Technische Praxis der Computersysteme

Teil 1

Einführung in die Benutzung und Installation

eines Linux/Unix-Systems

Roland Steinbauer, Martin Piskernig, Andreas Nemeth, Bernhard Lamel

Version 1.5, Oktober 2005

Inhaltsverzeichnis

Vorbemerkungen zu Version 1.5

Version 1.5 dieses Skriptums basiert nach wie vor auf dem bewährten Originalentwurf von Roland Steinbauer. Es enthält die ersten grösseren Änderungen seit 2003. Ich habe begonnen, Beispiele aus den Folien in den Buchteil des Skriptums zu übernehmen; in der Vorlesung werden zum Teil nicht idente, aber entsprechend analoge Beispiele vorgeführt.

Die Versionsnummer liegt noch immer unter der nächsten ganzzahligen Grenze; für Version 2.0 plane ich insbesondere auch einen Index, der in dieser Version allerdings noch fehlt.

Insgesamt konnte sich Linux in den letzten Jahren als vollwertige Betriebssystem-Alternative etablieren, und die Entwicklungsgeschwindigkeit hat um ein vielfaches zugenommen. Umso wichtiger ist es, mit den Grundlagen vertraut zu sein–man kann mit den entsprechenden Kenntnissen produktiver arbeiten, und zur verantwortungsvollen Administrationstätigkeit sind sie unerlässlich. Noch mehr als in den Vorgängerversionen wählen wir einen Weg, der weder distributionsspezifisch ist, noch die Verwendung des letzten technischen Tricks erfordert.

Inzwischen liegt die Verantwortung für sämtliche Fehler in diesem Skriptum bei mir; über Rückmeldungen freue ich mich, insbesondere über Korrekturen (Mail an `bernhard.lamel@univie.ac.at`).

Bernhard Lamel, September 2005

Vorwort

Die vorliegende Version 1.0 ist eine überarbeitete Neuauflage des Skriptums vom vorigen Wintersemester. Ich habe vor allem versucht die Kapitel 2, 3 und 6 klarer zu gestalten und einige zu komplizierte Formulierungen zu glätten. Außerdem hoffe ich den Großteil der vorhandenen Druckfehler gefunden und ausgebessert zu haben. Schließlich habe ich auch einige Aktualisierungen und kleinere Anpassungen über das ganze Skriptum hinweg vorgenommen.

Mein besonderer Dank gilt Oliver Fasching, der das gesamte Skriptum mit viel Sorgfalt korrekturgelesen hat. Weiters möchte ich den vielen Hörern der Vorlesung im vergangenen Wintersemester danken, die durch ihre Anregungen und Vorschläge zur Verbesserung des Skriptums beigetragen haben. Leider war es aus zeitlichen Gründen nicht mehr möglich, einen Sachindex anzulegen. Schließlich ist es wiederum unserem Sekretariat zu verdanken, daß das Skriptum rechtzeitig fertiggestellt werden konnte.

Roland Steinbauer, Oktober 2001

Vorwort zur Version 0.98 (WS 2000/01)

Das vorliegende Skriptum ist anläßlich meiner Vorlesung „Praxis der Computersysteme, Teil 1“ im Wintersemester 2000/2001 am Institut für Mathematik der Universität Wien entstanden. Es baut auf der Rohfassung des Handouts zur (einstündigen) Vorlesung aus dem Wintersemester 1999/2000 auf, ist aber wesentlich umfangreicher und (hoffentlich) reifer geworden. Insbesondere sind nicht nur alle in der Vorlesung verwendeten Folien, sondern auch viel zusätzliches Material enthalten, und so kann das vorliegende „Werk“ (fast) als eigenständige Einführung in die Benutzung und Installation eines Unix/Linux-Systems gelten. Sowohl die Vorlesung als auch das Skriptum sind zweisemestrig bzw. -teilig konzipiert; der vorliegende erste Teil bildet die Grundlage für den zweiten Teil mit Schwerpunkt Netzwerk-Systemadministration unter Linux.

Ich habe mich bemüht, die Erfahrungen und Rückmeldungen der letztjährigen Vorlesung in meine Vorbereitung einzubeziehen und hoffe, daß diesem Unterfangen zumindest teilweise Erfolg beschieden war. Essentielle Beiträge zu mehreren Kapiteln verdanke ich Martin

Piskernig und Andreas Nemeth. „Meinen“ Tutoren Matthias Zeichmann, Mark Heinzle und Barbara Pocza danke ich für die große Unterstützung. Florian Wisser möchte ich darüber hinaus für seinen kreativen und persönlichen Einsatz nicht nur bei der Entstehung dieses Skriptums danken. Michael Kunzinger hat große Teile des Skriptums korrekturgelesen, Hermann Schichl und Gerald Teschl haben mich in allen Linux-Fragen sehr unterstützt und Sascha Husa hat ursprünglich mein Interesse für Systemadministration geweckt. Schließlich wäre das Skriptum ohne die freundliche Mithilfe unseres Sekretariats nicht rechtzeitig fertigzustellen gewesen.

Viele Themen hätte ich gerne noch genauer und ausführlicher behandelt und manche Formulierung hätte ich noch gerne verfeinert; der zeitliche Rahmen des Projekts hat dies jedoch leider verhindert, sodaß ich die Versionsnummer gegenüber dem Vorjahr zwar erheblich erhöht aber doch unter 1.0 gewählt habe. Die Verantwortung für alle Fehler, Ungenauigkeiten und Unklarheiten liegt selbstverständlich bei mir: Alle Rückmeldungen, Anregungen und Beschwerden daher bitte an meine Adresse (`roland.steinbauer@univie.ac.at`). Und natürlich ergeht (wie schon im Vorjahr) an dieser Stelle an alle Interessierten die Einladung an einer verbesserten Version fürs nächste Mal mitzuarbeiten. Weiteres Material, Informationen und Links zur Vorlesung und zum Thema finden sich unter `http://www.mat.univie.ac.at/praxis`.

Roland Steinbauer, November 2000

Statt einer Einleitung

Statt einer Einleitung, die den Geist von Linux und/oder Unix beschwört, möchte ich hier eine kleine Reiseanleitung geben; denn es ist ein Reisebeginn, um den es sich handelt. Ein Betriebssystem wie Linux kann man weder in einer Vorlesung, noch durch Lesen eines Skriptums oder eines Buches lernen. Nur die Grundlagen kann man so erfahren, und durch Nachschlagen wird man sich hin und wieder gewisse Einzelheiten ins Gedächtnis (zurück)rufen; also genau das, was man von einem guten Reiseführer erwartet. Die Reise, die will schon selbst unternommen sein!

In diesem Sinne will die Vorlesung und das Skriptum dazu beitragen, einen guten Reisebeginn zu ermöglichen. Nicht die neuesten technischen Tricks (die vielleicht schon in der nächsten Version Schnee von gestern sind), sondern die grundlegenden Konzepte sollen im Vordergrund stehen. Mit einem (guten) Verständnis für das Funktionieren des Systems, seine Teile und ihr Zusammenwirken, wird es möglich, konkreten Problemen im Alltag erfolgreich zu begegnen. Hier soll soviel von den Grundlagen vermittelt werden, daß ein kreatives Weitergehen möglich wird, die selbständige Beschäftigung mit dem System nicht zur Frustration, sondern zur spannenden, herausfordernden und letztlich erfolgreichen Reise wird.

Wichtig ist allerdings, daß man sich wirklich *selbst* auf die Reise begibt, d.h. praktische Erfahrung sammelt. Dazu dienen natürlich die begleitenden Übungen, wo das Lösen erster praktischer Aufgaben trainiert wird. Sie können aber naturgemäß ebenfalls nur einen Anfang darstellen, eine Aufforderung weiterzugehen und selbst weitere Praxis zu sammeln.

Es gibt eine Fülle einschlägiger Literatur zum Thema, die für jeden Geschmack das Passende bieten kann. Neben Büchern sind Unmengen an Material im Internet zu finden (siehe auch und vor allem Kapitel 4). Bei der Vorbereitung der Vorlesung und des Skriptums habe ich mich hauptsächlich folgender Quellen bedient (deren Auswahl allerdings eher Auskunft über meinen Geschmack gibt).

1 Matt Welsh, Lar Kaufman, *Running Linux*, 4th Edition, O'Reilly & Associates, Inc, (Sebastopol, CA, 2002).

2 Lary Greenfield, *The Linux Users' Guide*, `http://www.ibiblio.org/pub/Linux/docs/linux-doc-project/users-guide/` (1996)

3 Evi Nemeth, Garth Snyder, Scott Seebass, Trent R. Hein *Unix System Administration Handbook*, 3rd Edition (Prentice Hall, New Jersey, 2000)
(Ein Klassiker! Die 3. Auflage behandelt erstmals auch Linux)

Weiteres Material findet sich unter `http://www.tldp.org`, Hinweise auf (viele) andere Bücher unter `http://www.linux.org/books/index.html` (siehe auch Kapitel 4).

Das Skriptum gliedert sich in 10 Kapitel. Zunächst wird in Kapitel 0 mit einer Begriffsklärung ein gemeinsamer Sprachgebrauch hergestellt. Vielen Hörern und Lesern wird das meiste darin bekannt sein. Gerade im EDV-Bereich ist es aber wichtig, den Anfänger nicht durch vorzeitigen Gebrauch (nicht erklärter) technischen Fachsprache zu „erschlagen". Kapitel 1 bietet eine historische Einleitung ins Thema und soll das Verständnis dafür wecken, warum Unix/Linux so ist, wie es ist. Wichtig ist es mir, das Konzept der „Freien Software" (Abschnitt 1.2) gründlich zu vermitteln. Das folgende Kapitel 2 bietet eine Einführung in den technischen Aufbau eines Unix/Linux-Systems. Eine erste Begegnung mit der Bedienung eines Linux-PCs findet in Kapitel 3 statt. In Kapitel 4 wird erklärt, wie man sich selbst weiterhelfen kann, wenn man einmal nicht sofort weiter weiß. Kapitel 5 beinhaltet ein kleines Tutorial im Umgang mit dem wichtigen Editor `vi`, der schon im Kapitel 6 (dringend) benötigt wird, um weitere Grundelemente der Bedienung eines Linux-Systems zu erlernen. Kapitel 7 führt in die grafische Benutzeroberfläche von Linux ein. Die Installation des Systems und das Management der grundlegenden Systemdienste wird in den Kapiteln 8 und 9 detailliert dargestellt.

Beginnen möchte ich unsere Reise mit zwei meiner Lieblingszitate aus den oben angesprochenen Texten.

Statt einer Einleitung (1)

There is little else that you need for your trip. The following three items may come in handy: a towel, a strong cup of coffe, and (optionally) a computer...

Linux itself is somewhat of a rebellion against the world of commercial software. The world of Linux is not an organized one. You must expect the unexpected. You must always yield the driving force behind free software: that being the desire—no, *need*—to develop and maintain the most succinct and powerful system anywhere. To put it in a nutshell: you must hack.

(Welsh, Kaufman, „Running Linux“)

Folie 1

Statt einer Einleitung (2)

The best way to learn Unix and Linux is by using them.

Use Linux for everything you can. Experiment. Don't be afraid—it's *possible* to mess things up, but you can always reinstall. Keep backups and have fun!

(Greenfield, „The Linux Users' Guide“)

Folie 2

0 Begriffsklärung

Aufgabe dieses Kapitels ist es, die Bedeutung grundlegender Begriffe wie Hardware, Software, Betriebssystem, Anwendung, etc. zu erklären. Dabei sollen weder besondere Finessen behandelt noch spezielle Feinheiten erklärt werden, sondern eine einfache operationale Klärung der später verwendeten Begriffe erfolgen; sozusagen ein gemeinsamer Sprachgebrauch festgelegt werden.

Unter der *Hardware* eines Computers versteht man alle (physikalisch) vorhandenen Bestandteile; mit anderen Worten: „alles was in den Computer eingebaut ist“, bzw. „das man angreifen kann“.

Das sind mindestens der *Prozessor* (bzw. die Prozessoren; CPU, Central Processing Unit), der das eigentliche Herzstück des Computers darstellt. Dieser befindet sich, wie auch der *Hauptspeicher* (Memory, RAM) auf der *Hauptplatine* (Motherboard, Mainboard). Auf dieser wiederum befinden sich weitere Hardwaregeräte wie z.B. der *IDE-Controller* bzw *SATA-Controller*, der den Datentransfer mit *Massenspeichergeräten* wie *Festplatten* oder *CDROM-, CDRW-,DVD-,usw Laufwerken* herstellt, der *Floppy-Controller*, der für den Datentransfer mit den *Diskettenlaufwerken* zuständig ist, sowie *parallele* (Druckerport) und *serielle* (Maus, Gameport) *Schnittstellen*. Weitere Schnittstellen auf dem Motherboard sind *PS/2*- (Maus, Tastatur) und *USB-Schnittstellen* (Maus, Tastatur, Webcam, Drucker, etc.; der USB-Bus ersetzt inzwischen fast alle alten parallelen, seriellen und PS/2 Schnittstellen die auch als *Legacy Ports* bezeichnet werden). Eine weitere wichtige serielle Verbindung, die in vielen modernen Mainboards integriert wird, ist die *Firewire-Schnittstelle* (IEEE 1394, wird zum Beispiel für digitales Video verwendet). Weiters findet man auf vielen Mainboards bereits integrierte Ethernet-Schnittstellen für die Netzwerkverbindung.

Zusätzlich zu diesen sogenannten *Onboard*-Komponenten verfügt die Hauptplatine über *Steckplätze* (Slots), an denen weitere Hardwarekomponenten (Platinen) angebracht werden können. Meistens ist die *Grafikkarte* (Video Chipset) an einem *AGP*- oder (veraltet) an einem *PCI-Slot* angebracht. Moderne Mainboards stellen inzwischen wieder spezielle, rasche PCI-Slots für Grafikkarten zur Verfügung.

Weitere optionale Geräte, die meist auf *PCI*- oder (älter) *ISA-Slots* angebracht sind, können z.B. *Soundkarten, Netzwerkkarten,* oder *SCSI-Adapter* sein, die die Datenübertragung zu externen *Audio-Geräten* (z.B. Boxen, Sterioanlage), einem *Netzwerk* (z.B. Ethernet, Token Ring) oder *SCSI-Massenspeichergeräten* (Festplatten, CDROM-, CDRW-Laufwerke) ermöglichen.

Weiters werden natürlich auch *Maus, Tastatur* und *Bildschirm* (Monitor) als Hardwarekomponenten eines Computersystems bezeichnet. Unter *Systemressourcen* versteht man alle von der Hardware zur Verfügung gestellten Funktionen.

Unter *Software* versteht man die Gesamtheit aller Programme, die auf dem System laufen. Die wichtigste und grundlegende Komponente davon ist das *Betriebssystem* (Operating System, OS). Es stellt den *Applikationen* (Anwendungssoftware, also den Programmen, „mit denen man arbeitet“) die Systemressourcen zur Verfügung. Es verwaltet die Verwendung der Hardware des Systems, teilt verschiedenen Prozessen verschiedene Ressourcen zu, räumt den Prozessen verschiedene Rechte ein, auf Ressourcen zuzugreifen und ist für die Abwicklung der Kommunikation zwischen den Prozessen verantwortlich. Dabei versteht man unter einem *Prozeß* ein laufendes Programm (Instanz) mit seinem privatem Datenbereich (vom OS zugeteilt). Wenn ein Programm z.B. zweimal gestartet wird, laufen zwei Prozesse, mit ihrem jeweiligen eigenen privaten Datenbereich.

Wichtige und weit verbreitete Betriebssysteme in der PC-Welt sind unter anderem MsDOS, *Microsoft Windows* in seinen verschiedenen Versionen (Win3x, Win9x, WinME, WinNT, Win2000, WinXP...), *OS/2* (IBM), *MacOS*, *Unix* in seinen verschiedenen Spielarten, etc.

Hardware (1)

- CPU
- Speicher (RAM)
- Hauptplatine (Motherboard)
- Schnittstellen
  - Legacy ports:
    - Parallel (Drucker)
    - Seriell (Maus, Joystick, Modem,...)
    - PS/2 (Maus, Tastatur)
  - USB (Maus, Tastatur, Drucker, Scanner, ...)
  - Firewire
  - Ethernet (LAN)
- Massenspeichergeräte
  - Diskettenlaufwerk
  - Festplatten (IDE, SCSI)
  - CDROM-,CDRW-Laufwerke
  - Bandlaufwerke (Streamer,...)

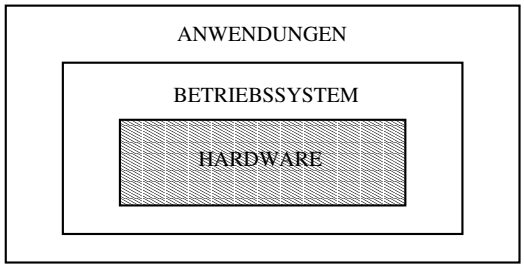
Folie 3

Hardware (2)

- Controller (Steckkarten oder Onboard)
  - Floppy
  - IDE
  - SCSI
- Grafikkarte
- Netzwerkkarte
- Soundkarte
- ...
- Bildschirm, Tastatur, Zeigegerät (Maus)

Folie 4

Betriebssystem



- stellt den Anwendungen Systemressourcen zur Verfügung
- wichtige Betriebssysteme: MsDOS/Windows, OS/2, MAC, Unix,...

Folie 5

1 Historische Einleitung

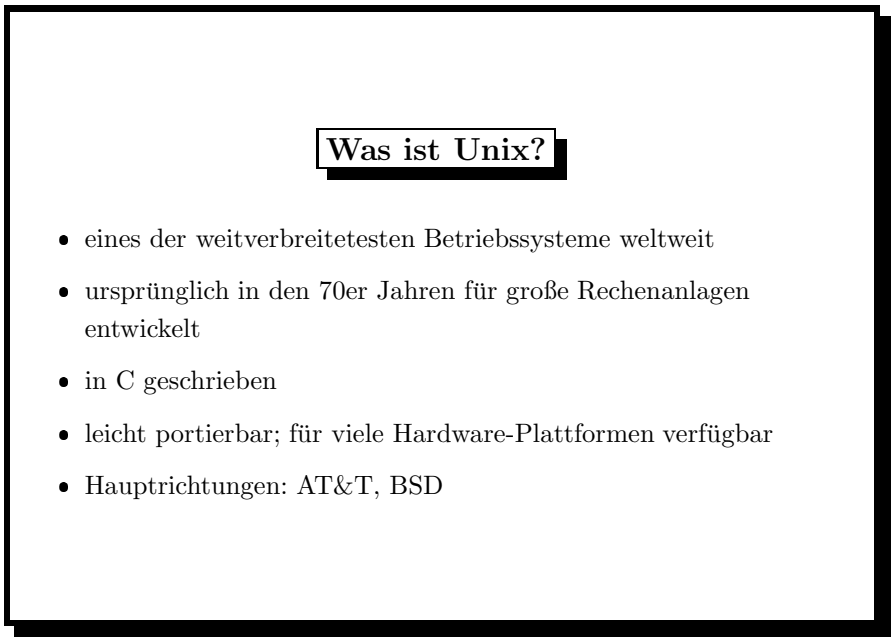
Dieses Kapitel versucht sich dem Thema Unix/Linux von seiner Entstehungsgeschichte her zu nähern. Hier wird erklärt, was Unix und Linux sind, wie sie entstanden sind und warum es wichtig ist, das zu wissen.

Unix ist eines der weltweit verbreitetsten Betriebssysteme und gleichzeitig auch eines der ältesten. Es geht zurück auf ein privates Forschungsprojekt einer kleinen Gruppe von Entwicklern aus dem Jahre 1969.

1.1 Eine kurze Geschichte von Unix

Bereits Mitte der 60er Jahre arbeiten *Bell Telephone Laboratories* (Bell Labs), eine Abteilung von *AT&T*, General Electric und eine Abteilung des MIT (Massachusetts Institute of Technology) an einem gemeinsamen Betriebssystem mit Namen *Multics*. Als 1969 Bell Labs diese Kooperation verlassen, konzipieren *Ken Thompson* und *Dennis Ritchie* ein Betriebssystem für Bell, das 1970 von Ken Thompson (in Assembler auf einer DEC-PDP11/20) implementiert wird. *Bill Kernighan*, ebenfalls von Bell, tauft dieses Betriebssystem *Unix* (als Anspielung auf Multics).

1973 erfindet Dennis Ritchie die Programmiersprache *C* und Unix wird von Ritchie und Thompson in C neu geschrieben. Damit wird Unix als erstes Betriebssystem portierbar (d.h. leicht auf andere Hardware-Plattformen übertragbar), was stark zur weiten Verbreitung des Systems beiträgt. Im Gegensatz zu anderen damaligen Betriebssystemen kann ein Großteil des Quellcodes unverändert übernommen und für eine neue Hardware (re)compiliert werden.



Folie 6

In den 70er Jahren verbietet ein Gerichtsurteil der in der Telephonbranche ansässigen Firma AT&T am Computermarkt zu konkurrieren. Daraufhin wird Unix inklusive Quellcode für geringe Lizenzgebühren im akademischen Bereich freigegeben und findet dort weite Verbreitung. AT&T-Unix erreicht 1983 seinen Höhepunkt mit dem Release des *System V* durch die von Bell gegründete Unix Support Group (USG).

Aufgrund der Ankündigung von AT&T, Unix doch zu kommerzialisieren (1979), entschließt sich die „Computer System Research Group"(CSRG) der *University of California* in *Berkeley* eine eigene Unix-Variante namens *BSD* (Berkeley Software Distribution) herauszubringen. Die Entwicklung von BSD-Unix kulminiert 1993 mit der Veröffentlichung von *4.4BSD*.

Unix +/- (1)

- flexibel, gehört nicht *einer* Firma
- gute Entwicklungsumgebung
- Netzwerkfähigkeit (1. TCP/IP-Implementation in Unix)
- viel freie Software verfügbar
- Open System
  - Application-Portability
  - User-Portability
  - System-Interoperability

Folie 7

Unix +/- (2)

- unhandliche Benutzeroberfläche
- wenig einheitliche Standards für Admin
- PC-Welt ?!?

„[...] The real reason of Unix's popularity? Many hackers[a] feel that Unix is the Right Thing—the One True Operating System."
(Welsh, Kaufman, „Running Linux")

[a] What I mean by a "hacker" is a feverishly dedicated programmer, a person who enjoys exploiting computers and generally doing interesting things with them. This is in contrast to the common connotation of "hacker" as a computer wrongdoer or outlaw.

Folie 8

System V von AT&T und 4.2BSD (1983) sind gewissermaßen die Stammväter aller aktuellen Unix-Versionen; ihr Einfluß ist bei allen Unix-Varianten festzumachen. So hat z.B. Linux sowohl System V-artige (Programmierinterface), wie auch BSD-artige (Systemadministration) Komponenten.

Da BSD-Unix ursprünglich Teile des AT&T-UnixCodes verwendet, war zum Betreiben des Systems eine AT&T-Lizenz nötig. Diese Konstellation führt Mitte der 80er Jahre zu einer Serie von gerichtlichen Klagen und Gegenklagen zwischen AT&T und der University of California. Die CSRG setzt sich daraufhin das Ziel, allen AT&T-Code aus BSD-Unix zu entfernen. Bevor diese ebenso aufwendige wie auch aufreibende Arbeit abgeschlossen ist, stellt die University of California die CSRG ein. Zuvor (1991) wird noch die zweite Version eines AT&T-freien (eingeschränkten) BSD-Unix unter dem Namen *Net/2* veröffentlicht. BSD-Unix wird von *BSDI* (Berkeley Software Design, Inc.) kommerziell weitergeführt. Erst im Jahr 1994, zwei Jahre nachdem AT&T „sein“ Unix an *Novell* verkauft hat, können die rechtlichen Auseinandersetzungen mit der University of California endgültig beigelegt werden.

Eine kurze Geschichte von Unix (1)

- **AT&T-Unix**

1965 *Multics: Bell Labs*, (AT&T), General Electric und ein Projekt des MIT; Bell Labs steigen aus

1969 *Ken Thompson* und *Dennis Ritchie* skizzieren Betriebssystem für Bell

1970 Thompson implementiert die gemeinsamen Ideen (Assembler); Bill Kernighan nennt das System Unix (als Wortspiel auf Multics)

1973 Ritchie erfindet „C”; Unix in „C” neu geschrieben (→ portierbar)

197*-8* Verbreitung im akademischen Bereich, großer Einfluß auf moderne Systeme

1983 USG: *System V*

1992 AT&T verkauft Unix an Novell

1995 Novell verkauft Unix an SCO

Folie 9

Im Jahr 1995 verkauft Novell „sein“ mittlerweile *UnixWare* genanntes Unix an die *Santa Cruz Operations, Inc.* (SCO, `http://www.sco.com`), die es bis zur Version 7 vertreibt. Schließlich wird SCO von *Caldera Systems Inc.* gekauft und in *Tarantella, Inc.* umbenannt. Caldera und Tarantella stellten im März 2001 *UNIX 8* vor, das UnixWare 7 kompatibel ist und gleichzeitig eine vollständige Linux-Implementierung und -Anwenderumgebung bietet.

Im Sommer 2003 nimmt die Lizenzierungsgeschichte eine weitere interessante Wendung. Nachdem sie sich wieder in SCO umbenannt hatte, klagt jene Firma, welche die Lizenzrechte an Unix System V besitzt, IBM wegen Lizenzverletzungen. IBM habe ihr Urheberrecht verletzt, als es die von SCO für die Entwicklung des eigenen Betriebssystem AIX erworbenen Quellcodes in die Entwicklung von Linux einfliessen liess. SCO behauptet nun, dass ein guter

Eine kurze Geschichte von Unix (2)

- **Berkeley-Unix**

1977 CSRG (University of California) aufbauend auf AT&T-Unix eigene Distributionen: *BSD-Unix*

1983 4.2BSD

198* Lizenzprobleme und Rechtsstreitigkeiten mit AT&T

1991 Net/2 (AT&T-freier Code)

1993 4.4BSD (BSDI)

80er Hardwareverkäufer entwickeln eigene Unix-Systeme aufbauend auf System V oder BSD mit vielen Erweiterungen → Chaos...

90er Bemühungen um einheitliche Standards, POSIX, OSF...
freie Unix-Versionen: FreeBSD, Linux

Folie 10

Teil der Quellcodes des Linux-Kernels von Code der SCO gehört kopiert wurde, und klagte IBM auf 1 Milliarde US-Dollar; dieser Betrag wurde inzwischen auf 3 Milliarden erhöht. Dies hat nachhaltigen Einfluss auf die Linux-Szene, da SCO nun auch von Linux-Usern Lizenzgebühren einheben will; schliesslich benutzen diese ja den laut SCO geklauten Code.

Wie man sich leicht vorstellen kann, führt dies zu einiger Aufregung; schliesslich wird, wie wir später noch erklären werden, die Entwicklung von Linux nicht von einer Firma, sondern von einer grossen Gemeinschaft von Programmierern getragen, die an diesem Projekt unentgeltlich arbeiten und ihre Produkte der Allgemeinheit unentgeltlich zugänglich machen. Diese sind empört über die Unterstellungen von SCO - schliesslich wird ihnen Diebstahl geistigen Eigentums unterstellt. Wie sich diese Klage in den USA auf die Rechtslage hier auswirken wird, ist unklar - ein deutsches Gericht hat auf jeden Fall in einer einstweiligen Verfügung SCO untersagt ihre Behauptungen in Deutschland öffentlich zu machen.

