welchen Dateien „love“ abhängt, und wie bitteschön der Rechner damit „love“ machen soll:

```
#
#   Makefile to make love
#

love: love.c
      gcc love.c -c love
```

Damit weiß der Rechner jetzt, wie er zu „love“ kommt: dazu wird die Datei „love.c“ benötigt. Ist sie vorhanden, wird das Kommando

```
gcc love.c -c love
```

ausgeführt, das den GNU-C-Compiler mit „love.c“ startet und „love“ als ausführbares Programm generiert.

18.1.2 Noch ein Makefile

18.1.3 Regel-Werk

Im Makefile werden die Abhängigkeiten eines Programms zu den dazu gehörenden Sourcen, Bibliotheken und Objekt-Datein anhand von Regeln („rules“) definiert. Eine Regel hat dabei folgendes Aussehen:

```
target: dependencies
<tab>   actions
```

`target` ist das Zeil, das gebildet werden soll, die `dependencies` geben an, wovon das `target` abhängig ist. Die `actions` geben an, wie dieses `target` gebildet werden kann. Wichtig dabei ist, daß die `actions` mit einem Tabulator-Zeichen eingeleitet werden. Es dürfen durchaus mehrere `actions` angegeben werden, aber alle müssen mit dem Tabulator-Zeichen beginnen.

---

2. mit großen oder kleinem ´M´

Dies ist eine tückische Fehlerquelle, da sich das Tabulatorzeichen optisch nicht von 8 Leerzeichen unterscheidet. Leider sieht dies das *make*-Kommando etwas anders, und bringt stattdessen oftmals eine wenig hilfreiche Fehlermeldung.

Kommentare werden mit # eingeleitet. Das Ende des Kommentars ist das Zeilenende.

18.1.4 Abhängigkeiten

Woher weiß *make*, was es überhaupt zu machen hat, wenn ich ihn mit „`make love`“ beauftrage? Nun, zum einen gebe ich ihm ja die Regeln und Abhängigeiten mit. Aber wenn das Target „love“ erfolgreich generiert wurde, und ich aus Versehen nochmals ein „`make love`“ starte, wird er es mit

```
´love´ is up to date
```

ablehnen. *make* erkennt dies daran, wenn das Target („love“) ein neueres Datum als die abhängigen Dateinen („love.c“) hat.

18.1.5 Pseudo-Targets

Darunter versteht man Targets, die keine Abhängigkeiten besitzen. Damit können Sie aber nie „up to date“ werden, d.h. beim Aufruf von *make* mit diesem Target werden die entsprechenden Aktionen immer ausgeführt.

Hierzu ein Beispiel: nach erfolgreicher Compilation sollen sämtliche Objekt-Dateien und *core*-Dateien[3] gelöscht werden. Dies erreicht man über folgende Regel:

```
clean:
        rm *.o core
```

3. *core*-Dateien werden bei einem Programm-Absturz angelegt („*core dumped*“)

und anschließendem Aufruf von „`make clean`“.

Weil diese Target nie erzeugt wird, bezeichnet man solche Targets als *Pseudo*-Targets.

18.2 Makros

18.2.1 Syntax

Makros innerhalb eines Makefiles sind eigentlich nichts anderes wie Variablen. Sie werden mit

```
NAME = text
```

angelegt, wobei `text` aus mehreren Wörtern bestehen kann. Er kann sich auch über mehrere Zeilen erstrecken, wenn die Zeilen als letztes Zeichen ein „\“ enthalten. Fehlt `text`, wird damit ein leeres Makro angelegt.

Das „\“ muß als allerletztes Zeichen in der Zeile stehen, da es die Bedeutung des nächsten Zeichens (das Zeilenende) aufhebt.

Es hat sich eingebürgert,daß Makro-Namen großgeschrieben werden. Auch sollte man Sonderzeichen vermeiden, auch wenn viele *make*-Implementationen damit zurechtkommen. „#“ in Makros sind nicht möglich, da damit ein Kommentar eingeleitet wird.

Die Verwendung eines Makros kann mit

```
$NAME
$(NAME)
${NAME}
```

erfolgen.

18.2.2 Vordefinierte Makros

Es gibt schon eine Reihe von vordefinierten Makros. Z.B. ist auf vielen Systemen CC als

```
CC = cc
```

definiert. Wenn man wissen will, welche Makros vordefiniert sind, kann man in folgenden Dateien nachschauen:

```
/usr/lib/make/default.mk                (SunOS 4)
/usr/share/lib/make/make.rules          (Solaris 2)
```

Sollten Sie nicht fündig werden, fragen Sie ihren System-Administrator oder das Unix-System („man make“).

18.2.3 Makro-Übergabe

Es gibt auch die Möglichkeit, Makros beim Aufruf von *make* mit zu übergeben:

```
make love CC=cc
```

Soll ein Makro aus mehreren Wörten bestehen, muß dies beim Aufruf gequotet werden:

```
make love CFLAGS="-O -DNDEBUG"
```

18.2.4 Environment-Variablen

Environment-Variablen können im Makefile wie Makros angesprochen werden. Angenommen, ich setzte eine Variable CCC, die den GNU-C++-Compiler enthalten soll:

```
setenv CCC g++              (C-Shell)
CCC=g++; export CCC         (Bourne-Shell)
```

Diese können dann im Makefile wie Makros angesprochen werden, z.B. wie in folgender Regel:

```
hello: hello.cc
        $(CCC) hello.cc -o hello
```

18.2.5 Prioritäts-Regeln

Gleichnamige Variablen werden nach folgenden Prioritätsregeln (in aufsteigender Reihenfolge) aufgelöst:

1. interne Variablen
2. Environment-Variablen
3. Variablen über Kommandozeile

D.h. Variablen, die über die Kommandozeile gesetzt werden, gewinnen immer.

18.2.6 String Substitution[4]

Makros können wiederum von anderen Makros abhängen oder aus ihnen zusammengesetzt sein. Dabei ist auch „Makro-Substitution“ möglich:

```
#   Makros
C_FILES   = life3.c genlist.c
H_FILES   = genlist.h
SRC_FILES = ${C_FILES} ${H_FILES}
PROG      = life3

#   Beispiel fuer „String Substitution“
OBJ_FILES = ${C_FILES:.c=.o}
CPP_FILES = ${C_FILES:.c=.i}
```

Ohne String-Subsitution müßte man `OBJ_FILES` und `CPP_FILES` konventionell definieren:

```
#   ohne „Makro Substitution“
OBJ_FILES = life3.o genlist.o
CPP_FILES = life3.i genlist.i
```

18.2.7 Interne Makros

Hier eine Liste der internen Makros:

4. nicht auf allen make-Varianten vorhanden

`$@`	Name des aktuellen Targets
`$$@`	Name des aktuellen Targets (nur auf der rechten Seite einer Regel, in den Abhängigkeiten, zulässig)
`$?`	Name der abhängigen Dateien, die neuer als das Target sind (nur in normalen Regeln zulässig)
`$<`	Name der abhängigen Dateien, die neuer als das Target sind (nur in Suffix-Regeln zulässig)
`$*`	Name der abhängigen Dateien, die neuer als das Target sind - aber ohne Endung (nur in Suffix-Regeln zulässig)
`$%`	Name der entsprechenden **.o**-Datei, falls das Target eine Bibliothek ist

Außer bei „$?“ kann bei Pfadnamen sowohl auf den Dateiname, als auf auf den Verzeichnisnamen über folgende Modifier zugegriffen werden[5]:

D	der Verzeichnisteil (z.B. ${*D}, ${<D}, ${@D}, $${@D})
F	der Dateiname (z.B. ${*F}, ${<F}, ${@F}, $${@F})

Beispiel für interne Makros:

```
#   Beispiel fuer „$?“
libonew.a : libonew.o lwp_new.o
        ar r $@ $?
```

Aufgelöst (ohne „$@“) lautet diese Regel:

```
libonew.a : libonew.o lwp_new.o
        ar r libonew.a $?
```

5. nicht auf allen Systemen vorhanden

Das Makro „$?“ hängt davon ab, ob `libonew.a` oder `lwp_new.a` (oder beide) neuer als `libonew.a` sind.

18.3 Suffix-Regeln

Jede Programmiersprache bringt i.d.R. ihre eigene Datei-Endung mit. So erkennt man C-Souren daran, daß sie mit *.c* aufhören, Pascal-Programme hören mit *.p* auf, Fortran-Programme mit *.f* auf (s. Tabelle 17: »Datei-Endungen«).

18.3.1 Was sind Suffix-Regeln?

Mit Suffix-Regeln kann man *make* beibringen, wie er z.B. aus C-Sourcen (*.c*) *.i*-Dateien basteln kann:

```
#
#   Beispiel fuer „Suffix Rules“
#

.SUFFIXES : ${SUFFIXES} .c .o .i

.c.i:
        $(CPP) $(DEBUGFLAGS) $< > $@

#   Null Suffix
.c :
        ${CC} $< -o $@
```

Die Zeile „`.SUFFIXES : ...`“ gibt an, welche Endungen alle bekannt gemacht werden sollen. Man kann auch, wie im obigen Beispiel geschehen, die vorhanden Liste erweitern, in dem man auf der rechten Seite mit `${SUFFIXES}` aufführt.

Die Regel „`.c.i:`“ gibt an, wie *make* aus einer *.c*-Datei eine *.i*-Datei erzeugen soll: Aufruf des Preprozessors `$(CPP)` mit der *.c*-Datei („`$<`“) und Umlenkung der Ausgabe in die *.i*-Datei („`> $@`“).

Beispiel:

```
make life3.i
```

Endung	Bedeutung
.c	C Source
.cc, .C, .cpp	C++ Source
.f	Fortran
.gz	komprimierte Datei (gzip)
.h	Header-Datei (C, C++)
.i	Dateien nach dem C-Preprozessorlauf
.l	lex-Dateien
.o	Objekt-Dateien
.p	Pascal-Dateien
.s	Assembler-Dateien
.y	yacc.Dateien
.Z	komprimierte Dateien (compress)

Tabelle 17: Datei-Endungen

Falls *life3.i* nicht existiert oder älter als *life.c* ist, wird *make* folgendes Kommando starten:

```
cc -E life3.c > life3.i
```

18.3.2 Null-Suffix-Regeln

Bei den Null-Suffix-Regeln fehlt die zweite Endung:

```
#   Null Suffix
.c :
        ${CC} $< -o $@
```

Damit verrät man *make*, wie es einer *.c*-Datei eine ausführbares Programm (ohne Endung) erzeugt.

Beispiel

```
make world
```

Falls *world.c* existiert, wird folgendes Kommando gestartet:

```
cc world.c -o world
```

18.3.3 Eingebaute Suffix-Regeln

Wie *make* aus einer *.c*-Datei eine *.o*-Datei erzeugen soll, brauch man ihm nicht erzählen - diese Suffix-Regel hat er schon eingebaut. Daneben kennt es eine Reihe von weiteren Suffix-Regeln. Je nach System sind diese Regeln fest eincompiliert oder er bezieht sie aus einer der folgenden Dateien:

```
/usr/lib/make/default.mk              (SunOS 4)
/usr/share/lib/make/make.rules        (Solaris 2)
```

Sollte die Datei unauffindbar sein, hilft manchmal auch ein Blick in’s Online-Manual („*man make*“).

C(++)

SCCS und RCS

Bibliotheken

lex und yacc

18.3.4 Konflikt-Behandlung

Betrachten wir folgende Regel:

```
life3 : life3.o genlist.o
```

Dummerweise sind mehrere Suffix-Regeln definiert, wie ein *.o*-Datei erzeugt werden kann, z.B. aus einer *.c*-Datei oder einer *.cc*-Datei oder anderen Datei-Typen. Wenn nur eine entsprechende *.c*-Datei da ist, ist alles klar. Aber was, wenn nicht? Dann wird die Such-Reihenfolge durch die *SUFFIXES*-Liste bestimmt:

```
.SUFFIXES : .o .c .cc .f
```

Es wird zuerst nach einer *.c*-Regel gesucht (*.o* nach *.o* macht keinen Sinn), dann nach einer *.cc*-Regel, und so fort.

18.3.5 Eigene Suffix-Regeln

Eigene Suffix-Regeln wird man dann einführen, wenn

- die eingebauten Regeln nicht ausreichen
- die eingebauten Regel Schrott sind

18.4 Kommandos

Kommandos werden in einer eigenen Sub-Shell ausgeführt, d.h. es stehen alle Features wie Wildcards, Ein-/Ausgabe-Umlenkung, ... zur Verfügung.

18.4.1 Wildcards

Man muß unterscheiden zwischen Wildcards ('*', '?', '[]') innerhalb eines Kommandos und zwischen Wildcards in den Abhängigkeiten einer Regel:

```
print : *.c
        lpr *.c
```

Das Wildcard „`*.c`“ als Abhängigkeit wird von *make* aufgelöst, das Wildcard im Kommando („`lpr  *.c`“) wird von der Shell aufgelöst. Dummerweise rechnet *make* die „*Hidden Files*“[6] dazu, während die Shell da anderer Meinung ist.

18.4.2 Zeilenende

Jede Zeile im Aktions-Teil wird in einer eigene Sub-Shell ausgeführt. Das ist aber nicht immer gewünscht.

Beispiel:

```
clean :
        cd backup
        rm *
```

Dieses Kommando das `cd`- und `rm`-Kommando in zwei unterschiedlichen Sub-Shells ausführen mit der Folge, das sämtliche Dateien im aktuellen Verzeichnis (und damit letztendlich auch das Makefile) gelöscht werden würde.

Will man mehrere Kommandos in derselben Shell ausführen, so müssen sie durch eine Strichpunkt (';') getrennt werden und in einer Zeile stehen:

---

6. das sind die, die mit einem Punkt ('.') beginnen

```
clean :
        cd backup ; rm *
```

Man kann sie auch auf mehreren Zeilen verteilen, indem man sie mit dem Backslash ('\') als allerletztes Zeichen in der Zeile verbindet:

```
clean :
        cd backup ;  \
        rm *
```

18.4.3 Skript-Programmierung

Sämtliche Programmierkonstrukte, die die (Bourne-)Shell anbietet, stehen auch in *make* zur Verfügung, da ja für jedes Kommando eine Sub-Shell gestartet wird.

Beispiel:

```
print :
        for f in ${SRC_FILES} ; do \
             lpr $$f ;             \
        done
```

Wichtig bei solchen Konstruktionen ist, daß die einzelnen Anweisungen durch einen Strichpunkt (';') getrennt werden und mehrere Zeilen über den Backslash ('\') verbunden werden.

Will man auf Shell-interne Variablen zugreifen, so muß man noch ein zusätzliches Dollar-Zeichen ('$') davorsetzen (siehe „`$$f`“ im obigen Beispiel).

18.4.4 Fehlercode

Wenn ein Kommando mit einen Fehlercode zurückliefert, wird *make* an dieser Stelle abgebrochen. Dabei ist unter Unix alles, was einen Rückgabewert ungleich 0 zurückliefert, ein Fehlercode.

Meistens ist dieses Verhalten ja gewünscht, z.B. beim Compilieren (was der Normalfall sein dürfte). Aber es gibt auch Situationen, in dem der Rückgabewert eines Kommandos piepegal ist, z.B. in

```
clean :
        - rm core
        - rm *.o
```

ist es nicht weiter tragisch, wenn „rm core“ nicht erfolgreich war (weil es beispielsweise nicht existiert). Der Abbruch im Fehlerfall wird durch ein vorangestelltes „-“ vermieden.

18.4.5 Ausgabe unterdrücken

Normalerweise wird jedes Kommando, das ausgeführt wird, vorher ausgegeben. Mit einem vorangestellten „@“ kann man das unterdrücken:

```
statistic :
        @ echo ""
        @ echo "      S T A T I S T I C S"
        @ echo ""
        @ echo "   Lines   Words   Bytes Modul"
        @ echo "   -------------------------------"
        @ wc ${SRC_FILES}
        @ echo ""
```

Vor allem beim *echo*-Kommando macht es keinen Sinn, daß vor der eigentlichen Ausgabe das *echo*-Kommando samt Ausgabe-String ausgegeben wird.

18.4.6 Shell-Abhängigkeit

18.4.7 Pfadnamen und Suchpfad

18.5 Besonderheiten

18.5.1 Optionen

18.5.2 Spezielle Targets

18.6 Projekt-Management

Für kleinere Programmpakete ist das Makefile meist noch recht übersichtlich. Allerdings verleiten die vielen Möglichkeiten, die *make* bietet, daß man sich schnell im Regel-Dschungel die Orientierung verliert und das Makefile immer mehr ein undurchsichtiges Eigenleben entwickelt. Und wehe, das Wissen über den Aufbau und Ablauf der Makefiles konzentriert auf einen einzelnen Spezialisten. Wenn dieser dann nicht mehr zur Verfügung steht, kann die weitere Pflege und Wartung der Makefiles den eigentlichen Entwicklungsaufwand übersteigen.[7]

Während man bei kleineren Projekten das Makefile notfalls nochmals aufsetzen kann, ist dies bei größeren Projekten oft mit erheblichem Aufwand verbunden. Daher sollte man auch (oder gerade) bei Makefiles nach dem Motto

> *„So einfach wie möglich - aber nicht einfacher“*

handeln.

Dieses Kapitel beschäftigt sich mit verschiedenen Aspekten, wie man mit *make* größere Projekte verwalten kann. In einem späteren Kapitel (18.9 »Richtlinien«) beschäftigen wir uns dann mit Makefile-Richtlinien, die die Einarbeitung und Wartung von (eigenen oder fremden) Makefiles erleichtern soll.

18.6.1 Schwierigkeiten

Einige der Probleme, die den Einsatz von *make* erschweren, sind:

- Verzeichnis-Baum

7. alles schon mal dagewesen!

- Bedingte Compilierung (`#if ... #endif`)
- versteckte Abhängigkeiten (z.B. über Header-Dateien)
- Versionierung

Manche der Probleme resultieren aus Unzulänglichkeiten des Compilers, manche resultieren aus Annahmen und Beschränkungen einiger Unix-Tools. Diese spiegeln sich z.T. in den eingebauten Suffix-Regeln wieder.

Ursprünglich war *make* nur zur Vereinfachung der Compilierung gedacht, hat sich aber über die Jahre zu einem mächtigen Entwicklungswerkzeug gemausert. Nicht zuletzt auch deswegen, weil es inzwischen eine Reihe von Tools gibt, die um *make* herum gebaut wurden, um die Einschränkungen aufzuheben (z.B. *makedepend*, s. a. Kap. 18.10 »Tools rund um „make“«).

18.6.2 Dummy-Targets

In der Praxis werden Dummy-Targets recht häuig eingesetzt, um mehrere Targets zusammenzufassen:

```
# compile all
all : anna berta carmen

anna : anna.c
        $(CC) -g -o anna anna.c

berta : berta.c
        $(CC) -g -o berta berta.c

carmen : carmen.c
        $(CC) -g -o carmen carmen.c
```

Der Entwickler braucht nur „make all“ einzugeben, und sämtliche Programme werden übersetzt. Daneben gibt es oft noch ein Target „install“, mit der die übersetzten Programme installiert werden (s. Kap. 18.9.5 »Standard Targets«).