Das Verfahren ist zur Zeit (September 2005) nach wie vor nicht entschieden. Eine ziemlich komplette und äusserst interessante Berichterstattung zu dem rechtlichen Verlauf findet man auf `http://www.groklaw.net`.

Ab Mitte der 80er Jahre beginnen große Hardwareproduzenten wie Sun, DEC, IBM und andere, aufgrund des Marktdrucks, „ein Unix haben zu müssen“, aufbauend entweder auf AT&T oder BSD-Unix, ihre eigenen Unix-Varianten zu entwickeln. Es entsteht eine Fülle von verschiedenen Dialekten und Spielarten des Betriebssystems mit jeweils verschiedenen Erweiterungen und Kombinationen aus System V- und BSD-artigen Features. Das Resultat ist eine fast unüberschaubare Vielfalt verschiedenster Produkte (siehe z.B. den Unix-Stammbaum unter `http://www.levenez.com/unix`), die sich alle mehr oder weniger ähneln und das Bemühen, über (brüchige) Koalitionen von Herstellern zu (Quasi-)Standards zu gelangen, die die Situation allerdings eher noch verkomplizieren (und weitere Varianten hervorbringen).

Eine praktische Konsequenz dieser Vielfalt an Unix Entwicklungslinien ist, daß in fast allen Dialekten mehrere Tools und Utilities zur Erledigung derselben Aufgabe vorhanden sind. Insgesamt ist *Unix nicht ein Betriebssystem, sondern eine generische Bezeichnung für eine Familie von verwandten Betriebssystemen.* Vom Standpunkt des Benutzers/Anwenders aus betrachtet, weisen die verschiedenen Unixe wenig Unterschiede auf, von Innen betrachtet– vom Standpunkt des Systemadministrators–sind die Unterschiede allerdings gravierend (siehe dazu auch `http://www.ugu.com`).

Seit Anfang/Mitte der 90er Jahre sind auch einige „freie“ Unix-Varianten verfügbar; die beiden wichtigsten sind *FreeBSD* und *Linux*. Wir wenden uns im nächsten Abschnitt dem Thema der freien Software zu.

1.2 Freie Software

Die Geschichte der *freien Software* in der Unix-Welt beginnt um 1984; *Richard Stallman*, ein ehemaliger MIT-Entwickler gründet das *GNU*[1]*-Projekt* (`http://www.gnu.org`) mit dem Ziel ein vollständiges Unix-artiges Betriebssystem zu programmieren und frei verfügbar zu machen. Wobei hier Software „frei“ heißt, wenn sie alle vier auf Folie 13 aufgezählten Merkmale aufweist („the four liberties“).

Im Jahr 1985 wird die *Free Software Foundation* (FSF, `http://www.fsf.org`) als gemeinnützige Organisation zur Unterstützung des GNU-Projekts gegründet. In der Folge werden vom GNU-Projekt wichtige Programme (Editoren, Textverarbeitungssysteme, Unixtypische Tools und Utilities) für Unix-Plattformen entwickelt bzw. andere freie Software vom GNU-Projekt (mit-)vertrieben (X-Window, TEX, etc.). Weiters entwickelt GNU die *General Public License* (GPL, `http://www.gnu.org/copyleft/gpl.html`), unter der „freie“ Software vertrieben wird. Sie ist eine Lizenz „to guarantee freedom, not to take it away“ und gibt oft Anlaß zu Mißverständnissen. Die wichtigsten Bestimmungen der GPL sind auf Folie 16 zusammengefaßt.

Heute wird „Open Source“ oft sysnonym mit „freier Software“ verwendet; eigentlich tragen diese zwei Ausdrücke eine unterschiedliche Bedeutung (siehe zum Beispiel `www.gnu.org/philosophy/free-software-for-freedom.html`). Eine Übersicht über ver-

[1] Das Akronym „GNU“ ist rekursiv definiert als „GNU's not Unix“.

Unix History Links

- `http://www.bell-labs.com/history/unix/`
  Bell Labs
- `http://cm.bell-labs.com/cm/cs/who/dmr/hist.html`
  The Evolution of the Unix Time-sharing System by Dennis M. Ritchie
- `http://www.levenez.com/unix/`
  Unix-Stammbaum
- `http://www.hsrl.rutgers.edu/ug/unix_history.html`
  NT und Unix
- `http://www.crackmonkey.org/unix.html`
  $7 History of Unix

Folie 11

Aktuelle Unix-Versionen

- Solaris 10 (Sun)
- HP-UX, Tru64 (HP)
- AIX (IBM)
- HP-UX (HP)
- UnixWare 7,OpenServer 6 (SCO)
- IRIX (SGI)

- FreeBSD
- Linux
- OpenSolaris

Folie 12

schiedene andere Lizenzen, und ihre Verträglichkeit mit der GPL, findet man auf der Webseite der FSF `http://www.fsf.org/licensing/licenses/index_html`.

Etwa 1990 ist das GNU-Projekt beinahe am Ziel angelangt; zu einem vollständigen Unix-artigen Betriebssystem fehlt nur mehr ein funktionstüchtiger Betriebssystemkern (für eine technische Erklärung der Begriffe siehe nächstes Kapitel). GNU entwickelt eine Sammlung von Servern, die auf dem *MACH*-Betriebssystemkern aufsetzen und auf den Namen *HURD* getauft werden. Eine produktionsfertige Version ist aber bis 1991 nicht verfügbar (kommt aber nächstes Jahr heraus), als der erste Linux-Kernel das Licht der Welt erblickt. Das aber ist eine andere Geschichte, die im nächsten Abschnitt erzählt wird.

Free Software

„a matter of liberty not price“
„free as in speech, not beer“

umfaßt die Freiheit, die Software

0 für jeden Zweck zu benützen

1 zu analysieren und für eigene Zwecke zu adaptieren

2 weiterzugeben

3 zu verbessern und die verbesserte Version weiterzuverbreiten

Punkt 1 und 3 setzen insbesondere Zugang zum Quellcode voraus.

Folie 13

1.3 Kurze Geschichte von Linux

Die Urgeschichte von Linux beginnt mit *Minix*, einem kleinen Unix-System für Intel-Hardware. Entwickelt von *Andrew S. Tannenbaum* war und ist es als Studienbetriebssystem gedacht, an dem Studenten die Funktionsweise von (Unix-artigen) Betriebssystemen kennenlernen können. Aus diesem Grund ist Minix bewußt einfach gehalten und wird nur marginal weiterentwickelt. Weitere Informationen über Minix findet man im Internet unter `http://www.cs.vu.nl/pub/minix/index.html`.

Im Jahr 1991 installiert der finnische Informatikstudent *Linus Torvalds* Minix auf seinem ersten PC. Weil er vor allem mit der Hardwarenutzung von Minix unzufrieden ist–es ist die Zeit der ersten 80386er Prozessoren, die erstmals am PC Hardewareunterstützung für Multitasking bieten–beginnt er als Hobbyprojekt sein eigenes Unix-Betriebssystemkern zu programmieren; der Name Linux ist eine Kombination aus Unix und Linus' Vornamen. Schon in einem frühen Stadium beschließt Linus nicht alleine zu arbeiten, sondern lädt Entwickler und Programmierer über das Internet ein, an seinem Projekt mitzumachen (siehe Folie 18). Ein wichtiges Merkmal von Linux ist, daß der (C-)Quellcode von Anfang an freigegeben und über das Internet verbreitet wird (`http://www.kernel.org`); Linux ist GPL lizenziert. Jeder Interessierte mit ausreichenden Programmierkenntnissen konnte und kann zur Weiterentwicklung von Linux beitragen; das ganze Projekt wird nach wie vor von Linus koordiniert und betreut.

Dieser Umstand trug und trägt zur dynamischen, schnellen Weiterentwicklung von Linux bei und ist Grundlage für seinen Erfolg und seine Verbreitung. Mittlerweile wird fast jede

Eine kurze Geschichte des freien Unix

84 Gründung des GNU-Projekts (R. Stallman)

85 Gründung der FSF

90 HURD (Betriebssystemkern)

91 erste Linux-Version

Folie 14

Software Lizenztypen

- proprietär (geschützt)
- Shareware (kostenpflichtig)
- Public Domain (kein Copyright, im Besitz der Öffentlichkeit)
- GNU-General Public License (copyleft)

  „Copyright to guarantee freedom, not to take it away“

Folie 15

General Public License

`http://www.gnu.org/copyleft/gpl.html`

- **nicht** Public Domain (diese ist nicht lizenziert)
- **nicht** Shareware (diese ist nicht frei)
- **ist** lizenziert und urheberrechtlich geschützt
- darf frei benutzt und weitergegeben werden
- darf **verkauft** werden, bleibt aber unter der GPL
- Sourcecode ist inkludiert
- darf verändert und weitergegeben werden, aber nur wieder unter der GPL

Folie 16

(Intel-basierte–und nicht nur diese) Hardware unterstützt und für neue Hardwarekomponenten steht meist bereits nach wenigen Wochen der entsprechende Treiber bereit.

Im Unterschied zu kommerziellen Unix-Varianten, die erst nach langen Entwicklungs- und Testphasen veröffentlicht werden, erscheint ca. alle Monate ein neuer Linux-Kernel im Internet (Sourcecode ($\approx$ 30 MB gezippt) auf `http://www.kernel.org`). Der aktuelle Kernel (Stand 10.9.2005) hat die Nummer 2.6.13. Bis zur Kernel-Version 2.4 gab es parallel zum stabilen Kernel (mit gerade zweiter Versionsnummer) einen Entwicklerkernel (mit ungerader zweiter Versionsnummer).

In der 2.6er Kernelserie gab es erstmals eine vierte Versionsnummer; da im Kernel 2.6.8 ein Fehler sofort zu beheben war, erschien kurz darauf die Version 2.6.8.1. Diese Numerierung wird seitdem fortgeführt; grosse Änderungen werden in der dritten Nummer angezeigt, die vierte Nummer dient zur Verwaltung von Fehlerbereinigungen und von sicherheitsbedingten Änderungen.

Ursprünglich für den 80368-Prozessor entwickelt, gibt es mittlerweile fast keine Hardware mehr, auf der Linux prinzipiell nicht läuft. Es werden alle gängigen Prozessortypen unterstützt: SPARC (Sun), Alpha (Digital), Motorola (Macintosh, Amiga, Atari), RISC (Silicon Graphics, IBM), PalmPilot (3COM), Intel-Architektur (Intel, AMD), und auch die x86_64 Architektur. Linux unterstützt auch Symmetric Multiprocessing (SMP, d.h. mehrere Prozessoren pro System). Wir beschäftigen uns hier im Detail nur mit (für den „normalen“ Benutzer interessantesten) 80x86-kompatiblen Systemen. Da Linux ursprünglich für diese Architektur entwickelt wurde, hat es hier auch den höchsten Entwicklungsstand. Trotzdem kann eine exotische Hardware (zu neu, sehr selten, zu alt) zu Schwierigkeiten in Betrieb oder Installation führen. Grundlegende Hardware wie Festplatte, CD-ROM, Prozessor/Speicher, Grafikkarte, Diskettenlaufwerk, etc. bereitet keine Schwierigkeiten, da die Entwicklung der Treiber mittlerweile sehr rasch vorangeht. Allerdings gibt es spezielle Hardware (Modems und Drucker), die nicht/nie unterstützt werden, da sie speziell für Windows entwickelt wurden. Eine Liste mit all der Hardware, auf der Linux sicher läuft, wird seit den Anfängen auf dem aktuellen Stand gehalten (die Linux Hardware Compatibility List; `http://www.linuxdoc.org/HOWTO/Hardware-HOWTO.html`). Viele Distributionen (siehe Abschnitt 1.4) führen eigene, detailliertere Hardware Compatibility Listen.

1.4 Linux-Distributionen

Linux wird üblicherweise als komplettes Betriebssystem mit Tools, Utilities, und Anwendungssoftware in Form von sogenannten *Distributionen* angeboten. Diese Distributionen werden entweder von Non-Profit-Organisationen oder Firmen vertrieben und weisen untereinander beträchtliche Unterschiede auf. Linux-Distributionen können entweder von der Homepage des Vertreibers aus dem Internet gratis heruntergeladen oder auf CDs gekauft werden.

Neben dem Download ist eine Fülle von Informationen, Dokumentation und Hilfe zu verschiedenen Linux-Themen im Internet zu finden. Eine kleine Auswahl wichtigster Links finden sich auf Folie 20 (siehe auch Kapitel 4).

Gemeinsam ist allen Distributionen, daß sie die offiziellen Linux-Kernels und eine weite Palette von GNU-Tools verwenden; die Installation, Administration und (Standard) grafische Oberfläche können allerdings sehr verschieden sein. Obwohl technisch gesehen Linux nur der Betriebssystemkern ist, wird oft das gesamte Betriebssystem, bestehend aus dem Kernel *und* vielen GNU-Tools und anderer freier Software, das von verschiedenen Organisationen und Firmen vertrieben wird, als Linux bezeichnet; Besser wäre die Bezeichnung GNU-Linux.

Zum Schluß dieses Kapitels werfen wir einen Blick auf verschiedene aktuelle Linux-Distributionen; vorweg folgende Bemerkung: Die Frage nach der „besten“ Distribution ist ähnlich sinnlos, wie die Frage nach dem „besten“ Betriebssystem. Verschiedene Distributionen/Betriebssysteme haben verschiedene Stärken und Schwächen und werden für verschiedene Benutzerkreise mit verschiedenen Zielen und verschiedenen Ansprüchen entwickelt. Für die (persönliche) Auswahl einer Distribution werden verschiedene Kriterien ausschlaggebend sein, wie etwa: Vollständigkeit/Umfang der mitgelieferten Software, Verbreitung, Sprachanpassung, eigene Ziele/Ansprüche in der Verwendung, leichte Administrierbarkeit, möglichst

Was ist Linux?

- Implementierung eines Unix-Kernels für i386-PCs,...
- frei verfügbar (GNU-GPL)
- geschrieben hauptsächlich von *Linus Torvalds* (Helsinki) ohne Verwendung von AT&T- oder BSD-Code
- Entwicklung weltweit über Internet (siehe */usr/src/linux/CREDITS*)
- weiterhin von Linus koordiniert

Obwohl streng genommen Linux nur der Betriebssystemkern ist, wird üblicherweise die ganze mitgelieferte Softwaredistribution (hauptsächlich GNU-Software) mit Linux bezeichnet. Besser wäre die Bezeichnung

GNU-Linux

für das ganze Betriebssystem.

Folie 17

Linus 1991 in `comp.os.minix`:

> "Do you pine for the nice days of Minix-1.1, when men were men and wrote their own device drivers? Are you without a nice project and just dying to cut your teeth on an OS you can try to modify for your needs? Are you finding it frustrating when everything works on Minix? No more all-nighters to get a nifty program working? Then this post might be just for you." [...]
> "As I mentioned a month ago, I'm working on a free version of a Minix-look-alike for AT-386 computers. It has finally reached the stage where it's even usable (though may not be, depending on what you want), and I am willing to put out the sources for wider distribution. It is just version 0.02... but I've successfully run `bash`, `gcc`, `gnu-make`, `gnu-sed`, `compress`, etc. under it."

Folie 18

Linux Hardware

Ursprünglich für Intel, mittlerweile...

- Intel (80x86)
- AMD
- DEC Alpha (Digital)
- SPARC (Sun)
- Motorola (Macintosh, Amiga, Atari)
- RISC (Silicon Graphics, IBM)
- PalmPilot (3Com)
- Pocket-Computer, Waschmaschinen (Embedded Systems), Echtzeit-Systeme
- ... siehe `http://www.kernel.org`

Folie 19

Einige Linux Links

- `http://www.linux.org` (Linux-Homepage)
- `http://www.kernel.org` (Linux-Kernel)
- `http://linuxtoday.com/`
- `http://www.linuxjournal.com`
- `http://www.linuxgazette.com` (Linux Zeitschrifen)
- `http://slashdot.org` (Linux News)
- `http://www.linux.com` (kommerzielle Linux Seite)
- `http://www.linuxplanet.com/linuxplanet/`
- `http://www.freshmeat.net` (Software)

Folie 20

Linux-Distributionen

- RedHat/Fedora
- Novell/SUSE/openSUSE
- Slackware (Urdistribution)
- Debian (Non-Profit-Organisation)
- Ubuntu
- Mandriva
- Gentoo
- TurboLinux
- Knoppix
- nicht englischsprachige Distributionen
- spezialisierte und Mini-Distributionen
- viele, viele mehr...

Folie 21

große Konfigurationsmöglichkeiten,...

Informationen über verschiedene Distributionen findet man im Internet unter `http://www.linux.org/dist/index.html` oder auf `http://distrowatch.com/` oder im Distribution-HOWTO; hier stellen wir (klarerweise ohne Anspruch auf Vollständigkeit) einige der wichtigsten vor.

- **RedHat/Fedora** (`http://www.redhat.com`): RedHat/Fedora ist eine der am weitesten verbreiteten Linux-Distributionen und wendet sich sowohl an Anfänger als auch an Profis. RedHat hat ein eigenes Softwarepacketmanagement System (RedHat-Package-Manager, rpm) entwickelt. Das rpm-Format ist GPL lizenziert und steht so allen anderen Distributionen zur Verfügung. Fedora ist ein von RedHat gesponsortes Projekt, auf dem die verschiedenen Enterprise-Distributionen, die RedHat anbietet, aufbauen.

- **SUSE/openSUSE** (`http://www.novell.com/linux/suse`): SUSE ist die führende Distribution in Europa, mit sehr großer Programmvielfalt. Derzeit enthält die Distribution mehr als 1500 Programme (auf 7 CDs/1DVD). Hat das rpm-Format schon verwendet, als RedHat es noch nicht GPL lizenziert hatte. SuSE hat die beste Unterstützung für deutsche Tastatur und ISDN/ADSL Anbindung. Die Installation ist wahlweise grafisch oder konsolenbasiert, aber jedenfalls sehr einfach. 2003 wurde SuSE von Novell übernommen. Ähnlich wie bei RedHat ist openSUSE (`http://www.opensuse.org/`) ein von Novell gesponsortes Gemeinschaftsprojekt.

- **Slackware** (`http://www.slackware.com`): Ist die Ur-Distribution von Patrick Volkerding mit einfacher Installation, aber sonst schwer administrierbar, bietet dafür aber (fast) unbegrenzte Konfigurationsmöglichkeiten. Slackware ist für technisch begeisterte und schon erfahrene Linux-Anwender geeignet, die sich nicht scheuen, Programme selbst zu kompilieren. Nur für Intel erhältlich.

- **Debian GNU/Linux** (`http://www.debian.org`): Debian ist eine Non-Profit-Organisation, deren Distribution nur „Open-Source“ Software (bei der die Quellen offen erhältlich sind) enthält. Verwendet ein eigenes Packetmanagement System (atp), das ähnlich einfach zu handhaben ist wie rpm. Debian-Distributionen zeichnen sich durch intensives Pre-Release-Testen aus und sind besonders verläßlich.

- **Mandriva** (`http://www.mandriva.com`): Diese Distribution basiert schon auf einer Distribution: RedHat oder Debian. Der Umfang wurde erweitert und die Konfiguration auf ein Optimum getrimmt. Erhältlich für Intel.

- **TurboLinux** (`http://www.turbolinux.com`): Diese Distribution ist spezialisiert für den High-Performance-Bereich und im Clustering. Marktführer in Asien mit großem Wachstumspotential.

- **Gentoo** (`http://www.gentoo.org`): Gentoo's Paketmanagement (portage) beruht auf den Programmsourcen; diese werden bei der Installation erst kompiliert, was maschinenspezifische Optimierungen erlaubt. Die Installation ist etwas zeitaufwändig, aber durchaus lehrreich (es gibt kein Installationsprogramm; alles wird per Hand ausgeführt).

- **Ubuntu** (`http://www.ubuntulinux.org`): Ubuntu ist eine Distribution, die auf Debian basiert und auf den Desktop-Bereich zielt; Einbindung neuer Programme und gute Sprachunterstützung sind damit Ziele dieser Distribution. Es gibt inzwischen schon ein Projekt, um für Ubuntu gepackte Programmpakete auch für Debian verfügbar zu machen.

- **Knoppix** (`http://www.knoppix.org/`): Knoppix ist eine bootfähige CD mit Hardwareerkennung, die ein relativ vollständiges System zur Verfügung stellt (die Programme auf der CD sind gepackt, und damit passen ungefähr 2 GB auf die CD). Es gibt schon viele spezialisierte „live“-CDs die auf Knoppix basieren.

Neben diesen „vollen“ Distributionen, die ein vollständiges System mit einer Fülle von Anwendungssoftware anbieten, gibt es etliche spezialisierte Entwickler- und Mini-Distributionen für ganz bestimmte Zwecke wie etwa Ein-Floppy-Distributionen für Notfälle aller Art, etc. Beispiele dafür sind: **ChainSaw Linux**, **DemoLinux**, **hal91 Floppy Linux**, **tomsrtbt**, und viele andere mehr.

Viele der großen Linux-Distributionen haben (nicht-englische) Sprachunterstützung, z.B. hat TurboLinux spezielle Japanisch-Unterstützung. **Connectiva Linux** ist eine „mainstream“ Distribution, die in Brasilien entwickelt wird und besonders auf den spanischen und portugisischen Sprachraum zugeschnitten ist.

Es gibt aber auch Distributionen, die auf diverse Sprachunterstützungen spezialisiert sind; z.B. sind **Kheops Linux** und **Eurielec Linux** französische bzw. spanische RedHat-Distributionen, **KSI** „is the very first Linux distribution in post-soviet which “speaks” Russian and Ukrainian fluently“.

Ein Kuriosum am Rande: **Red Flag** ist die offizielle Linux-Distribution der Volksrepublik China mit spezieller Unterstützung für chinesische Schriftzeichen. (Ob Taiwan eine ähnliche Distribution plant, ist nicht bekannt.)

2 Technische Einleitung

Jetzt geht's also richtig los. Hier wird die Philosophie und das Design von Linux/Unix kurz erklärt und die Unterschiede zu (aber auch die Gemeinsamkeiten mit) anderen Betriebssystemen skizziert. Weiters werden Hardwareanforderungen und Softwarefeatures von Linux vorgestellt und der grundlegende Aufbau von Linux/Unix beschrieben.

Im generellen Unix-Design wird *Machbarkeit und Benutzerfreundlichkeit* oft der Vorzug gegenüber Sicherheit gegeben; es ist ein sehr „offenes“ und „freundliches“ Betriebssystem. Diese Philosophie stösst heute, da Datensicherheit vor allem im Netzwerkbereich eine immer größere Rolle spielt, an ihre Grenzen; warum, wird im historischen Kontext klar. Viele Unix-Konzepte wurden schon in der „Urzeit“ des Computers und der Netzwerke entwickelt, als Sicherheitsfragen noch keine große Rolle spielten. Trotz der großen Weiterentwicklung auf dem Sicherheitssektor bleibt diese „Offenheit“ ein Merkmal aller Unix-Varianten.

Ein weiteres Merkmal von Unix-Systemen ist, daß (auf der Benutzerebene) *kleinen, funktionellen Programmen*, (Tools, Utilities) der Vorzug vor großen „monolithischen Programmklötzen“ gegeben wird, was zur Flexibilität und Stabilität (aber auch der Wartbarkeit) des gesamten Systems beiträgt.

Unix ist von Grund auf als volles *Multiusersystem* konzipiert. Das bedeutet, daß jedem Benutzer nicht nur eine eigene Arbeitsumgebung zur Verfügung gestellt wird, die er an seine Bedürfnisse anpassen kann; es bedeutet auch die Vergabe privater Datenbereiche und eine ganze Palette spezifischer Zugriffsrechte auf das System. Nur ein echtes Multiuser-Konzept ermöglicht die Verwaltung einer großen Anzahl von Benutzeraccounts mit geringem Aufwand und die gleichzeitige Verwendung des Systems durch mehrere Benutzer über ein Netzwerk. Zusätzlich können Benutzer zu *Gruppen* zusammengefaßt werden, für die dann gemeinsame Zugriffsrechte auf das System definiert werden können.

Unix ist auch multitaskingfähig, das heisst, dass mehrere Prozesse scheinbar gleichzeitig ausgeführt werden können. Wir werden dieses Konzept noch später im Detail erläutern; unter anderem ist ohne Multitaskingunterstützung eine gleichzeitige Verwendung des Systems durch mehrere Benutzer undenkbar.

2.1 Linux-Design

Um die Designmerkmale von Linux besser zu verstehen, befassen wir uns kurz mit älteren Betriebssystemen. Einfache Betriebssysteme wie etwa CP/M oder MsDOS sind komplett in Assembler geschrieben, wobei als Entwicklungsstrategie ein „Bottom-Up-Konzept“ zum Einsatz kommt. Ausgehend von einer kleinen Sammlung von Systemroutinen werden im Laufe der Zeit weitere Routinen eingebaut, die innerhalb des Betriebssystems keinerlei innere Struktur aufweisen. Jeder Teil kann prinzipiell jeden anderen Teil benutzen, wobei hinsichtlich der Implementierung rasche und maschinenspezifische Ad-hoc-Lösungen vor durchdachten Konzepten rangieren. So sind etwa einige Teile innerhalb von MsDOS nicht reentrant (d.h. können sich nicht direkt oder indirekt rekursiv selbst aufrufen), was speziell die Einbindung von Multitasking-Erweiterungen sehr erschwerte.

Ein weiterer großer Designmangel von MsDOS ist der nicht vorhandene Schutz der Tasks untereinander und des Betriebssystems selbst vor den Anwendungsprogrammen. Geschützte Betriebssysteme wie Unix, Windows-NT oder OS/2 unterscheiden mindestens zwei Betriebsarten, den *privilegierten Modus* (das Betriebssystem selbst) und den *nichtpriveligierten Modus* (alle Anwendungsprogramme). Bei den meisten Betriebssystemen hat sich die Bezeichnung *User-Modus* (nichtpriveligiert) und *Kernel-Modus* (privilegiert) eingebürgert. Dieses Konzept ist aber nur bei entsprechender Unterstützung durch die Hardware möglich, was bei Intels 8086/8088, der Ur-PC-CPU, noch nicht gegeben war; deswegen mußte MsDOS noch ohne diesen Schutz auskommen.

Allerdings können sich auch Betriebssysteme, die dieses Grunddesign aufweisen, erheblich voneinander unterscheiden. Die ersten größeren Betriebssysteme waren diesbezüglich ähnlich wie MsDOS aufgebaut, nämlich *monolithisch*, was nichts anderes bedeutet, als daß es im allgemeinen unmöglich ist, einen Teil des Systems auszuwechseln. So muß bei einer größeren Änderung an der Hardware oft der gesamte Betriebssystemkern neu compiliert

Unix-Philosophie

- Machbarkeit und Benutzerfreundlichkeit vor Sicherheit
- kleine funktionelle, leicht kombinierbare Einheiten statt großer „monolithischer Programmklötze"
- volles Multiuser-Konzept
- volles Multitasking-System

Folie 22

Multiuser-Konzept

- benutzerspezifische Vergabe von Rechten
- privater Datenbereich (Homedirectory)
- private Arbeitsumgebung (Desktop)
- Benutzerverwaltung
- gleichzeitiges Arbeiten mehrerer Benutzer (Netzwerk, `at, cron`)

Folie 23

und/oder gelinkt werden. Dazu benötigt man sowohl den Quelltext des Betriebssystems, als auch ein Entwicklungssystem (Compiler, Linker), das im kommerziellen PC-Bereich üblicherweise nicht im Lieferumfang eines Betriebssystems enthalten sind.

Zu Beginn war auch Linux monolithisch aufgebaut, was Linus Torvalds als den wahrscheinlich größten Fehler in der Entwicklung des Betriebssystems bezeichnete. Mittlerweile sind nur die wichtigsten Gerätetreiber fest an den Betriebssystemkern gebunden, andere werden *modular* eingebunden, sodaß sich die Problematik wesentlich entschärft hat.

Bei der Trennung zwischen privilegiertem und normalem Modus geht man davon aus, daß der privilegierte Teil (mit erhöhten Befugnissen) fehlerfrei ist und daß Übergriffe nur durch die Anwendungsprogramme zu erwarten sind; praktischerweise wird nur ein möglichst kleiner Teil des Betriebssystems mit vollen Privilegien ausgestattet. Intel CPU's ab dem 80386 unterstützen bis zu vier unterschiedliche Privilegienstufen, was es möglicht macht weitere, verfeinerte Schichtungen im Betriebsystem einzuführen. Linux verwendet aber nur die Unterteilung in User- und Kernel-Modus.

Das dominierende Paradigma der Software-Erstellung—die *objektorientierte Programmierung*—findet auch im Betriebssystemdesign große Anwendung. Man verwendet *Client-Server-Konzepte*, bei dem Nutzer- und Anbieterobjekte (Clients und Server) miteinander durch Nachrichten kommunizieren. Eine Hauptaufgabe des Betriebssystemkerns der im privilegiertem Modus läuft, ist dabei die Verteilung der Nachrichten; die Clients und Server dagegen werden im nichtprivilegiertem Modus betrieben. Ein prominentes Beispiel für eine objektorientierte Client-Server-Architektur bei Betriebssystemen ist der Mach-Kernel in einigen Unix-Varianten und dem MacOS und die grafische Oberfläche X-Window unter Unix. Große *Ausführungssicherheit* wird dadurch gewonnen, daß ein Server als Teil des Betriebssystems ausfallen kann, ohne daß zwangsweise das gesamte System abstürzt (Server arbeiten nicht im privilegierten Modus). Im Fehlerfall (etwa Zugriff auf unzulässige Speicherbereiche) wird der Server durch das Sicherheitssystem im Betriebssystemkern abgebrochen. Prinzipiell kann der Server erneut gestartet werden.

Ein weiterer Aspekt ist die Überwachung sicherheitsrelevanter Operationen. Durch *Protokollierung* ausgewählter Nachrichten, die durch den Betriebssystemkern verteilt werden, kann eine fast beliebig genaue Überwachung stattfinden, ohne daß dazu das Design geändert werden muß.

Weiters bietet eine Client-Server-Architektur die Möglichkeit, die Objekte an verschiedenen Stellen in einem Netzwerk zu betreiben. So kann etwa ein Kommunikationsserver auf einem Computer mit Verbindung zu externen Netzen seine Dienste allen erreichbaren Teilnehmern anbieten, wobei sich an der Bedienung und der Organisation zwischen lokalen und nichtlokalen Services nichts ändert (vgl. Kapitel 7).

2.2 Multitasking

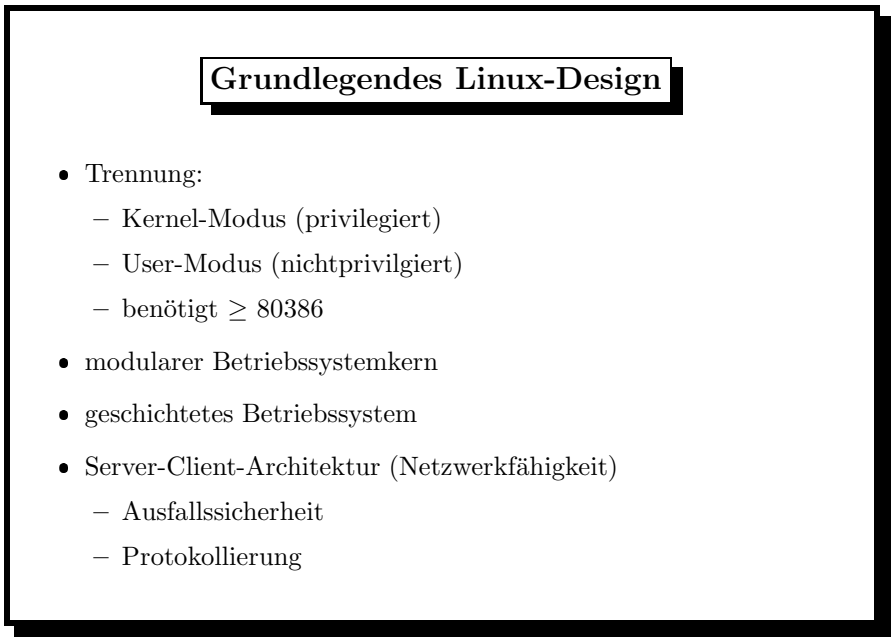
Moderne Betriebssysteme erlauben es, mehrere *Prozesse scheinbar gleichzeitig* auszuführen. Ein Prozeß ist dabei die Instanz eines Programms mit seiner eigenen *Prozeßumgebung* (Datenbereich, Prozeßberechtigungen, Umgebungsvariablen, etc.). Es ist wichtig hier zwischen Programm und Prozeß zu unterscheiden. Wird ein Programm mehrfach gestartet, erhält jeder Prozeß seine eigene Prozeßumgebung. Die Prozeßumgebungen werden durch den Betriebssystemkern verwaltet und sind sauber getrennt. Im Fall von Linux werden die Prozesse vom Betriebssystem durch eine eindeutige Zahl, die PID (Process Identifier), identifiziert.

Der Eindruck, daß mehrere Prozesse gleichzeitig auf einem System mit nur einem Prozessor ablaufen kommt dadurch zustande, daß mehrmals innerhalb einer Sekunde ein Wechsel zwischen den Prozessen stattfindet. Ein solcher Wechsel wird *Kontextwechsel* genannt und verläuft in mehreren Schritten.

1. Der aktuell laufende Prozeß wird vom Betriebssystem angehalten. Sein Kontext wird für die spätere Fortsetzung an gleicher Stelle gesichert.

2. Der nächste Prozeß, der ausgeführt werden soll, wird ausgesucht.

3. Der Kontext dieses Prozesses wird wiederhergestellt, und der Prozeß setzt seine Ausführung fort. Aus der Sicht dieses Prozesses hat sich gegenüber dem Zustand vor



Folie 24

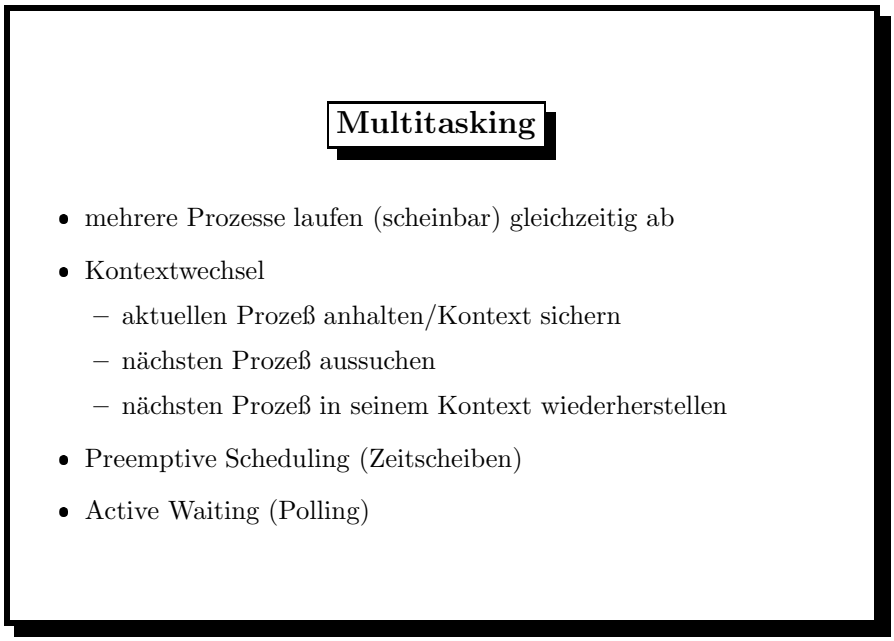
dem Kontextwechsel nichts ereignet (natürlich ausser Änderungen, die ausserhalb seiner Kontrolle liegen).

Gründe dafür, einen laufenden Prozeß zu unterbrechen sind:

- Der dem Prozeß zustehende Anteil an verfügbarer CPU-Zeit (Zeitscheibe) ist abgelaufen. Das Betriebssystem führt einen Kontextwechsel durch und zwar ohne daß der Prozeß selbst etwas tun muß oder sich dagegen wehren könnte. Dieses Verfahren wird *präemptive Ablaufkontrolle* (preemptive scheduling) genannt und stellt damit sicher, daß kein Prozeß auf Kosten aller anderen Prozesse die alleinigen Ausführungsrechte okkupieren könnte.
- Ein Prozeß möchte auf eine Ressource zugreifen, die (derzeit) nicht vorhanden ist oder etwas Zeit zur Bereitstellung erfordert. Ein prominentes Beispiel dafür sind alle Arten von Ein-/Ausgabeanweisungen.
- Der Prozeß wird durch eine Intervention angehalten (dies kann auf verschiedene Weisen durch einen anderen anderen Prozeß oder auch durch einen Benutzer mit den entsprechenden Berechtigungen geschehen).

Ein typisches Beispiel für den zweiten Punkt ist das folgende: Ein Prozeß will einen Block einer Datei von einer Festplatte lesen. Das Betriebssystem muss (über Treiber und Plattencontroller) die Daten von der Festplatte einlesen und der Prozeß ist gezwungen, auf die Daten zu warten. Tatsächlich macht MsDOS (und auch DOS/Windows) dies so, was nichts anderes bedeutet, als daß der Prozeß in dieser Zeit Däumchen dreht. Etwas technischer wird dies auch *aktives Warten* (Active Waiting) oder *Polling* genannt. Bei Unix (und allen anderen Multitasking Betriebssystemen) wird dem aktiven Warten durch einen Kontextwechsel entgangen; auch versucht man auf anderer Ebene diesem gezwungenen Stillstand durch eine Datenanforderung zu entgehen, indem zB die Daten strategisch günstig von der Festplatte gelesen werden.

Threads: Die Aufteilung von Systemaufgaben in einzelne Prozesse ist also ein Mittel, Ressourcen effektiv (ohne aktives Warten) zu organisieren. Dieselbe Strategie, auf das Niveau der Prozesse selbst angewandt, führt zur Idee der Threads. Ein Thread verhält sich von der Organisation der Rechenzeit her genauso, wie wir bislang einen Prozeß beschrieben haben. Tatsächlich besteht auch jeder Prozeß aus mindestens einem Thread. Threads können aber



Folie 25

auch nur Teile eines Prozesses sein. Sie arbeiten im Kontext des Prozesses (Speicher, etc.) und dienen dazu, die Aufgaben des Prozesses in kleinere funktionelle Einheiten aufzuteilen, die dann in einer für die Gesamtperformance des Systems günstigen Weise abgewickelt werden können.

2.3 Aufbau eines Unix/Linux-Systems

Wie wir gesehen haben, besteht jedes Betriebssystem aus mehreren Komponenten, die im Folgenden am Beispiel eines Unix-Systems mit besonderer Berücksichtigung von Linux vorgestellt werden.

Grob gesehen besteht ein Unix-Betriebssystem aus einem Betriebssystemkern, dem sog. *Kernel*, dem *Filesystem*, *Systemprogrammen* und eventuell weiteren *Applikationen*.

Der Kernel ist das Herz des Betriebssystems. Er verwaltet die Dateien, die im *Filesystem* gespeichert sind, ist für das *Scheduling* und die *Systemressourcenvergabe* an die Prozesse zuständig (insbesondere die *Speicherverwaltung*) und bewerkstelligt die *Kommunikation mit der Hardware*.

Der Kernel führt nur wenige Funktionen selbst aus, bietet aber die notwendigen Werkzeuge an, mit deren Hilfe alle Dienste erzeugt werden können. Auch verhindert der Kernel das direkte Ansprechen der Hardware und zwingt die Prozesse, die bereitgestellten Werkzeuge zu verwenden. Damit bietet der Kernel den Prozessen und Benutzern Schutz vor einander und stellt somit die grundlegende Systemsicherheit her. Die bereitgestellten Werkzeuge des Kernels können mittels *„Systemaufrufen“* (System Calls) verwendet werden.

Wichtige Aufgaben des Kernels sind Prozeßmanagement und Speichermanagement. Letzteres kümmert sich um die Vergabe von Speicherbereichen und Swapbereichen (Auslagerunsgspeicher auf der Festplatte; virtueller Speicher in der MS-Welt) an Prozesse, den Kernel selbst und an den Buffer. Das Prozeßmanagement erzeugt Prozesse und implementiert das Multitasking.

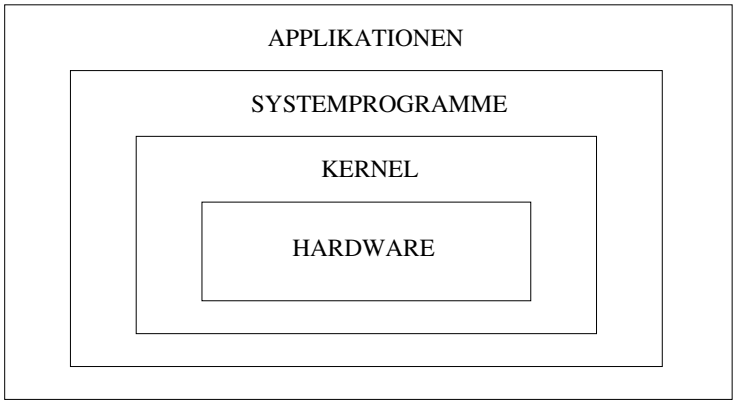
In der untersten Schicht enthält der Kernel einen *Treiber* für jede Hardwarekomponente, die er unterstützt. Da die Welt voll von verschiedener Hardware ist, ist die Anzahl der einzelnen Treiber riesig. Allerdings gibt es für spezifische Hardwaregeräte standardisierte Funktionen, die sich nur in der untersten Schicht der Implementation unterscheiden. Zum Beispiel haben alle Festplattentreiber Befehle, wie „initialisiere Festplatte“, „lese Sektor N“ und „schreibe Sektor N“. Diese Ähnlichkeiten machen es möglich, die Treiber in Klassen

Hauptkomponenten eines Unix-Betriebssystems

- Betriebssystemkern (**Kernel**)
- Filesystem
- Systemprogramme
  stellen einfache Systemfunktionen zur Verfügung
- Applikationen
  Programme für produktive Aufgaben

Folie 26

Aufbau eines Unix-Systems



Folie 27

zusammenzufassen, die alle dieselben Befehle unterstützen. Das trifft auch auf die vom Kernel zur Verfügung gestellten Dienste zu, die dann zu abstrakten Klassen zusammengefaßt werden. So erhielten die verschiedenen Netzwerkprotokolle eine Programmierschnittstelle und über die Dateisysteme wurde das VFS (Virtual Filesystem) gesetzt, sodaß sich die Befehle unabhängig von deren Implementation benutzen lassen. Wenn ein Prozeß auf ein Dateisystem zugreifen will, so geht der Befehl an das VFS, das ihn an den entsprechenden Dateisystemtreiber weiterleitet.

Das *Filesystem* ist eine hierarchische, baumartige Struktur, die die Dateien und Verzeichnisse (auf den Festplatten) des Systems repräsentiert. Unter Unix gibt es einen gewissen Pseudostandard für die Organisation des Filesystems mit allerdings erheblichen (wenn auch oft intuitiv zu erratenden) Abweichungen. Anders als in der Windows-Welt laufen alle Directories in einem einzigen Mutterverzeichnis, genannt *Rootverzeichnis,* / zusammen. Trennzeichen für Pfade sind „forward Slashes“, / (statt Backslashes, \). Folie 29 zeigt die wichtigsten Verzeichnisse in einem Unix-System; die Details folgen in Abschnitt 3.7.

Aufgaben des Kernel

- Prozeßverwaltung
- Verwaltung der Systemressourcen
  - Speichermanagement
- Kommunikation mit der Hardware
  - Schnittstellen (z.B. VFS)
  - HW-Treiber
- Verwaltung des Filesystems
- Grundlegende Systemsicherheit

Folie 28

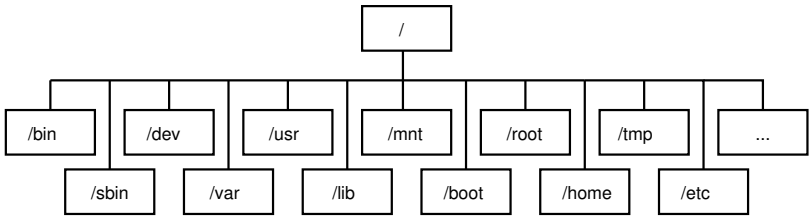
Die *Systemprogramme* benutzen die vom Kernel bereitgestellten Werkzeuge, um die verschiedenen Dienste des Betriebssystems zu implementieren, wie etwa die grundlegende Netzwerkkommunikation, die grafische Benutzeroberfläche etc. Die Systemprogramme und alle anderen Programme laufen „über” dem Kernel im Usermodus.

Als *Applikationen* bezeichnet man Programme, die zur Ausführung produktiver Aufgaben gedacht sind (oder fürs Spielen, wenn die Applikation nun mal ein Spiel ist), im Gegensatz zu den Systemprogrammen, die grundlegende Systemfunktionen zur Verfügung stellen. Eine Textverarbeitung ist eine Applikation, Telnet ein Systemprogramm. Die Unterscheidung ist allerdings nicht in allen Fällen klar und im Detail nicht immer zielführend.

Das wichtigste Unix-Systemprogramm ist die *Shell*. Sie ist das Benutzerinterface des Systems, also das Programm, mit dem der User Befehle an das System (den Kernel) eingibt.

Filesystem

- hierarchische, baumartige Struktur
- repräsentiert Dateien und Verzeichnisse auf der Festplatte



Folie 29

Linux-Systemprogramme und Applikationen

- Shell (Kommandointerpreter)
- Grafische Benutzeroberfläche (GUI)
  X-Window
- Textverarbeitung
- Programmiersprachen
- Kommunikation/Netzwerk
- Systemmanagement
- Online Dokumentation

Folie 30

Die Shell wird deswegen auch als *Kommandointerpreter* bezeichnet.

Unter Unix wie auch unter Linux gibt es eine Vielzahl verschiedener Shells mit zum Teil unterschiedlicher Syntax. Manche sind ein reines Userinterface, andere bieten auch eine (einfache) Scriptsprache an, können Variablen verwalten und bieten Flußkontrolle (if-then-else-Konstruktionen, Schleifen).

Die Standard-Shell unter Unix ist die *Bourne Shell* (`sh`), unter Linux die GNU-Variante davon, die *Bourne Again Shell* (`bash`). Sie bietet ein komfortables Interface (inklusive *Command-Completion*) und eine umfangreiche Programmiersprache, deren Beherrschung (zumindest) für den Systemadministrator unerläßlich ist. Weitere beliebte Shells sind die `tcsh`, die ebenfalls einen sehr komfortablen Interpreter mit einer anderen, von der *C-Shell* `csh` abgeleiteten Shell-Sprache besitzt, die *Kornshell* (`ksh`) und die *Z-Shell* (`zsh`).

Die grafische Benutzeroberfläche (Graphical User Interface, GUI) in der Unix-Welt heißt X-Window (ohne „s"[!!]) und ist in einer der Versionen *XFree86* oder *Xorg* im Umfang aller Linux-Distributionen enthalten; mehrere Window-Manager bzw. Desktopsysteme (die den Look and Feel des GUI bestimmen) stehen zur Verfügung: *fvwm2, KDE, GNOME,....* Wir besprechen das X-Window-System im Detail in Kapitel 7.

Zur *Textverarbeitung* stehen unter Linux die Standard-Unix-Editoren wie etwa *ed* (zeilenorientiert), *vi, vim*, *axe*, *pico* (Pine-Look and Feel), *joe*, *nedit*, *(x)emacs* (GNU, mächtiger Editor für alle Zwecke) etc. zur Verfügung. Für Systemadministratoren ist vi trotz seiner etwas gewöhnungsbedürftigen Bedienung ein mächtiges, unverzichtbares Werkzeug, unter anderem deswegen, weil er auf (fast) allen Unix-Systemen auch im Notfall verfügbar ist; wir geben in Kapitel 5 eine kleine vi-Einführung.

Weiters steht unter Linux das professionelle Textverarbeitungssystem (eigentlich Typensatzsystem) *TEX* mit seiner leicht zu handhabenden Erweiterung LaTeX (`http://www.latex-project.org`) zur Verfügung. Seine Funktionsweise ähnelt der einer Programmiersprache: ein ASCII-Quelltext wird vom Interpreter in den Textsatz verwandelt (dvi oder pdf-format; die Ausgabe ist nicht auf diese Formate beschränkt, so ist unter anderem auch Ausgabe in HTML möglich). TeX eignet sich vor allem zur Darstellung komplizierter mathematischer Formeln und ist daher im naturwissenschaftlichen Bereich weit verbreitet. Dieses Skriptum ist z.B. in LaTeX geschrieben; LaTeXist vor allem in der Mathematik zum Standardprogramm für den Typensatz geworden. Für LaTeX sind auch einige WYSIWYG (What you see is what you get) Oberflächen erhältlich (lyx).

WYSIWYG Editoren bzw. Textprozessoren sind *Oowriter*, *Kword* und *AbiWord* die Bestandteile der Office Pakete *Open Office* (kommt vom früher durch SUN hergestellten StarOffice), bzw. *K-Office* (KDE) und *Gnome-Office* (Gnome) sind. Open Office umfaßt äquivalente Programme zu allen Komponenten von MS-Office, Koffice ist noch jünger und etwas weniger weit entwickelt.

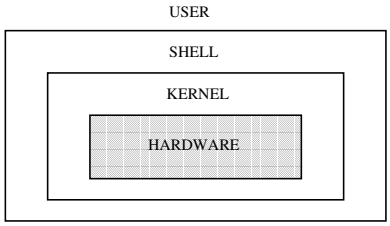
Linux besitzt hervorragende *Entwicklereigenschaften.* Ein C/C++ Compiler *gcc* (GNU-C-Compiler) mitsamt Bibliotheken (*glibc*) ist Standard und zum Compilieren wichtiger Systemprogramme unerläßlich. Weiters sind unter Linux alle gängigen Programmiersprachen verfügbar (Fortran (f77 and f90/95), Java, Pascal, Perl, Python, TCL, Lisp, ...); es gibt inzwischen auch schon freie Implementierungen des .net-Frameworks (mono).

Die *Netzwerkunterstützung* ist generell bei Unix-Systemen sehr gut und bereits in sehr tiefen Schichten im Kernel implementiert; die TCP/IP-Protokollfamilie (auf der praktisch der gesamte Datenverkehr sowohl im LAN (Local Area Network) wie auch im Internet beruht) wurde historisch (Mitte der (80er Jahre) erstmals in einem Unix-Kernel implementiert. Alle gängigen Internet-Kommunikationstools, wie etwa Telnet, Ftp, Ssh, Talk etc. sind praktisch in allen Linux-Distributionen im Lieferumfang enthalten; Unterstützung für eine Vielzahl von Instant-Messaging-Protokollen wird auch geboten (zB durch die Programme gaim und Kopete, die zum Gnome- bzw. KDE-Desktop gehören. Der Standard-Webbrowser in der Unix-Welt ist *Netscape*, bzw. sein Open-Source-Pendant *Mozilla (Firefox).* Einige der grafischen Oberflächen bieten aber auch ihren eigenen File/Webbrowser (z.B. *Konqueror* von KDE oder Galeon von Gnome).

Der Standard-Mailclient an der Konsole ist *pine* von der Washington University, *mail* und *mutt* sind ebenfalls weit verbreitet. Auch steht eine Vielzahl graphischer Mailclients zur Verfügung, KDE stellt *Kmail* zur Verfügung, in Gnome ist das Outlook-Pendant *Evoluti-*

Die Shell

- Benutzerinterface zum System
- Kommandointerpreter und Programmierfähigkeiten



- übliche Unix-Shells
  - bash (Linux default)
  - csh, tcsh
  - ksh
  - zsh, ...

Folie 31

Grafische Oberfläche

- X-Window
  entwickelt am MIT
  (TCP/IP Server-Client-Architektur)
- X-Server: XFree86, Xorg (HW-Treiber)
- Window-Manager Desktop (Look and Feel)
  - twm
  - fvwm2, fvwm95
  - KDE
  - Gnome
  - Windowmaker
  - Enlightenment
- Motif
- Siehe auch `http://xwinman.org/`

Folie 32

Textverarbeitung/Office

- Unix-Standardeditoren
  - ed
  - vi (vim = Vi Improved)
  - pico
  - joe, nedit
  - (x)emacs (mächtiger GNU-Editor)
- TEX, LATEX, (naturwiss. Bereich)
- Spellchecking: ispell
- Postscript: Ghostview, Ghostscript
- Pdf: Acrobat Reader
- Office
  - Open Office
  - K-Office (KDE)
  - Gnome-Office

Folie 33

on integriert, beliebte Stand-alone-MCs sind zB *Sylpheed* oder *Thunderbird*. Weiters sind alle in der Unix-Welt verbreiteten Netzwerkdienste wie *Nfs, Nis, Lpd,...* sowie eine *SMB*-Protokollimplementation (Standard-MsWin Netzwerkprotokoll) namens *Samba* unter Linux verfügbar. Schließlich ist der *Apache Webserver* (Marktführer) fixer Bestandteil aller Linux-Distributionen.

Alle Linux-Distributionen stellen Programme zur *Systemadministration* und -*konfiguration* zur Verfügung. Neben kleinen flexiblen Tools zur Erfüllung einer bestimmten Aufgabe gibt es auch große Programmpakete, die die Verwaltung des Gesamtsystems ermöglichen (*Linuxconf*, *Yast* (SuSE)).

Linux als Programmierumgebung

- Shellprogrammierung
  - bash, sh
  - csh, tcsh, ksh
- Programmiersprachen
  - C/C++
  - Fortran (77, 90)
  - Java
  - Pascal
  - Lisp
  - Perl
  - tcl/tk
  - Python
  - PHP
  - Basic...

Folie 34

2.4 Hardwareanforderungen, Dualboot

Zum Schluß dieses Kapitels wenden wir uns kurz den Hardwareanforderungen von Linux- und Dualboot-Systemen (zwei oder mehrere Betriebssysteme auf einem Computer) zu.