„Timestamp“-Targets

Eine etwas subtile Variante von Dummy-Targets sind „Timestamp“-Targets. Damit bezeichne ich Targets, die zwar angelegt werden, aber nicht als Target gebraucht werden. In Wirklichkeit werden sie zur Synchronisation von Aktivitäten verwendet:

```
TIMESTAMP.strip : anna berta carmen
        strip $?
        touch $@
```

Was macht diese Target? Falls „`TIMESTAMP.strip`“ nicht existiert oder eines der abhängigen Dateien „`anna`“, „`berta`“ oder „`carmen`“ neuer ist, werden die entsprechenden Dateien ge*strip*t[8] und danach „`TIMESTAMP.strip`“ ge*touch*t, d.h. es erhält einen neuen Zeitstempel bzw. wird als leere Datei angelegt, falls sie noch nicht existiert.

Die Datei „`TIMESTAMP.strip`“ dient also nur dazu, um festzustellen, ob „`anna`“, „`berta`“ oder „`carmen`“ schon einen *strip* hinter sich haben. Diesen Trick findet man häufiger in Makefiles. Wenn Sie sich also schon immer gefragt haben, zu was Dateien der Größe „0“ gut sein sollen, hier ist eine mögliche Antwort.

Allerdings hat diese Lösung auch einen Haken: man sieht der Datei „TIMESTAMP.strip“ nicht an, zu was sie gut sein soll und ein ordnungsliebender Mensch könnte leicht auf die Idee kommen, diese Datei zu löschen, da sie die Größe „0“ hat - weg damit! Daher ist es besser, dieser Datei einen sinnvollen Inhalt zu verpassen, damit sie

1. eine Größe > 0 hat *und*
2. damit der ahnungslose Benutzer einen Schimmer bekommt, zu was diese Datei gut sein könnte.

Dies kann man z.B. durch folgende Regel erreichen:

```
TIMESTAMP.strip : anna berta carmen
        strip $?
```

8. `strip` entfernt die Symboltabelle aus dem Programm. Dadurch wird das Programm kürzer, defür kann man es nicht mehr debuggen.

```
        echo "last strip of anna, berta or carmen:"\
             > $@
        date >> $@
```

18.6.3 Rekursives *make*

Am unproblematischten ist es, wenn man sämtliche Dateien in einem einzigen Verzeichnis hat. Leider ist dieses Vorgehen bei größeren Projekten nicht praktikabel und üblicherweise hat man seine Dateien über mehrere Verzeichnisse verteilt (s. Abbildung 30: »rekursive Verzeichnis-Struktur«).

Verteiltes *make*

Eine Möglichkeit, mit dem Verzeichnisbaum fertig zu werden, besteht darin, in jedes Verzeichnis ein Makefile zu plazieren, das über das Makefile im übergeordneten Verzeichnis aufgerufen wird.

Das „Top-Level“-Makefile könnte dabei folgendermaßen aussehen:

```
SUBDIRS = src lib

all :
        for d in $(SUBDIRS); do \
            (cd $$d; make all)  \
        done
```

Voraussetzung dafür ist natürlich, daß die drunterliegende Makefiles ein Target „all“ besitzen.

18.6.4 Tips

Weitergabe von Makros

Der rekursive Aufruf von *make* ist auch dazu geeignet, Informationen und Flags durchzureichen.

Beispiel:

```
CFLAGS     =
DEBUGFLAGS = -g $(CFLAGS)
```

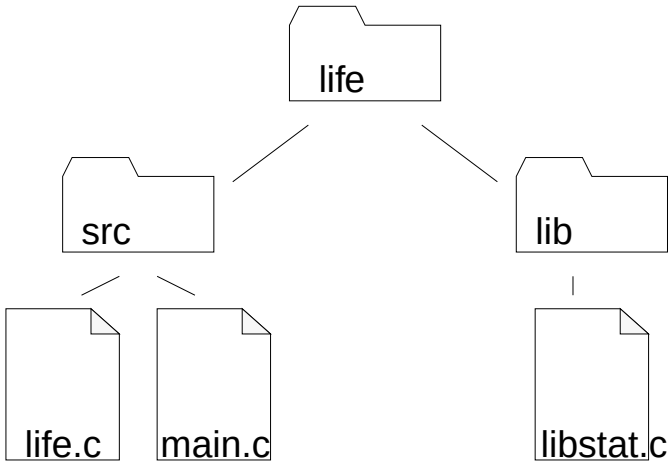


Abbildung 30: rekursive Verzeichnis-Struktur

```
testbin :
        make bin "CFLAGS=$(DEBUGFLAGS)"
```

In diesem Beispiel wird durch `make testbin` dasselbe Makefile nochmal aufgerufen, jedoch mit geänderten *CFLAGS*. Mit demselben Verfahren können auch Makros in drunterliegenden Makefiles überschrieben werden.

MAKE-Makro

Die meisten *make*-Versionen besitzen auch ein internes MAKE-Makro. Damit lautet die obere „testbin“-Regel:

```
testbin :
        $(MAKE) bin "CFLAGS=$(DEBUGFLAGS)"
```

Der Vorteil des internen MAKE-Makros ist die Weitergabe der Optionen beim Aufruf von *make*. Wird beispielsweise *make* mit der Option „-n“ aufgerufen[9], so wird damit auch alle weiteren *make*s mit „-n“ aufgerufen.

Wichtige Makros

Jedes Makefile hat seine eigene Makros. Um die Verwaltung und Verwirrung gering zu halten, sollte jedes Makefile dieselben Makro-Namen besitzen. Jeder *make*-Aufruf sollte dann diejenigen Makros weiterreichen, die wichtig sind.

Beispiel:

```
all :
        for d in $(SUBDIRS); do       \
            (cd $$d;                  \
            make all "CFLAGS=$(CLAGS) \
                LDFLAGS=$(LDFLAGS)    \
                LIBFLAGS=$(LIBFLAGS)) \
        done
```

9. „-n“ zeigt nur die Kommandos an, die *make* ausführen würde, führt sie aber nicht aus.

18.6.5 Zentrales Makefile

Der verteilte Ansatz über rekursive *make*-Aufrufen ist eine Möglichkeit. Eine andere Möglichkeit ist ein zentrales Makefile, in dem die einzelnen Dateien über ihren vollen Pfad angesprochen werden. Glücklicherweise gibt es dazu einige Tricks, die diese Sache vereinfacht.

Verzeichnisse in Makros

Die internen Makros (außer `$?`) kennen zwei zusätzliche Modifier:

- `D` für Directory (Verzeichnis)
- `F` für File (Datei)

Beispiel:

```
.c.o:
        cd ${<D}; $(CC) -c ${<F}
```

Bei Dateien im aktuellen Verzeichnis wird D durch den Punkt (.) ersetzt.

VPATH

Manche *make*-Varianten kennen einen „Viewpath“ (VPATH). Damit kann man - ähnlich wie dem Suchpfad in der Shell - Verzeichnisse angeben, in der *make* nach den entsprechenden Dateien suchen soll. Die einzelnen Dateien werden durch einen Doppelpunkt (:) getrennt.

Beispiel:

```
VPATH = test/src work/src

main.o : main.c

life.o : life.c
```

make sucht jetzt die Dateien `main.c` und `life.c` zuerst im aktuellen Verzeichnis, dann im Verzeichnis `test/src`, und zum Schluß im Verzeichnis `work/src`.

VPATH ist schon ein sehr altes Feature, das von vielen *make*s verstanden wird, aber seltsamerweise fast nie dokumentiert ist. Hier heißt es ausprobieren.

Obwohl VPATH sehr nützlich ist, sollte man damit sorgsam umgehen. Es enthält einige Fallstricke bei relativen Pfadnamen und Unterverzeichnissen. Auch besteht die Gefahr, daß man sich im Verzeichnis-Dschungel verheddert und es schwer nachvollziehen läßt, welche Dateien aus welchen Verzeichnissen das aktuelle Programm gebildet haben.

18.6.6 Bedingte Compilierung

Rekursives *make*

Oft hat man das Problem, das man seine Sourcen mit verschiedenen Flags übersetzen will. Zu Testzwecken will man sein Programm beispielsweise mit den Optionen „-g -DDEBUG“, während für das fertige Programm die Option „-O“ verwendet werden soll.

Um dies zu bewerkstelligen, gibt es mehrere Möglichkeiten. Die erste Möglichkeit verwendet den rekursiven *make*-Aufruf:

```
bin:
        make $(PROG) "CFLAGS=-O"

testbin:
        make $(PROG) "CFLAGS=-g -DDEBUG"
```

Nachteil dieser Lösung ist eine schlechtere Performance: es dauert länger, bis endlich der Compiler gestartet wird. Dafür hat diese Lösung den großen Vorteil, daß sie auf allen Rechnern mit allen *make*-Varianten funktioniert.

Conditional Makros

Eine andere Möglichkeit, mit diesem Problem fertig zu werden, bieten die „Conditional Makros“:

```
bin : $(PROG)
bin := CFLAGS = -O

testbin : $(PROG)
testbin := CFLAGS = -g -DDEBUG
```

Ursrünglich nur Bestandteil des BSD-*make*s hat es inzwischen Einzug in viels System V-*make*s gefunden. `make testbin` sucht nach den Regeln für `$(PROG)` (genauso wie `make bin`), aber mit dem Unterschied, daß die `CFLAGS` entsprechend umgesetzt werden.

Remake

Ein großes Problem bei der bedingten Compilierung sind die Objekt-Dateien: ihnen sieht man leider nicht an, mit welchen Flags sie übersetzt wurden. *make* kann leider nicht erkennen, ob es diese Objekt-Dateien nochmals neu übersetzen muß, weil sich die Optionen beim Compilieren geändert haben.

Es gibt mehrere Möglichkeiten, um aus diesem Dilemma raus zu kommen. Die erste Möglichkeiten besteht darin, erstmal „klar Schiff“ zu schaffen:

```
testbin : clean $(PROG)
testbin := CFLAGS = -g -DDEBUG

clean :
        rm -f *.o $(PROG)
```

Bevor das Programm gebildet wird, werden sämtliche Objekt-Dateien und das Programm selber gelöscht. Damit müssen sämtliche Objekt-Dateien nochmal erzeugt werden.

Der Nachteil dieser Methode ist offensichtlich: die Turnaround-Zeit steigt an, da sämtliche Sourcen nochmals übersetzt werden müssen, obwohl das vielleicht gar nicht unbedingt nötig ist.

Die „klassische“ Lösung ist etwas aufwendiger: es werden extra Abhängigkeiten eingeführt:

```
rebuild:
        make life.o FRC=FORCE

FORCE :

life.o : $(FRC)
```

Der Trick bei der Sache ist: `FRC` ist normalerweise leer, d.h. falls `life.o` existiert und „up-to-date“ ist, passiert nichts. Falls nicht, wird es aus `life.c` genieriert.

Etwas anderes sieht es aus, wenn `make rebuild` gestartet wird: jetzt hängt `life.o` plötzlich von `FORCE`, einem Dummy-Target, ab. Jetzt wird `life.o` auf jeden Fall neu generiert.

Dieses Verfahren funktioniert nur, wenn man

1. rekursive *make*s mit dem Argument `"FRC=$(FRC)"` aufruft
2. jede Datei von `$(FRC)` abhängt

Verschieden Varianten über verschiedene Targets

Eine weitere (und aufwendigere) Möglichkeiten ist die Einführung verschiedener Targets für die verschiedenen Übersetzungswege:

```
life : life.o
        $(CC) -o life life.o

life.o : life.c
        $(CC) -c -O -o life.o life.c

testlife : testlife.o
        $(CC) -o testlife testlife.o

testlife.o : life.c
        $(CC) -c -g -DDEBUG -o testlife.o life.c
```

In diesem Beispiel können zwei Versionen des „life“-Programms erzeugt werden: eine normale Version („life“) und eine Testversion („testlife“). Jede dieser Versionen hat seinen eigenen Satz von Objekt-Dateien, die zwar von denselben Sourcen abhängen, aber unterschiedlich übersetzt werden.

Verschiedene Varianten in verschiedenen Verzeichnissen

Eine Variante von der vorherigen Lösung besteht darin, die unterschiedlichen Versionen in unterschiedlichen Verzeichnisse zu plazieren, zusammen mit dem entsprechenden Makefile:

```
life : life.o
        $(CC) -o life life.o

life.o : ../src/life.c
        $(CC) -c -O -o life.o life.c
```

Dies ist das Makefile für die „Normal“-Version. Die „Test“-Version enthält folgendes Makefile:

```
life : life.o
        $(CC) -o life life.o

life.o : ../src/life.c
        $(CC) -c -g -DDEBUG -o life.o life.c
```

Gegebenenfalls kann man noch ein „Top-Level“-Makefile mit den entsprechenden Regeln aufstellen:

```
normal:
        cd normal; make life

test:
        cd test; make test
```

Varianten über Suffix-Regeln

Eine elegante Lösung ist die Einführung eigener Suffixe, z.B. das Suffix „**.tobj**“ für „Test“-Objekt:

```
.SUFFIXES : .c .o .tobj

.c.o :
        $(CC) -c -O $<

.c.tobj :
        $(CC) -c -g -DDEBUG -o $@ $<

life : life.o
        $(CC) life.o -o life

testlife : life.tobj
        $(CC) life.tobj -o testlife
```

Vorraussetzung dafür ist, daß der Compiler mitspielt. Manche Compiler sind pingelig beim Erstellen einer Objekt-Datei, während beim Linken die Endung oftmals keine so große Rolle mehr spielt.

18.6.7 Header-Dateien

Obekt-Dateien hängen nicht nur von C-Sourcen, sondern auch von Header-Dateien ab, d.h. man müßte diese eigentlich mit in den Abhängigkeiten mit aufnehmen:

```
anna.o : anna.c anna.h global.h util.h
        $(CC) -c anna.c
```

Dies wird man aber in den seltesten Fällen in Makefiles antreffen, und zwar meist aus folgenden Gründen:

- Faulheit des Programmierers
- versteckte Abhängigkeiten
- zu dynamisch
- zu großer Overhead

Glücklicherweise erhält der Programmierer hier Unterstützung vom Compiler (BSD-Compiler und Sun-Compiler): Mit der Option -M generiert der Compiler eine Liste von Abhängigkeiten, die ins Makefile übernommen werden können:

```
anna.o: anna.c
anna.o: ./anna.h
anna.o: ./global.h
anna.o: ./util.h
anna.o: /usr/include/stdio.h
anna.o: /usr/include/stdlib.h
    ...
```

Daneben gibt es auch ein Programm „`makedepend`“, das auch diese Abhängigkeiten generiert.

Eine weitere Möglichkeit bietet *make* unter SunOS 4 (und wohl auch Solaris 2): Fügt man die Zeile

```
.KEEP_STATE:
```

in das Makefile ein, erkennt *make* solche versteckte Abhängigkeiten wie die von Header-Dateien selbstständig.

Ein Nachteil all dieser Lösung will ich nicht verschweigen: die Turnaround-Zeiten steigen, da bei jeder kleiner Änderung in einer Header-Datei (und sei es nur ein zusätzliches Leerzeichen) plötzlich eine ganze Reihe von C-Sourcen übersetzt werden, die direkt oder indirekt von dieser einen Header-Datei abhängen. Macht man es allerdings nicht, unterläuft man der Gefahr, daß bei Schnittstellen-Änderungen oder geänderten Datenstrukturen nicht alle davon abhängigen Dateien nochmal übersetzt werden. Dies kann tödlich sein, zumal der Linker ahnungslos alles zusammenlinkt.

18.6.8 Globale Definitionen (include-Anweisung)

Viele Makefiles innerhalb verschiedener Verzeichnisse eines Projekts ähneln sich in großen Teilen: es werden die gleichen CFLAGS definiert, der gleiche Compiler aufgerufen, die gleichen Suffix-Regeln verwendet, usw. Was liegt näher, als diese Gemeinsamkeiten in einer gemeinsamen Datei zu verwalten?

Glücklicherweise kennen viele *make*-Varianten eine „include“-Anweisung, mit der diese gemeinsame Datei eingebunden werden kann:

```
include common.mk
```

Hiermit wird die Datei „common.mk“ eingebunden. Syntaktisch sieht das ganze dann so aus, daß diese Datei hier an diese Stelle hineinkopiert wird.

Bei der Verwndung der *include*-Anweisung ist darauf zu achten, das `include` am Zeilenanfang steht und dahinter mindestens ein Leerzeichen oder Tabulator-Zeichen folgt.

Leider sind Makefiles nichts statisches - sie ändern sich mit zunehmender Projektdauer, mit geänderten Anforderungen,

18.7 Fehlersuche

18.7.1 Debug-Options

18.7.2 Syntax-Error

18.7.3 „Don´t know how to make“

18.7.4 Target „up to date“

18.7.5 „Command not found“

18.7.6 Syntax Error in mehrzeiligen Kommandos

18.7.7 „Too many lines“

18.7.8 Unerkannte Makros

18.7.9 Ignorierte Regeln

18.7.10 NFS-Problematik

18.8 Portierungen

18.8.1 GNU-make

18.8.2 Kommentare

Normalerweise können Kommentare überall beginnen. Allerdings gibt es auf manchen Systemen (z.B. HP-UX) Probleme, wenn ein Kommentar nicht am Zeilenanfang steht, sondern mit einem Tabulator-Zeichen eingeleitet wird.[10]

18.8.3 imake

18.8.4 Unterschiede verschiedener *make*s

18.8.5 Test

18.9 Richtlinien

Viele der Richtlinien in diesem Kapitel sind aus den „Makefile Conventions“ für GNU-Programme entnommen.

10. Offensichtlich wird dies dann mit einer „action“ verwechselt.

18.9.1 Allgemeine Konventionen

Standard-Shell

Normalerweise ist die Bourne-Shell (/bin/sh) als Standard-Shell vordefiniert. Es gibt aber auch *make*-Varianten, die die Standard-Shell über die Environment-Variabeln SHELL vererbt bekommen. Um Probleme zu vermeiden, sollte daher die Bourne-Shell über folgendes Makro

```
SHELL = /bin/sh
```

als Standardshell definiert werden.

Suffix-Liste

Die Suffix-Liste sollte explizit gesetzt werden:

```
.SUFFIXES:
.SUFFIXES: .c .o
```

Die erste Zeile löscht die Suffix-Liste, die zweite Zeile setzt dann explizit die gewünschten Endungen.

Grund:	Verschiedene *make*-Varianten haben unterschiedliche und inkompatible Suffix-Listen. Dies kann zu Verwirrungen auf unterschiedlichen Systemen führen.

Programmstart

Programme im Arbeitsverzeichnis müssen mit `./programm` gestartet werden.

Grund:	Das aktuelle Verzeichnis gehört nicht immer zum Suchpfad dazu.

18.9.2 Utilities

Shell-Kommandos

Auch wenn manche *make*-Varianten Korn-Shell- oder Bash-Syntax verstehen, sollte man bei der Bourne-Shell bleiben.