In der ersten Version des Skriptums stand hier:

> Linux stellt generell sehr niedrige Hardwareanforderungen. Prinzipiell läuft es bereits auf einem 80368-Prozessor mit 4MB RAM und ca. 100MB Speicherplatz auf der Festplatte. Die untere Grenze für eine durchschnittliche Workstation mit „akzeptabler Performance“ ist etwa ein Pentium II mit 64MB RAM. Für Server (Web-Server, File-Server, Firewall/Gateway) können aber auch noch Pentium Rechner herangezogen werden.

Heute wird man die Eckdaten für eine Workstation mit akzeptabler Performance doch etwas höher ansetzen. Insbesondere die inzwischen sehr umfangreichen Desktopumgebungen verbrauchen Systemressourcen sehr rasch.

Linux Networking

- TCP/IP-Implementation im Kernel
- Basiskommunikationstools
  - Ping
  - Telnet, Ssh
  - Ftp, Ncftp, Sftp, Scp
  - Browser: Firefox, Konqueror, Galeon, ...
  - Mailclients: Pine, Thunderbird, Kmail, Evolution, ...
- Netzwerkdienste
  - Nfs (Network File System)
  - Nis (Network Information System, yp, Sun)
  - Webserver: Apache
  - SMB: Samba

Folie 35

Systemmanagement

- Software Packetmanagement (rpm, apt)
- Benutzerverwaltung
- Konfiguration von Peripheriegeräten
- Backup
- Performancekontrolle
- Tools:
  - vim
  - Yast (SuSe)
  - Linuxconf
  - Webmin

Folie 36

Probleme hat Linux mitunter bei exotischer oder brandneuer Hardware, da nicht alle Hersteller alle relevanten Spezifikationen ihrer Produkte bekanntgeben, was das Programmieren der entsprechenden Treiber erschwert. Vor dem Kauf einer solchen Hardwarekomponente empfiehlt sich jedenfalls ein Blick auf die Hardware Compatibility Liste der entsprechenden Distribution.

Linux kann gemeinsam *mit anderen Betriebssystemen* (MsWin, OS/2,...) auf einem Computer betrieben werden. Bei diesen sogenannten Dualboot-Systemen kann man das Betriebssystem beim Booten auswählen. Weiters weist Linux gute Kompatibilitätseigenschaften zur MsDOS/Windows-Welt auf. Mit einer Ausnahme (NTFS, die Unterstützung für Schreiboperationen ist nach wie vor experimentell) kann Linux alle in der Windows-Welt verbreiteten Filesysteme lesen und schreiben, sodaß im Dualboot-Betrieb Linux auf die entsprechenden Windows-Dateien zugreifen kann. Außerdem kann ein Linux-Rechner mittels des *Samba*-Pakets in Windows-Domänen eingegliedert werden und die typischen Windows-Datei- und Druckerdienste in Anspruch nehmen bzw. anbieten (und sogar als Primary Domain Controller fungieren). Darüberhinaus verfügt Linux über DOS- und Windows-Emulatoren und es ist eine (allerdings Lizenzpflichtige) Linux-Version von `VMware` (emuliert einen virtuellen PC im PC, `http://www.vmware.com`) erhältlich.

Linux und andere OS

- kann neben anderen OS (DOS, Win*, OS/2,...) auf einem PC installiert werden (Dualboot-Systeme).
- FAT16/32 (Win*)-Partitionen lesen und schreiben
- NTFS (WinNT) Partitionen lesen (experimentell auch schreiben)
- HFS (MAC) Partitionen lesen
- kann mittels `Samba` (SMB Protokoll) in NT-Domänen eingegliedert werden; sogar PDC
- verfügt über DOS/Win* Emulatoren `dosemu, wine`
- mittels *Appletalk* als MAC Fileserver
- `VMware` (Lizenzpflichtig)

Folie 37

3 Unix/Linux benutzen 1 (Grundlagen)

Dieses Kapitel bietet eine Einführung in die Benutzung eines Unix/Linux-Systems. Eine gute Beherrschung dieser einfachen Bedienungselemente (die großteils unabhängig vom verwendeten Unix-Derivat gelten) ist Grundlage für das Verständnis aller Systemadministrationstätigkeiten.

Wir beginnen mit einer kurzen allgemeinen Einführung in die Besonderheiten der Benutzung von Multiuser/Multitasking-Systemen geben eine Einführung in den Umgang mit Dateien (erzeugen, kopieren, löschen), erklären die einfachsten Shell-Befehle, zeigen wie man sich im Filesystem orientiert und bewegt und machen eine kleine Tour durch ein typisches Linux-Filesystem.

3.1 Allgemeines

Multiuser/Multitasking-Systeme haben eigene Gesetze. Jeder Benutzer, der das System verwenden will, muß sich *einloggen* (anmelden). Er benötigt dazu ein gültiges *Benutzerkonto* (Account); üblicherweise identifiziert sich der Benutzer am System durch einen *Loginnamen* (Username, Benutzername) und authentifiziert sich durch ein *Paßwort*, die beim Login-Vorgang abgefragt werden. Loginnamen und Anfangspaßwort erhält man vom Systemadministrator. Loginnamen werden üblicherweise klein geschrieben und sind auf den meisten Systemen bis zu acht Zeichen lang. Gute Paßwörter (und ein entsprechend sorgsamer Umgang mit ihnen) sind essentiell für die gesamte Sicherheit des Systems. Tips und Regeln zur Wahl von Paßwörtern und zum Umgang mit Ihnen geben wir ein bißchen später.

Ein Unix-System kennt neben den „normalen“ Benutzeraccounts sogenannte privilegierte Accounts. Der Loginname des Systemadministrators ist *root*; dieser Benutzer darf wirklich ALLES. Ein typisches Unix-System gibt jedem Benutzer eine eindeutige Kennung (UID, User Identifier), die eine ganze Zahl ist. Traditionell hat root die UID 0. Die Kompetenzen von root werden nur durch die physischen Möglichkeiten (auch root kann mit einem DVD-Spieler keine CDs brennen) und die—von Person zu Person verschiedenen—ethischen Grundlagen eingegrenzt: root hat Zugriff auf alle persönlichen Dateien aller Benutzer einschließlich E-Mails, kann diese filtern, durchsuchen und verändern, ohne daß es der Benutzer merkt; root kann sogar die Tastatur „abhören“.

Es gibt für solche Massnahmen normalerweise keinen Grund, ausser die Systemsicherheit macht sie notwendig: Darunter fallen so wichtige Tätigkeiten wie zum Beispiel das Filtern der E-Mail nach Viren, das Untersuchen der missbräuchlichen Benutzung eines Accounts, etc. Benutzer können aber besonders sensible Daten mittels entsprechender Software (z.B. *PGP*) verschlüsseln. Weiters können im System weitere Accounts mit (gewissen Abstufungen von) erweiterten Rechten existieren.

Ein typischer (textbasierter) Loginvorgang läuft wie folgt ab: Nachdem die Verbindung zum Rechner besteht (entweder physisch oder über ein Netzwerk), erwartet den Benutzer ein *Login* genanntes Programm. Unter dem Begrüßungstext, der etwa Auskunft über die verwendete Distribution, den installierten Kernel und die Art des Logins (etwa: „Welcome to SuSE Linux 6.2 (i386) -Kernel 2.2.13 (tty2)“) steht eine schlichte Aufforderung: „hell login:“, wobei in diesem Fall der *Hostname* des Systems „hell“ ist. Nun ist der richtige Zeitpunkt gekommen, um den Benutzernamen einzugeben. Jetzt noch das Passwort eingeben (nachdem „Password:“ erschienen ist, das Eingetippte wird nicht angezeigt) und man gelangt in eine Shell, genannt die *Login-Shell*. Die Eingabeaufforderung (Prompt) kann, je nach Shell und Konfiguration, ein völlig anderes Aussehen haben (meist jedoch „user@hostname aktuelles Verzeichnis>“ also z.B. „karli@hell karli>“).

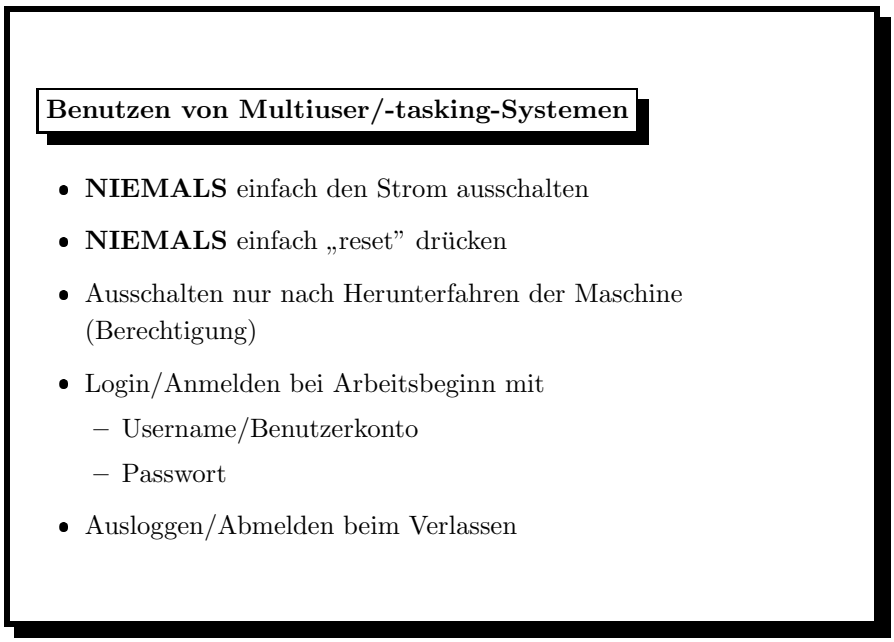
Typischerweise stellen Unix-Systeme neben dem hier dargestellten textbasierten Login auch eine grafische Benutzeranmeldung zur Verfügung. Das Loginprogramm läuft in einem Fenster auf einer grafischen Oberfläche ab; neben der Tastatur kann auch die Maus zur Bedienung herangezogen werden. Nach dem Login auf der grafischen Oberfläche gelangt der User statt in eine Shell in seinen Desktop. Beim grafischen Login besteht meist die Möglichkeit, verschiedene Desktopvarianten mittels eines Menüs auszuwählen. Oft besteht auch die Möglichkeit, das System aus dem grafischen Login herunterzufahren; auf der Textkonsole

wird dies durch die Tastenkombination `<ctrl><alt><del>` bewerkstelligt. Dieses Feature kann aber je nach Aufgabenbereich des Systems vom Administrator abgestellt werden.

Linux-Systeme verfügen in der Regel gleichzeitig über mehrere (oft 6) textbasierte Terminals und eine grafische Oberfläche. Man kann zwischen den textbasierten Konsolen mittels `<alt><Fn>` ($n = 1, 2, \ldots, 6$) wechseln. In den grafischen Modus gelangt man mittels `<alt><f7>`; von dort zurück auf die entsprechende Textkonsole mittels `<ctrl><alt><fn>`.

Nach dem Arbeiten muß man sich vom System *abmelden* (ausloggen). Dies erfolgt mittels einem der Befehle `exit`, `logout` oder durch die Tastenkombination `<ctrl>d`, beziehungsweise im grafischen Modus über ein Menü. Vergißt ein Benutzer, sich nach Arbeitsende auszuloggen, so hat jede Person, die physischen Zugriff auf das System erlangt, auch uneingeschränkten Zugang zu den Daten des Benutzers und überhaupt Zugang zum System im Umfang der Rechte des Benutzers. Sinngemäß gilt dasselbe für das Verlassen des Arbeitsplatzes während Pausen. Auf der grafischen Oberfläche besteht die Möglichkeit den Desktop mittels Menü zu sperren, ohne auszuloggen. Beim Wiedereinstieg gelangt man nach Eingabe des Paßworts wieder in die alte Session.

Ein Benutzer hat im Allgemeinen keinen Einblick oder Einfluß auf die tatsächliche Arbeit des Systems. So kann es z.B. vorkommen, daß die Ausführung der gleichen Tätigkeiten zu unterschiedlichen Zeitpunkten verschieden lange dauert. Diese Tatsache wird vor dem Hintergrund, daß mehrere Benutzer zugleich über das Netzwerk auf dem Rechner arbeiten können, verständlich. Es können dem Betriebssystem aber auch Aufgaben zur späteren Bearbeitung übertragen werden. Auf jeden Fall sollte man die folgenden grundlegenden Regeln befolgen:

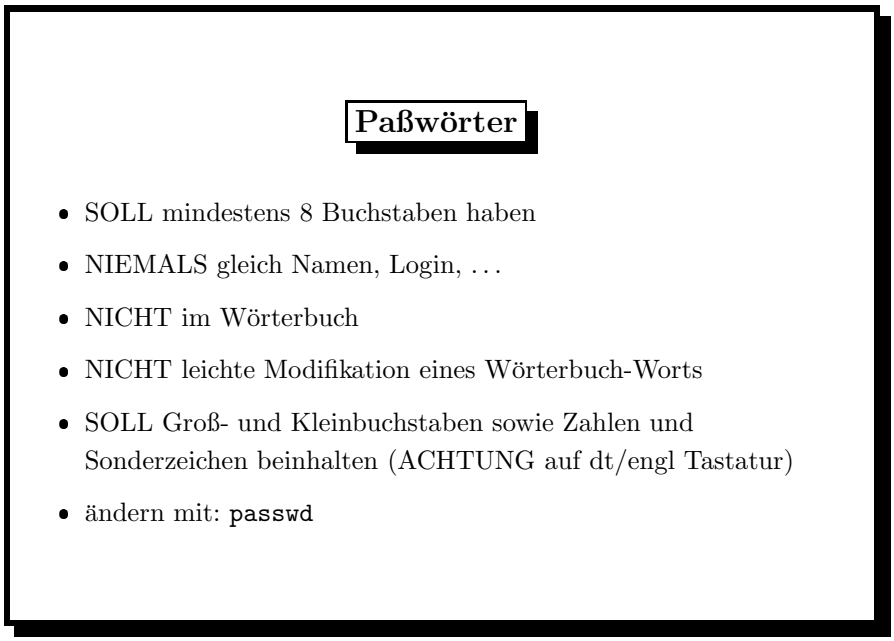


Folie 38

- NIEMALS den Computer „einfach so" ausschalten oder gar Reset drücken! Es könnten auch andere Benutzer, die nicht physisch am Computer anwesend sind, das System benutzen. Daß man sich in diesem Fall den geballten Zorn dieser Leute zuzieht, ist klar. Weiters laufen eine Menge Hintergrundprozesse, die auf die Hardware zugreifen können; zum Beispiel werden Daten nicht sofort auf die Festpaltte geschrieben (Multitasking). Es kann durch nicht ordnungsgemässes Herunterfahren durchaus passieren, dass das Dateisystem inkonsistent ist; im schlimmsten Fall ist es unwiderruflich zerstört. (Und wieder ist einem der Zorn der Mitbenutzer sicher.)
- Wenn man sich vom Arbeitsplatz entfernt, sollte man das System vor unbefugtem Zugriff schützen (siehe oben). Man sollte aber auch nicht durch exzessives Benutzen der

Sperrfunktion des Desktops den Zugang zum System unnötig blockieren. Weiters sollte man im Auge behalten, dass Benutzer mit entsprechenden Rechten (root) ein Herunterfahren des Systems erzwingen können (und manchmal müssen, um notwendige Wartungsarbeiten durchzuführen). Dieses Herunterfahren beendet dann natürlich die noch laufenden Prozesse aller Benutzer und kann so zum Verlust von Benutzerdaten führen. Gute Systemadministratoren warnen zwar rechtzeitig vor solchen Massnahmen, es soll aber auch Benutzer geben, die auf diese Warnungen nicht zeitgerecht reagieren.

- Nicht zuletzt gibt es auf vielen Multiuser-Systemen Benutzungsordnungen, die ein möglichst reibungslosens Miteinander der Benutzer ermöglichen soll. Ist eine solche nicht vorhanden, sollte man zumindest versuchen, die offensichtlichen und oben angesprochenen Regeln zu beachten.



Folie 39

Gute Paßwörter sind essentiell für die Sicherheit des gesamten Systems; wohlgemerkt nicht nur für die Sicherheit der eigenen Daten. (Hat ein Hacker einmal Zugang zu einem Benutzeraccount, so fällt es ihm *wesentlich* leichter root-Zugang zum System zu erhalten...). Daher sollten bei der Wahl von Paßwörtern einige grundsätzliche Regeln befolgt werden, die auch auf Folie 39 zusammengefaßt sind. Das Passwort kann zwischen 5 und 8 Zeichen lang sein (Standard-Unix-Verschlüsselungsalgorithmus) oder bis zu 255 Zeichen (MD5-Verschlüsselung), man sollte aber (als Schutz gegen Wörterbuchattacken, siehe unten) immer mindestens 8 Zeichen verwenden. Es sollte unter keinen Umständen aus leicht erratbaren persönlichen Daten bestehen; dazu zählen der Benutzername, Namen von Verwandten, Haustieren, des Freundes/der Freundin oder Zahlenkombinationen wie die eigene Telefonnummer, Matrikelnummer oder das eigene Geburtsdatum sein.

Wörter, die einen Sinn ergeben (also im Wörterbuch stehen), sind ebenfalls zu vermeiden, da der Verschlüsselungsalgorithmus bekannt ist (Open Source) und so Wörterbuch-Attacken leicht möglich sind. (Dabei werden alle Wörter in einem Wörterbuch verschlüsselt und mit dem verschlüsselten Paßwort verglichen.) Auch Ähnlichkeiten mit bekannten Wörtern (zum Beispiel durch Voranstellen einer Zahl) sind nicht empfehlenswert. Andererseits sollte das Paßwort leicht tippbar und einfach zu merken sein, da es keine Möglichkeit gibt, es wieder herauszufinden, sollte man es vergessen haben. Fazit: Vergisst man sein Paßwort, bleibt einem der Canossagang zum Systemadministrator nicht erspart; der kann zwar das alte Paßwort auch nicht herausfinden, aber er kann ohne Kenntnis des vergessenen Paßworts ein

neues setzen. Auch sollte das Paßwort keine Sonderzeichen enthalten, die es auf englischen Tastaturen nicht gibt, da der aktuelle Zeichensatz nicht immer funktionieren muß. Weiters sollte man Paßwörter *nicht aufschreiben*, und niemals an andere weitersagen. Eine gute Methode der Paßwortwahl ist daher die folgende: Man denkt sich einen Satz aus und nimmt Anfangsbuchstaben der einzelnen Wörter in Groß- und Kleinschreibung und ersetzt eventuell noch einzelne Buchstaben durch ähnlich lautende Sonderzeichen (siehe oben) oder Zahlen.

Außerdem sollte man Paßwörter in unregelmäßigen Abständen (Größenordnung ca. drei Monate) ändern (und zwar *nicht* durch hinzufügen von 1,2,3 etc.). Viele Systeme sind so konfiguriert, daß eine Paßwortänderung nach einer gewissen Zeit erzwungen wird. Der entsprechende Befehl lautet `passwd`. Sicherheitshalber wird das aktuelle Paßwort abgefragt (es könnte ja jeder kommen,...) und das neue gleich zweimal eingegeben, um Tippfehlern vorzubeugen (da es *nicht* angezeigt wird). Positiv zusammengefaßt: Das Paßwort sollte mindestens 8 Zeichen lang sein, Groß-/Kleinbuchstaben, Zahlen und einfache Sonderzeichen (Satzzeichen, Bindestrich) enthalten und trotzdem einfach zu merken sein.

3.2 Shell-Grundlagen

In diesem Abschnitt erklären wir die Grundlagen des Umgangs mit der Shell. Da die Shell *das* Interface des Benutzers zum System schlechthin ist, ist die Kenntnis der Shell schon fast die Kenntnis vom Umgang mit Unix überhaupt. Generell ist die Philosophie von Unix (vgl. Abschnitt 3) viele einfache Shell-Befehle (Tools) zur Verfügung zu stellen, die durch geschicktes Kombinieren zur Bewältigung komplexer Aufgaben herangezogen werden können. Die folgenden Befehle und ihre Syntax funktioniert in allen Unix-Shells.

Die Shell ermöglicht das grundlegende interaktive Benutzen des Systems; sie gibt einen *Prompt* aus (z.B. [roli@pablo tmp]$) und wartet auf die Eingaben des Benutzers. Diese werden mittels <ENTER> an die Shell übergeben, die sie daraufhin bearbeitet und sich dann wieder mit dem Prompt zurückmeldet. Zunächst sei gesagt, daß in der Unix-Shell (im Unterschied zu MsDOS bzw. einem DOS-Fenster in Win*) zwischen Groß- und Kleinschreibung unterschieden wird; die Unix-Shell ist *case sensitive*. Das grundlegende Kommandoformat in Unix ist:

```
Kommando Optionen Argumente
```

Im weiteren werden Kommandos mit ihren Optionen und Argumenten im Text durch Verwenden von `typewriter`-Fonts hervorgehoben. Ein einfaches Beispiel eines Kommandos, das den Inhalt des Verzeichnisses `/home/fred` im „langen Format“ ausgibt ist:

```
ls -l /home/fred
```

Der Name des Befehls (das Kommando; hier `ls`, was für List steht; analog zum DOS Kommando `DIR`) kommt also zuerst. Dann wird die Option mit einem vorangestellten „-“ angegeben. Eine *Option* verändert die Funktionsweise des Kommandos (hier: liste die Dateien und gib weitere Informationen, wie Größe, Datum der letzten Änderung, etc. an).

Sollen mehrere Optionen angegeben werden, können sie hinter einem „-“ (ohne Abstand) gruppiert oder jedes separat mit einem eigenen „-“ angegeben werden. Wichtig ist, daß zwischen „-“ und der Option *kein Abstand* eingegeben wird. Schließlich gibt das *Argument* an, worauf sich das Kommando bezieht (hier: das Directory).

Es können mehrere Argumente getrennt durch einen Abstand eingegeben werden. Ein weiteres Beispiel (liste die Verzeichnisse `/home/karin` und `/home/fred` im „langen Format“ und zeige versteckte Dateien an):

```
ls -la /home/karin /home/fred          oder auch
ls -l -a /home/karin /home/fred
```

Abhängig vom Befehl können Optionen und/oder Argumente *optional* sein, was durch eckige Klammern angedeutet wird; in unserem Falle also:

```
ls [-a -l] [Datei]
```

Die Unix-Shell

„Knowing the Unix-Shell *is* knowing Unix."

- *das* essentielle Werkzeug des Benutzers
- wartet mit einem Prompt auf Eingaben, z.B.

  [praxis0@macondo praxis0]$
- interpretiert Kommandos und führt diese aus (Kommandointerpreter)
- ist case sensitive !!!!!!
- ist „viel mehr" als ein DOS-Fenster

Folie 40

Kommandosyntax

Kommando Optionen Argumente

Beispiele:

```
ls -l /home/fred
```

```
ls -l -a /home/karin /home/fred
```

```
ls -al /home/karin /home/fred
```

allgemein

```
ls [-a -l] [Datei]
```

Folie 41

Die Angabe einer Datei oder eines Directories (unter Unix ist ein Directory nichts anderes als eine spezielle Datei, und wir müssen hier nicht dazwischen unterscheiden) als Argument ist also optional; wird nichts angegeben so wird als Default das aktuelle Directory angenommen.

Weiters gibt es auch sogenannte *lange Optionen*, die aus ganzen Wörtern bestehen; diese werden üblicherweise mit vorangestelltem „--“ angegeben, z.B.:

```
ls --help
```

Word-Completion

... wenn das lange Tippen zu fad wird...

- tippe die Anfangsbuchstaben des Kommandos (ohne ENTER) und drücke dann <TAB>
- die Shell vesucht zu vervollständigen
- eindeutige Vervollständigung
  - Kommando wird vervollständigt
  - ausführen mit <ENTER>
- nicht eindeutige Vervollständigung
  - bis zum „Verzweigungspunkt” vervollständigt
  - Warnung und Anzeige der Möglichkeiten

Folie 42

Eine wichtige Eigenschaft der Shell, die das Leben erheblich erleichtert, ist die *Word-Completion*. Man gibt z.B. die ersten Buchstaben eines Kommandos ein und drückt dann die $<TAB>$-Taste. Die Shell versucht Kommandos automatisch zu vervollständigen. Ist diese Vervollständigung eindeutig, dann wird das Kommando einfach ergänzt, ist sie nicht eindeutig, so wird bis zum „Verzeigungspunkt“ vervollständigt und es gibt eine optische (Blinken) oder akustische Warnung. Drückt man daraufhin nochmals die $<TAB>$-Taste, so werden die verschiedenen Möglichkeiten angezeigt; ist die Liste zu lang, wird man sicherheitshalber noch einmal gefragt („Display all 3856 possibilities? (y or n)“). Man sieht so, wieviele weitere Zeichen eingegeben werden müssen, um eine eindeutige Vervollständigung zu erzielen. Ein analoges Verhalten zeigt die Shell auch bei der Angabe von Dateinamen (etwa als Argumente zu Kommandos). Durch effizientes Nutzen der Word-Completion kann der geübte Benutzer eine sehr große Arbeitsgeschwindigkeit erreichen, in der Regel höher als mit grafischer Bedienung. Außerdem kann die Word-Completion (Anzeigen der Alternativen) verwendet werden, wenn man etwa den genauen Namen eines Befehls oder einer Datei vergessen hat.

Neuere Versionen der Bash kennen auch die „programmable Completion“die es ermöglicht, auch Optionen zu vervollständigen oder bei Argumenten die Auswahl auf Dateien mit einer bestimmten Endung zu beschränken.

Wichtige Keyboard-Tips, die das Arbeiten in der Shell komfortabler und schneller machen, sind auf Folie 43 zusammengefaßt.

Wir zählen nun einige kleine und nützliche Shell-Kommandos zu folgenden Themenbereichen auf

- einfache Shell-Kommandos (Folie 44)
- User-Information (Folie 45)

Keyboard Tips

<ARROW-UP>	gibt das letzte Kommando zurück
<CTRL-c>	Kommandoabbruch und Rückkehr zur Shell
<CTRL-d>	exit
<CTRL-s>	stoppt Output auf Bildschirm
	und Input, „merkt“ sich aber Zeichen
<CTRL-q>	gibt Input/Output wieder frei
<CTRL-u>	löscht Zeile
<SHIFT-PG UP>	scrollt in der Shell zurück
<CTRL-PG DOWN>	scrollt in der Shell vor

Folie 43

- User-Kommunikation (Folie 46)
- System-Information(Folie 47) .

Wir geben meist nur den Befehl ohne Optionen bzw. Argumente an; diese können über die Manpages durch die Einagbe

```
$ man Befehl
```

online abgefragt werden. Das Thema Hilfe und Dokumentation wird systematisch in Kapitel 4 abgehandelt.