Standard-Kommandos

Folgende Unix-Kommandos können direkt verwendet werden:

```
cat cmp cp echo egrep expr false grep
ln mkdir mv pwd rm rmdir sed test touch true
```

Bei den Optionen sollte man sich auf die gängigen Optionen beschränken, die auf allen Systemen vorhanden sind[11].

Grund: diese Kommandos sind auf allen Unix-Derivaten vorhanden

Kommandos

Alle übrigen Kommandos wie Compiler-Aufruf und andere Programme sollten in Variablen abgespeichert werden.

Grund: zum einen änderungsfreundlicher, zum anderen kann bei Bedarf das Kommando über die Kommandozeile mitgegeben werden.

18.9.3 Variablen

Kommando-Variablen

Nach Möglichkeit sollten Variablen genauso wie das Kommando heißen und dem Kommando vorbelegt sein.

Beispiel: `YACC = yacc`

---

11. „`mkdir -p`“ sollte man z.B. vermeiden, da diese Option nicht auf allen Systemen vorhanden ist.

Vordefinierte Variablen

(s. Tabelle 18: »Vordefinierte Variablen«)

Flags.

Flags zu Kommandos sollten in einer Variable mit dem Namen des Kommandos und der Endung „FLAGS“ gekennzeichnet werden (s Tabelle 19: »Flag-Variablen«)

Abweichend von der obigen Namensgebung werden die Flags für den C-Compiler (CC) und für den C++-Compiler (CCC) mit CFLAGS bzw. CCFLAGS benannt. Dies hat historische Gründe.

Lebenswichtige Flags

Optionen, die für die Compilation bzw. Erzeugung des Targets unbedingt notwendig sind, werden **nicht** in diesen Variablen abgespeichert.

Grund:	Es sollte weiterhing möglich sein, Variablen über den *make*-Aufruf oder über Environment-Variablen zu setzen, ohne daß die Compilation schief läuft.

Beispiel:

```
hugo.o : hugo.c
        $(CC) -c $(CPPFLAGS) $(CFLAGS) hugo.c
```

Install-Variablen

Die Variable INSTALL muß in jedem Makefile definiert sein und zum Installieren von Dateien dienen.

Variable	Wert	Beschreibung
AR	ar	Archiver (zum Bibliotheken bauen)
BISON	bison	Parser-Generator
CC	cc	C-Compiler
CCC,CXX, C++C	CC	C++-Compiler
CPP	cpp	C-Preprocessor
FC	fc, f77	Fortran-Compiler
FLEX	flex	lexikalische Analyse
INSTALL	install	Installierung
LD	ld	linker
LEX	lex	lexikalische Analyse
MAKE	make	Make-Kommando
PC	pc	Pascal-Compiler
YACC	yacc	Parser-Generator

Tabelle 18: Vordefinierte Variablen

Kdo.	Flags	Beispiel	Bemerkung
AR	ARFLAGS		
BISON	BISONFLAGS	-y	
CC	CFLAGS	-g	
CCC	CCFLAGS	-Wtemplates	
CPP	CPPFLAGS	-I/usr/include	
FC	FCFLAGS		
FLEX	FLEXFLAGS		
INSTALL	INSTALLFLAGS		
LD	LDFLAGS	-lm	
LEX	LEXFLAGS LFLAGS		
MAKE	MAKEFLAGS	-k	
PC	PCFLAGS		
YACC	YACCFLAGS YFLAGS	-v	Verbose-Flag

Tabelle 19: Flag-Variablen

Daneben sollten die Variablen INSTALL_PROGRAM und INSTALL_DATA definiert werden, die zur Installation von Programmen und Daten dienen. Der Default-Wert dafür sollte $(INSTALL) sein.

Name	Default	Anmerkung
INSTALL	install cp	
INSTALL_PROGRAM	$(INSTALL)	Installation von Programmen
INSTALL_DATA	$(INSTALL)	Installation von Daten

Tabelle 20: Installations-Makros

Für die Installation von Programmen und Dateien sollte immer der Dateinamen, und nicht der Verzeichnisnamen, verwendet werden.

Beispiel:

```
install:
        $(INSTALL_PROGRAM) hugo $(bindir)/hugo
        $(INSTALL_DATA) libhugo.a $(libdir)/libhugo.a
```

18.9.4 Installations-Verzeichnis

Installations-Verzeichnisse sollte immer in Variablen abgelegt werden, so daß sich die Installation auch leicht an anderen Zielverzeichnisse anpassen läßt.

Standardnamen für solche Variablen werden in diesem Abschnitt beschrieben. Sie basieren auf Standard-Dateisystemen von SVR4, 4.4BSD, Linux, Ultrix V4 und anderen modernen Betriebssystemen.

Root-Installations-Verzeichnis

Aus folgenden zwei Varablen sollten alle weitere Installations-Variablen abgeleitet werden:

`prefix` Diese Variable enthält das Basis-Verzeichnis.

`exec_prefix` Diese Variable enthält das Basis-Verzeichnis für einige ausführbare Programme.

Als Default-Wert enhätlt diese Variable den Wert von `prefix`.

Variable	Default	Beschreibung
	Ausführbare Programme	
bindir	$(exec_prefix)/bin	hier werden die ausführbare Programme für den Benutzer abgelegt
sbindir	$(exec_prefix)/sbin	hier werden die Programme für den System-Administrator abgelegt
libexecdir	$(exec_prefix)/libexec	hier werden die Programme abgelegt, die von anderen Programmen benötigt werden
	Daten-Files	
datadir	$(prefix)/share	für Read-Only Architektur-unabhängigen Daten-Files
sysconfdir	$(prefix)/etc	Read-Only Konfigurations-Files, die zu einer Single-Maschine gehören
sharedstatedir	$(prefix)/com	Architektur-unabhängige Daten-Files, die vom Programm verändert werden können
localstatedir	$(prefix)/var	lokale Architektur-unabhängige Daten-Files, die vom Programm verändert werden können
libdir	$(exec_prefix)/lib	Objekt- und Bibliotheks-Dateien (keine ausführbaren Programme)
infodir	$(prefix)/info	Info-Dateien
lispdir	$(prefix)/share/emacs/site-lisp	Emacs-Lisp-Dateien
includedir	$(prefix)/include	Header-Dateien
	Manual-Seiten	
mandir	$(prefix)/man	Verzeichnis für die Manual-Seiten
man1dir	$(mandir)/man1	„1“er-Manual-Seiten
man2dir	$(mandir)/man2	„2“er-Manual-Seiten

Tabelle 21: Installations-Verzeichnisse

Variable	Default	Beschreibung
...		
manext	.1	Manual Extension
man1ext	.1	„1“er-Manual Extension
man2ext	.2	„2“er-Manual Extension
...		
	Source-Verzeichnis	
srcdir	-	Verzeichnis, in dem die Sourcen kompiliert werden

Tabelle 21: Installations-Verzeichnisse

18.9.5 Standard Targets

Notwendige Targets

all	Compilation des gesamten Programms (möglichst mit der Option „-g“) sollte das Default-Target sein
install	Compilation und Installation des Programms, der Bibliotheken, usw...
uninstall	„install“ wieder rückgängig machen
install-strip	Installation mit ge*strip*ten Programmen[12]
clean	Löschen aller Dateien, die beim Erstellen des Programms erzeugt werden
distclean	lösche alle Dateien, die nicht mehr benötigt werden
dist	Distributions-Tarfile erstellen; das Tar-File sollte ein Unterverzeichnis mit dem

12. es gibt ein Kommando „strip“, das die Symbol-Tabelle aus dem Programm entfernt.

	Toolname und Versionsnummer enthalten, in dem sämtliche Programme und Dateien enthalten sind (z.B. chicco-1.0.1)
check	Selbsttest (Überprüfung des Programmes)

Nützliche Targets

mostlyclean	wie „clean“, außer Bibliotheken und andere Dateien, die zeitintensiv zu erstellen sind
TAGS	Erstellen/Update einer Tags-Tabelle[13]
info	Info-Dateien erstellen
installcheck	Installation von Test-Dateien und -Verzeichnisse, die für „check“ benötigt werden
installdirs	Verzeichnisse, die für „install“ erzeugt werden müssen

18.9.6 Pfade

18.10 Tools rund um „make“

18.10.1 makedepend

18.10.2 autoconfig

13. siehe „man ctags“

18.10.3 automake

18.10.4 imake

18.10.5 Versionskontrolle

SCCS

RCS

18.11 Aufgaben

VI

Das X-Window-System

19 X-Windows

19.1 Geschichte

X11 ist aus dem „Athena“-Projekt am M.I.T. (Massachusetts Institute of Technologie) hervorgegangen. Es sollte die Möglichkeit geschaffen werden, Grafikausgaben transparent im Netzwerk zu ermöglichen, d.h. die graphische Ausgabe ist nicht mehr an den Rechner gebunden, auf dem die Anwendung abläuft. Gedacht war das ganze als die Basis für einen Fenster-Manager, der auf diese hardware-unabhänigige Schnittstelle dann aufsetzen kann. Der Fenster-Manager selbst war nicht Gegenstand dieser Entwicklung. Später wurde die Weiterentwicklung des X-Window-Systems einer firmenneutralem Konsortium, dem X-Konsortium, übertragen.

19.1.1 Entwicklungsziel

- Hardware-Unabhängigkeit
- Netzwerk-Transparenz
- Multitasking / Multiuser
- Multiwindow / Multiscreen
- firmenneutral

- Basis für darauf aufbauenden Window-Manager

19.1.2 Eigenschaften

- HW-unabhängig
- Betriebssystem-unabhängig
- Netzwerk-unabhängig
- Multitasking / Multiuser
- Multiwindow / Multiscreen
- standardisiert durch ANSI
- X-Lib-Bibliothek ist Schnittstelle der X-Clients zu X
- kein eigenes Look & Feel, kein Window-Manager
- Client/Server-Architektur
- Unterstützung von Text und Graphik in hierarchisch angeordneten Fenstern
- Sourcen frei verfügbar (X11R6: ca. 87 MB)

19.2 Architektur

19.2.1 Client-Server-Architektur

Der X-Server ist für die graphische Ausgabe verantwortlich. Er gliedert sich in:

DIP:	Device Independent Part (geräteabhängige Abbildunge wird in geräteunabhängige Darstellung übertragen)
DDP:	Device Dependant Part (HW-abhängiger Teil; wird vom Hersteller gestellt)

Als X-Client wird meist das eigentliche Programm bezeichnet. Dies kann z.B. ein CAD-Programm, eine Textverarbeitung oder aber auch der Window-Manager selbst sein.

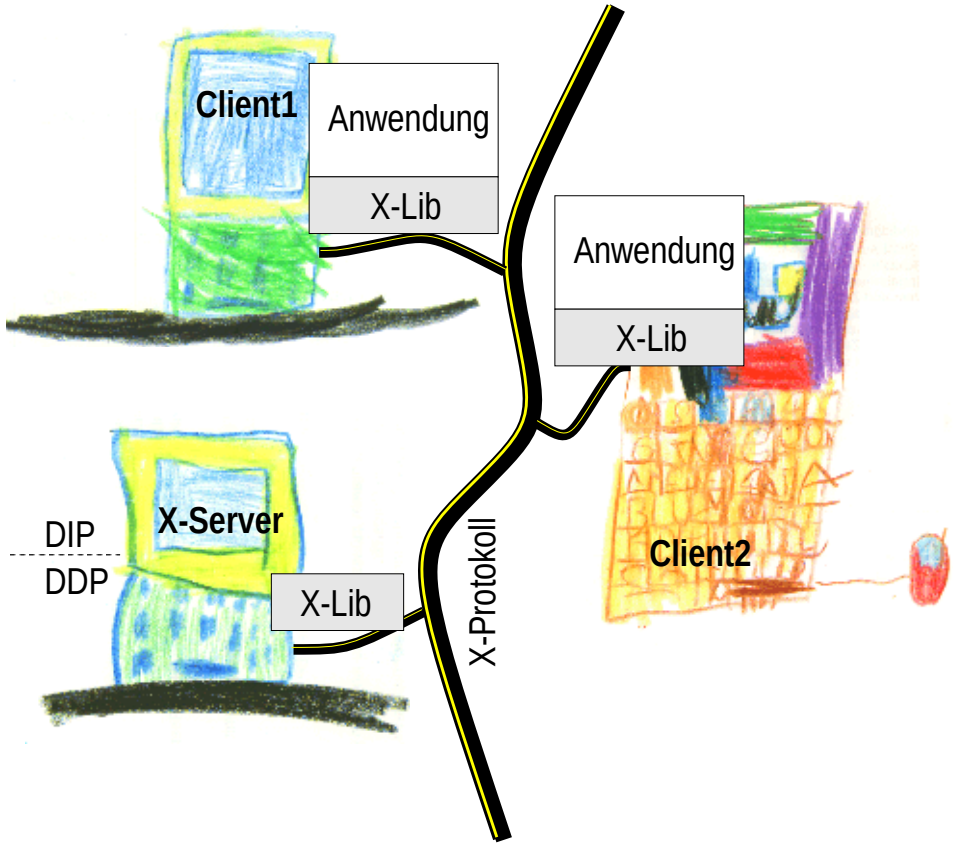


Abbildung 31: X-Client-Server-Architektur

Die Kommunikation von X-Client und X-Server findet über das X-Protokoll statt:

19.2.2 Architektur-Modell

Das X-Architektur-Modell gliedert sich in mehrere Schichten, ähnlich dem OSI-Schichten-Modell (s. Abbildung 32: »Schichten-Modell«).

X-Protokoll

asynchrone Übermittlung (keine Bestätigung)

X-Lib

Schnittstelle zum X-Protokoll; das X-Protokoll und die X-Library sind Bestandteil des X-Window-Systems.

Xt (X-Toolkit-Intrinsics)

Die Xt-Bibliothek ist das „Kernstück“ oberhalb der X-Lib, die als Basis für darauf aufbauende weitere „Tool-Kits“.

- vermindert Komplexität durch Objekt-Orientierung
- vereinheitlichte Verwaltung von Resourcen (Resourcefelder)
- vereinheitliches Geometrie-Management
- dient zum Aufbau von Objekten („Widgets[1]“)

19.2.3 Fenster-Hierarchie

X-Windows verfährt nach dem Motto „Fenster sind billig“, wobei ein Fenster nicht viel mehr als ein Rechteck bzw. Bildschirmausschnitt bedeutet, der zur Darstellung von Text oder Graphik dient:

Fenster:	Rechteck, hierarchisch angeordnet, „objektorientiert“,

1. Widget = Window Gadget (Fenster-Dingsbums)

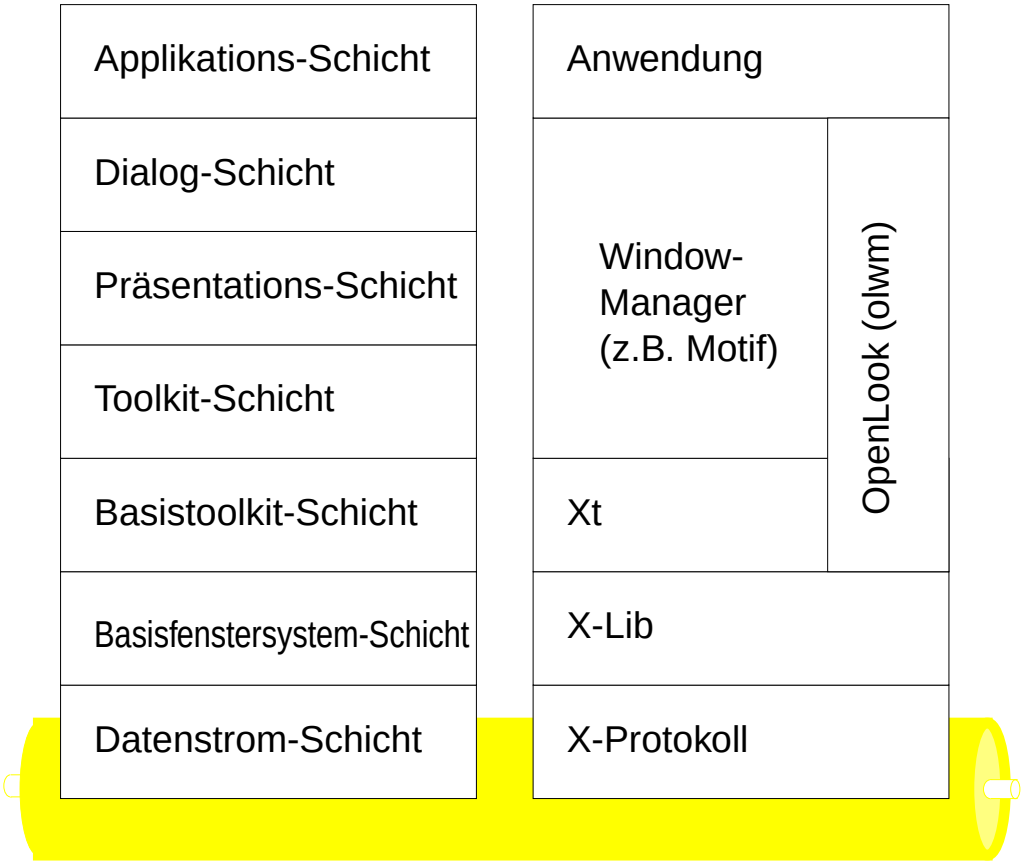


Abbildung 32: Schichten-Modell

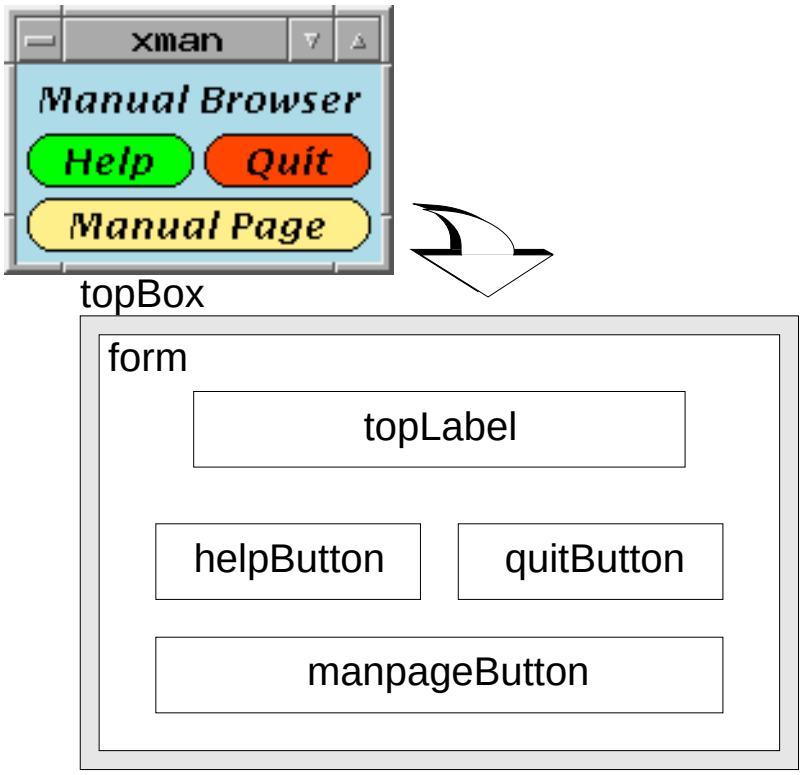


Abbildung 33: Fenster-Hierarchie von „xman“

„Fenster sind billig“

Der Fensterinhalt wird vom Anwenungsprogramm verwaltet, nicht vom X-Server.

Fenster-Merkmale

- Fenster = Rechteck
- hierarchisch angeordnet (Baum-Struktur)
- „objekt-orientiert“
- „Fenster sind billig“

19.3 Window-Manager

Window-Manager sind nicht Bestandteil von X11, sondern setzen darauf auf.

- verschiedene Window-Manager für X11 erhältlich
- verantwortlich für das „Look & Feel“
- verwalten die Fenster auf dem Bildschirm
- dienen zur Steuerung und Manipulation der Fenster
- kann auch auf einem anderen Rechner laufen

19.3.1 OSF/Motif

Der Motif-Window-Manager „mwm“ ist der Window-Manager der OSF („Open Software Foundation“). Die OSF ist eine Stiftung von Forschung und Entwicklung, die 1988 von mehreren Firmen gegründet wurde und zum Ziel hat(te), eine komplette offene Software-Plattform für die neunziger Jahre und darüber hinaus zur Verfügung zu stellen.

19.3.2 OpenWindows

Der OpenWindow-Manager „openwin“ bzw „olwm“ ist der Window-Manager, der von Sun ab dem Betriebssystem SunOS 4 ausgeliefert wird und „SunView“ abgelöst hat. OpenWindows wird manchmal als „Open-

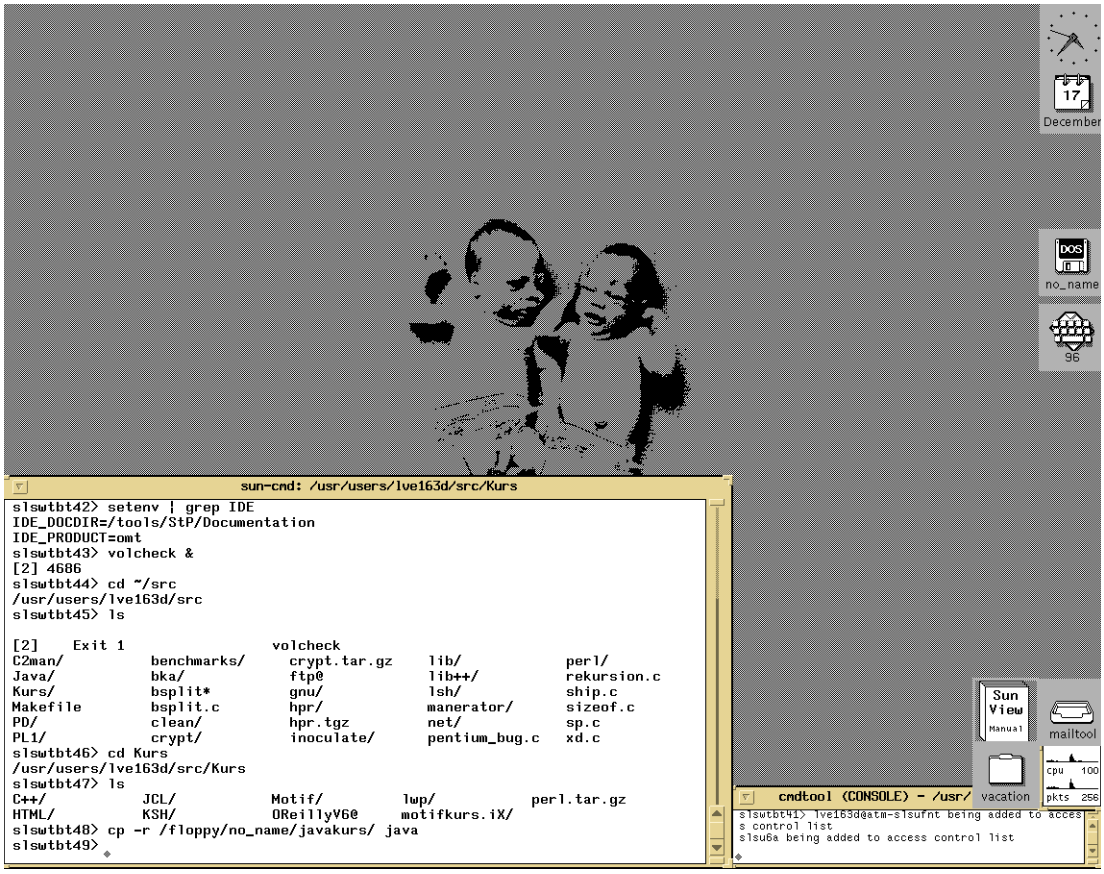


Abbildung 34: OpenWindow-Manager „openwin“

Look“ bezeichnet. Damit ist aber mehr das „Look & Feel“ von OpenWindow gemeint. Gegenüber Motif ist es ergonomischer zu bedienen, ist aber auf Sun-Maschinen beschränkt.

Daneben gibt es auch den „olvwm“ (OpenLook Virtual Window Manager). Im Gegensatz zu „openwin“ ist er frei verfügbar und bietet die Möglichkeit, mehrere „virtuelle“ Bildschirme zu verwalten.

19.3.3 VUE

„VUE“ ist der Standard-Desktop bei HP. Er ist gekennzeichnet durch eine „Bedienleiste“ am unteren Fensterrand, über den sich die wichtigsten Programme starten lassen, der Auskunft über Uhrzeit und Systemzustände liefert, und der einen Mülleimer enthält.

19.3.4 CDE

„CDE“ (Common Desktop Environment) ist die Antwort mehrerer führender Unix-Hersteller (IBM, HP, Sun, Novell) auf die Bedrohung des Unix-Marktes durch Windows-NT. Die „Offenheit“ von Unix hat leider auch den Nachteil, daß verschiedene Window-Manager exisistieren. Um nun *eine* einheitliche graphische Oberfläche zu schaffen, hat man sich zusammengesetzt und aus den verschiedenen existierenden Window-Manager die Rosinenstücke rausgepickt. Heraus kam dabei CDE, das sich vom Aussehen sehr stark an HP´s VUE anlehnt.

Von anderen Window-Managern unterscheidet sich CDE u.a. auch durch seine konsequente Drag&Drop-Unterstützung. Daneben stehen mehrere (standardmäßig vier) virtuelle Bildschirme zur Verfügung, die sich über die „Bedienleiste“ auswählen lassen.

Ein weiterer Bestandteil von CDE ist der „Session Manager“. Er sorgt dafür, daß beim Verlassen des Desktops auf Wunsch die Arbeitsumgebung abgespeichert wird. Damit ist beim nächsten Start von CDE diese Arbeitsumgebung wieder verfügbar, und man kann mit seiner Arbeit weitermachen.

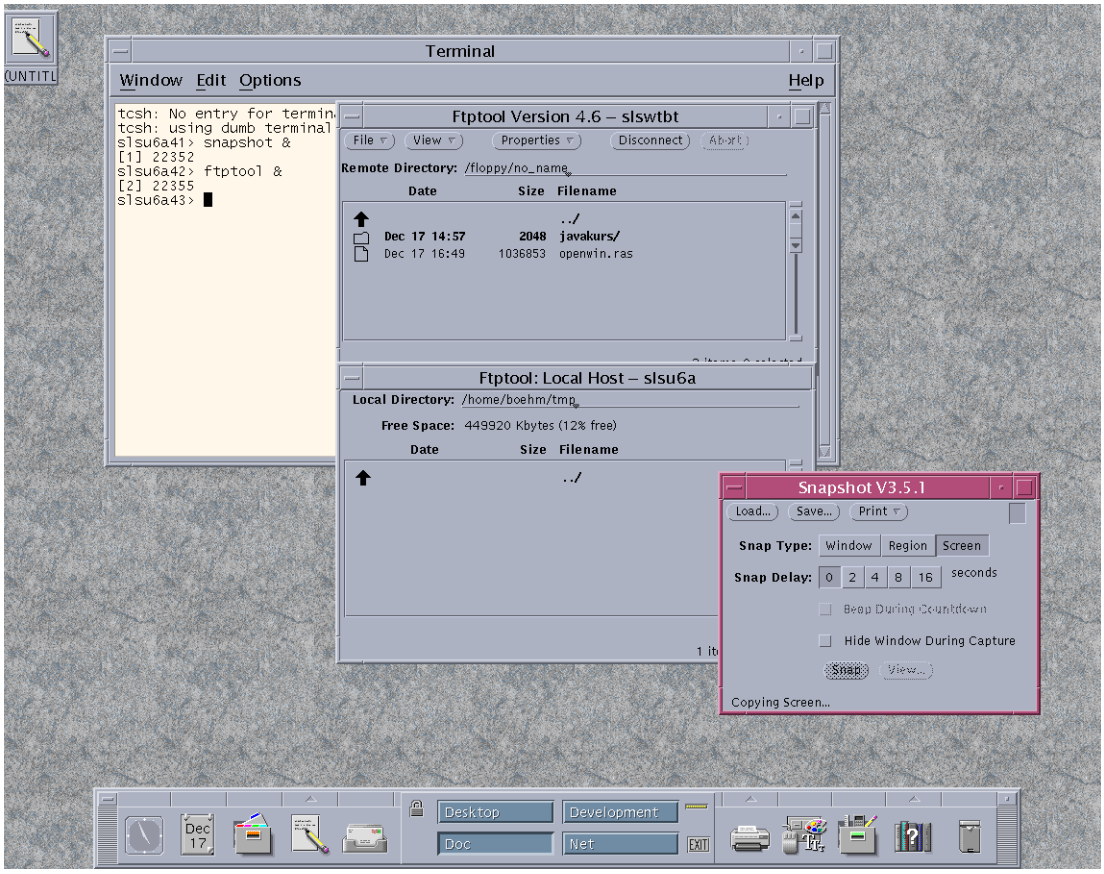


Abbildung 35: CDE

Die Realisierung der CDE-Entwicklung durch das X-Consortium wurde von der OSF koordiniert. CDE basiert noch auf X11R5 und Motif 1.2. Laut Bob Schleifler (Leiter des X-Konsortiums) soll die nächste offizielle Version auf X11R6 und Motif 2.0 basieren.

19.3.5 fvwm

ist ein frei verfügbarer Desktop, der unter Linux weit verbreitet ist. Auch er bietet mehrere virtuelle Bildschirme an, die sich über eine Icon-Leiste anwählen lassen.

Den „fvwm“ gibt es auch im Windows 95-Look. Er heißt dann „fvwm95“.

Ein weiterer Window-Manager unter Linux ist der

19.3.6 bowman

dessen Vorbild der Desktop des NEXT-Würfels ist.

19.3.7 Weitere Windowmanager

Es gibt noch weitere Window-Manager wie der „twm“, „uwm“, „wm“, „dxwm“, ..., dessen Verbreitung und Bekanntheitsgrad sehr gering ist, und die daher in der Praxis so gut wie keine Rolle spielen.

19.3.8 X ohne Window-Manager

X11 kann auch ohne Window-Manager betrieben werden. Es ist zwar mühsam, da man sämtlich Fenster beim Aufruf in Position und Größe festlegen muß, aber möglich ist es.

19.4 X-Start

Um mit X-Windows arbeiten zu können, muß erstmal der X-Server gestartet werden. Dazu dient das Kommando „xinit“ (unter Linux:

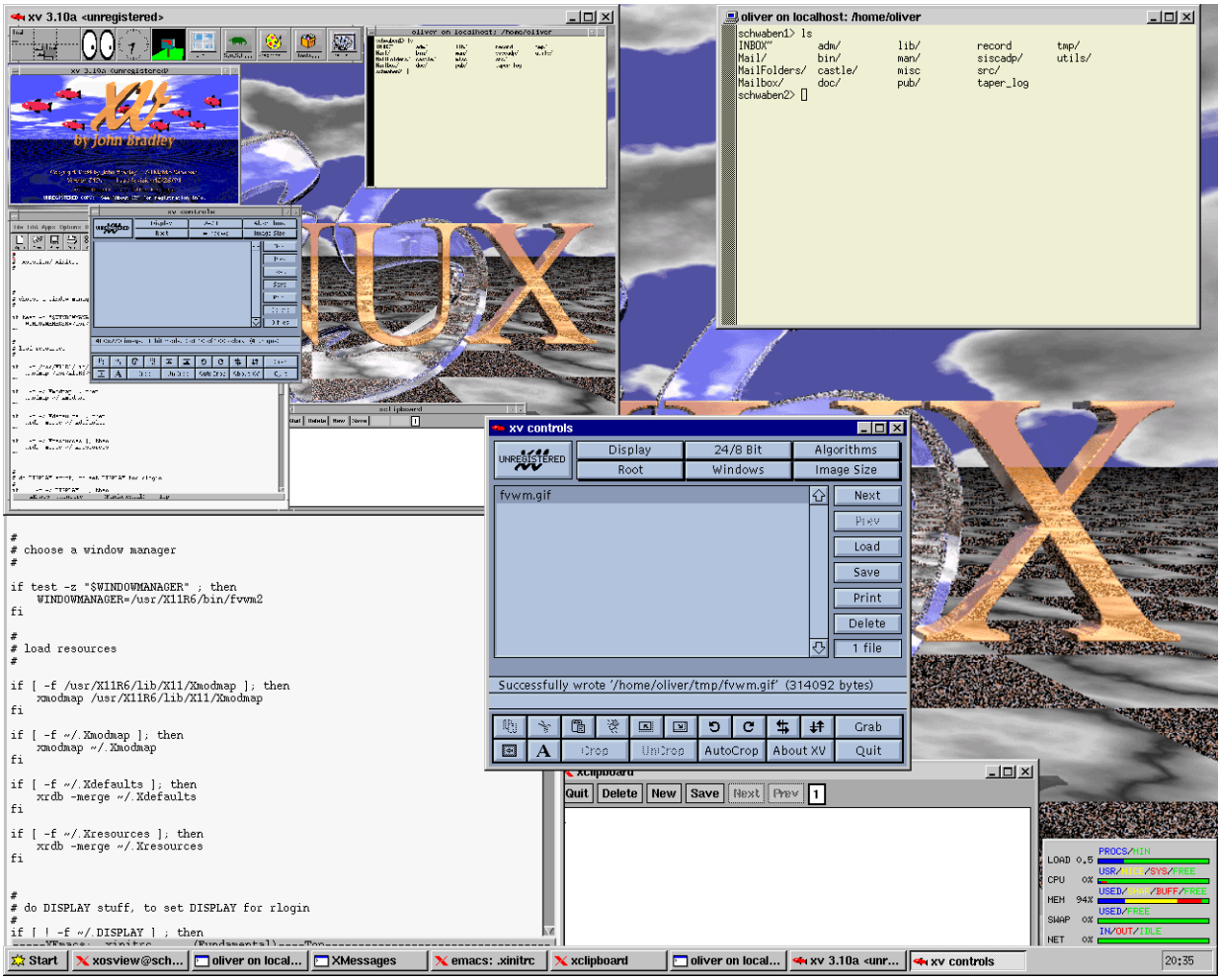


Abbildung 36: fvwm95

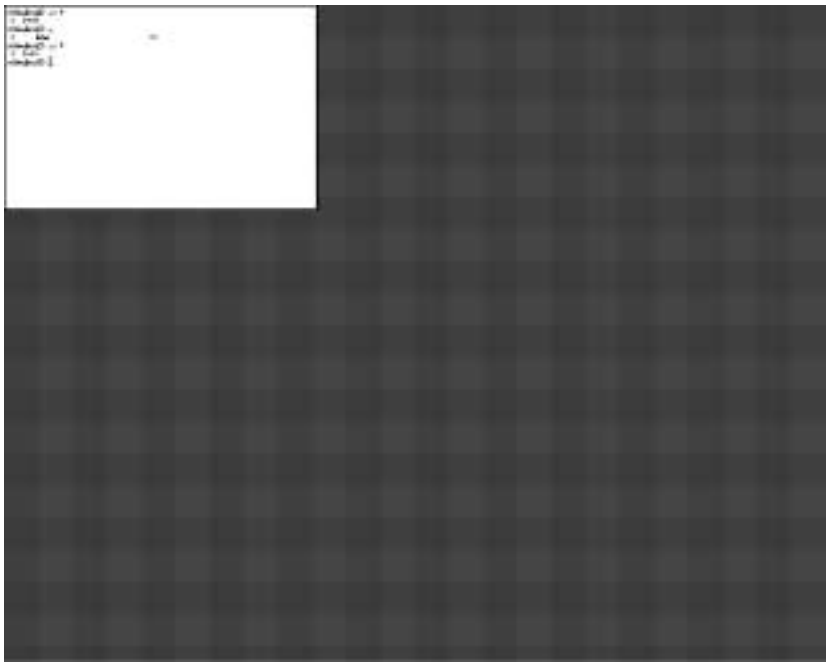


Abbildung 37: X11 pur

„startx“), dem man über die Datei „.xinitrc“ den gewünschten Fenster-Manager und/oder die gewünschten Fenster mitgeben kann.

Ist der X-Server schon hochgefahren und der X-Display-Manager „xdm“ gestartet, erhält man einen graphischen Login-Begrüßungsbildschirm. Dann kann man die Datei „.xsession“ die gewünschten Kommandos absetzen.

19.5 X11-Konfigurierung

19.5.1 Resourcen

- Anpassung an individuellen Benutzerwünschen
- Anpassung an den verwendeten X-Server oder Hardware

Resourcen bestimmen das Erscheinungsbild einer X-Anwendung und

- lassen sich hart ins Programm kodieren,
- über Kommandozeilenparameter setzen[2] oder
- über Dateien („.Xdefaults“) setzen

Resource-Dateien können

- anwendungs-
- benutzer-
- display- *oder*
- hostabhängig

sein. Ein Eintrag in einer Resource-Datei setzt sich aus den Komponenten

- Programm
- Widget(s)
- Resource *und*
- Wert

---

2. z.B.`xterm -sb`
`xman -bg yellow`

zusammen. Es gibt Regeln für die Auswertungsreihenfolgen von Resource-Dateien.