3.3 Vom Umgang mit Dateien

Hier soll der grundlegende Umgang mit Dateien und Verzeichnissen vorgestellt werden. Es gibt zwar verschiedene grafische Tools (die sich ähnlich dem Windows-Explorer verhalten), insbesondere ist in jeder der grossen Desktopumgebungen ein leistungsfähiger Filebrowser integriert. Für den Systemadministrator ist das Bewegen auf der Konsole allerdings Pflichtprogramm; da die Konsole für viele Aufgaben auch wesentlich leistungsfähiger ist und die Arbeit mit ihr in der Regel rascher geht, ist es auch für den „normalen“ Benutzer interessant, sich eingehend mit ihr auseinanderzusetzen.

Jedes Objekt in einem Unix Betriebssystem ist entweder ein Prozess oder eine Datei. Unix unterscheidet prinzipell drei Typen von Dateien: *gewöhnliche* Dateien, *Directories* und *spezielle Files*. Letztere repräsentieren die Hardware des Systems und werden im Abschnitt 3.7 kurz behandelt. Gewöhnliche Files können entweder Textfiles, die angesehen oder ausgedruckt werden können, oder Code, d.h. exekutierbare (ausführbare) Programme sein. Directories (Verzeichnisse) sind unter Unix nichts anderes als Files, die das Inhaltsverzeichnis des Directories enthalten (und die Information, wo sich die Dateien befinden).

Oft gibt die „Endung“ einer Datei (das ist der Teil des Dateinamens, der nach dem letzten Punkt kommt, hat aber unter Unix keine spezielle Bedeutung) Indizien darauf, welche Art von Daten sich dahinter verbirgt. Weiss man nicht, was in einer Datei steckt, so kann der leistungsfähige Befehl `file` helfen

```
file filename
```

Einfache Shell-Kommandos

`clear`,	
`<CTRL-l>`	löscht Terminal
`echo string`	gibt `string` aus (Scripts!)
`wc [-lwc] file`	zählt die Anzahl der Zeilen, Wörter bzw. Zeichen in `file`
`sort file`	sortiert die Zeilen von `file` alphabetisch
`grep Muster file`	gibt die Zeilen in `file` aus, die `Muster` enthalten

Folie 44

User-Information

`who`	wer ist am System eingeloggt
`w`	wer ist am System eingeloggt und tut was
`finger [user]`	Information über den Benutzer
`who am i`	
`whoami`	wer bin ich

Folie 45

User-Kommunikation

`talk Benutzer[@Host]`	beginnt interaktives „Gespräch“mit Benutzer auf Host
`write Benutzer[@Host]`	schreibt Nachricht an Benutzer
`wall`	schreibt Nachricht an alle eingeloggten Benutzer
`mesg [y|n]`	erlaubt/verbietet Nachrichten mittels `wall, write, talk`; root überschreibt `mesg`

Folie 46

System-Information

`date`	gibt Systemzeit und -datum aus
`uname [-a]`	gibt Systeminformation aus
`hostname`	gibt Hostname aus
`uptime`	gibt Uptime, Anzahl der Benutzer, etc. aus

Folie 47

zeigt den Typ der Datei `filename` an. Dem Betriebssystem ist es *vollkommen* egal, was sich wirklich hinter einer Datei verbirgt; eine Datei ist immer eine Aneinanderreihung von Bytes.

Eine Folge von Datei- beziehungsweise Verzeichnisnamen getrennt durch „forward Slashes“, / heißt *Pfadname*. Wir unterscheiden *absolute* und *relative* Pfadnamen: erstere beginnen mit dem *Root-Directory* /, in dem der ganze Verzeichnisbaum zusammenläuft, letztere beginnen mit dem aktuellen Verzeichnis (Beispiele finden sich auf Folie 49). Steht am Ende eines Pfadnamens ein /, so handelt es sich dabei um ein Directory. (Traditionelle) Unix-Dateinamen können bis zu 255 Zeichen lang sein und können keinen, einen oder auch mehrere Punkte beinhalten (und dürfen auf keinen fall Slashes enthalten). Richtlinien zur Wahl von Filenamen sind auf Folie 50 zusammengestellt. Wie schon oben erwähnt, gibt es unter Unix wie unter Win* Konventionen, die den Typ eines Files an den Endungen erkennen lassen. Diese sind aber nicht durch die 8+3-Konvention beschränkt (`index.html` vs. `index.htm`). Files, deren Namen mit einem Punkt beginnen sind versteckt (üblicherweise Konfigurationsfiles, die man im Normalfall nicht sehen will; sie können mit `ls -a` gelistet werden). Das aktuelle Verzeichnis wird mit „.“ abgekürzt, das übergeordente Verzeichnis (Parentdirectory) mit „../“.

Dateitypen

- gewöhnliche Files
  - Textfiles, Grafikfiles
  - (ausführbarer) Code
  - ...
- Verzeichnis
  - Inhaltsverzeichnis
  - Liste der Files
- spezielle Files
  - repräsentieren Hardware des Systems
- `file filename` gibt Typ von Datei `filename` aus

Folie 48

Um sich im *Unix-Verzeichnisbaum zu bewegen*, verwendet man das Kommando `cd` (Change Directory), das völlig analog zum `DOS`-Äquivalent funktioniert. Hat man sich „verirrt“, so zeigt einem das Kommando `pwd` (Present Working-Directory) das aktuelle Verzeichnis an. Den Inhalt eines Verzeichnisses sieht man mit dem `ls`-Kommando. Die wichtigsten Optionen von `ls` sind auf Folie 51 zusammengefaßt. `dir` ist oft ein Alias (siehe 6.2) für `ls -al`, oder überhaupt für `ls`.

Wichtige Kommandos zum *Anlegen, Löschen* und *Bewegen von Dateien* und *Verzeichnissen* `touch, mkdir, cp, mv, rm` werden auf den Folien 52 und 53 erklärt. Wichtig ist, daß unter Unix kein einfaches Rückgängigmachen des Löschens möglich ist. Daher ist die Option `-i` (interaktiv) bei `rm` und `mv` äußerst wichtig; sie erzwingt eine Rückfrage vor dem Löschen bzw. Überschreiben vorhandener Dateien. Üblicherweise ist deshalb `rm` bzw. `mv` ein Alias auf `rm -i` bzw. `mv -i` (siehe Abschnitt 6.2).

Die Unix-Shell kennt verschiedene *Metacharacters*; das sind Zeichen, die für die Shell eine besondere Bedeutung haben. Wichtige Metacharacters sind *Wildcards* (z.B. „ ?“, „*“ „[]‘ „[-]“), die besonders gut geeignet sind, um Dateinamen–oder Dateinamenmuster–zu finden und daher ein besonders praktisches Verwenden der oben erklärten Kommandos

Pfadnamen

- absolute Pfadnamen
  - beginnend von `/` aus, z.B.
  - `/home/fred/Texte/brief17.txt`
  - `/home/fred/Bilder/Urlaub/`
- relative Pfadnamen
  - vom aktuellen Verzeichnis aus,
  - z.B. von `/home/fred/` aus
    - `Texte/brief17.txt`
    - `Bilder/Urlaub/`
  - z.B. von `/home/fred/Texte/` aus
    - `brief17.txt`
    - `../Bilder/Urlaub/`

Folie 49

Unix-Dateinamen

- bis zu 255 Zeichen lang, case sensitive
- soll(t)en Inhalt beschreiben (*nicht* `text.txt`!)
- sollen nur alphanumerische Zeichen enthalten
- keine Leerzeichen, Shell-Metacharakter (*,?,<, >,;,&,[,],...)
- kein +-
- kein Systemkommando
- versteckte Files beginnen mit „. “
- „. “ aktuelles Verzeichnis
- „../ “übergeordnetes Verzeichnis

Folie 50

Bewegen in Filesystem

`cd [directory]`	wechselt ins Verzeichnis `directory`
`pwd`	gibt das aktuelle Verzeichnis aus
`ls [-aldh...] [file]`	listet die Datei `file`
`-a` (all)	listet auch versteckte Dateien
`-l` (long)	langes Format (Typ, Rechte, Größe, etc.)
`-d` (directory)	Verzeichniseintrag statt Inhalt
`-h` (human readable)	(mit `-l`) zeigt Größe in kB, MB an
`--color[=when]`	färbiges Listing, wobei when= always, never, auto

Folie 51

ermöglichen. „*“ersetzt eine beliebig lange Folge von beliebigen Zeichen, „?“ ersetzt *genau ein* Zeichen. Das gilt aber nicht für versteckte Dateien; den Punkt am Anfang muß man unabhängig von der verwendeten Wildcard spezifizieren. Mittels eckiger Klammern kann man die Auswahl einengen. So bewirkt zum Beispiel `ls [a,b,c]*`, daß alle Dateien mit den (kleinen) Anfangsbuchstaben a, b, oder c gelistet werden. „`!`“ verneint den Ausdruck in der Klammer; „`-`“ steht für alle Zeichen dazwischen (z.B.: `[!a-z]` = kein Kleinbuchstabe).

Um nicht nur den Inhalt von Verzeichnissen, sondern auch den der sich darin befindenden *Dateien (resp. ihren Inhalt) auszugeben*, bedarf es im Falle von schlichten Textfiles nur der richtigen Auswahl der nun beschriebenen Programme. `cat` zeigt ungebremst den gesamten Inhalt der angegebenen Datei an, was vor allem bei größeren Texten schon an die Grenzen der menschlichen Leistungsfähigkeit stößt.

`more` gliedert die hereinbrechende Informationsflut in bildschirmgroße Happen, die man mit <SPACE> weiterverfolgen oder mit <q> beenden kann.

`less` bietet einen größeren Funktionsumfang und Komfort (ein Wortspiel auf `more`; kann mehr, obwohl es nach weniger klingt). Die Pfeiltasten scrollen hinauf bzw. hinunter, `/ Muster` sucht (abwärts) nach `Muster`, `? Muster` aufwärts und mit `q` gelangt man wieder in die Shell.

Alle drei Kommandos können auch benutzt werden, um beliebige (nicht nur Text-) Dateien anzuzeigen, was allerdings nicht immer Sinn macht.

Schließlich erwähnen wir den Befehl `lpr` der das Ausdrucken von (Text- Postscript- und anderern) Files erlaubt, sowie die Kommandos `lpq` und `lprm` zum Verwalten von Druckaufträgen (siehe Folie 55).

3.4 Umleitungen und Pipes

Hier ist nun ein guter Zeitpunk, um einen ersten Einblick in die Funktionalität der Unix-Shell zu gewähren, und *Umleitungen (Redirects)* und *Pipes* einzuführen. Diese sind die Basisstrukturen, mit denen in der Shell mehrere Kommandos verknüpft werden können, um komplexere Aufgaben zu bewältigen.

Wenn die Shell ein Kommando ausführt werden automatisch drei damit asoziierte Dateien, die sogenannten *Standard File Deskriptoren* geöffnet: diese sind der *Standard Input (STDIN)*, der *Standard Output (STDOUT)* und der *Standard Error (STDERR)*. Der Standard Input ist, woher sich das Kommando seinen Input erwartet; das ist per default die

Erzeugen, Kopieren und Verschieben

`cp [-iR...] src1 [... srcN] destination`	kopieren
`-i` (interactive)	Rückfrage vor Überschreiben
`-R`	rekursiv
`-p`	preserve (erhält Attribute)
`mv [-i] oldname newname`	umbenennen
`mv [-i] file1 [...fileN] new-directory`	verschieben
`mkdir name`	erzeugt Verzeichnis `name`
`touch name`	aktualisiert Datumsstempel; erstellt leere Datei, falls nicht vorhanden

Folie 52

Löschen

`rmdir`	löscht Verzeichnis (wenn leer!)
`rm [-irf] name`	löscht Datei `name`
`-r`	rekursiv (für nichtleere Verzeichnisse)
`-f` (force)	ohne Rückfrage

- ACHTUNG: KEIN EINFACHES UNDELETE!!!
- Wildcards
  - ?... ersetzt genau *ein* beliebiges Zeichen
  - ∗... ersetzt eine Folge von beliebig vielen beliebigen Zeichen

Folie 53

Textdateien anzeigen

- `cat filename` einfaches Ausgeben
- `more filename` etwas komfortabler
  - <SPACE> scrollen
  - <q> (quit) `more` verlassen
- `less filename` komfortabel
  - `/ Muster` sucht vorwärts nach `Muster`
  - `? Muster` sucht rückwärts nach `Muster`
  - <q> (quit) `less` verlassen

Folie 54

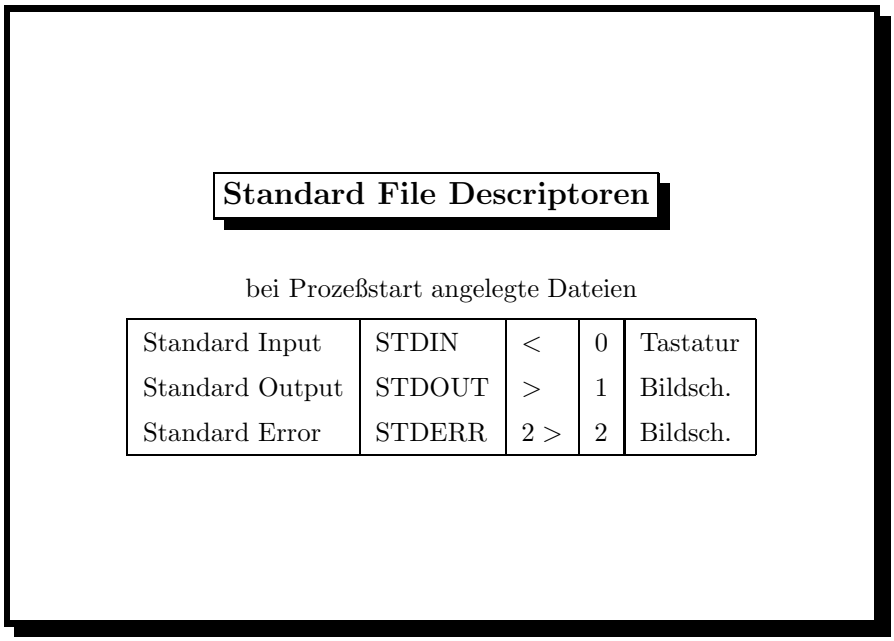
Drucken

`lpr [-P Name] Datei`	druckt `Datei` auf Drucker `Name`
`lpq [-P Name]`	zeigt Druckerjobs auf Drucker `Name` an (inkl. Nr.)
`lprm [-P Name] Nr`	löscht Druckerjob mit der Nummern `Nr`

Folie 55

Tastatur. Der Standard Output und Error sind, whohin der Prozeß den Output und die Fehlermeldungen schreibt; per default der Bildschirm. Die Abkürzungen und Symbolik der Filedeskriptoren ist auf Folie 56 dargestellt.

Ein Weg die Funktionalität der Shell zu erhöhen ist es nun, diese Defaults umzuleiten. Mit der Syntax `Kommando < InFile` wird der STDIN für das Kommando statt von der Tastatur aus der Datei `InFile` bezogen. Die Syntax `Kommando > OutFile` lenkt den STDOUT in die Datei `OutFile` um. `ls -l > ls.out` z.B. schreibt die Ausgabe des `ls` Befehls in die Datei `ls.out`. Mittels `2>` kann der STDERR in eine Datei ungeleitet werden. In beiden Fällen bewirkt `>> OutFile` statt `> OutFile`, daß der Output an das Ende von `OutFile` angehängt wird; sonst wird die eventuell existierende Datei `OutFile` überschrieben. Es können auch STDOUT und STDERR in dieselbe Datei umgeleitet werden (*kombinierter Redirect*); die entsprechende Syntax ist `Kommando > OutFile 2>&1`; Achtung, die Reihenfolge ist wichtig! (warum?). Die Syntax für Umleitungen ist auf Folie 57 zusammengefaßt, wo man auch die Antwort zu dem vorangegangenen warum findet.



Folie 56

Noch größere Flexibilität erreicht man durch das Umleiten des Standardoutputs eines Kommandos in den Standardinput eines weiteren Kommandos; diese Struktur wird *Pipe* genannt. So können zwei (oder mehrere) Kommandos hintereinander auszuführen, wobei der Output des ersten Kommandos in das zweite Programm „gefüttert“ wird. Die Syntax ist:

```
Kommando1 | Kommando2 | Kommando3 | ...
```

Ein wichtiges Beispiel ist das komfortable Listen langer Verzeichnisse:

```
ls -l /etc | less
```

Hier wird der Output des `ls`-Kommandos statt auf den Bildschirm in das `less`-Kommando umgeleitet; dieses gibt das formatierte Listing auf dem Bildschirm aus und ermöglicht u.a. ein interaktives Suchen nach bestimmten Dateien.

Um weitere sinnvolle Beispiele zu konstruieren verwenden wir die Kommandos `wc` und `grep` aus dem vorigen Abschnitt.

- `ls -l /tmp | grep roland | less` zeigt ein Listing aller Files mit Besitzer roland in `/tmp`.
- `ls -l | wc -l` zählt die Files im aktuellen Directory.

Umleitungen

- `command < in.file > out.file 2> error.file`
- `command < in.file >> append.out 2>> append.err`
- `command > out.file 2>&1`
  zuerst STDOUT nach `out.file`, dann STDERR nach STDOUT
- NICHT `command 2>&1 > out.file`
  zuerst STDERR nach STDOUT=Bildsch., dann STDOUT nach `out.file`

Folie 57

Pipes

- eine Folge von Kommandos getrennt durch „|“
- STDOUT des vorigen Kommandos als STDIN des nächsten
- Syntax

  ```
  Kommando 1 | Kommando 2 | ...| KommandoN
  ```

- Beispiele
  - `ls -l|less`
  - `ls -l|grep 'Nov 4'|wc -l`

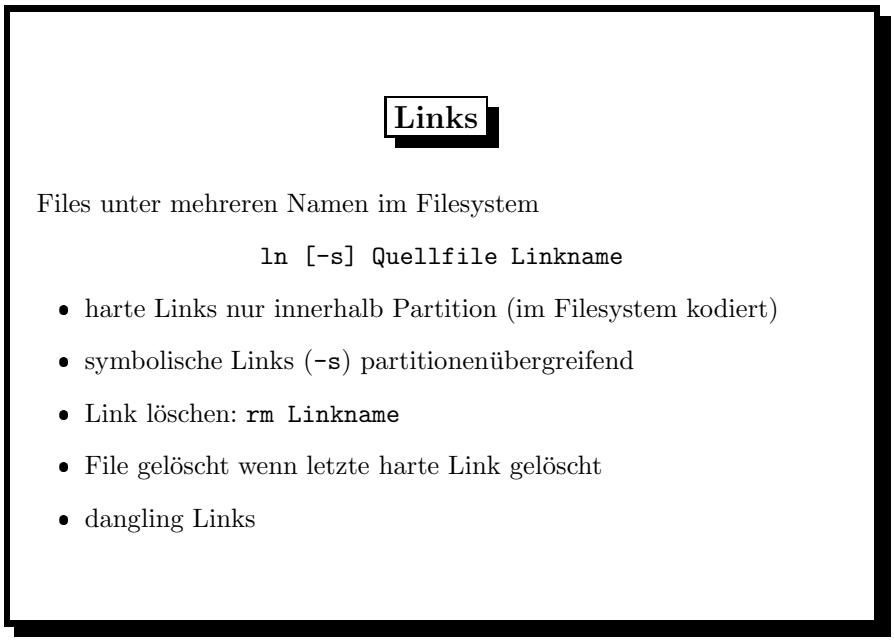
Folie 58

- `ls -l | grep 'Nov 4'| wc -l` zählt die Files im aktuellen Directory, die zuletzt am 4. November verändert wurden.

3.5 Links

Oft ist es sinnvoll, ein und dieselbe Datei unter zwei oder mehreren Namen an verschiedenen Stellen im Filesystem ansprechen zu können; diese Überlegung führt uns zum Konzept von *Links*. Links sind „Dummy-Files“, die nichts anderes tun, als auf das richtige File zu zeigen und werden mit dem Kommando `ln` angelegt; die Synatx ist auf Folie 59 erklärt.

Wichtig ist die Unterscheidung in *harte* und *symbolische* Links. Jedes File ist „in Wirklichkeit“ ein harter Link. Es kann aber mehrere (harte) Links geben, die auf dasselbe File zeigen; ein File ist dann gelöscht, wenn der letzte harte Link auf das File gelöscht ist. Harte links sind „hart“ in der logischen Struktur des Filesystems kodiert und können daher nur *innerhalb* einer Partition verwendet werden (vgl. Abschnitt 3.7). *Symbolische* (Soft-, Sym-) Links können über die Grenze von Partitionen hinausgehen. Wird ein *symbolischer* Link gelöscht, so ist nur der Link verschwunden, nicht das File (auch wenn alle Symlinks auf ein bestimmtes File gelöscht werden, gibt es das File noch immer). Wird hingegen das File selbst gelöscht (d.h. der letzte harte Link der darauf zeigt), bleibt der Symlink als Zeiger erhalten, der allerdings ins Leere zeigt, also unbrauchbar geworden ist; Man spricht von „broken“ oder „dangling“ Links.



Folie 59

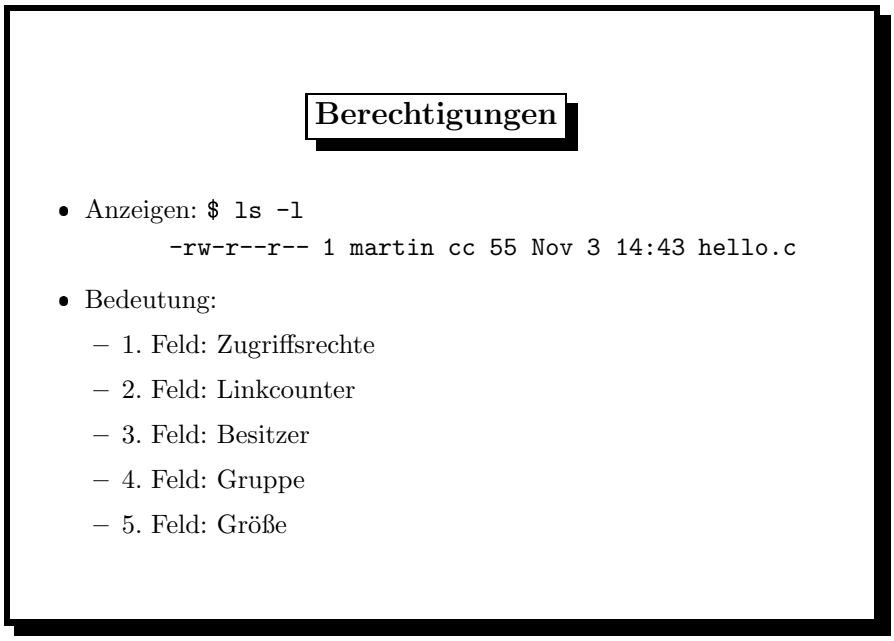
3.6 Dateiberechtigungen

Unter Unix hat jede Datei (und somit auch jedes Verzeichnis) im Dateisystem einen bestimmten Besitzer und eine bestimmte Gruppe. Das ermöglicht eine Vergabe von spezifizierten Dateiberechtigungen an verschiedene Benutzer des Systems. Der Unix-Standard ist ein dreistufiges Berechtigungsystem, das zwischen dem *Besitzer*, der *Gruppe* und allen *anderen* unterscheidet, die jeweils drei Rechte haben können: *Lesen, Schreiben, Ausführen (r,w,x)*. Die Bedeutung dieser Rechte für Dateien ist evident, für Verzeichnisse gilt das folgende: Ausführberechtigung bedeutet, daß man ins Verzeichnis wechseln darf. Leseberechtigung steht für das Ansehen des Verzeichnisinhalts, i.e., Listen des Directories. Hat man nur die Berechtigung `x` auf einem Directory, kann man nur auf Dateien zugreifen, deren Namen man kennt (und die lesbar sind!; Listings des Verzeichnisses sind verboten). Schreibberechtigung schließlich ist das Recht, weitere Dateien im Verzeichnis zu erstellen oder zu entfernen

(komplett entleeren kann man sie auch wenn man nur auf der Datei Schreibrecht hat). Die Berechtigungen werden bei jedem Zugriff auf die Datei geprüft, gleichgültig, ob der Benutzer direkt auf die Datei zugreifen will oder über eine Applikation. Ansehen kann man diese Informationen mit dem alten Bekannten `ls -l`, *verändern* kann sie jeweils der Besitzer mit `chmod, chown` und `chgrp`. Die Syntax dieser Befehle wird auf Folie 62 erklärt.

Neben der *symbolischen Darstellung* von Dateiberechtigungen (rwx) für die drei Benutzergruppen (Besitzer (User), Gruppenbesitzer (Group), andere (Others)=(ugo)) wird auch die *oktale Darstellung* verwendet. Für die drei Benutzergruppen wird die Folge (rwx) jeweils als Binärzahl aufgefasst (z.B. 101 für r-x); das ergibt im okatlen Zahlensystem jeweils eine einstellige Zahl (hier: $1 \times 4 + 0 \times 2 + 1 \times 1 = 5$). Eine Folge von drei oktalen Ziffern gibt also die gesamte Dateiberechtigng wieder; z.B. steht 765 für 7 für den Besitzer, 6 für den Gruppenbesitzer und 5 für andere Benutzer. Das bedeutet also 7=111=rwx für den Besitzer, 6=110=rw- für den Gruppenbesitzer und 5=101=r-x für alle anderen; also insgesamt: 765=rwxrw-r-x. Weitere Beispiel befinden sich auf Folie 63.

Die Berechtigungen neu erstellter Dateien werden über das `umask`-Kommando geregelt, dessen Syntax auf Folie 64 erklärt ist. Die `umask` ist auf einen Modus gesetzt (Standard: `002`), der von der vollen Berechtigung `666` bei Dateien und `777` bei Verzeichnissen subtrahiert wird. Also haben in diesem Fall neu erstellte Dateien die Berechtigung `664=rw-rw-r--`.