Resource-Typen

- Resourcen zur Verhaltenssteuerung
  (z.B. *KeyboardFocusPolicy*)
- Resourcen zur Steuerung des Aussehen
  (z.B. *foreground*)
- Resourcen, die in Verbindung mit einem Client aktiviert werden
  (z.B. *clientdecoration: none*)

Setzen von Resourcen

über Kommandozeilenparameter / Optionen:

```
xterm -sb
xman -bg yellow
```

über Option `-xrm` *resource_string*

```
xman -xrm "*.background: yellow"
```

über Resource-Datei

19.5.2 Resource-Datei

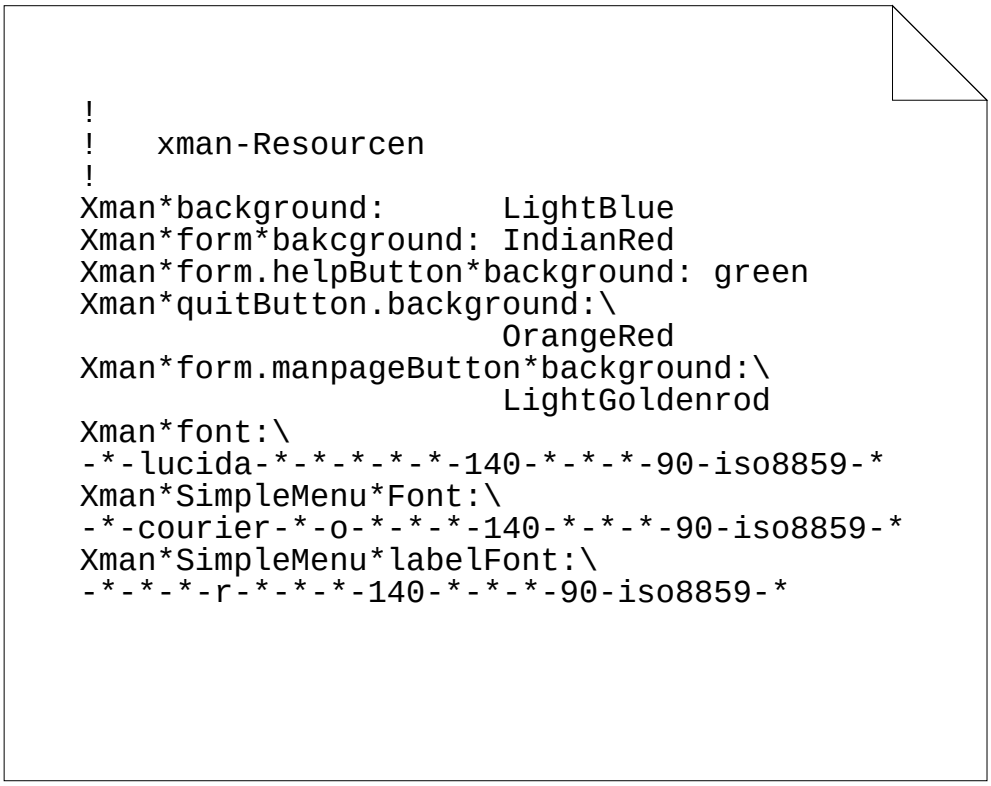
Aufbau der Resource-Datei:

```
! Kommentar
program{[.|*|?[3]]widget}+[.|*|?]resource: value
```

Bedeutung der Einträge in der Resource-Datei:

`program`	Name des Programms (1. Buchstabe meist groß, z.B. `Xterm`)
`widget`	Instanzname (wird durch die Programmierung vorgegeben) oder Klassennamen (z.B. `XmLabel`)

3. Seit X11R5 kann statt `widget` auch `?` als Wildcard verwendet werden



```
!
!   xman-Resourcen
!
Xman*background:      LightBlue
Xman*form*bakcground: IndianRed
Xman*form.helpButton*background: green
Xman*quitButton.background:\
                      OrangeRed
Xman*form.manpageButton*background:\
                      LightGoldenrod
Xman*font:\
-*-lucida-*-*-*-*-*-140-*-*-*-90-iso8859-*
Xman*SimpleMenu*Font:\
-*-courier-*-o-*-*-*-140-*-*-*-90-iso8859-*
Xman*SimpleMenu*labelFont:\
-*-*-*-r-*-*-*-140-*-*-*-90-iso8859-*
```

Abbildung 38: .Xdefaults

`resource` durch Widget-Set (z.B. Motif) vorgegeben (z.B. `labelString`)

Beispiele

```
! alles grau (Hintergrund)
*background: gray

! alle Label-Hintergründe rot
*XmLabel.background: red

! grüner Hintergrund des Help-Buttons (xman)
Xman.topbox.form.helpButton.background: green

Xman*form.Command.background: yellow
```

19.5.3 Resource-Regeln („Precedence Rules“)

Vorrang hat der Eintrag, der eine Resource am genauesten spezifiziert:

1. Instanz/Klassenname, „?“ genauer als „*“

   Beispiel:
   ```
   label.background,
   XmLabel.background,
   ?.background
       genauer als
   *.background
   ```

2. Instanzname genauer als Klassenname,
   Klassenname genauer als „?“

   Beispiel:
   ```
   *label.background,
   XmLabel.background,
   ?.background
   ```

3. Verbindung durch einen Punkt („.“) ist genauer als durch einen Stern („*“)

   Beispiel:
   ```
   *label.background
   *label*background
   ```

19.5.4 Hierarchie der Resourcen

Eine Reihe von Resource-Dateien und Environment-Variablen regeln die Einstellungen von Resourcen. Wird dabei eine Resource mit verschiedenen Werten vorbelegt, so gewinnt die Resource-Datei, die als letztes durchlaufen wird:

1. Anwendungs-abhängig:

   ```
   /usr/lib/X11/$LANG/app-defaults/prog-name[4]
   /usr/lib/X11/app-defaults/prog-name
   ```

2. Benutzer-abhängig:

   ```
   $XUSERFILESEARCHPATH/prog-name
   $XAPPLESRESDIR/$LANG/prog-name
   ~/$LANG/prog-name
   ~/prog-name
   ```

3. Display-abhängig:

   ```
   ~/.Xdefaults
   ```

4. Host-abhängig:

   ```
   $XENVIRONMENT
   ~/.Xdefaults-hostname
   ```

19.5.5 Resourcen - Fehlerquellen

Folgende Fehlerquellen sind möglich:

- Widget nicht vorhanden bzw. falsch geschrieben (Groß/Kleinschreibung wird unterschieden!)
- falsche Hierarchie
- Resource nicht vorhanden

Leider führen Fehler beim Setzen von Resourcen zu keiner Fehlermeldung, sondern werden von X11 stillschweigend ignoriert. Um festzustellen, ob eine Resource gesetzt wurde, kann man sich des *appres*-Kommandos („Application Resource“) bedienen (s. nächster Abschnitt).

---

4. z.B. *XMeter*

19.5.6 Resource-Kommandos

Die eingestellten Resourcen werden von X11 in einer internen Datenbank verwaltet. Folgende Funktionen existieren, um auf diese interne Datenbank zuzugreifen:

appres

appres (*List Application Resource Database*) listet die Resourcen für ein bestimmtes Programm auf. Leider ist es nicht auf allen Systemen[5] vorhanden.

Beispiel:

```
appres Xman
```

xrdb

Mit *xrdb (X-Server Resource DataBase Utility)* kann man diese Resource-Datenbank manipulieren. Die wichtigsten Optionen sind dabei:

-load *file*	liest die Resource-Datei *file* ein und baut daraus die interne Resource-Datenbank auf
-merge *file*	fügt die Resourcen aus *file* zu den bestehenden Resourcen hinzu bzw. überschreibt die internen Werte mit den Werten aus *file.*

19.6 X-Fonts

19.6.1 Font-Name

Um einen gewünschten Font auszuwählen, kann man das Kommandos „*xlsfonts*“ benutzen. Die gewünschten Eigenschaften des Fonts (Größe, Typ, ...) lassen sich über verschiedene Menüs auswählen und als Ergebnis erhält man ein Fontname mit folgendem Aufbau:

5. z.B. nicht unter *HPUX*

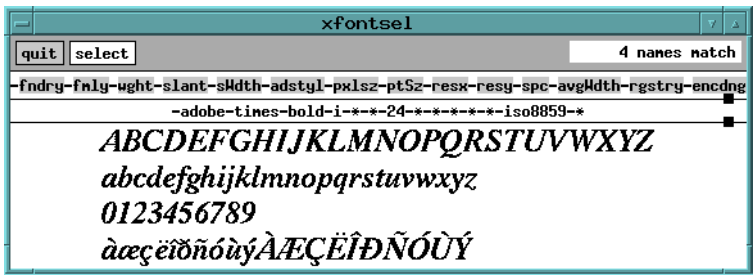


Abbildung 39: xfontsel

-fndry-fmly-wght-slant-sWdth-adstyl-pxlsy-ptSz-resx-resy-spc-avgWdth-rgstry-encdng

	Bedeutung	**Beispiel(e)**
fndry	Foundry	Adobe
fmly	Family	courier helvetia lucida times
wght	Weight	bold light
slant	Neigung	i (italics) o (oblique) r (roman)
sWdth	Width	normal condensed
adstyl	Style	sans serif
pxlsz	Pixelsize	
ptSz	Pointsize	
resx	Resolution x	
resy	Resolution y	
spc	Spacing	p (proportional) m c (fixed/courier)
avgWdth	Average Width	
rgstry	Registry	Adobe iso8859
encdng		dingbats

Tabelle 22: X-Fonts

Für Einträge, die man offen lassen möchte, kann man „*“ als Wildcard verwenden.

Beispiel für einen ISO8859-Font aus der *Lucida*-Familie mit der Größe *140 Point* und durchschnittlicher Breite von *90 Point*:

```
-*-lucida-*-*-*-*-*-140-*-*-*-90-iso8859-*
```

19.6.2 Font-Kommandos

xfd (X-Font-Displayer)

Damit läßt sich ein Font anschauen. Wenn man nicht weiß, welche Fonts es gibt, kann man z.B. mit →*xlsfonts* nachfragen.

Beispiel:

```
xfd -fn 12x24
```

xfontsel (X-Font-Selector)

Damit kann man sich den Namen eines Fonts zusammenbauen (s. Abbildung 39: »xfontsel«).

xlsfonts (List Fonts)

Damit lassen sich sämtliche Fonts des X-Servers auflisten. Aber Achtung: je nach Geschwindigkeit und Umfang der Fonts kann es mehrere Minuten(!) dauern, ehe sämtliche Fonts aufgelistet werden. Und solange ist der Bildschirm blockiert!

Optionen:

-l, -ll, -lll	alle Fonts auflisten (im long-Format)
-fn *pattern*	nur die Fonts mit *pattern* auflisten

19.6.3 X-Font-Server

Normalerweise sind die Fonts Bestandteil des X-Servers. Da es aber ein ziemlicher Verwaltungsaufwand darstellt, wenn man allen X-Servern

einen neuen Font (z.B. „Futura“) beibringen will, gibt es die Möglichkeit, einen X-Font-Server einzurichten, über den dann sämtliche X-Server ihre Fonts beziehen können.

19.7 X-Settings

19.7.1 xset

Mit *xset* lassen sich eine Reihen von Benutzer-Einstellungen vornehmen. Die wichtigsten Optionen sind:

q	gibt die aktuelle Einstellung (current settings) aus
m	damit läßt sich die Maus beschleunigen oder auch verlangsamen (accellerator), abhängig von einem einstellbaren Schwellwert (treshold)
s	screenblank
led on/off	schaltet die LEDs auf der Tastatur an/aus

Beispiele:

xset m 5/2 12	beschleunigt die Maus um den Faktor 2,5, falls 12 Pixel innerhalb kurzer Zeit überschritten werden
xset s 300 120	nach 5 Minuten ohne Eingabe wird der Bildschirm-Schoner gestartet, wobei alle 2 Minuten der Hintergrund gewechselt wird

19.7.2 xsetroot

Mit *xsetroot* lassen sich die „Root-Window“[6]-Parameter setzen.

Optionen:

6. Mit „Root-Window“ wird der Desktop-Hintergrund bezeichnet.

-help	Hilfe
-cursor *cursorfile maskfile*	Damit läßt sich der X-Cursor für den Bildschirm-Hintergrund setzen
-bitmap *file*	damit läßt sich der Bildschirmhintergrund setzen
-gray/grey	grauer Bildschirmhintergrund
-solid *color*	*color*-farbiger Bildschirmhintergrund

19.7.3 xmodmap

Mit *xmodmap (Modify KeyMaps)* läßt sich die Tastaturbelegung beeinflußen.

Optionen:

-pm	print modifier map
-pk	print key map
-pp	print pointer map (Maus-Buttons)

Beispiele:

```
xmodmap -e 'keysym F1 = Help'
xmodmap -e 'keycode 50 = Backspace'
xmodmap .xmodmaprc
```

Um festzustellen, welcher Keycode durch welche Taste erzeugt wird, kann man das Programm *xev* verwenden (leider auch nicht auf allen Systemen vorhanden).

19.8 Authorisierung

19.8.1 Display-Umlenkung

Wie schon angedeutet, ist es mit X11 möglich, daß Programm und Ausgabe auf unterschiedlichen Rechnern ablaufen können. Um X11 mitzutei-

len, auf welchem Rechner die Ausgabe stattfinden soll, muß die Environment-Variable „DISPLAY“ gesetzt werden:

```
setenv DISPLAY   gold:0
setenv DISPLAY   gold:0.0[7]
```

Damit wird die Ausgabe auf den Rechner *gold* umgeleitet. „`.0`“ in der zweiten Zeile gibt dabei den Bildschirm an, auf dem die Ausgabe erfolgen soll[8]. Aber nachdem man meistens nur mit einem Bildschirm arbeitet, kann man diese Angabe auch getrost weglassen (erste Zeile).

Läuft das Programm und die Ausgabe auf demselben Rechner, läßt man die Angabe des Rechners weg.

19.8.2 Rechnerbasierte Authorisierung

Mit xhost, dem „Server Access Control Program“, kann man anderen Rechnern den Zugriff auf den eigenen X-Server erlauben oder verweigern:

xhost +garfield	Rechner „garfield“ darf auf den X-Server zugreifen
xhost -garfield	Rechner „garfield“ darf nicht mehr zugreifen
xhost +boehm@archimedes[9]	Benutzer „boehm“ in der Domain „archimedes“ darf auf den X-Server zugreifen
xhost -boehm@archimedes	„boehm“ darf nicht mehr
xhost +	jeder darf

7. C-Shell Syntax; in der Bourne-Shell (und verwandten Shells) setzt man diese Environment-Variablen mit

```
export DISPLAY=gold:0
export DISPLAY=gold:0.0
```

8. X11 kann mehrere Bildschirme verwalten

9. funktioniert nicht auf allen Systemen

xhost -	keiner darf (außer der lokale X-Server selbst natürlich)

19.8.3 Benutzerbasierte Authorisierung

Die Zugangskontrolle kann auch über einen benutzerspezifischen Schlüssel („Magic Cookie“) stattfinden, der in der Datei *~/.Xauthority* abgelegt wird.

xdm

Wird der X-Server über *xdm* gestartet, hängt es von der *„xdm-config“*-Datei ab, ob ein Schlüssel automatisch in *~/.Xauthority* abgelegt wird.

```
DisplayManager._0.authorize: true
```

Ist diese xdm-Resource gesetzt, so wird beim Einloggen automatisch ein Schlüssel generiert.

xinit/startx

Wird der X-Server vom Benutzer hochgefahren, kann er mit dem Kommando *xauth* einen Schlüssel in der Datei *~/.Xauthority* ablegen:

```
xauth add ‘hostname‘:0 MIT_MAGIC-COOKIE-1 1234abcd
```

`‘hostname‘:0`	Display-Name (`‘hostname‘/unix:0` für lokalen Eintrag)
`MIT_MAGIC-COOKIE-1`	Protokoll-Name
`1234abcd`	Hexkey (sollte zufällig sein)

Loggt man sich zwischenzeitlich auf einen anderen Rechner ein, so hat man damit automatisch Zugriff auf seinen lokalen Rechner, wenn auf dem anderen Rechner dasselbe Heimatverzeichnis (und damit dieselbe *.Xauthority*-Datei) sichtbar ist. Ist dies nicht der Fall, kann man den Schlüssel mit

```
xauth extract $DISPLAY | rsh rechner2 xauth merge -
```

auf den Rechner *rechner2* exportieren.

19.9 X-Tools

19.9.1 Terminal-Programm

xterm	Terminal-Emulator (VT100, VT102, Tektronix 4014)

19.9.2 Status

xbiff	zeigt Mailbox an, ob sie leer oder voll ist
xclock	Uhr (-update 1: Sekundenzeiger)
xload	load average display - zeigt die Systemlast als Histogramm an (-update 1: sekündlicher Update)
xlsclients[10]	List X-Clients (= Fenster)
xwinfo	X-Window-Information
xlswins[11]	Information über den X-Fenster-Baum des Bildschirms
xprop	Ausgabe von X-Properties

19.9.3 Editoren

xedit	einfacher Texteditor
bitmap	Bitmap-Editor
xclipboard	Clipboard

10. nicht auf allen Systemen (z.B. nicht auf HP)

11. nicht auf allen Systemen

xfig	Zeichenprogramm
xpaint	Malprogramm

19.9.4 Utilities

xcalc	Taschenrechner
xlock	Lockscreen
xman	Auswahl und Anzeige von Manual-Seiten
xdpr	„dump to printer“ (Ausdruck des Bildschirms)
xpr	Ausdruck eines „Window Dumps“
xwd	„Window Dump“ (Bildschirmabzug) Bsp.: xwd > /tmp/w
xwud	„Window Undump“ (Anzeige eines Bildschirmabzugs) Bsp.: xwud -in /tmp/w
showrbg	zeige die bekannten Farb-Namen
xrefresh	Bildschirminhalt neu ausgeben
xkill	X-Client beenden

19.9.5 Demos

xeyes	X-Eyes (beobachtet den Cursor)
xlogo	X-Logo
xmag	Vergrößerungsglas (Magnifier)
xev	gibt die X-Events aus
xgc	X-Graphics-Demo

19.9.6 Freeware

xsnow	mehr für die winterliche Adventszeit

xengine	simuliert einen Kolben-Motor auf dem Bildschirm und zeigt die U/min an (gut als Test für die Leistungsfähigkeit des X-Servers geeignet)

19.10 Aufgaben

Aufgabe 27: Wie können Sie beim Hochfahren des X-Servers erreichen, daß Sie unter verschiedenen Window-Manager auswählen können? Welche Datei müssen Sie dazu anpassen?

Aufgabe 28: Welche Farben haben die einzelnen Buttons von „xman“, wenn die Einstellungen aus Abbildung 38: ».Xdefaults« zu Grunde gelegt werden? Verändern Sie die Farben des Help-Buttons zu „lilablassblau“ (oder einer Farbe, der diesem Farbton ähnlich kommt).

Aufgabe 29: Wie kann man *xterm* dazu bringen, standardmäßig mit einer „Scrollbar“ zu erscheinen?

Aufgabe 30: Für hektische Zeitgenossen gibt es die Möglichkeit, die Mausgeschwindigkeit zu beeinflussen.

a) Wie läßt sich die Maus um 250% beschleunigen?

b) Was bewirkt dabei ein „Treshold“ von 5?

Aufgabe 31: Wie kann man Hintergrundbild als „Root Window“ laden? Welche Bildformate sind möglich?

Aufgabe 32: Versuchen Sie mit Hilfe der Dateien
~boehm/lib/bitmaps
~boehm/lib/bitmaps/point_hand_mask.bm
einen Cursor für den Desktop zu setzen.

Aufgabe 33: Basteln Sie Ihren eigenen Cursor (z.B. ein Fadenkreuz) und setzen Sie ihn für den Desktop.

Aufgabe 34: Wie kann man die Auflösung des Bildschirms ermitteln?

20 KDE

KDE ist ein recht junger Windowmanager. Motivation für KDE war die unbefriedigende Lage im Unix-Windows-Markt. Es gibt zwar viele Window-Manager für das X-Windows-System, aber die freien Window-Manager sehen entweder zu altbacken aus, bieten zu wenig oder sind nicht mehr als eine Machbarkeitsstudie. Auf der anderen Seite gibt es die kommerziellen Window-Manager (OSF/Motif, CDE), die für den Privatmann einfach zu teuer sind und zudem zu viele HW-Resourcen verschlingen. Anstatt aber diese Mißstände wehmütig zu beklagen, schritt man mutig zur Tat und stampfte das KDE-Projekt aus dem Boden mit dem Ziel, den ultimativen Window-Manager auf die Beine zu stellen und eine gemeinsame Grundlage für KDE-Anwendungen zu schreiben. Beide Ziele wurde erreicht.

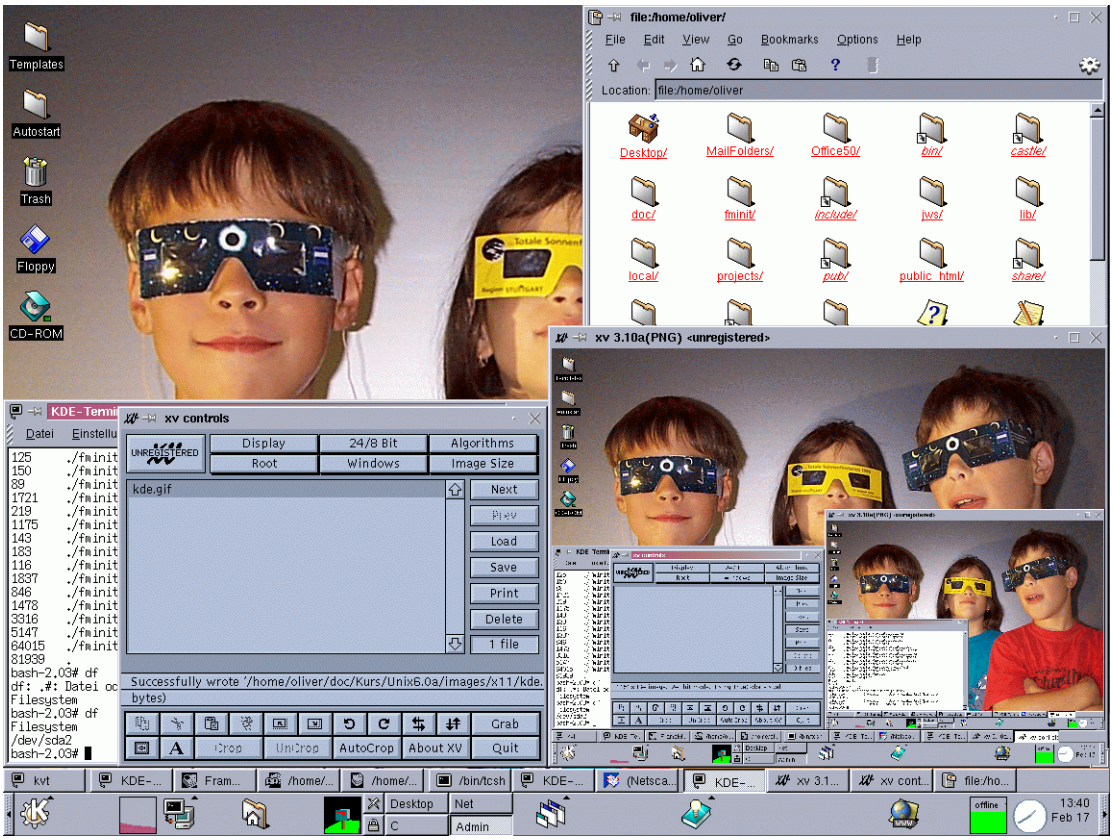


Abbildung 40: KDE

20.1 Konfiguration

20.1.1 K→Settings

Es gibt mehrere Möglichkeiten, den KDE-Desktop zu konfigurieren. Eine-Möglichkeit führt über das K-Menü: unter Settings geht ein weiteres Unter-Menü auf, mit dem man dann die verschiedenen Sachen einstellen kann.

20.1.2 KDE Control Center

Über das KDE Control Center kann der KDE-Desktop ebenfalls konfiguriert. Beim KDE Control Center hat man alles unter einer einheitlichen Oberfläche zusammengefaßt, was nicht nur Anfänger sehr entgegenkommt.

Keys

Schauen Sie sich doch einmal die Einstellungen von Keys an: hier finden Sie die Tastatur-Kürzel (unter “Global Keys” und “Standard Keys”), mit der sich der Desktop steuern läßt. Und falls Ihnen diese Einstellung nicht zusagt, können Sie sie hiermit auch abändern.

Windows

Ebenfalls interessant sind unter Windows die Mouse-Einstellungen: hiermit sehen Sie, was welcher Maus-Button bewirkt. Gefällt es Ihnen nicht, können Sie es ja abstellen oder umkonfigurieren.

20.1.3 Panel-Einstellungen

Ruft man im Panel[12] das Kontext-Menü (rechte Maus-Taste) auf, so kann das Panel konfiguriert werden. Hier kann z.B. festgelegt werden, dass das

12. das ist der Bereich, in dem sich das K-Menü und die verschiedenen Icons und andere Menüs timmeln

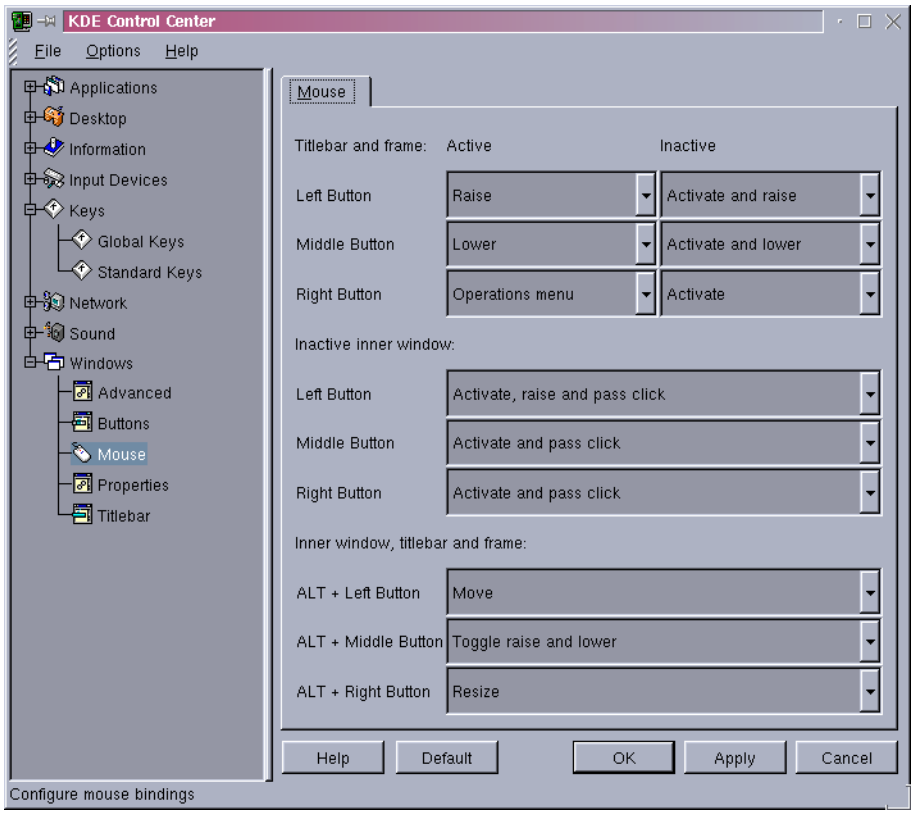


Abbildung 41: KDE Control Center (kcontrol)

Panel und die Taskbar unten (Bottom) erscheinen soll, dass die Uhr angezeigt werden soll u.v.m.

20.2 kfm – mehr als ein Datei-Manager

20.2.1 kfm – der KDE File Manager

Der KDE File Manager (kfm) ist eine wertvolle Hilfe, um sich im Unix-Dateibaum zurechtzufinden. Die Bedienung ist intuitiv und läßt sich ähnlich wie ein Web-Browser bedienen. Man kann mittels Drag&Drop Dateien kopieren, verschieben, oder sogar auf dem Desktop ablegen.

Zum Öffnen eines Verzeichnisses oder einer Datei reicht bereits ein einfacher Maus-Klick. Dies ist am Anfang sicherlich gewöhnungsbedürftig, aber man gewöhnt sich schnell daran.

Eine Datei kann auch über das Kontext-Menü, das mit der rechten Maus aktiviert wird, geöffnet werden. Dies hat den Vorteil, dass man gleich sieht, mit welchem Programm die Datei geöffnet wird. So wird z.B. ein GIF-Bild mit dem Image Viewer geöffnet, während eine einfache ASCII-Datei mit einem Text-Editor geöffnet wird.

20.2.2 kfm als Web-Browser

Sobald mit kfm eine HTML-Datei oder ein Verzeichnis mit “index.html” geöffnet wird, fungiert kfm als Web-Browser und zeigt die Datei entsprechend an. Dies funktioniert mit lokalen Dateien, wie auch mit Dateien im Internet (hierzu einfach die URL in die obere Eingabezeile eingeben).

kfm kann auch als Web-Browser über Options → Configure Browser... konfiguriert werden (z.B. Proxy-Einstellungen, ...).

20.2.3 kfm als FTP-Browser

Auch als FTP-Client ist kfm einsetzbar. Einfach die URL in die obere Eingabezeile eingeben, und schon kann man ihn als FTP-Browser zum

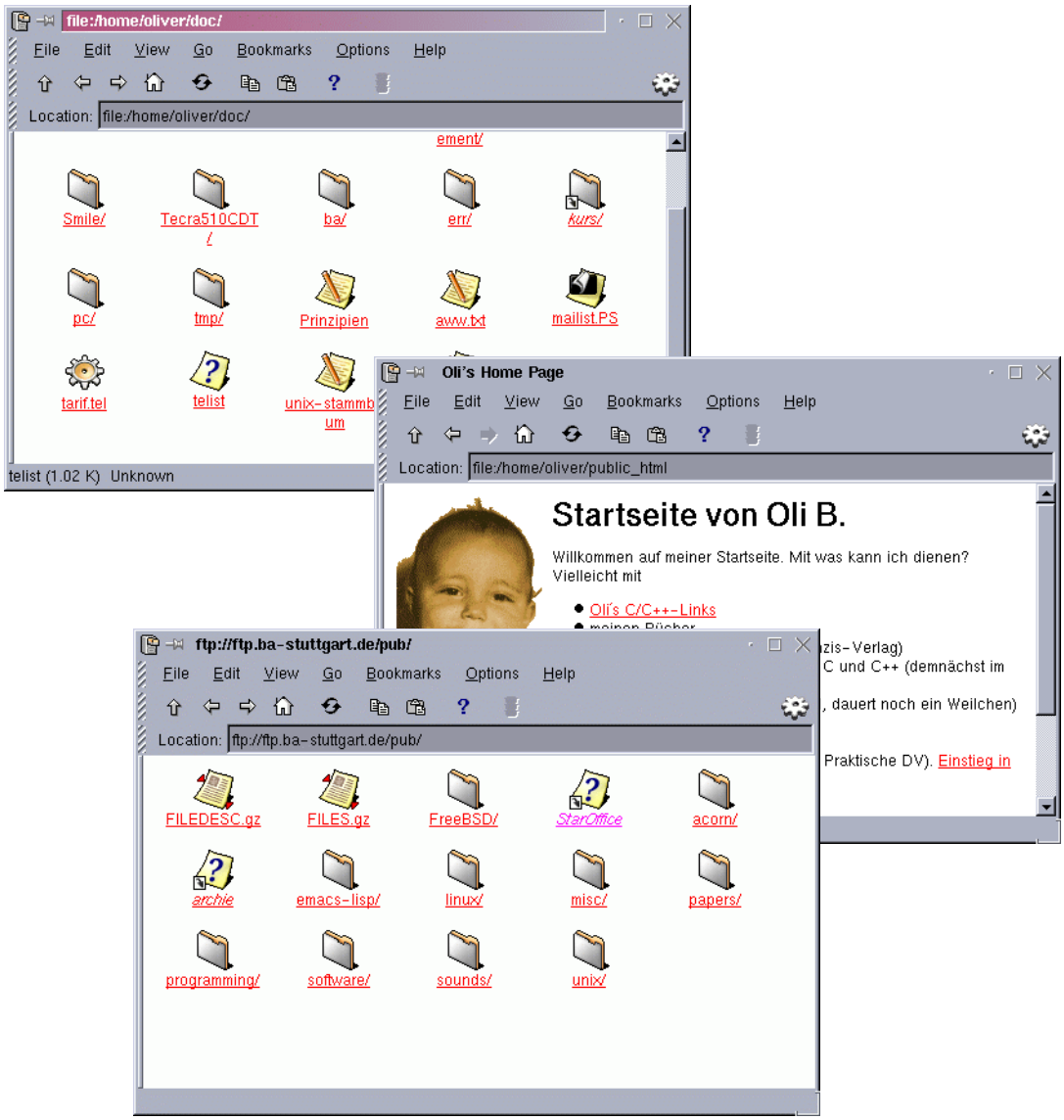


Abbildung 42: KDE File-Manager (kfm)

Up- and Download einsetzen. Aber nicht nur “anonymous ftp” (s. Kap. 22.1.5) beherrscht kfm, man kann sich mit kfm auch als normaler Benutzer anmelden, wenn man den Benutzernamen vor dem Rechnernamen in der URL angibt (durch @ getrennt), z.B.

```
ftp://boehm@amadeus.ba-stuttgart.de
```

Nach der Eingabe des Passworts wird dann die Verbindung aufgebaut und man kann ihn dann wie einen lokalen Datei-Manager benutzen. Sogar Drag&Drop funktioniert damit über Rechnergrenzen hinweg.

20.2.4 kfmclient-Kommandos

kfm selber können Sie nicht sehen. Das, was Sie in Wirklichkeit sehen, ist *kfmclient*, die graphische Schnittstelle zu kfm. kfmclient können Sie auch direkt von der Kommandozeile aufrufen. Hierzu einige nützliche Argumente:

kfmclient openURL file:/
: Öffnen des Datei-Browser mit dem angegebenen Verzeichnis

kfmclient openURL trash:/
: Öffnen des Papierkorbs

kfmclient openURL *http://www.ba-stuttgart.de/~boehm*
: Verwendung als Web-Browser

kfmclient openURL *ftp://ftp.kde.org*
: Einsatz als FTP-Browser (anonymous ftp)

kfmclient openURL *ftp://boehm@ftp.ba-stuttgart.de*
: Einsatz als FTP-Browser; das Passwort für den angegebenen Benutzer wird dabei am Anfang in einer Dialogbox abgefragt

kfmclient move *von nach*
: entspricht im Wesentlichen dem *mv*-Befehl; jedoch können mit diesem Befehl nicht nur Dateien, sondern auch WWW- oder FTP-

Adressen verschoben werden (sofern man die Berechtigung dazu hat)

VII

Vernetzung

21 Netzwerkdienste

Die folgenden Utilities:

- sind die Umsetzung bestehender UNIX-Kommandos für eine vernetzte Umgebung
- wurden an der UCB entwickelt
- werden „**r-Utilities**“ genannt, weil sie alle mit ’r’ für „remote“ beginnen
- sind auf UNIX-Rechner beschränkt
- setzen voraus, daß der Zielrechner auf dem lokalen Rechner bekannt ist (/etc/hosts)
- setzen voraus, daß der Benutzer auf dem Zielrechner bekannt ist (/etc/passwd)

Ein wichtiges Merkmal ist die Gleichsetzung von Benutzerkennungen über Rechnergrenzen hinweg, die das Eintippen von Benutzername und Paßwort vermeidet.

21.1 Rechnername und IP-Adresse

Für die Kommunikation mit anderen Rechnern muß man die IP[1]-Adresse kennen. Eine IP-Adresse hat die Form *a.b.c.d*, wobei a, b, c und d Zahlen zwischen 0 und 255 sein können, beispielsweise

149.204.82.1

Da man sich mit solchen Nummerngebilden als normaler Benutzer etwas schwer tut, werden diese unhandlichen Nummern auf Rechnernamen (hostname) abgebildet, z.B. „neckar“, unter der diese Maschine dann ansprechbar ist. Diese Zuordnung findet lokal über die Datei „/etc/hosts“ statt oder wird in eine netzwerkweiten Datenbank eingetragen, die mit

ypcat hosts

aufgelistet werden. Mit dem Kommando

hostname

bekommt man dann den Namen des Rechners raus, auf dem man sich gerade befindet. Falls man mehr an der IP-Adresse interessiert ist, die verrät u.a. das Kommando

ifconfig -a (Sun)

ifconfig lan0 (HP)

21.2 Unterdrückung der Paßwortabfrage

Die Unterdrückung der Paßwortabfrage ist für eine Reihe von „r-Utilities“ notwendig, da sie nicht in der Lage sind, das Paßwort abzufragen. Zwei Dateien legen fest, welche Client-Benutzerkennung mit welcher Server-Benutzerkennung arbeiten darf:

- /etc/hosts.equiv
- .rhosts

1. Internet Protocoll

Die Datei /etc/hosts.equiv:

- ist systemweit nur einmal vorhanden
- kann nur vom Systemverwalter verändert werden

/etc/hosts.equiv kann folgende Einträge haben:

+	alle Benutzer, dürfen sich von jedem Rechner aus unter ihrer jeweiligen Kennung einloggen
host	alle Benutzer dürfen sich vom Rechner ’host’ aus unter ihrer Kennung einloggen
host user	der Benutzer ’user’ darf sich vom Rechner ’host’ aus unter jeder (!!) Kennung (außer root) einloggen

Die Datei ~/.rhosts:

- kann von jedem Benutzer selbst angelegt werden
- enthält Einträge bestehend aus Rechnername und Benutzerkennung

~/.rhosts kann folgende Einträge haben:

+	der Besitzer von ’.rhosts’ darf sich von jedem Rechner aus unter seiner Kennung einloggen
host	der Besitzer von ’.rhosts’ darf sich vom Rechner ’host’ aus unter seiner Kennung einloggen
host user	der Benutzer ’user’ darf sich vom Rechner ’host’ aus unter der Kennung des Besitzers von ’.rhosts’ einloggen

21.3 Anmelden auf einem anderen Rechner

Das Anmelden auf einem anderen Rechner im Netzwerk ist mit dem Kommando ***rlogin*** möglich.