Folie 60

Berechtigungen (symbolisch)

- Aufschlüsselung des 1. Feldes (`-rw-r--r--`):
  - 1. Bit: Spezifizierung des Datei-Typs
    - `-` Datei
    - `d` Verzeichnis
    - `l` Symbolischer Link
    - `c` oder `b` spezielles File
  - jeweils drei Bits für die Rechte von Besitzer(u), Gruppe(g) und anderen(o)
    - `r` Lesen (bei Verzeichnissen: Listen)
    - `w` Schreiben (Dateien im Verz. erstellen/überschreiben/Löschen)
    - `x` Ausführen (Hineinwechseln)

Folie 61

Rechte verändern (Symbolisch)

```
chmod modus datei
```

Modus zusammengesetzt aus:

- `u` Besitzer
- `g` Gruppe
- `o` Andere
- `a` Alle
- `+` Rechte setzen
- `-` Rechte entfernen
- `=` Rechte spezifiziert setzen
- `r` Lesen
- `w` Schreiben
- `x` Ausführen

Beispiel: `chmod ug+rw hello.c`

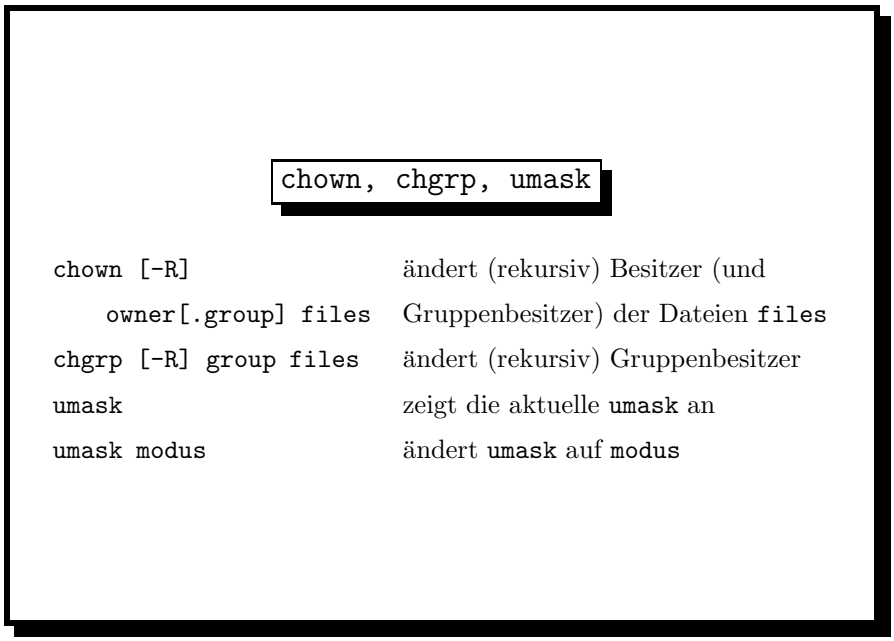
Folie 62

Berechtigungen (oktal)

	Besitzer	Gruppe	Andere
symbolisch	rwx	rw-	r-x
binär	111	110	101
oktal	7	6	5

Beispiel: `chmod 644 bericht.txt`
entspricht: `chmod u=rw bericht.txt`
`chmod go=r bericht.txt`

Folie 63



Folie 64

3.7 Tour durchs Filesystem

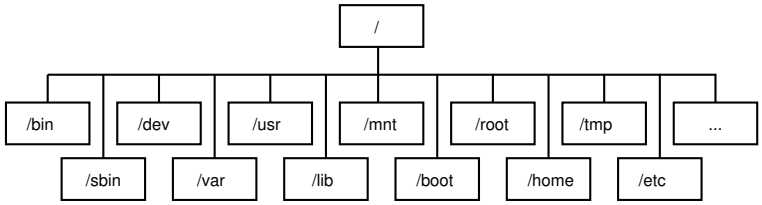
In diesem Abschnitt unternehmen wir einen kleinen Rundgang durch das Filesystem eines „typischen“ Linux-Systems. Im Großen und Ganzen folgen die meisten Distributionen dem unter `http://www.pathname.com/fhs` festgelegten Filesystemstandard, der sich eng an die entsprechenden Unix-Konventionen anlehnt. Die Abweichungen auf individuellen Systemen und von Distribution zu Distribution sind allerdings teilweise doch relativ groß. Hier geht es darum, einen ersten Überblick zu bekommen, sodaß diese Unterschiede nicht so ins Gewicht fallen.

Das Filesystem kann aus einer oder mehreren Partitionen (logische Einheit auf der Festplatte, die wie eine eigene Platte angesprochen werden kann; für Details siehe Kapitel 8) bestehen. Gibt es nur eine Partition, so ist dies die `root`-Partition, d.h. dort, wo das `/`-Verzeichnis liegt, in dem alle anderen Verzeichnisse zusammenlaufen. Es gibt aber gute Gründe, gewisse Teile des Filesystems (i.e. bestimmte Verzeichnisse) auf eigenen Partitionen anzulegen. Diese Partitionen werden dann an den entsprechenden Stellen in die `/`-Partition gemountet (d.h. „angehängt“, Details in Teil 2). Praktisch sieht das so aus, daß auf der `/`-Partition ein leeres Verzeichnis angelegt wird (z.B. `/home`). In dieses *Mountpoint* genannte Directory wird dann beim Systemstart dann die entsprechende Partition, die in diesem Fall üblicherweise die Homedirectories der Benutzer enthält, gemountet. Wir diskutieren im Detail, welche Teile des Filesystems auf einer eigenen Partition angelegt werden können und für welche dies nicht ratsam ist.

Oft gibt es auch weitere Directories in einem Linux-System, die wir auf Folie 72 diskutieren.

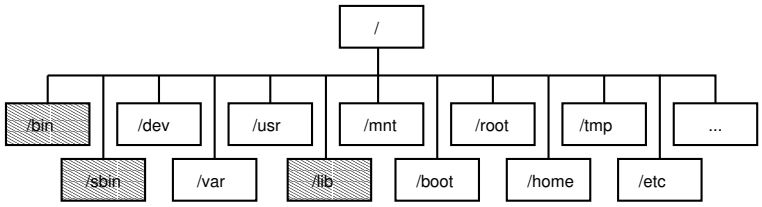
Weitere wichtige Befehle (neben `cd, ls`) mit denen man sich einen guten Überblick im Dateisystem verschaffen kann sind `du, locate, which` und das mächtige Kommando `find`. Wir stellen Syntax und wichtigste Optionen dieser Befehle auf Folien 73, 74 zusammen.

Überblick



Standards unter `http://www.pathname.com/fhs`

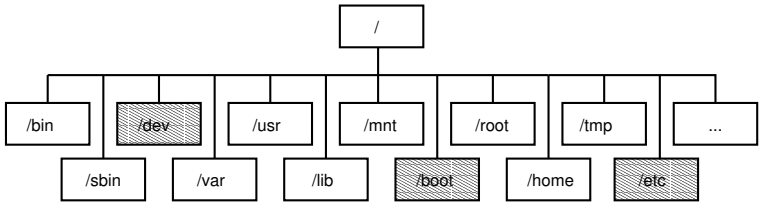
Folie 65



- `/bin` wichtige Systemprogramme
- `/sbin` Systemprogramme für root (static binaries)
- `/lib` C-Bibliotheken

auf der /-Partition (sollen immer vorhanden sein)

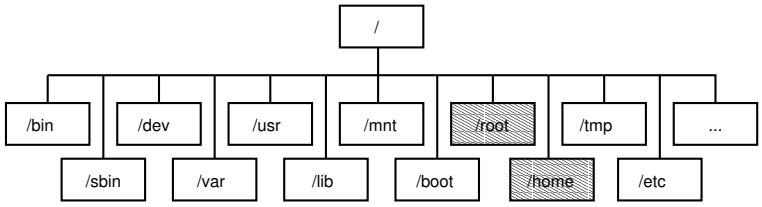
Folie 66



- `/boot` Kernel (u.U. eigene Partition)
- `/dev` spezielle Files, repräsentieren Hardware
- `/etc` systemweite Konfigurationsfiles

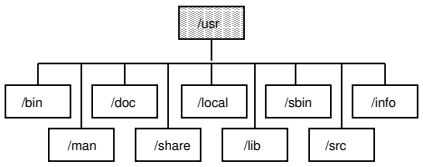
  keine eigenen Partitionen

Folie 67



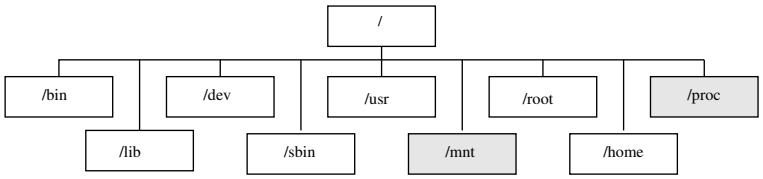
- `/home` beinhaltet Homeverzeichnisse der Benutzer
  oft eigene Partition (Sicherheit, Neuinstallation)
- `/root` Homeverzeichnis von root
  keine eigene Partition

Folie 68



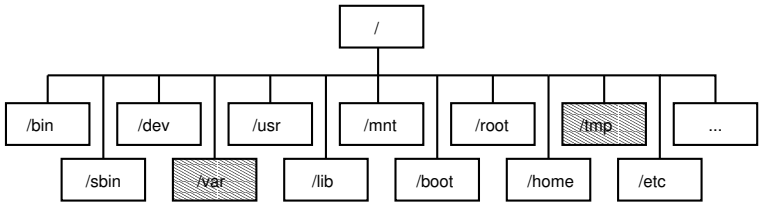
- `/usr` enthält alle nicht essentiellen Systemprogramme
- oft eigene Partition, wichtige Unterverzeichnisse:
  - `/usr/X11` X-Window
  - `/usr/local` lokale verwendete Dateien
  - `/usr/(share/)doc` Dokumentation
  - `/usr/info` Online Dokumantationssystem
  - `/usr/man` Manpages
  - `/usr/share` Architekturunabhängige Dateien
  - `/usr/src` Quellcode

Folie 69



- `/mnt` enthält Mountpoints für Floppy, CD-Rom ,...
  keine eigene Patition
- `/proc` virtuelles Filesystem (Kernel- und Prozeßinformation)

Folie 70



- `/tmp` temporäre Dateien
  oft eigene Partiton (Defragmentierung)
- `/var` variable Größe, eigene Partition auf Servern
  - `/var/log/` Systemlogfiles
  - `/var/spool/` Spoolfiles (Drucker, Mail,...)

Folie 71

Weitere Verzeichnisse

- `/opt` oft von großen Programmpaketen verwendet
- `/disks`, `/media` alternativer Mountpoint für CD-Rom etc.
- `/usr/src/linux` Kernel-Quellcode
- `lost+found` auf jeder Partition vorhanden, kaputte Files nach Festplattencheck

Folie 72

du, locate, which

`du [-hsm] Files` (disk usage)	zeigt benötigten Plattenplatz von Files an
`-s` (summarize)	Zusammenfassen für Verzeichnisse
`-h` (human readable)	netter Output
`-m` (mega)	gibt Größe in MB an
`locate [dir] string`	durchsucht Filenamen nach `string` ab Verzeichnis `dir`
`which command`	sucht Pfad des Kommandos

Folie 73

find

`find [path...] [expression]`

- mächtiges Suchwerkzeug
- Sucht nach `expression` im Pfad `path`; `expression` kann Optionen enthalten
- einige Optionen
  - `-name expression` sucht nach Filenamen
  - `-cmin n` File zuletzt verändert vor n Minuten
  - `-cnewer file2` File neuer als `file2`
  - vieles, vieles mehr ...
- Beispiel: `find . -name *.txt`

Folie 74

4 Dokumentation

Linux Dokumentation gibt es wie Sand am Meer. Tatsächlich ist die Verfügbarkeit von viel und viel gutem Dokumentationsmaterial (Online und Offline) auf verschiedenen technischen Niveaus eine der großen Stärken von Linux. Dieses kurze Kapitel gibt einen Überblick über die verschiedenen Informationsquellen und erklärt, wo was zu finden ist.

Um etwas Ordnung in die Fülle von vorhandenen Dokumentationsquellen zu bringen, unterscheiden wir zwischen *lokaler Online-Dokumentation*, Dokumentation im *Internet* und *Offline-Dokumentation* also Bücher etc.

Drei gute allgemeine Bücher über Linux/Unix-Systemadministration sind in der Einleitung erwähnt. Darüberhinaus gibt es viele weitere „allgemeine“ und auch Bücher zu speziell(er)en Themen, wie etwa Linux-Netzwerkadministration, Webserver-Administration unter Linux oder Treiberprogrammierung für Linux etc. Eine sehr umfangreiche Liste von Linux Büchern findet sich auf der Linux Homepage unter `http://www.linux.org/books/index.html`. Klarerweise sind die meisten Bücher in englischer Sprache erschienen; eine gute Beherrschung des Englischen ist aus naheliegenden Gründen gerade im EDV-Bereich sehr wichtig. Eine Übersicht über deutsche Linux-Bücher gibt es z.B. auf der SuSE Homepage unter `http://www.suse.de/de/products/books/linux/index.html`.

Als wichtigster Teil der *lokalen Online-Dokumentation* sind die *Manpages* (Manual Pages) zu nennen. Zu jedem Unix Befehl gibt es eine eigene Manpage, die eine kurze Beschreibung der Funktion, der Syntax und aller Optionen enthält und auf *jedem* Unix-System installiert ist (siehe auch Folie 76). Manpages sind unverzichtbar in der täglichen Arbeit mit Unix-Befehlen und werden mit `man befehl` aufgerufen. Vielen Anfängern fällt wegen der sehr knappen und technischen Sprache der Umgang mit den Manpages schwer; nichtsdestotrotz

Will man in der Unix Shell eine bestimmte Aufgabe erledigen, hat aber den adäquaten Befehl vergessen, so bietet `man -k begriff` resp. `apropos begriff` die Möglichkeit, die Manpages nach dem Begriff zu durchsuchen.

Eine neuere Alternative zu Manpages bieten die *Info*-Seiten. Sie unterscheiden sich hauptsächlich in der `emacs`-artigen Bedienung und der Möglichkeit, über Hyperlinks direkt zu verwandten Seiten zu wechseln. Findet man mit `man` oder `info` eine Option nicht, von der man aber glaubt, daß sie vom Kommando unterstützt wird, hilft oft ein an das Kommando angehängtes `--help`, oder `-h`; z.B. `ls --help`.

Eine weitere lokale Informationsquelle ist die bei den meisten größeren Programmpaketen inkludierte Dokumentation, die auf den meisten Systemen unter `/usr/(share/)doc/` zu finden ist. Bei den meisten Paketen ist eine beträchtliche Informationsfülle vorhanden, die von Hilfe bei der Installation bis hin zu einer Bedienungsdokumentation oder Verweisen auf die Homepage der Software alles enthält.

Neben den bisher genannten speziellen Hilfetexten gibt es eine Reihe von allgemeineren Texten, die breitere Themen in einem weiteren Rahmen behandeln; nämlich die *FAQs (Frequently Asked Questions)*, und die *HOWTOs (How To...)*.

Die FAQs sind eine Sammlung von häufig gestellten Fragen *mit* den jeweiligen Antworten. Die Themenbereiche sind recht allgemein und reichen von „Was ist Linux?“ bis etwa zu „Wie kann ich gelöschte Files restaurieren?“. Hat man sich also schon einmal gefragt, wie dieses oder jenes funktioniert, ist es ziemlich wahrscheinlich, hier eine Lösung für das Problem zu finden.

Die *HOWTOs* bieten Erklärungen zur Bewerkstelligung größerer Aufgaben, wie etwa die Installation eines Web- oder Mailservers. Auch der technische Hintergrund wird ausführlich erörtert. HOWTOs sind unter anderem zu folgenden Themenbereichen vorhanden: Netzwerk, Hardware, Systemadministration, Programmieren, ...

FAQs und HOWTOs sind oft lokal auf Linux-Systemen installiert und dann meist im Text- oder Html-Format unter `/usr/(share/)doc/FAQ` resp. `/usr/(share/)doc/HOWTO` zu finden. Es empfiehlt sich aber jedenfalls im Internet nach der neuesten Version zu suchen. Diese befindet sich immer auf der „ersten Adresse“ für Linux Dokumentation, der Homepage

Dokumentation

- Online lokal
  - Manpages (Online-Kommandobeschreibung, lokal)
  - Infoseiten (Hypertext-Kommandobeschreibung, lokal)
  - `Befehl --help` (Hilfe für `Befehl`)
  - Programmdokumentation (`/usr/(share/)doc/programm`)
  - FAQs (Frequently asked Questions) (`/usr/(share/)doc/FAQ`)
  - HOWTOs (`/usr/(share/)doc/HOWTO`)
- Internet
  - Linux Documentation Project (LDP, `www.ldp.at`)
  - Usenet (z.B. `at.linux`) , Email
  - Distributionssupport

Folie 75

Manpages

- `man befehl`
- in `/usr/man`
- Inhalt
  - **Name:** Name und einzeilige Beschreibung
  - **Synopsis:** Kommandosyntax
  - **Description:** Erklärung der Funktionsweise
  - **Files:** Mit dem Kommando assoziierte Dateien
  - **Bugs:** Bekannte Bugs
  - **See also:** Verwandte Befehle

Folie 76

des *Linux Documentation Projects (LDP)* unter `http://www.tldp.org`. Deutsche Übersetzungen (die allerdings nicht immer sehr empfehlenswert sind) findet man z.B. bei SuSE unter `http://www.suse.de/de/support/howto/index.html`.

Das Internet und hier besonders das World Wide Web bieten eine riesige Flut an Informationen an und so verwundert es nicht, daß es auch die bei weitem größte Quelle an Linux-Hilfestellungen darstellt. Von Artikeln zu besonderer Hardware über Anleitungen zur Programmierung unter Unix bis hin zu detaillierten Erklärungen zur inneren Funktionsweise eines Unix-Systems findet man alles, was das Herz begehrt. Ein guter Startpunkt ist die (schon erwähnte) Homepage des *Linux Documentation Project* (LDP, `http://www.tldp.org`). Dort befinden sich die neuesten Versionen der Manpages, HOWTOs, FAQs und vieles mehr. Besonders hervorzuheben sind noch die *Guides*, lange, buchartige Dokumente zu großen Themenbereichen, z.B. „Linux Kernel Internals“, „The Linux Network Administrator's Guide, Second Edition“, etc.

Für ganz spezifische Probleme, kann es oft hifreich sein, mit einer Suchmaschine die entsprechenden Begriffe im Internet zu suchen. Weitere Links befinden sich auf Folie 79.

FAQs

- Frequently Asked Questions *mit* Antworten
- lokal: `/usr/(share/)doc/FAQ`
- www: `http://www.tldp.org`
- Format: ACSII und html, ...
- Antworten auf Fragen wie
  - Was ist Linux?
  - Kann ich Win* Programme unter Linux verwenden?
  - Java on Linux FAQ
  - ...

Folie 77

Ein anderer geeigneter Internetdienst ist das Usenet. In Newsgroups kann man Fragen zu seinen Problemen stellen (allerdings sollte man das erst tun, nachdem man auf den bisherigen Wegen keine Lösung gefunden hat) und bekommt meist innerhalb kürzester Zeit brauchbare Ratschläge. Ein guter Startpunkt ist hier die Newsgroup `at.linux`.

Weiters gibt es unzählige Mailinglisten zum Thema Linux. Es gibt Entwicklerlisten, Listen zu vielen Softwarepaketen und Mailinglisten zu verschiedenen Distributionen.

Last but not least sei der Support mancher Distributionen genannt, den man über www, per Telefon oder Mail in Anspruch nehmen kann und den in der überwiegenden Zahl der Fälle fachlich versierte Teams betreuen.

HOWTOs

- behandeln detailliert Aspekte der Konfiguration oder Benutzung
- lokal: `/usr/(share/)doc/HOWTO`
- www: `http://www.tldp.org`
- Format: ASCII, html, pdf, ...
- z.B.:
  - PCMCIA HOWTO
  - DOS-Win-to-Linux-HOWTO
  - Infrared-HOWTO
- mini-HOWTOs
  - MP3-CD-Burning
  - Linux+DOS+Win95+OS2

Folie 78

Internet

- Linux Documentation Project
  `http://www.tldp.org, http://www.ldp.at`
- `http://www.linux.org`
- `http://lwn.net/`
- `http://www.kernel.org, http://www.kernel.at`
- `news://at.linux`
- `news://comp.os.linux.*` (14 Gruppen, z.B. misc, Hardware,...; viel Traffic)
- diverse Mailinglisten (Kernel, Security, Distributionen...)

Folie 79

5 Editoren

Dieses Kapitel bietet einen kleinen Einstieg in das fundamental wichtige Thema „Editieren“. Wir stellen einige der unter Linux verfügbaren Editoren zusammen und geben ein Tutorial für den wichtigsten Unix-Editor, den `vi`.

Eine der allerhäufigsten und wichtigsten Aufgaben eines Benutzers—insbesondere unter einem System wie Unix—ist das Editieren von Textdateien. Das rudimentäre Beherrschen zumindest einiger Texteditoren ist auf einem System wie Unix daher einfach überlebensnotwendig. Neben den Aufgaben, die jeder Benutzer auf dem System erledigen will, gilt vor allem folgendes: praktisch die gesamte Systemkonfiguration ist in ASCII-Dateien gespeichert. Die einfachste und in gewissem Sinne effektivste Art der Konfiguration besteht im Editieren dieser Dateien.

Unter Unix stehen eine Vielzahl von Editoren sowohl für die Textkonsole (bzw. Terminal Emulation am GUI) sowie auch für die grafische Oberfläche zur Verfügung; ein Überblick befindet sich auf Folie 80. Die wichtigsten Editoren unter Linux sind Emacs und vor allem `vi` (Visual Editor). Letzterer ist als *der* Standardeditor unter *jedem* Unix-System, in *jeder* Situation verfügbar. Daher ist für den Systemadministrator die Beherrschung des `vi` Pflicht! Der `vi` ist ein einfacher aber sehr mächtiger Texteditor, der allerdings auf den ersten Blick schwierig zu bedienen ist. Nach einer Eingewöhnungsphase stellen sich aber die vermeintlich komplizierten Bedienungselemente als gut durchdachte Konzepte heraus. Es gibt kaum einen anderen Editor, der es gestattet mit weniger Aufwand komplexe Aufgaben zu erledigen.

Unter Linux wird meist der rückwärtskompatible Editor `vim` verwendet. Das Binary liegt in `/bin/` und ist daher auch im Notfall verfügbar, im Unterschied zu den meisten anderen Editoren, die in `/usr/bin` liegen. (Insbesondere wird bei tiefen Eingriffen im System sicher keine grafische Oberfläche vorhanden sein!)

Unix Editoren

- vi (Visual Editor), vim (vi Improved), gvim (grafisch)
- emacs, xemacs (GNU)
- Konsole
  - pico
  - joe
  - axe
  - zed
  - ...
- GUI
  - xedit
  - nedit
  - gedit
  - xcoral
  - kwrite
  - ...

Folie 80

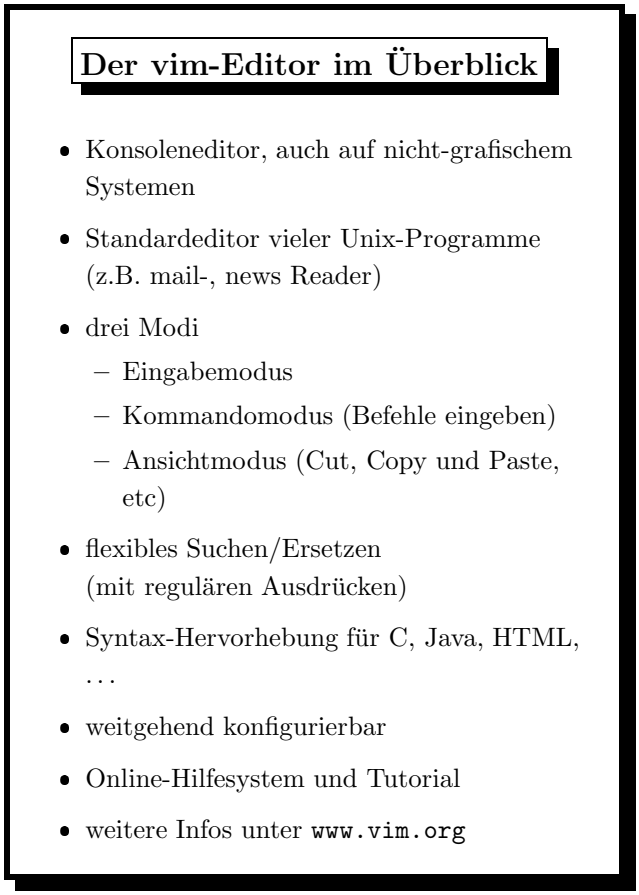
Wichtige Editoren

- vim (=vi Improved)
  - Standard-Unix-Editor
  - schnell zu bedienen
  - immer verfügbar
- (X)Emacs
  - sehr mächtiger Editor für viele Zwecke
  - Durch „Plugins“ erweiterbar
  - Texteditor, Mailreader, Programmierumgebung, ...
- pico
  - Einfach zu erlernen
  - Look and Feel von pine
- nedit
  - Motif-basiert
  - umfangreich
  - flexibel konfigurierbar

Folie 81

Meist ist unter Linux auch eine `vi`-Version für die grafische Oberfläche, der `gvim` verfügbar. Er kombiniert die Vorzüge des reinen konsolenorientierten `vi` mit den Features einer grafischen Anwendung.

Im folgenden wollen wir uns aber aus den oben angeführten Gründen dem `vi` zuwenden und einen Schnellkurs in 11 Folien absolvieren. Will man fortgeschrittenere Editiertechniken einsetzen, muß man die Arbeitsweise des `vi` zumindest grundlegend verstehen.



Folie 82

vim-Schnellkurs in 11 Folien

Grundlagen

- Starten von `vim`:
  - `vi Dateiname`
  - `vi`
- `i` ... in den Eingabemodus wechseln
- Text beliebig eingeben bzw. bearbeiten
- `ESC` ... in den Kommandomodus wechseln
- `:wq` oder `:x`
  Datei speichern und `vim` verlassen
- `:q!` ... `vim` verlassen ohne Datei speichern

Folie 83

Wechseln zwischen Modi

- vom Eingabe- oder Ansichtmodus in den Kommandomodus: `ESC`
- vom Kommandomodus in den Ansichtmodus: `v`
- vom Kommandomodus in den Eingabemodus:
  - `a` Text nach Cursor einfügen
  - `A` Text am Ende der Zeile einfügen
  - `i` Text an der Cursorposition einfügen
  - `I` Text am Anfang der Zeile einfügen
  - `o` leere Zeile unterhalb des Cursors einfügen
  - `O` leere Zeile oberhalb des Cursors einfügen
  - `s` Zeichen mit Text ersetzen

Folie 84

Bewegen in der Datei

. . . effizienter als Cursortasten . . .

- `0` Beginn der Zeile
- `$` Ende der Zeile

Bewegung über Wörter:

- `w` nächstes kleines Wort
- `W` nächstes großes Wort
- `b` voriges kleines Wort
- `B` voriges großes Wort
- `e` zum Ende des nächsten kleinen Wortes
- `E` zum Ende des nächsten großen Wortes

Bewegung über Zeilen:

- `k` Zeile rauf
- `j` Zeile runter
- `:num` gehe in Zeile `num`

Folie 85

Text bearbeiten

im Kommandomodus:

- `x` einzelnes Zeichen löschen
- `dw` bis zum Ende des Wortes löschen
- `d$` bis zum Ende der Zeile löschen
- `d0` bis zum Beginn der Zeile löschen
- `dd` ganze Zeile löschen
- `:n1,n2d` Bereich von Zeile `n1` bis `n2` löschen
- `R` in Eingabemodus wechseln und Text überschreiben
- `cw` bis zum Wortende löschen und in Eingabemodus wechseln
- jedem Kommando kann eine Zahl vorangestellt werden, die angibt, wie oft das Kommando wiederholt werden kann; z.B. `3x` löscht die nächsten 3 Zeichen.

Folie 86

Allgemeine Kommandosyntax

Kommandos im `vim` folgen der simplen Logik:

`:ZeilenAktionBereichOptionen`

alle Felder sind optional
spezielle Zeilenbereiche:

- anfang,ende (:1,3 ...)
- $ Letzte Zeile in der Datei (:1,$ ...)
- % Die gesamte Datei (:% ...)