% **rlogin** [-l Benutzer] Hostname

Wird *rlogin*:

- ohne Option aufgerufen, melden Sie sich unter Ihrem Namen auf dem Zielrechner ’Hostname’ an
- mit der Option ’-l’ und einem Benutzernamen aufgerufen, melden Sie sich unter der angegebenen Kennung auf dem Zielrechner ’Hostname’ an

Aktuelles Verzeichnis auf dem Zielrechner ist:

- Ihr Home-Verzeichnis, wenn Sie auf dem Zielrechner eines haben
- ansonsten das Root-Verzeichnis

Beendet wird *rlogin* durch Ctrl D, *logout* oder *exit*.

Beispiel:

```
% hostname
Venus
% rlogin Jupiter
% hostname
Jupiter
% logout
```

21.4 Dateitransfer im Netz

Das Kopieren von Dateien über Rechnergrenzen hinweg ist möglich mit dem Kommando ’rcp’.

% **rcp** [Option] [[Benutzer@]Hostname:]Datei(en) [[Benutzer@]Hostname:]Verzeichnis

rcp:

- ist eine Erweiterung des *cp*-Kommandos
- erlaubt das Kopieren von lokalen Dateien auf einen Zielrechner und umgekehrt sowie das Kopieren zwischen Drittrechnern
- ermöglicht mit der Option ’-r’ auch das rekursive Kopieren ganzer Dateibäume
- überträgt auch die Zugriffsberechtigungen auf die Zieldatei
- kann keine Paßwörter abfragen

Werden auf der Kommandozeile Metazeichen verwendet, die erst auf dem Zielrechner interpretiert werden sollen, müssen diese ge*quote*t werden (s. Beispiele).

Als Benutzerkennung für den Zielrechner:

- wird standardmäßig dieselbe Benutzerkennung verwendet
- kann eine andere angegeben werden

Beispiele:

```
% rcp jupiter:/home/tarzan/work/src/\* .
% rcp ~/entwickl/src/* mars:/home/tarzan/work/source
% rcp paul@jupiter:/home/jupiter/work/source/file1.c
.
```

21.5 Remote Shell

„rsh“ erlaubt das Absetzen eines Kommandos auf einem anderen Rechner, ohne sich dort einloggen zu müssen.

% **rsh** [-l Benutzer] Hostname [Kommando][2]

rsh:

- ist ein Prozeß, der die Kommandozeile an die Shell des Zielrechners übergibt
- verbindet stdin, stdout und stderr des gestarteten Kommandos mit dem lokal ablaufenden Prozeß
- ohne Kommando entspricht es einem *rlogin*

Es gilt einschränkend anzumerken, daß:

- terminalorientierte Kommandos nicht oder nur eingeschränkt lauffähig sind
- UNIX-Signale nicht funktionieren
- das Environment der aktuellen Shell nicht mit übergeben wird

2. Auf manchen Unix-Derivaten (z.B. HP-UX) verbirgt sich hinter „*rsh*“ die „*restricted shell*“. Dort wird die „Remote Shell“ meist mit „*remsh*“ abgekürzt.

Metazeichen müssen auch hier vor der Interpretation durch die lokale Shell geschützt werden.

Beispiele:

```
% rsh jupiter who
% rsh mars ls -l \~/work/source
```

21.6 Auflisten aller aktiven Maschinen und Benutzer

Mit dem Kommando ***rusers*** können sowohl die aktiven Rechner des Netzwerkes, als auch die dort eingeloggten Benutzer angezeigt werden.

% **rusers** [Option] [Hostname]

'rusers':

- ohne Argument listet alle Maschinen und alle eingeloggten User auf diesen Maschinen auf
- mit 'Hostname' als Argument, listet die auf dem Zielrechner eingeloggten User auf
- mit der '-l' Option, listet neben Usernamen auch die Terminalnamen und Einloggzeiten auf

Beispiel:

```
% rusers
jupiter paul:console
venus tarzan:console
pluto jane:console
```

Falls *rusers* nicht zum Erfolg führt bzw. nicht auf dem Rechner vorhanden ist, kann man sein Glück noch mit ***rwho*** versuchen.

21.7 Weitere Netzwerk-Kommandos

Um festzustellen, ob eine Maschine im Netzwerk ansprechbar ist, kann mit

ping Host

festgestellt werden, ob *Host* erreichbar ist und ob er „alive“ oder „down“ ist (*Host* kann dabei als Rechnername oder als IP-Adresse angegeben werden). Aber Achtung: wenn eine Maschine „alive“ ist, heißt es noch nicht, daß sie auch tatsächlich läuft. Es heißt lediglich, daß die Schnittstellenkarte noch ein Lebenszeichen von sich gibt (der Rechner kann trotzdem gerade abgestürzt sein und im Boot-Monitor vor sich hin dümpeln).

Ein weiteres Kommando, mit der sich (zumindest auf der Sun) die Netzwerklast feststellen läßt, ist

netstat -i

Es gibt eine Statistik über den aufgelaufenen Netzwerkverkehr aus. Falls man den Verdacht hat, daß das Netzwerk mal wieder ganz schön dicht ist, kann man mit diesem Kommando die Kollisions-Rate ermitten:

Kollisions-Rate = Collis / Opkts

Zwischen 1 und 2 Prozent sind normal. Steigt sie allerdings über 5% an, sollte man seinen Netzwerkadministrator davon unterrichten, bevor das Netzwerk schlapp macht.

22 Das Internet

Mit **Internet** wird die Gesamtheit aller Rechner bezeichnet, die weltweit untereinander vernetzt sind. Es wird auch als *„das Netz der Netze“* bezeichnet. Einige Begriffe, die dabei auftauchen, werden hier in diesem Kapitel erläutert:[3]

22.1 Internet Protokolle

22.1.1 TCP/IP (Transmission Control Protocol / Internet Protocol)

Dies ist das zugrundeliegende Protokoll, über den der Datenaustausch stattfindet. Darauf aufbauend gibt es eine Reihe von Dienste wie z.B. →*telnet*, →*FTP* usw.

3. Die hier aufgeführten Begriffe und Dienste sind zwangsläufig nur eine sehr kleine Auswahl. Es gib noch eine Reihe von weiteren Begriffe und Diensten im Internet. Dies würde jedoch den Umfang dieses Buches sprengen.

Abbildung 43: Das Internet

22.1.2 telnet

telnet ist zum einen ein Protokoll, zum anderen ein Programm, mit dem man sich auf anderen Rechnern einloggen kann. Dazu benötigt man den Namen des gewünschten Rechners (z.B. „ba-stuttgart.de“) bzw. dessen Internet-Adresse (z.B. 141.31.1.1).

22.1.3 FTP (File Transfer Protocol)

FTP ist ein Protokoll, das auf →TCP/IP aufsetzt und den Austausch von Dateien über Rechnerwelten hinweg definiert.

22.1.4 ftp (file transfer program)

ftp ist ein interaktives Programm, das über →FTP eine Verbindung zu einem anderen Rechner aufbaut. Nach dem Aufbau der Verbindung kann mit dem Datei-Transfer begonnen werden.
Drei Dinge braucht man für eine erfolgreiche Verbindung: eine Benutzerkennung (immer), das entsprechende Passwort (fast immer), und noch eine Account-Information (fast nie). Fehlen diese Informationen, kann man es auch mit →*anonymous ftp* versuchen.

22.1.5 anonymous ftp

Man spricht von *anonymous ftp*, wenn es einen Benutzer *anonymous* gibt, mit der man sich auf der Gegenseite anmelden kann. Als Passwort wird üblicherweise die eigene Email-Adresse verwendet.

22.2 Internet Dienste

22.2.1 finger

finger ist ein Programm, das einige Informationen über einen Benutzer am lokalen Rechner oder aber auch an einem anderen Rechner ausgibt.

22.2.2 archie

Archie ist eine Internet-Datenbank. Abfragen können über →telnet, aber auch über Email erfolgen.

22.2.3 WWW (World Wide Web)

WWW (oder kurz 3W) ist ein hypertext-basiertes Informationswerkzeug im Internet. WWW-Dokumente werden in HTTP erstellt, die Graphiken enthalten, aber auch Verweise auf Resourcen von anderen Rechnern enthalten können. Um solch ein HTTP-Dokumente anzeigen zu können, gibt es ASCII-Browser (die natürlich nur den Text anzeigen könne) und Multimedia-Browser. Der bekannteste Vertreter unter Unix ist MOSAIC und Netscape.

Es gibt eine Reihe sogenannter WWW-Server, die solche HTTP-Dokumente bereithalten (u.a. auch im Alcanet), die sich dann über solche eine WWW-Browser auf dem Bildschirm angezeigt und ausgedruckt werden können. Und täglich kommen neue WWW-Server hinzu.

Als Manko gilt noch das Erstellen solcher Dokumente, da die Dokumente oft von Hand erstellt oder nachbearbeitet werden müssen. Aber auch hier tut sich einiges: so gibt es z.B. von der *Cyberleaf* von der Firma *Interleaf*, mit der sich Dokumente in HTTP-Dokumente konvertieren lassen, und es nur eine Frage der Zeit, bis hier andere Firmen nachziehen und vielleicht sogar einen Editor auf HTTP-Basis anbieten.

Bei all der Euphorie, die sich im Moment bezüglich dem *World-Wide-Web* ausbreitet, darf eines nicht vergessen werden: da Dokumente auch Bilder und Geräusche enthalten können, wächst die zu übertragene Datenmenge leicht ins Unermeßliche. *World-Wide-Web* wird von einigen Fachleuten schon als *die* Killer-Applikation für´s Internet angesehen, und es ist in der Tat so, daß die übertragenen Datenmengen mit Einführung von *World-Wide-Web* überproportional gestiegen sind (die Netzwerkbetreiber wird´s freuen).

22.2.4 News

Unter News werden im Zusammenhang mit Internet die verschiedene Newsgruppen verstanden. Eine Newsgruppe ist eine Art Zusammenschluß Gleichgesinnter zu einem bestimmten Thema. Es gibt tausender solcher Themen mit der dazugehörenden Newsgruppe. Der überwiegende Teil beschäftigt sich mit technischen Themen allgemeiner Art (z.B. „de.comp.misc“) oder sehr spezieller Themen (z.B. „alt.comp.windows.setup“). Aber es gibt auch nicht technische Themen, von denen jedoch einige in Veruf gekommen sind (z.B. „alt.pictures.erotica.female“).

Innerhalb einer Newsgruppe kann man Fragen stellen, Fragen von anderen beantworten, Erfahrungen austauschen, oder ganz einfach seinen Senf dazu geben (aber bitte recht freundlich).

Um an den Newsgruppen teilnehmen zu können, müssen die Newsgruppen vom Administrator eingerichtet sein. Ferner braucht man noch einen „Newsreader“ (z.B. „mxrn“, s. Abb. 44), um die News lesen und am Meinungsaustausch teilnehmen zu können.

22.2.5 GOPHER, WAIS (Wide Area Information Server)

Diese Dienste dienen zur netzwerkweiten Verteilung von Dokumenten und Informationen, die sich mit Hilfe dieser Programme abrufen lassen. Im Gegensatz zu →WWW handelt es sich hierbei um ASCII-Dokumente mit entsprechender geringerer Anforderung an das Terminal und der Übertragungsrate.

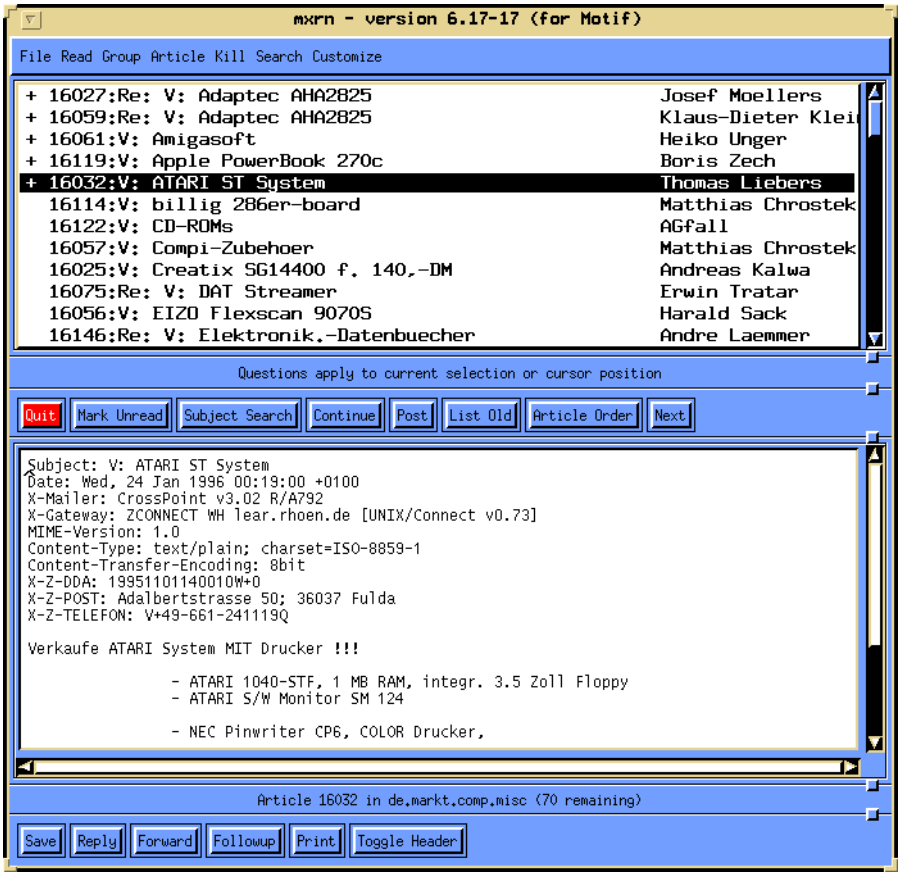


Abbildung 44: Newsreader „mxrn“

22.3 Begriffe

Die Netzgemeinschaft ist eine Welt für sich mit eigenen Regeln und eigenem Dialekt. Dazu gehören neben den „Smileys“ (s. Kap. 12.5.1) und „Akronyme“ (s. Kap. 12.5.2) u.a. folgende Begriffe:

Flame	Beschimpfung
Lurker	einer, der eine Diskussion verfolgt, ohne sich zu beteiligen
Netiquette	der Netz-Knigge
Thread	eine Reihe von Beiträgen zu einem bestimmten Thema

Tabelle 23: Internet-Slang

VIII

Anhang

A Glossary

In diesem Kapitel sind einige Begriffe und Abkürzungen aufgelistet, mit denen man ab und zu bei der Arbeit mit UNIX konfrontiert wird. Die Liste ist nicht vollständig, sondern stellt nur einen kleinen Ausschnitt dar. Beiträge und Berichtigungen zu dieser Liste sind jederzeit willkommen.

AT&T	in den Bell-Laboraties von AT&T erblickte Unix das Licht der Welt
BSD	Berkley Software Distribution
FAQ	Frequently Asked Question: Fragen, die öfters gestellt werden, werden in sogenannte *FAQ´*s zusammengefaßt, zusammen mit den entsprechenden Antorten. Für jedes größere Wissensgebiet existieren heute *FAQ´*s, teilweise auch in Deutsch.
Internet	Die Gesamtheit sämtlicher IP-basierter Netze
IP	Internet Protocoll; eine IP-Adresse besteht aus 4 Zahlen zwischen 0 und 255, die jeweils durch einen Punkt voneinander getrennt sind, z.B. 149.204.218.142

	Damit ist jeder Rechner eindeutig im →Internet identifiziert (vergleichbar mit einer Telefonnummer)
MIT	Massachusetts Institute of Technologie Geburtsort des X-Windows-System
RFC	Request-For-Comment: die meisten Standards in UNIX sind aus sogenannte RFC´s entstanden. Die RFCs enthalten Vorschläge, zu denen jeder seinen Senf dazugeben kann. Irgendwann wird der RFC dann mal abgesegnet und gilt als neuer Standard.
Root Window	Damit wird der Desktop-Hintergrund bezeichnet
SMTP	Simple Mail Transfer Protocol Protokoll, das auf →TCP/IP aufsetzt und den Mail-Austausch regelt
TCP/IP	Transmission Control Protocol / Internet Protocoll
UCB	University of California in Berkley

B man-Beispiel

NAME

ulk - Unbekanntes Linux-Kommando

SYNOPSIS

ulk [-option]
... filename

AVAILABILITY

bekannte Einschränkungen über die Verfügbarkeit des beschriebenen Kommandos

DESCRIPTION

Ein Versuch, das Kommando möglichst einleuchtend zu erklären. Dazu zählt auch die Aufzählung der Ein-/Ausgabe-Dateien, der erwartete Eingabe, die zu erwartende Ausgabe (bzw. Fehlerausgabe).

Internas und Implementierungsdetails werden nicht beschrieben. Vielmehr liegt der Schwerpunkt auf die Beantwortung der Frage „Was macht dieses verdammte Kommando?“

OPTIONS

-option
hier kommt jetzt die Beschreibung der einzelnen Optionen

ERRORS

(alphabetische) Auflistung aller generierten Fehler-Codes

USAGE

Commands
Modifiers
Variables
Expressions
Input Grammar
Environment Variables

EXAMPLES

Falls ein Beispiel mit angegeben sein sollte, so wird der Prompt gewöhnlich als

```
example% ulk ...
```

angezeigt, bzw. wenn es sich um ein Beispiel für den Superuser handelt, mit

```
example# ulk ...
```

FILES

eine Liste von Dateien, auf die von dem Kommando zugegriffen, angegriffen oder vergriffen wird

SEE ALSO

ein Verweis auf ein verwandtes Programm, das einem genausowenig weiterhilft

DIAGNOSTICS

Ursachenforschung, warum dieses Kommando möglicherweise fehlgeschlagen haben könnte

WARNINGS

Dieses Beschreibung gefährdet Ihr geistiges Wohlbefinden und sollte nur in äußersten Notfällen durchgelesen werden, falls kein Ansprechpartner ansprechbar ist.

Zu Risiken und Nebenwirkungen fragen Sie Ihren Arzt oder plagen Sie Ihren zuständigen Systemadministrator.

NOTES

Nehmen Sie nicht alles für bare Münze, was Sie in diesem Kapitel lesen.

BUGS

Der interessanteste Teil mancher Kommandos. Leider fehlt er oft.

AUTHOR

Oliver Böhm
(boehm@informatik.ba-stuttgart.de, http://ba-stuttgart.de/~boehm)

C Konfigurationsdateien

In diesem Anhang finden Sie einige Konfigurationsdateien aus dem /etc-Verzeichnis. Sie dienen lediglich als Beispiel um einen Eindruck zu vermitteln, wie diese Dateien aussehen können.