Verfügbare Kommandos erhält man mittels

- `:help x`

Die zweite Bereichsangabe ist etwa möglich durch:

- `0` Beginn der Zeile
- `$` Ende der Zeile
- `w` bis zum Ende des Wortes

Folie 87

Dateioperationen

- `:e datei` Lädt die angegebene Datei in den Editor
- `:w [datei]` Datei schreiben; wenn eine Datei angegeben wurde, wird der Text dort abgespeichert
- `:q` `vi` beenden
- `:wq` Datei speichern und beenden
- `:x` Wenn es Änderungen gibt, Datei speichern; beenden
- `ZZ` dito

Diese Befehle sind durch `!` erzwingbar (z.B. `:q!`)

- `:!command` Externes Programm ausführen (also etwa `!ls`)
- `:rdatei` Inhalt der angegebenen Datei an die Cursorposition kopieren; (z.B. :r!ls)

Folie 88

Suchen und Ersetzen

Suchen

- `/muster` sucht angegebenes Muster vorwärts
- `n` wiederholt Suche ab der Cursorpostion, d.h. findet nächsten Treffer
- `?muster` sucht angegebenes Muster rückwärts
- `N` findet nächsten Treffer rückwärts

Ersetzen

- `:g/muster/s/neuertext`
  sucht `muster` und ersetzt es mit `neuertext`
- `:Zeilen/altertext/neuertext/g`
  sucht in den angegebenen Zeilen nach `altertext` und ersetzt es mit `neuertext`

Folie 89

Undo, Copy, Cut Paste

- `u` mache letze Eingabe rückgängig
- `Strg+R`
  rückgängig gemachte Änderungen wiederherstellen

- `yy` Kopiert die Zeile in den Buffer
- `dd` Kopiert die Zeile in den Buffer und löscht sie dann
- `p` Fügt die Inhalte des Buffers an der Cursorposition ein

Folie 90

Konfiguration

Optionen in `vi` sind durch das Kommando `set` konfigurierbar.
nützliche Einstellungen:

- `set bs=2`
- `set ai`
- `set history=50`
- `set ruler`
- `set incsearch`
- `set ignorecase`
- `set smartcase`
- `set scrolloff=3`

bequemes Setzen von Einstellungen: `~/.vimrc`

Folie 91

Beispiel `.vimrc`

```
set bs=2         " allow backspacing over everything
set ai           " always set autoindenting on
set history=50   " keep 50 lines of command line history
set ruler        " show the cursor position all the time
set incsearch    " incremental search
set ignorecase   " ignore case
set smartcase    " smart case
set scrolloff=3  " scroll offset
set cindent      " c indenting
set gdefault     " Substitute /g is always on
```

Folie 92

Onlinehilfe und Tutorial

- `F1` oder `:help`
  ruft Starseite der Online-Hilfe auf; weiters selbsterkärend
- `:help kommando` gibt Hilfe zu einem Kommando aus
- `:help TAB` Kommandline-Completion falls genauen Namen des Befehls vergessen
- `vimtutor` an der Kommandozeile startet ein Online-Tutorial (Pflichtlektüre!)

Folie 93

6 Unix/Linux benutzen 2

In diesem Kapitel erklären wir den etwas fortgeschritteneren Umgang mit der Shell, zeigen wie die Benutzer-Umgebung (Environment) konfiguriert werden kann und wie man Prozesse kontrolliert. Schließlich stellen wir noch einige weitere nützliche Tools und Utilities vor.

6.1 Shell-Variablen

Ganz allgemein sind Variablen einem Programm bekannte Namen, die Werte repräsentieren, die sich während der Laufzeit des Programms ändern können. Auch die Shell kennt verschiedene Variablen; neben Standard-Variablen, die bei der Initialisierung der Shell gesetzt werden und welche für das Arbeiten notwendig sind (z.B. `HOME, TERM, PATH ...`, siehe Folie 95) können auch benutzerdefinierte Variablen verwendet werden. Erstere werden in der Regel groß geschrieben, letztere sollten zu Unterscheidung klein geschrieben werden.

Für einen Überblick über alle aktuell in der Shell gesetzten Variablen gibt man `set` ein.

Um eine Variable zu definieren ist es notwendig, ihren Namen und ihren Anfangswert anzugeben. Man schreibt in der Shell einfach `NAME=WERT`. Will man sich die neu gesetzte Variable ansehen, kann man entweder wie oben `set`, oder aber `echo $NAME` eintippen. Es wird dann der Wert der Variablen gezeigt; `$NAME` steht also für den Wert der Variablen `NAME`.

Verwendet man eine nicht zuvor definierte Variable, so erhält man anders als in manchen Programmiersprachen keine Fehlermeldung; in diesem Fall ist der Inhalt der Variablen einfach leer. Es ist auch nicht notwendig, den Typ einer Variablen anzugeben, und er kann sich im Laufe der Zeit ändern, falls das notwendig ist.

Um eine Variable nicht nur für die aktuelle Shell sondern auch für alle *Subshells* (aus der aktuellen Shell aufgerufene Shells; insbesondere Shellscripts; siehe dazu Abschnitte 6.2, 6.3 und Teil 2) zu setzen verwendet man die Syntax `export NAME=WERT`. Der Unterschied wird im folgenden Beispiel genau demonstriert. Insbesondere muss man Umgebungsvariable, die die Ausführung eines Shellscripts beeinflussen sollen, exportieren!

- Nicht exportierte Variable sind in der Subshell nicht gesetzt:

```
[roli@pablo roli]$ xy=world
[roli@pablo roli]$ echo xy
xy
[roli@pablo roli]$ echo Hello $xy
Hello world
[roli@pablo roli]$ bash
 [roli@pablo roli]$ echo Hello $xy
 Hello
 [roli@pablo roli]$ exit
 exit
[roli@pablo roli]$ xy=
[roli@pablo roli]$ echo $xy
```

- Exportierte Variable schon:

```
[roli@pablo roli]$ export xy=world
[roli@pablo roli]$ bash
 [roli@pablo roli]$ echo Hello $xy
 Hello world
 [roli@pablo roli]$
```

Die Standardvariable `PATH` (der *Suchpfad*) besteht aus einer Aneinanderreihung von Verzeichnissen, in denen die Shell nach ausführbaren Dateien sucht. Diese können dann direkt,

Variablen

einem Programm bekannte Namen, die veränderliche Werte repräsentieren

- Shell-Variablen: Standard (groß geschrieben), benutzerdefiniert
- setzen: `NAME=WERT`
- für alle Subshells setzen: `export NAME=WERT`
- Wert abrufen: `echo $NAME`
- gesetzte Variablen ansehen: `set`
- löschen: `unset NAME` oder `NAME=`
- Standard Variablen: `PATH, HOME, PS, USER, TERM, ...`

Folie 94

Shell-Standardvariablen

typischerweise groß geschrieben

- `HOME` oder `~` ... Homedirectory
- `PS1` ... Primary Prompt
- `PS2` ... Secondary Prompt
- `PATH` ... Suchpfad
- `HOSTNAME` ... Hostname
- `HOSTTYPE` ... CPU Typ
- `LOGNAME` ... Loginname
- `USER` oder `USERNAME`
- `PWD` ... Current Directory
- `SHELL` ... Shell-Typ
- `TERM` ... Terminaltyp
- ...

Folie 95

durch Eingabe ihres Namens, aufgerufen werden, und es steht die Kommandovervollständigung für sie zur Verfügung. Zum Beispiel kann das Verzeichnis `/sbin/` dem Suchpfad wie folgt hinzugefügt werden:

```
guest@boernslaptop:~$ ifconfig
bash: ifconfig: command not found
guest@boernslaptop:~$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games
guest@boernslaptop:~$ PATH=$PATH:/sbin
guest@boernslaptop:~$ ifconfig
ath0      Link encap:Ethernet ...
```

Ein kleines Detail am Rande: Für Dateien im aktuellen Verzeichnis `.` steht die Dateinamenvervollständigung zur Verfügung, ausführbare Dateien können aber nur dann direkt aufgerufen werden, wenn sich `.` in `PATH` befindet (davon sollte man aber aus Sicherheitsgründen absehen!).

Mit dem Prompt `PS1` kann man beeinflussen, wie sich die Shell meldet. Die Bash bietet hier eine Vielzahl von Möglichkeiten (`man bash`, siehe Abschnitt Prompting). Ein kleines Beispiel:

```
boern@boernslaptop:~$ echo $PS1
\u@\h:\w\$
boern@boernslaptop:~$ PS1='\u@\h:\w \t \$ '
boern@boernslaptop:~ 18:38:06 $
```

Möchte man eine Variable nicht mehr verwenden (=löschen), verwendet man `unset` (also `unset NAME`). Ist eine Variable nicht gesetzt, so ist, wie schon oben bemerkt, ihr Wert leer; daher kann eine Variable auch mittels `NAME=` gelöscht werden. Im Unterschied zu höheren Programmiersprachen führt das Verwenden nichtdefinierter Variablen zu keiner Fehlermeldung; ob das gewünschte Resultat erzielt wird, ist aber eine andere Frage. Auch kennt die Shell nur einen einzigen Variablentyp. Variablen spielen klarerweise in der Shellprogrammierung (siehe Teil 2) eine große Rolle.

Eine sehr nützliche Funktion ist die Umleitung der Ausgabe eines Befehls in eine Variable (*Kommando-Substitution*, z.B. `now=`date``; siehe Folie **??**). Achtung: die hier eingesetzten Hochkommas sind sog. *Backticks* (von links oben nach rechts unten). So gesetzte Variablen können dann z.B. mit `echo` ausgegeben oder später (etwa in Scripts) verwendet werden.

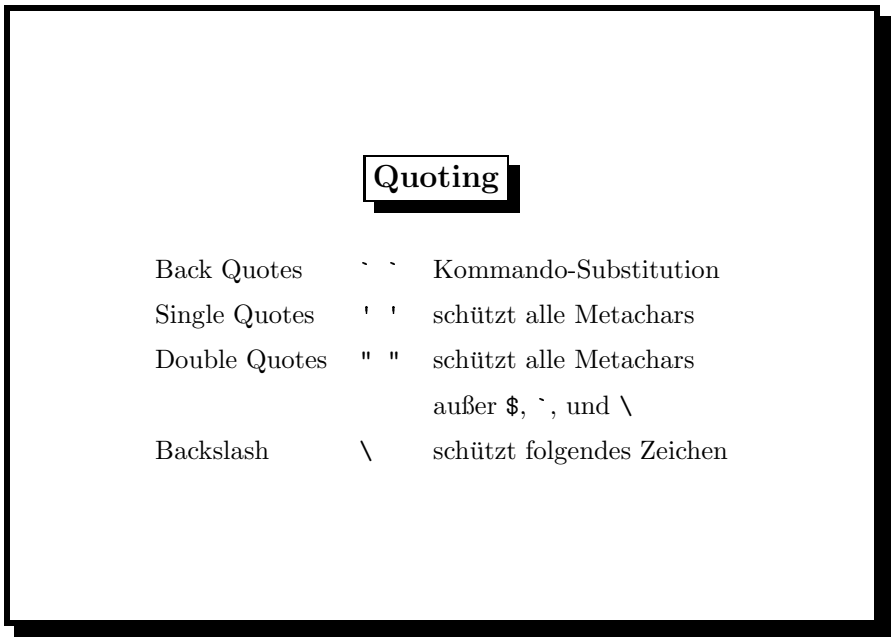
Enthalten Variablen Shell-Metacharacters (Leerzeichen, Wildcards, etc.), so müssen diese mit Hochkommas geschützt werden, wobei folgende Regel gilt: *einfache Hochkommas (Single Quotes)* schützen alle Shell-Metacharacters, *doppelte Hochkommas (Double Quotes)* schützen alle Shell-Metacharacters außer `$` (und erlauben so Variable zu verwenden), `` ` `` und `\`. Wobei der *Backslash* `\` als Metacharacter eingesetzt wird um die spezielle Funktion des folgenden Zeichens aufzuheben (siehe dazu Folie 96).

Ganz besonders wichtig ist es, daß Leerzeichen in Dateinamen geschützt werden müssen– für die Shell ist das Leerzeichen der Metacharakter, der Befehle von Optionen und Argumenten trennt. Das ist übrigens nicht nur unter Unix der Fall, fast alle Betriebssysteme zeigen dieses Verhalten wenn man sie auf der jeweiligen Konsole bedient. Natürlich ist es deswegen ratsam, *keine Leerzeichen in Dateinamen* zu verwenden. Diese Konvention wird leider oft gebrochen.

Besonders effektiv können Variablen in Schleifen eingesetzt werden. Wir geben weiter unten einige einfache, aber wichtige Beispiele. (Im zweiten Teil der Vorlesung wird dann eine kleine Einführung in die (Bash-) Shellprogrammierung gegeben.) Will man mehrere Kommandos in einer Zeile eingeben, so müssen sie durch einen Strichpunkt getrennt werden (z.B. `date; pwd`).

6.1.1 Einige Beispiele zum Quoting

- Datum durch Kommando-Substitution in eine Variable einlesen:



Folie 96

```
[roli@pablo roli]$ date
Sat Oct 20 14:30:47 CEST 2001
[roli@pablo roli]$ now=`date`
[roli@pablo roli]$ echo $now
Sat Oct 20 14:30:56 CEST 2001
[roli@pablo roli]$ date
Sat Oct 20 14:31:06 CEST 2001
[roli@pablo roli]$ echo $now
Sat Oct 20 14:30:56 CEST 2001
```

- Eine Liste mit 10 Einträgen erzeugen:

```
[roli@pablo roli]$ list=`seq 10`
[roli@pablo roli]$ echo $list
1 2 3 4 5 6 7 8 9 10
```

- * muss geschützt werden:

```
[roli@pablo DIR_1]$ xy=world
[roli@pablo DIR_1]$ echo * Hello $xy *
file1 file2 Hello world file1 file2
```

- ' schützt sämtliche Metachars:

```
[roli@pablo DIR_1]$ echo '* Hello $xy *'
* Hello $xy *
```

- " erlaubt Variablenauswertung:

```
[roli@pablo DIR_1]$ echo "* Hello $xy *"
* Hello world *
```

- \ schützt die spezielle Bedeutung des folgenden Zeichens:

```
[roli@pablo DIR_1]$  echo "Hello \$xy"
Hello $xy
[roli@pablo DIR_1]$ echo \* Hello $xy \*
* Hello world *
```

- Ähnlich mit der Kommandosubstitution:

```
[roli@pablo DIR_1]$ echo "`seq 2`"
1
2
[roli@pablo DIR_1]$ echo "'seq 2'"
'seq 2'
```

Schleifen

- Die `for` Schleife: `for i in liste; do ...; done`
  Der Anweisungsblock, der auf das `do` folgt, wird für jedes `i` in der Liste durchlaufen. Der jeweilige Wert von `i` steht wie immer unter `$i` zur Verfügung.
- Die `while` Schleife: `while (ausdruck); do ...; done`
  Der Anweisungsblock wird solange durchlaufen, wie `ausdruck` „wahr“ist (d.h. Exit-Code `0` hat). Wenn `ausdruck` einen Exit-Code ungleich `0` hat, wird die Ausführung der Schleife abgebrochen und nach dem `done` wieder aufgenommen.
- Die `until` Schleife: `until (ausdruck); do ...; done`
  Wie die `while` Schleife, nur „umgekehrt“: Die Abarbeitung der Schleife erfolgt solange `ausdruck` Exit-Code gleich `0` hat.

Folie 97

6.1.2 Einfache Schleifenkonstruktionen

Schleifen sind natürlich wichtig für Shellscripts, können aber auch oft auf der Kommandozeile gewinnbringend angewandt werden.

- Die `for` Schleife: Kann dazu verwendet werden, Kommandos auf mehrere Dateien anwenden. Syntax: `for i in liste; do ... ;done`. Ein einfacher MP3-Player:

```
[roli@pablo gianna]$ for musicfile in *.mp3; do
> mpg123 "$musicfile"
> done
```

- `n` mal durchlaufene Schleife, mit Hilfe einer Liste und einer `for` Schleife:

```
[roli@pablo DIR_1]$ ls
[roli@pablo DIR_1]$ for i in `seq 10`
```

```
> do
> touch file$i
> done
[roli@pablo DIR_1]$ ls
file1  file10  file2  file3  file4
file5  file6  file7  file8  file9
```

- `while`-Schleife, `n`-mal durchlaufen:

```
[roli@pablo ]$ i=1; n=10
[roli@pablo ]$ while ( test $i -le $n )
> do echo -n $i; i=$($i+1)
> done
12345678910
```

- `until`-Schleife, wird jetzt nur 9mal durchlaufen:

```
boern@boernslaptop:~$ i=1; n=10
boern@boernslaptop:~$ until [ $i -ge $n ];
> do
> echo -n $i
> i=$(( $i + 1 ))
> done
123456789boern@boernslaptop:~$
```

6.2 Benutzerumgebung

Unter Benutzerumgebung versteht man die spezifisch und pro User eingerichtete und konfigurierte Arbeitsumgebung, die der Benutzer am System vorfindet. Die Konfiguration der Benutzerumgebung geschieht vorwiegend durch Shell-Variablen. In diesem Abschnitt erklären wir, welche Konfigurationsdateien die Benutzerumgebung (in der Shell) beeinflussen.

Grundsätzlich ist zwischen *systemweiten* und *benutzerspezifischen* Konfigurationsfiles zu unterscheiden. Erstere befinden sich in `/etc/`, gelten für alle Benutzer und können nur vom Systemadministrator geändert werden. Benutzerspezifische Konfigurationsfiles befinden sich im Homedirectory des Users (meist als versteckte Dateien), beeinflussen nur die Konfiguration seiner Umgebung und liegen vollständig in der Verantwortung des Benutzers selbst, können also von ihm geändert werden.

Die genaue Reihenfolge der *Initialisierung* der Benutzerkonfiguration hängt teilweise von der Distribution ab. Unter RedHat wird beim Login zunächst `/etc/profile` ausgeführt, dann `$HOME/.bash\_profile`; diese rufen ihrerseits `/etc/bashrc` bzw. `$HOME/.bashrc` auf. Beim Aufrufen einer Subshell wird nur `/etc/bashrc` und `$HOME/.bashrc` ausgeführt (siehe auch Folie 98). Beim Logout wird die Datei `$HOME/.bash\_logout` abgearbeitet.

Eine weitere bequeme Konfigurationsmöglichkeit stellen die sogenannten *Aliases* dar. Sie dienen zum Abkürzen häufig verwendeter Befehle bzw. Befehlsketten. Indem man ein Alias setzt definiert man einen neuen (kurzen) Befehl, der als Abkürzung für den komplizierteren steht. Die grundlegende Syntax ist `alias shortcut='list of commands'`. Alle aktuell gültigen Aliases kann man mit `alias` anzeigen. Ein kurzes Beispiel:

```
boern@boernslaptop:~/mydir$ ls -l
total 0
-rw-r--r--  1 boern boern 0 2005-09-28 10:10 foo
boern@boernslaptop:~/mydir$ alias ll='ls -l'
boern@boernslaptop:~/mydir$ ll
total 0
-rw-r--r--  1 boern boern 0 2005-09-28 10:10 foo
boern@boernslaptop:~/mydir$
```

Benutzerumgebung

- beim Einloggen werden folgenden Dateien abgearbeitet:
  - `/etc/profile`
  - `$HOME/.bash\_profile`
  - `$HOME/.bashrc`
  - `/etc/bashrc`
- beim Ausloggen:
  - `$HOME/.bash\_logout`
- systemweite Einstellungen in `/etc/`
- benutzerspezifische Einstellungen in `$HOME`

Folie 98

Hat man z.B. `rm` als Alias für `rm -i` gesetzt, was durchaus empfehlenswert ist, so kann man das Kommando `rm` normal ausführen, wenn man den ganzen Pfad angibt; in diesem Fall wird also mit `/bin/rm` die Rückfrage, ob das File wirklich gelöscht werden soll, unterdrückt.

Seine Aliases definiert man sinnvollerweise nicht in jeder Session neu, sondern speichert sie in `$HOME/.bashrc` ab. Dann stehen sie immer zur Verfügung. Falls man seinen Suchpfad dauerhaft abändern will, so empfiehlt es sich auch, die entsprechende Definition in der `.bashrc` abzulegen.

6.3 Prozeßkontrolle

Wie wir bereits (aus Kapitel 0) wissen, ist ein Prozeß ein laufendes Programm; vom System werden ihm verschiedene Ressourcen zur Verfügung gestellt und verschiedene Rechte eingeräumt. Etwas detaillierter verfügt ein Prozeß über eine *Prozeßumgebung*; ihre wichtigsten Bestandteile sind auf Folie 99 aufgelistet, Folie 100 zeigt ein Beispiel.

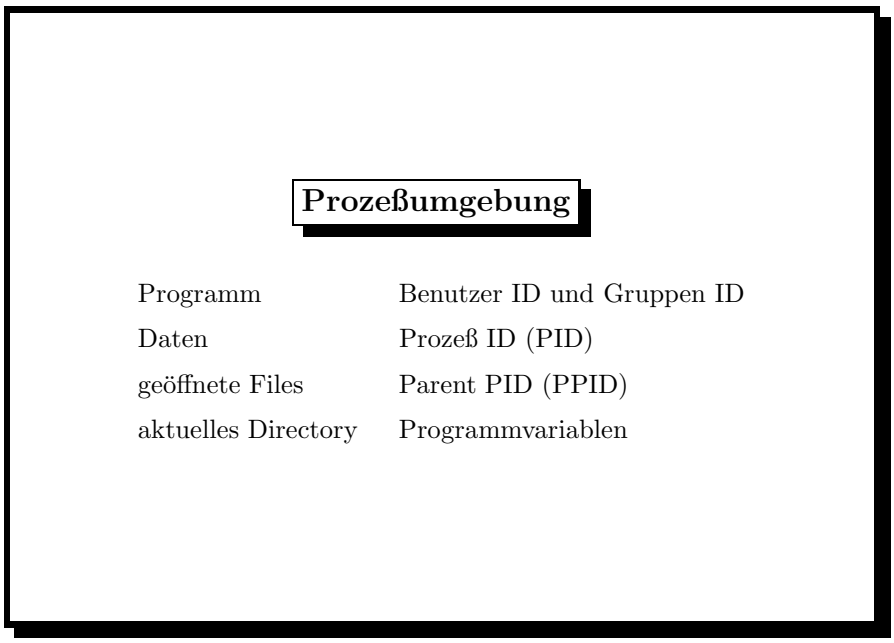
Prozesse werden systemintern über eine eindeutige Nummer, die *Prozeß ID (PID)* identifiziert, die beim Prozeßstart vom Kernel vergeben wird. Prozesse existieren in einer Eltern-Kind Hierarchie. Startet ein Prozeß einen neuen, so ist dieser sein Child-Prozeß; umgekehrt ist der Ursprüngliche Prozeß der Parent-Prozeß (wichtiges Bsp: Subshell).

Aus einer Shell kann man neue Prozesse auf verschiedene Arten aufrufen. Man kann sie im *Vordergrund* starten, so wie wir das bisher immer getan haben, wir können sie aber auch im *Hintergrund* starten; dann bleibt die Shell für weitere Eingaben frei. Sinnvoll ist das vor allem, wenn wir aus einer Shell oder einer Terminal Emulation Programme aufrufen, die dann auf einem GUI laufen (z.B. `netscape`). Die Syntax ist auf Folie 101 erklärt.

Informationen über die Umgebung laufender Prozesse können mit `ps` abgefragt werden, `ps aux` zeigt z.B. alle am System laufenden Prozesse an. Die Porzeßhierarchie kann mittels `pstree` angezeigt werden. Mit dem Kommando `kill [- signal] PID` können *Signale* an einen Prozeß geschickt werden, um diesen (vor allem—wie schon zu vermuten war—) zu beenden. Weiters können Prozesse vom Vordergrund in den Hintergrund befördert werden und vice versa. Die Folien 102, 103 geben Auskunft über die Syntax der entsprechenden Befehle.

Schließlich erwähnen wir an dieser Stelle die sogenannten Daemons (Disk And Execution MONitors); diese sind Prozesse, die dauerhaft im Hintergrund laufen und wichtige System-

funktionen erfüllen. Genauer besprechen wir Daemonen im Kapitel 9, auf Folie 104 wird ein erster Überblick gegeben.



Folie 99

Prozeßumgebung Beispiel

```
$ cat file
```

Programm:	`cat`
Prozeß ID (PID):	317
UID, GID:	franz, student
geöffnete Files:	`/dev/tty1`, `file`
Parent, PPID:	`bash`, 204
aktuelles Directory:	`/home/franz`

Folie 100

Prozesse starten

- im *Vordergrund:* „normales“ Ausführen von der Kommandozeile; blockiert Shell
  z.B. `$ ls`
- im *Hintergrund:* zusätzliches `&` am Ende des Kommandos; gibt Shell wieder frei

  ```
  $ xterm &
  [1] 417
  $ _
  ```

  `[1]` ist die (Job-)Nummer des im Hintergrund gestarteten Prozesses, `417` die ID des Prozesses.

Folie 101

Prozesse kontrollieren

Beenden:

- `Strg+c` beendet einen Vordergrundprozeß (= schickt ein Interrupt-Signal)
- `$ kill [-signal] PID` beendet einen Vorder- oder Hintergrundprozeß
  Signale:
  - `1` "Hangup": erhält einen Childprozeß, wenn sein Parent stirbt
  - `2` "Interrupt": Der Interrupt-Key (Strg+c) wurde gedrückt
  - `3` "Quit": Der Quit-Key (Strg+\) wurde gedrückt
  - `9` "Software terminate": Das mächtigste Signal; kann nicht ignoriert werden
  - `15` ist der Default-Wert
- `$ killall [-signal] name` beendet alle Prozesse mit dem angegebenen Namen

Folie 102

Prozesse kontrollieren 2

- Für Prozesse, die länger dauern und auch nach dem Ausloggen des Besitzers weiterlaufen sollen, empfiehlt sich `nohup`. z.B. listet `$ nohup ls -R / >out \&` rekursiv alle Unterverzeichnisse im Hintergrund auf und schreibt die Ausgabe in die Datei `out`.
- In der `bash` gibt es weitere Möglichkeiten zur Prozeßkontrolle:
  - `Strg+z` stellt den aktuellen Vordergrundprozeß in den Hintergrund
  - `$ jobs` gibt alle Prozesse ("Jobs") aus, die im Hintergrund laufen oder unterbrochen wurden. `%jobnr` kann auch etwa an `kill` übergeben werden.
  - `$ fg` stellt einen unterbrochenen Prozeß in den Vordergrund
  - `$ bg` stellt einen unterbrochenen Prozeß in den Hintergrund

Folie 103

Dämonen

Ein Dämon (Daemon, Disk And Execution MONitor) ist ein nicht-endender Prozeß, meistens ein Systemprozeß, der eine Systemresource (z.B. die Druckerwarteschlange) überwacht. Übliche Dämonen:

- `crond` führt in regelmäßigen Intervallen Programme aus
- `atd` führt einmalig an einem bestimmtem Datum und zu bestimmter Uhrzeit ein Programm aus
- `lpd` überwacht die Druckerwarteschlange
- `syslogd` schreibt Meldungen des Kernels und von Programmen nach `/var/log/messages`
- siehe Kapitel 9 und Teil 2

Folie 104

6.4 Weitere Tools & Utilities, Einfache Netzwerkbefehle

Zum Abschluß diese Kapitels stellen wir einige weitere wichtige Unix-Tools und Utilities vor und erklären auch die einfachsten Netzwerkbefehle.