C.1 XF86Config

```
# XF86Config auto-generated by XF86Setup
#
# Copyright (c) 1996 by The XFree86 Project, Inc.
#
# See 'man XF86Config' for info on the format

Section "Files"
   RgbPath    "/usr/X11R6/lib/X11/rgb"
   FontPath
"/usr/X11R6/lib/X11/fonts/misc:unscaled,/usr/X11R6/lib/X11/fonts/75d
pi:unscaled,/usr/X11R6/lib/X11/fonts/100dpi:unscaled,/usr/X11R6/lib/
X11/fonts/Type1,/usr/X11R6/lib/X11/fonts/Speedo,/usr/X11R6/lib/X11/f
onts/misc,/usr/X11R6/lib/X11/fonts/75dpi,/usr/X11R6/lib/X11/fonts/10
0dpi"
EndSection

Section "ServerFlags"
EndSection

Section "Keyboard"
   Protocol        "Standard"
   AutoRepeat      500 30
```

```
   LeftAlt         Meta
   RightAlt        Meta
   ScrollLock      Compose
   RightCtl        Control
   XkbKeycodes     "xfree86"
   XkbTypes        "default"
   XkbCompat       "default"
   XkbSymbols      "us(pc101)"
   XkbGeometry     "pc"
   XkbRules        "xfree86"
   XkbModel        "pc101"
   XkbLayout       "de"
   XkbVariant      "nodeadkeys"
EndSection

Section "Pointer"
   Protocol        "Mouseman"
   Device          "/dev/mouse"
   BaudRate        1200
   Emulate3Timeout 50
EndSection

Section "Monitor"
   Identifier      "MyMonitor"
   VendorName      "ViewSonic"
   ModelName       "P775"
   HorizSync       30-95
   VertRefresh     50-180
   Modeline  "1152x864"  137.65 1152 1184 1312 1536 864 866 885 902
-hsync -vsync
   Modeline  "1024x768"  115.50 1024 1056 1248 1440 768 771 781 802
-hsync -vsync
   Modeline  "800x600"    69.65 800 864 928 1088 600 604 610 640 -
hsync -vsync
   Modeline  "640x480"    45.80 640 672 768 864 480 488 494 530 -
hsync -vsync
EndSection

Section "Device"
   Identifier      "MyVideoCard"
   VendorName      "Tseng"
   BoardName       "Matrox Mystique"


EndSection

Section "Screen"
   Driver          "Accel"
   Device          "MyVideoCard"
   Monitor         "MyMonitor"
   DefaultColorDepth 32
   BlankTime       2
   SuspendTime     3
```

```
   OffTime         5
   SubSection "Display"
      Depth        8
      Modes        "1152x864" "1024x768" "800x600" "640x480"
   EndSubSection
   SubSection "Display"
      Depth        15
      Modes        "1152x864" "1024x768" "800x600" "640x480"
   EndSubSection
   SubSection "Display"
      Depth        16
      Modes        "1152x864" "1024x768" "800x600" "640x480"
   EndSubSection
   SubSection "Display"
      Depth        24
      Modes        "1152x864" "1024x768" "800x600" "640x480"
   EndSubSection
   SubSection "Display"
      Depth        32
      Modes        "1152x864" "1024x768" "800x600" "640x480"
   EndSubSection
EndSection

Section "Screen"
   Driver          "SVGA"
   Device          "MyVideoCard"
   Monitor         "MyMonitor"
   DefaultColorDepth 32
   BlankTime       2
   SuspendTime     3
   OffTime         5
   SubSection "Display"
      Depth        8
      Modes        "1152x864" "1024x768" "800x600" "640x480"
   EndSubSection
   SubSection "Display"
      Depth        15
      Modes        "1152x864" "1024x768" "800x600" "640x480"
   EndSubSection
   SubSection "Display"
      Depth        16
      Modes        "1152x864" "1024x768" "800x600" "640x480"
   EndSubSection
   SubSection "Display"
      Depth        24
      Modes        "1152x864" "1024x768" "800x600" "640x480"
   EndSubSection
   SubSection "Display"
      Depth        32
      Modes        "1152x864" "1024x768" "800x600" "640x480"
   EndSubSection
EndSection
```

```
Section "Screen"
   Driver          "VGA16"
   Device          "MyVideoCard"
   Monitor         "MyMonitor"
   BlankTime       2
   SuspendTime     3
   OffTime         5
   SubSection "Display"
      Depth        4
      Modes        "1152x864" "1024x768" "800x600" "640x480"
   EndSubSection
EndSection

Section "Screen"
   Driver          "VGA2"
   Device          "MyVideoCard"
   Monitor         "MyMonitor"
   BlankTime       2
   SuspendTime     3
   OffTime         5
   SubSection "Display"
      Depth        1
      Modes        "1152x864" "1024x768" "800x600" "640x480"
   EndSubSection
EndSection
```

D Nützliche Aliase

Zum Schluß noch einige Aliase für die C- und TC-Shell, die sich bei der Arbeit als nützlich erwiesen haben (getestet unter SunOS 4.1.3, SunOS 5.5.1, Linux 2.2 und HP-UX):

alias del	’chmod og-wxr \!*; mv \!* /tmp’
	Löscht eine Datei; zuvor wird aber noch eine Kopie unter /tmp angelegt (für alle Fälle; wird beim nächsten Booten automatisch vom System gelöscht)
alias undel	’mv \!* /tmp .’ Falls eine Datei mit dem *del*-Alias von oben gelöscht wurde, kann sie mit diesem Alias wieder zurückgeholt werden.
alias deldir	’chmod -R og-wxr \!*; /bin/cp -rp \!* /tmp; /bin/rm -r -f \!*’
	Ähnlich „del“, nur daß ein Verzeichnis gelöscht wird (mit Kopie unter /tmp).
	Achtung! Abhängig von der Installation werden Verzeichnisse unter /tmp werden nicht immer

	gelöscht, so daß damit eventuell Speicherleichen übrigbleiben können.
alias undeldir	’cp -rp /tmp/{\!*} .’ Holt das (die) betreffende(n) Verzeichnis(se) wieder zurück, sofern es mit den *deldir*-Alias gelöscht wurde.
alias showpath	’find $path -prune -print’ Zeigt den Suchpfad etwas übersichtlicher an als „echo $path“. Hat außerdem den angenehmen Nebeneffekt, das nicht existierende Verzeichnisse aufgedeckt werden.
alias showmanpath	„echo $MANPATH | tr ’:’ ’\012’ | sort“ Zeigt den Manual-Suchpfad etwas übersichtlicher an.
alias lw	„expand \!* | awk ’length > max { max = length } END {print max}’“ Bestimmt die Zeilenbreite (linewidth) eines Dokuments.
alias lpn2	’cat -n \!:1 | tr -d „\014“ | fold -88 | unexpand -a | pr -2 -w176 -f -h \!:1 | ...(Druck-Kommando)...’ Druckt ein Dokument (z.B. C-Source) 2-spaltig („pr -2 ...“) und mit Zeilennummerierung („cat -n“) aus. Die Seitenlänge kann man beim pr-Kommando mit -l notfalls anpassen. Formfeeds werden rausgefiltert (*tr -d “\014“*), da sie die Formatierung durcheinander bringen. Das Druck-Kommando (z.B. lpr...) ist so aufzusetzen, daß der Drucker 176 Zeichen nebeneinander ausdrucken kann.
alias lpdiff	“sdiff -w 172 \!:1 \!:2 | tr -d ’\014’ | ...(Druck-

	Kommando)...“
	Vergleicht zwei Dateien und druckt sie nebeneinander aus.
alias ztarc	’tar -cfvp - \!:2* | compress -v >! \!:1.tar.Z; ls -l \!:1.tar.Z’
	Erzeugt ein komprimiertes Tar-File mit der Endung „.tar.Z“. Der erste Parameter ist der Name des Tar-Files (ohne Endung), weitere Parameter geben den Namen der Dateien bzw. Verzeichnisse an, die in das Tar-File mit aufgenommen werden sollen.
alias ztarx	’zcat \!:1 | tar -xvfp - \!:2*’
	Liest den Inhalt eines komprimierten Tar-Files aus.
alias ztart	’zcat \!* | tar -tvfp -’
	Zeigt den Inhalt eines komprimierten Tar-Files an.
alias dircopy	’mkdir -p \!:2; pushd \!:1; tar -cvfp - . | (cd \!:2; tar -xfp -); popd’
	Kopiert ein Verzeichnis mit Hilfe des tar-Befehles. Dabei werden im Gegensatz zum cp-Befehl symbolische Links nicht aufgelöst, sondern bleiben erhalten.
alias tar2uucp	’tar -cfvp - \!:2* | compress -v | uuencode \!:2.tar.Z >! \!:1; ls -l \!:1; split -9000 \!:1 \!:1.; ls -l \!:1.??’
	Erzeugt ein komprimiertes Tar-File und wandelt es mit Hilfe von uuencode in handliche ASCII-encoded Dateien (kleiner 600kB), die per Mail verschickt werden können. Sollten die erzeugten Dateien zu groß sein, ist der erste Parameter beim *split*-Kommando

	entsprechend zu verkleinern (z.B. „spilt -1000“ für Dateien < 64kB).
alias file2uucp	’uuencode \!:1 \!:1 | split -9000 - \!:2; ls -l \!:2??’
	Wandelt eine Binär-Datei in ASCII-encoded Dateien um, die per Mail verschickt werden können.

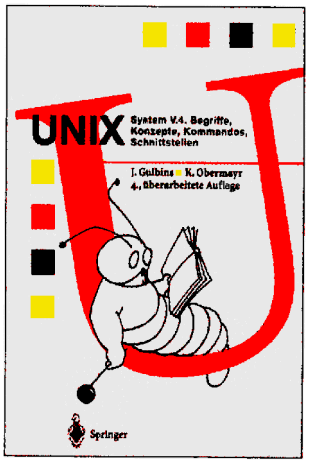
E Literaturverzeichnis

E.1 Bücher

Um die Liste der Bücher nicht zu groß werden zu lassen, versuche ich mich auf die Bücher zu beschränken, die ich empfehlen kann (habe allderdings so gut wie keins gelesen) oder die mir empfohlen wurden bzw. die mir in verschiedenen Zeitschriften durch positve Kritiken aufgefallen sind (das sind fast alle in dieser Liste).

E.1.1 Deutschsprachige Bücher

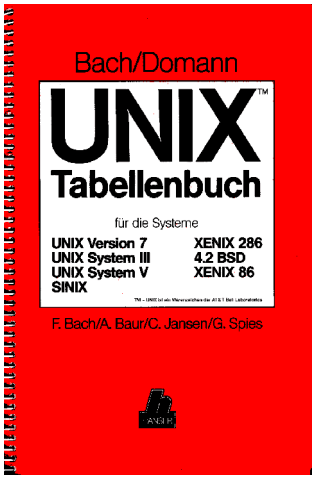
Die meisten guten Unix-Bücher stammen meist aus dem englischen Sprachraum (leider). Deutsche Unix-Bücher sind oft mehr oder weniger geglückte Übersetzungen aus dem Englischen. Aber es gibt auch Ausnahmen: da wäre z.B.



J. Gulbins und K. Obermayr
„**UNIX System V.4. Begriffe, Konzepte, Kommandos, Schnittstellen**“
4. überarbeitete Auflage
Springer Verlag
ISBN 3-540-58864-7
839 Seiten

In diesem Buch ist der Wurm drin, genauer gesagt „Wunix, der Unix-Wurm“, der dieses Buch ziert. Der „Gulbin“, wie er auch nur genannt wird, war lange Zeit eines der wenigen deutschsprachigen Bücher (von einigen wenigen Übersetzungen aus dem Amerikanischen einmal abgesehen), das locker und verständlich geschrieben ist und jetzt nochmal eine Renovierung erhalten hat.

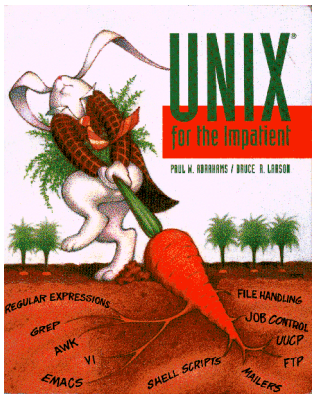
Dieses Buch ist unentbehrlich für jeden Unix-Aufsteiger und verdient unbedingt einen Ehrenplatz im Regal, zeigt es doch sowohl dem unbedarften Besucher als auch dem Unix-Experten, hier ist einer am Werk, der etwas von Unix versteht („Oh, ein Unix-Buch“ bzw. „Oh, ich sehe, sie haben den neuen Gublins, darf ich mal?“).



Fred Bach / Peter Domann
UNIX Tabellenbuch
für die Systeme
UNIX Version 7, XENIX 286,
UNIX System III, 4.2 BSD,
UNIX System V, XENIX 86
SINIX
Carl Hanser Verlag München Wien, 1986[1]
ISBN 3-446-14622-9
287 Seiten
78,- DM

Für den Anfänger gänzlich ungeeignet, aber als Nachschlagewerk hervorragend geeignet, vor allem dann, wenn man häufiger das (Unix-)System wechseln muß. Auch bei Portierungen zwischen den einzelnen Unixen leistet es wertvolle Hilfe (vor allem bleibt es durch die Ringbuch-Bindung auch an der Stelle offen, die man aufgeschlagen hatte).

E.1.2 Englischsprachliche Bücher



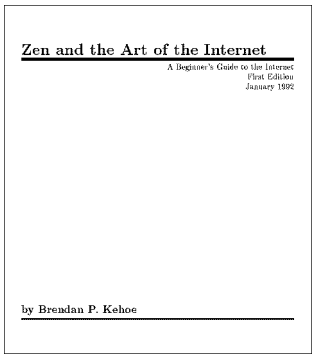
Paul W. Abrahams - Bruce R. Larson
Unix for the Impatient
Addison-Weslsy Publishing Company, 1992
ISBN 0-201-55703-7
560 Seiten

Schon lange her, daß ich dieses Buch auf meinem Schreibtisch stellen durfte (zwischenzeitlich habe ich die Stelle gewechselt, das Buch nicht).

1. Hatte leider keine aktuellere Ausgabe zur Verfügung, aber ich nehme an, daß es inzwischen eine neuere Ausgabe wohl geben wird.

Das Buch wendet sich an den „Ungeduldigen“, der die Antwort auf sein Problem *sofort* haben will. Als Nachschlagewerk ist es jedenfalls ganz gut zu gebrauchen. Aber auch als Quer-Einstieg in Unix, das unbekannte Betriebssystem, ist es geeiget. Es erklärt die wichtigsten Dinge und Konzepte über Unix und dessen Umfeld, ohne den Leser mit unnötigen Ballast zu langweilen.

Inzwischen gibt es auch eine auf 825 Seiten angewachsene „Second Edition“ zum Preis 58,- DM.



Brendan P. Kehoe:
Zen and the Art of the Internet
(A Beginner´s Guide to the Internet)
First Edition
January 1992

Dieses Buch ist vor allem für den Einstieg in die Welt des Internets geeignet und kann (wie sollte es auch anders sein) über´s Internet bezogen werden[2]. Obwohl es nicht mehr ganz taufrisch ist[3] (so fehlt z.B. das „World Wide Web“), ist als Anfangslektüre hervorragend geeignet, um die erste Ausflüge ins Internet wagen zu können oder auch ganz einfach nur, um sich einen Überblick über´s Internet zu verschaffen.

Inzwischen gibt es wohl auch eine deutsche Übersetzung („Zen und die Kunst des Internets“) im Markt&Technik-Verlag, die vergriffen ist.

---

2. z.B. auf *ftp.stgl.netd.alcatel.de (149.204.42.51)* als komprimierte (→compress) Postscript-Datei unter */pub/misc/zen10.ps.Z*

3. kann sein, daß es inzwischen eine überarbeitete Ausgabe gibt; für sachdienliche Hinweise wäre der Autor dankbar

Andrew Oram and Steve Talbot:
Managing Projects with make
O'Reilly & Associates, Inc.
ISBN 0-937175-90-0

Was sie schon immer über *make* wissen wollten - hier steht´s drin. Leider auf englisch.

E.2 Zeitschriften

iX
Multiuser Multitasking Magazin
Heise Verlag
DM 7,50

Laut eigenem Bekunden die meistgekaufte Unix-Zeitschrift Europas. Das Hauptaugenmerk dieser Zeitschrift liegt im Unix-Bereich und Themen rundum um Unix, aber nicht nur. Die Artikel sind i.d.R. auch für nicht-Unix-Freaks recht verständlich geschrieben und man erhält recht gute Tips und Einblicke in die verschiedene Bereiche von Unix. Auch über

Entwicklungen rund um Unix sowie über die diversen Bezugsquellen wird man recht gut informiert.

UNIX^open
Praxiswissen für den Einsatz offener Computersysteme
AWi-Verlag
DM 6,80

Scheint mir mehr ein Presseorgan der anbietenden Firmen im Unix-Bereich zu sein. Viele Artikel stammen von Firmen, die auf diese Art ihre Produkte und Entwicklungen kundtun wollen. Aber es kommen auch Grundlagen- und Praxisberichte vor.

Mein Tip: die UNIX^open liegt oft auf diversen (Unix-)Messen zum Mitnehmen aus (oftmals die letzten 3 Ausgaben). Zugreifen und sich selber ein Bild davon machen.

E.3 Weitere Quellen

E.3.1 Internet

Das Internet selbst bietet eine Fülle von Artikeln und Informationen an. So gibt es unter

ftp://rtfm.mit.edu/pub/usenet/new.answers/books

eine Hitliste von Samuel Ko (sko@helix.net) über die 150 besten Bücher über und um Unix. Diese Liste ist u.a. eine Sammlung aus den verschiedenen Newsgroups und enthält neben den allgemeinen Daten auch jeweils eine Kurzkritik[4] zu den verschiedenen Büchern. Leider enthält diese Liste nur englisch-sprachige Bücher.

Diese Liste enthält auch Verweise auf weitere Bücher, so z.B. die YABL („Yet Another Book List“) von Mitch Wright:

ftp://ftp.rahul.net/pub/mitch/YABL

Hier werden hauptsächlich Bücher über Unix und C besprochen. Wer lieber am Thema „Internet“ interessiert ist, kann sich ja unter

ftp://rtfm.mit.edu/pub/usenet/news.answers/internet-services

die „Unofficial Internet Book List“ anschauen.

E.3.2 O’Reilly-Verlag

O’Reilly-Bücher genießen einen guten Ruf in der Unix-Welt. Sie beschäftigen sich fast ausschließlich mit Themen rund um Unix. Oft sind es nebenberufliche Autoren aus der Praxis, die „ihr“ Werk mit sehr viel Idealismus und Engagement (und oft auch unter Mithilfe der Internet-Gemeinde) erstellt haben. Man kann diese Bücher fast durchweg empfehlen, oftmals gibt es auch deutsche Übersetzungen der Bücher.

Will man sich einen Überblick über die Bücher des O’Reilly-Verlags verschaffen, so kann man unter *ftp.ora.com* fündig werden (nach einer Datei *book.catalog.Z* oder *bookcat.zip* Ausschau halten). Man findet auf diesem Server auch jede Menge Beispiele, die in ihren Bücher verwendet werden.

Auch über *Gopher* (`telnet gopher.ora.com`[5]) oder über’s *World Wide Web* (`http://www.ora.de/`) kann man in kann man in ihrem Katalog rumstöbern. Fragen richtet man am besten an *anfragen@ora.de*.

4. z.B. „*A fairly informal (funny) and non-technical introduction to Unix...*“ (über „Unix for Dummies“)

5. login: *gopher* oder *gopher.ora.com*

Index

Symbols

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z