Eine wichtige Aufgabe sowohl für den Benutzer als auch den Administrator ist das Archivieren, Bewegen und Komprimieren nicht nur einzelner Dateien sondern ganzer Verzeichnisbäume. *Das* Programm zum Archivieren unter Unix ist `tar`, was für *Tape Archive* steht und auf das beträchtliche Alter diese Werkzeugs hinweist. Unter Linux ist meist die GNU-Version von `tar` verfügbar. Die Grundfunktionsweise von `tar` ist es einen ganzen Verzeichnisbaum in eine einzige—Tarfile genannte—Datei zu „verpacken“, bzw. ein solches wieder zu entpacken. Die Dateiattribute wie Besitzer, Datumstempel etc. bleiben dabei erhalten. Daher eignet sich `tar` auch hervorragend zum Kopieren großer Date(ie)nmengen.

Die grundlegende Syntax (`tar` benötigt kein `-` vor den Optionen; es kann aber optional auch angegeben werden) ist `tar cf tarfile.tar filestotar` (`c`=create, `f`=file) für das Archivieren bzw. `tar xf tarfile.tar` für das extrahieren (`x`=extract). Die Standardextension für Tarfiles ist `.tar`. Das Tar-Programm bietet auch die Möglichkeit mittels der Option `z` resp. `j` mit `gzip` resp. `bzip2` (siehe unten) komprimierte Tarfiles zu erzeugen/extrahieren; diese haben dann die Standardendung `.tar.gz` oder `.tgz` bzw. `.tar.bz`. Weitere Details befinden sich auf Folie 105.

`tar` (Tape Archive)

wichtige Optionen

- `c` ... create
- `x` ... extract
- `v` ... verbose
- `z` ... compress mit `gzip`
- `j` ... compress mit `bzip2`
- `f` ... File (statt Tape)
- `t` ... liste Inhalt eines Tarfiles

Beispiele:

- `tar cvzf tarfile.tgz directory/`
  archiviert `directory/` in `tarfile.tgz`
- `tar xvzf tarfile.tgz`
  entpackt `tarfile` in `./`
- `tar cvfML /dev/fd0 1440 directory/`
  erstellt Multivolume-Tarfile auf Diskette mit Größe jeweils 1.4MB

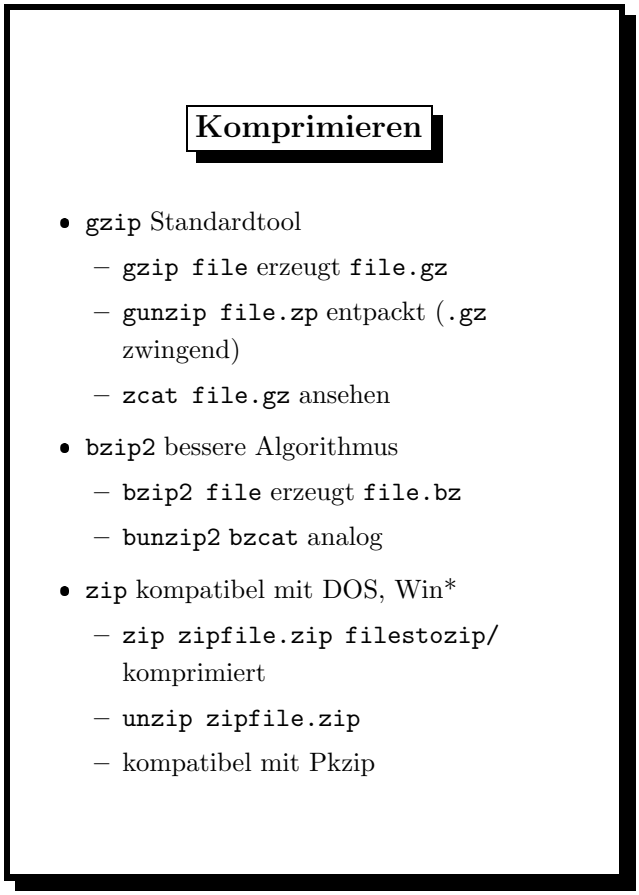
Folie 105

Bereits im Zusammenhang mit `tar` haben wir zwei Kompressionsprogramme angesprochen; `gzip` und `bzip2`. Ersteres benutzt den Lempel-Ziv Algorithmus um die Dateigröße zu reduzieren. `gzip file.ext` ersetzt die Datei `file.ext` durch die komprimierte Datei `file.ext.gz`; mittels der Option `-v` erfährt man, um wieviel Prozent die Datei komprimiert werden konnte. `gunzip` dekomprimiert die Dateien wieder—allerdings nur, falls die Endung `.gz` lautet. Komprimierte Dateien können mit `zcat` angezeigt werden.

Einen stärkeren Komprimierungsalgorithmus bietet `bzip2`. Die Verwendung ist analog zu `gzip`; die Standardextension für `bzip2`-komprimierte Dateien ist `.bz`, analog zu `gzip` gibt es die Kommandos `bunzip2` und `bzcat`.

Weiters ist auf (fast) allen Linux Systemen das auch unter DOS und Win* verfügbare `zip` vorhanden (kompatibel mit Pkzip), das die etwas unterschiedliche Syntax `zip zipfile.zip filestozip` aufweist und die Features von `tar` mit der Komprimierung verbindet. Dekomprimieren funktioniert hier mit `unzip zipfile.zip`. Kompressionsprogramme sind auf Folie 106 zusammengestellt.

Zum Schluß diese Abschnitts erklären wir noch die Funktionsweise der zur Textmanipulation nützlichen Tools `cut` und `sort`, sowie der Filter `grep`, `sed` und `awk` (Folien 107, 108) und besprechen weitere Befehle zum Ansehen von Dateien (`tail` und `head`; Folie 109) und das Suchen von Dateien und Kommandos auf den Folien 110 und 111. Schließlich befaßt sich Folie 112 mit einfachen Netzwerktools.



Folie 106

Textmanipulation

- `cut -f(elder) -d(trennzeichen) datei` oder
  `cut -c(zeichen) datei` schneidet einen bestimmten Bereich aus jeder Zeile der gegebenen Dateien.
  `$ cut -f1 -d: /etc/passwd` gibt die Benutzernamen des Systems
  `$ ps | cut -c-5,20-` gibt die PID und den zugehörigen Prozeß aus der Prozeßliste aus
- `grep string datei` sucht die Zeichenkette in der Datei und gibt jeden Treffer aus. Mehrere nützliche Optionen:
  - `-v` gibt Zeilen aus, die *nicht* den String enthalten.
  - `-n` zusätzlich Angabe der Zeilennummer
  - `-i` ignoriert Groß/Kleinschreibung

Folie 107

Textmanipulation 2

- `sort -t(trennzeichen) +feld -optionen datei` sortiert die Zeilen der Datei nach dem gegebenen Muster.
  Beispiel: `sort -t/ +5 /etc/shells` sortiert die Shells in der Datei.
  Einige Optionen:
  - `-d` wie im Wörterbuch sortieren (nur Buchstaben, Zahlen und Leerzeichen berücksichtigen)
  - `-r` Sortierung umdrehen
  - `-n` Numerische Felder arithmetisch sortieren
- `sed`, `awk` und `perl`: Sehr umfangreiche Skriptsprachen; zur Verarbeitung schwierigerer Strukturen notwendig
  Beispiel: `sed -n xp datei` gibt die x-te Zeile der Datei aus

Folie 108

Dateien ansehen

Neben `more`, `less` und Co gibt es noch andere Möglichkeiten, Dateien anzusehen:

- `head [-n(num)] datei` zeigt die ersten Zeilen der Datei an.
- `tail [-n(num)|`n(num)] datei+ zeigt die letzten Zeilen der Datei an; wenn `n` positiv ist, ab der Zeile n. Äußerst nützlich bei `tail` ist die Option `-f`, mit der der Dateiinhalt laufend neu eingelesen wird, was z.B. bei der Betrachtung von Log-Dateien vorteilhaft ist.
  Beispiel: `tail -f /var/log/messages`
- Mit `gzip` komprimierte Dateien kann man, ohne sie zu entpacken, mit `zcat datei` ansehen.

Folie 109

Befehle finden/Dateitypen ansehen

- `type befehl` bzw. `which befehl` zeigt an, wo im Pfad sich `befehl` befindet.
  ```
  $ type ls
  ls is /bin/ls
  ```
- `whereis befehl` gibt den Pfad zur ausführbaren Datei und die Manpage dazu an.
  ```
  $ whereis ls
  ls: /bin/ls /usr/man/man1/ls.1
  ```
- `file datei` versucht herauszufinden, um welchen Dateityp es sich handelt.
  ```
  $ file bild.jpg
  bild.jpg: JPEG image data
  ```

Folie 110

Suchen nach Dateien

- Mit `find verzeichnis -name datei` kann man Dateien (und Verzeichnisse) in bestimmten Verzeichnissen suchen, dabei dürfen auch Wildcards verwendet werden.
  z.B. `find /usr/include -name *io.h`
  Man kann auf die Treffer mit `-exec cmd` auch gleich Befehle ausführen:
  `find .` name c* -exec ls -l +-
  Ein simples Säuberungskommando:
  `find $HOME` name *.bak -ok rm +-
- `locate datei` durchsucht die regelmäßig mit `updatedb` erstellte Dateidatenbank und gibt alle Dateien zurück, die den Dateistring enthalten
  z.B. `locate libc.so`

Folie 111

Netzwerkbefehle

- `ping host` prüft Erreichbarkeit des angegebenen Hosts
- `traceroute host` verfolgt, über welche Router Netzwerkpakete an den Host geschickt werden
- `telnet host` baut eine Login-Verbindung zu einem Host auf; sollte aus Sicherheitsgründen nicht verwendet werden, da es Paßwörter im Klartext überträgt; Alternative `ssh`
- `ssh -l user host` (Secure Shell) baut eine sichere (verschlüsselte) Verbindung zum Host auf
- `ftp host` initiiert eine FTP Verbindung; dient zur Übertragung von Dateien; nicht-anonymes Ftp hat dieselben Schwächen wie `telnet`

Folie 112

7 X-Window

In diesem Kapitel besprechen wir detailliert die Funktionsweise der Standard-Unix grafischen Benutzeroberfläche (Graphical User Interface, GUI), dem X-Window-System.

Das *X-Window-System* (`http://www.x.org`; nicht X-Windows!) wurde 1984 am MIT vom X-Consortium entwickelt. Es ist nicht GPL lizenziert, wird aber unter einer ähnlichen Lizenz (X11-License) vertrieben. Unter der selben Lizenz wird der ursprünglich für 80x86 Prozessoren entwickelte *XFree86* (`http://www.xfree86.org`) vertrieben. Dieser wird wegen einem Problem mit einer Lizenzänderung nun langsam durch den Xorg-Server als das Standard-X-System unter Linux abgelöst.

Das X-Window-System ist modular aufgebaut; es benutzt eine Client/Server-Architektur. Der X-Server verarbeitet alle einkommenden Events (Tastatur, Maus,...) und leitet diese an die entsprechenden Anwendungen weiter. Ebenso kümmert sich der X-Server um die Ausgabe der Anwendungen (hier Clients genannt) auf dem Display. Der X-Server übernimmt also die Kommunikation mit der Hardware auf der einen Seite und mit den Applikationen auf der anderen Seite. Er läuft im Userspace aber mit root-Privilegien.

Ganz allgemein ist in einer Server/Client Architektur der Server immer der Teil, der eine Ressource zur Verfügung stellt, und der Client nimmt sie vom Server in Anspruch. Im Falle von X ist die Ressource die Ein/Ausgabe-Infrastruktur des Rechners; das führt immer wieder zur Verwirrung, da der X-Server lokal läuft, X-Clients aber oft auf der grossen Kiste im Rechenzentrum laufen. Im Falle eines Desktoprechners freilich koexistieren X-Clients und der X-Server in derselben Maschine.

Unter allen Anwendungen, die sich mit dem X-Server in Verbindung setzen (X-Clients) gibt es eine besonders wichtige: den *Window-Manager*. Dieser ist für das Handling der Fenster auf dem Display verantwortlich. Er erzeugt die Rahmen der Fenster und ist für ihr Verschieben, Vergrößern, Verkleinern etc., also für das Look and Feel der grafischen Oberfläche zuständig. Es gibt eine Vielzahl von Window-Managern; für eine kleine Übersicht empfehlen wir `http://www.xwinman.org`. Eine Weiterentwicklung der Window-Manager hin zum gesamten Oberflächenmanagement stellen die grossen Desktops `GNOME` und `KDE` dar. Man beachte, daß es durchaus möglich ist, während einer X-Session den Window-Manager zu stoppen und (unter Umständen einen anderen) wieder zu starten!

Die Konfiguration des X-Window-Systems zerfällt in mehrere Teile. Das Konfigurationsfile des X-Servers ist `/etc/X11/XF86Config-4` bzw. `/etc/X11/xorg.conf` und kann nur von root bearbeitet werden; dort wird die Konfiguration für die relevante Hardware in einzelnen Sektionen durchgeführt (Keyboard, Maus, Grafikkarte, Monitor, ...) und wichtige globale Einstellungen (Pfade, ...) vorgenommen. Wichtig ist die Konfiguration des Monitors, insbesondere die horizontale und vertikale Bildwiederholfrequenz; überschreitet der Konfigurationseintrag die tatsächlichen Monitorkapazitäten, so kann dieser **schwer beschädigt werden**. Viele Treiber überprüfen allerdings inzwischen selber die Bildwiederholraten, sodaß diese Gefahr nicht mehr so stark besteht. Die wichtigste Sektion ist die „Screen“ Sektion; hier wird die Auflösung und Farbtiefe festgelegt. Das Konfigurationsfile sollte bei der Installation richtig angelegt werden, kann aber später mit mehreren Tools, die auch die Grafikkarte prüfen und die Konfiguration testen können, verändert (nachjustiert) werden.

Darüberhinaus gibt es (analog zur Shell) systemweite und benutzerspezifische Konfigurationsfiles, die das Aussehen des GUI bestimmen; Details finden sich auf Folie 119. Selbstverständlich kann jeder Benutzer (soweit installiert) seinen Lieblings WM verwenden und dessen Aussehen weitgehend konfigurieren. Dies geschieht über Konfigurationsfiles im Homedirectory des Users. Je nach verwendeter Software sind folgende Dateien auschlaggebend: `.xsession`, `.xinitrc`, `.Xauthority`, `.Xdefaults`, `.wm_style`, und viele andere mehr. Die meisten modernen Window-Manager können jedoch auch bequem über ein Menü konfiguriert werden.

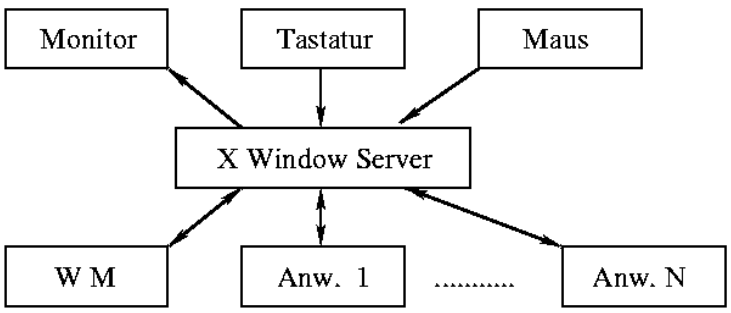
Je nach Konfiguration (siehe Runlevels, Kapitel 9), kann das System ein grafisches Login anbieten. Meldet man sich auf dieser grafischen Oberfläche an, so kommt man direkt ins X-Window-System. Bietet das System nur einen textbasierten Login (Textkonsole) an, so kann von der Konsole X-Window mit dem Befehl `startx` gestartet werden. Achtung: per Default läuft immer nur eine grafische Oberfläche!

Das X-Window-System

- nicht X-Windows!
- netzwerkbasiertes grafisches System
- Entwickelt im MIT 1984
- Frei verfügbar
- Aktuelles Release X11R6
- Xorg und XFree86 sind aktuelle X-Window-Implementationen unter Linux
- Erlaubt das Ausführen grafischer Anwendungen
- Definiert Protokoll zur Kommunikation mit (grafischen) Ein/Ausgabegeräten
- Client/Server-Architektur; beliebig viele (auch Remote-) Clients verbinden sich mit dem Server

Folie 113

X-Window-Architektur



Folie 114

X-Server

- Standard für Linux: *XFree86*, *Xorg*
- Open Source
- modulare Treiber; Treiber für Grafikkarten auch von Drittanbietern (nvidia, ATI,. . .)
- `http://x.org`, `http://www.xfree86.org`
- ein alternativer X-Window-Server für Linux: Xi Graphics `http://www.xig.org`

Folie 115

X-Clients

- Anwendungen, die unter X laufen
- starten von Kommandozeile oder spezielle Startskripts
- wichtigster Client: Window-Manager:
  - Fensterverwaltung (Verschieben und Vergrößern/Verkleinern)
- `xterm` emuliert Terminal, zur Eingabe von Befehlen
- `xeyes` zeigt zwei Augen an, die dem Mauszeiger folgen
- `xcalc` ein Taschenrechner
- `xedit` ein grafischer Editor
- `xwd` macht einen Screenshot des X Bildschirms
- . . .

Folie 116

Window-Manager

- `twm` Standard XFree86 Window-Manager; wenig Komfort
- `fvwm(95)` gute Konfigurierbarkeit und Geschwindigkeit, sieht Windows ähnlich
- `enlightenment` sehr schönes Aussehen, ressourcenaufwendig
- `sawmill` Resourcen-sparsam, gut konfigurierbar
- `windowmaker`, `olvm`, `icewm`, `fluxbox`, ...

Folie 117

Desktops

- Weiterentwicklung der WM
- Oberflächenmanagement
- weit verbreitet, einheitliches Look'n'Feel
- standardisierte Anwendungen und Kommunikationsprotokolle (z.B. für Drag'n'Drop)
- GNOME (Gnu Network Object Model Environment):
  - baut auf C und GTK auf
  - viele Anwendungen verfügbar, gut in die meisten Distributionen integriert
- KDE (K Desktop Environment):
  - in C++ mit dem Toolkit Qt geschrieben, modular und objektorientiert
  - viele Standardprogramme z.B. für WWW, Mail, News, MP3, Office ...

Folie 118

X Konfiguration

- X-Server
  - nur root
  - `/etc/X11/XF86Config`, `/etc/X11/xorg.conf`
  - viele Tools verfügbar
- WM, Desktop
  - benutzerspezifisch
  - `.xsession`, `.Xdefaults`, `.xinitrc`, ...
  - Menü des WM/Desktops

Folie 119

X starten

- Von der Textkonsole
  - Einloggen
  - `startx`
  - Führt Befehle in `/etc/X11/xinit/xinitrc` und `~/.xinitrc` aus
- Grafisches Login
  - Über Display-Manager (XDM, KDM, GDM, ...) einloggen
  - Führt Befehle in `/etc/X11/xdm/Xsession` und `~/.xsession` aus
- X verlassen
  - Menü (Logout)
  - killen des X-Servers `<ctrl><alt><backspace>`

Folie 120

Eine große Stärke des X-Window-Systems beruht auf seiner Client/Server-Architektur. Das X-Window-System ist „von Haus aus“ netzwerkfähig und es ist sehr einfach, Programme remote auszuführen, d.h. X-Clients nicht am lokalen Rechner laufen zu lassen, sondern die Fenster über ein Netzwerk an den X-Server am lokalen Rechner zu „schicken“. Das ist vor allem dann von großem Vorteil, wenn ein starker Rechner (Applikationsserver, auf dem die X-Clients laufen) im LAN vorhanden ist, währen die Arbeitsplatzrechner (Workstation, auf ihr läuft der X-Server) nur über schwächere Hardware verfügen.

Nachdem X-Clients grossen Einfluss auf das Verhalten des Gesamstsystems ausüben können, liegt es auf der Hand, daß sich die Clients beim Server als zugriffsberechtigt ausweisen können müssen. X sieht für diese Authentifizierung mehrere Möglichkeiten vor. Die beiden gebräuchlichsten sind die hostbasierte Authentifizierung mittels `xhost` und die benutzerbasierte Authentifizierung mittels `xauth`, daß Zugriffsschlüssel (meist `MIT-MAGIC-COOKIE-1`) in der Datei `.Xauthority` verwaltet.

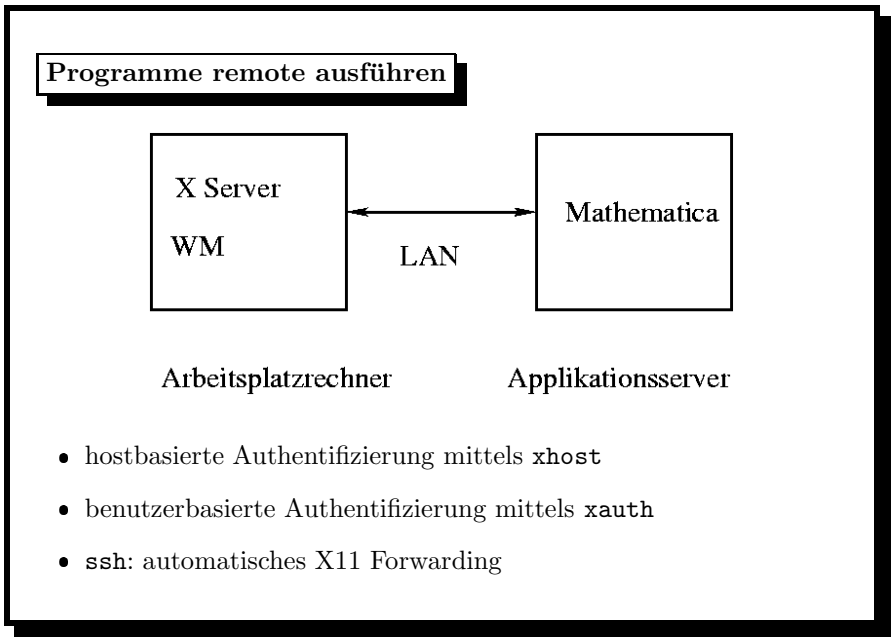
Wir stellen hier zunächst die hostbasierte Authentifizierung mittels `xhost` dar. Eine kleine Vorbemerkung: die Verbindung ist auf jeden Fall unverschlüsselt, und die Authentifizierung erfolgt auf Basis der IP-Adresse! Damit ist diese Authentifizierung von Haus aus so unsicher, daß man sie eigentlich nur in „sicheren“ LANs anwenden sollte. Konkret geht man wie folgt vor.

1. Zuerst muß man dem X-Server mitteilen, daß er X-Client-Verbindungen von anderen Rechnern (also X-Clients, die auf anderen Systemen laufen) entgegennehmen darf. Dazu dient das Kommando `xhost` (Syntax: `xhost [+|-] [hostname]`). Damit kann man entweder gezielt einzelnen Rechnern den Zugriff auf den lokalen X-Server ermöglichen oder verbieten oder gleich generell allen erlauben (nicht empfohlen!) oder untersagen, X-Fenster an den X-Server zu „schicken“ (`xhost [+|-]`).

2. Am Remote-System, wo die Applikation (der X-Client) laufen soll, muß angegeben werden auf welchem Display (X-Server) das Fenster erscheinen soll. Dafür gibt es die Shell-Variable `DISPLAY`, die man—nach dem Einloggen (etwa über Telnet) am Applikationsserver—auf den Hostnamen (oder die IP) des Rechners mit dem X-Server setzt (z.B. `export DISPLAY=xserver:0.0`, letzteres gibt die Screen- und X-Servernummer an). Nun kann man X-Programme (wie etwa `xeyes, xterm, \dots`) am Applikationsserver ausführen, die Ausgabe erfolgt jedoch am lokalen X-Server. Dies hat den Vorteil, daß man z.B. für umfangreiche Berechnungen `mathematica` am Server rechnen läßt, sich die Ergebnisse aber am langsameren Arbeitsplatzrechner anzeigen lassen kann.

   Will man nur für einzelne X-Programme die Ausgabe umleiten, kann man das dem Programm auch direkt durch die Option `-display` mitteilen (z.B. `xterm -display xserver:0.0`)

Wir haben schon oben bemerkt, daß X-Verbindungen unverschlüsselt und unsicher sind. Es stellt für einen Angreifer (eigentlich) überhaupt kein Problem dar, eine X-Verbindung zu übernehmen (Session-Hijacking), zu missbrauchen oder zu korrumpieren. Es ist deswegen sicherer, X nur über eine verschlüsselte Verbindung zu verwenden. Die Secure Shell `ssh` kann selbständig die Aufgabe des X11-Forwardings übernehmen (mit Hilfe der Option `-X`, sodaß das Einloggen per `ssh -X user@host` ausreicht, um sich X-Clients schicken lassen zu können. `ssh` beherrscht auch die benutzerbasierte Authentifizierung (aber nur die mittels `MIT-MAGIC-COOKIE-1`). Das ganze funktioniert so, daß der `ssh`-Server einen Dummy-X-Server am Zielhost einrichtet, und die entsprechenden Umgebungsvariablen setzt. Er vergibt auch einen Zugriffsschlüssel (allerdings nicht denselben, der auf dem X-Server-Host verwendet wird; der vom `ssh`-Server vergebene Schüssel ist nach Ende der `ssh`-Session wertlos, der Originalschlüssel bleibt immer am `ssh`-Client-Host).



Folie 121

8 Installation

In diesem Kapitel widmen wir uns der Installation eines Linux-Systems. Wir erklären, was vor der Installation zu bedenken und zu planen ist und warum das Dokumentieren der Installation wichtig ist. Verschiedene Installationsvarianten werden vorgestellt und die prinzipiell durchzuführenden Schritte besprochen. Als Beispiel (einer einfach zu installierenden Distribution) gehen wir eine RedHat-Installation im Detail durch. Damit sollte es leicht möglich sein, unter Zuhilfenahme der distributionsspezifischen Dokumentation die xy-Distribution erfolgreich zu installieren.

War die Installation eines Linux-Systems vor einigen Jahren manchmal noch ein kleines Abenteuer mit unbestimmtem Ausgang (vor allem wenn die Hardware etwas exotischer war), so haben in der Zwischenzeit viele Distributoren viel Aufwand betrieben, um die Installation so einfach wie möglich zu gestalten. Viele moderne Linux-Distributionen verfügen über ein komfortables (oft grafisches) menügesteuertes Installationstool (*Installer*), das praktisch ,,von jedermann" bedienbar ist. Trotzdem ist für den Systemadministrator und den interessierten Benutzer einiges an Hintergrundwissen nötig, um die vollen Konfigurationsmöglichkeiten einer Linux-Installation ausnützen zu können. Genau das soll hier vermittelt werden.

In den Übungen wird es Gelegenheit geben, die Installation eines RedHat-Linux-Systems praktisch durchzuführen.

8.1 Vorbereitung und Planung

Die Bedeutung der Planung und Vorbereitung für eine erfolgreiche Installation kann gar nicht überschätzt werden. Langjährige Erfahrung zeigt, daß eine gut durchdachte Planung später viel an Konfigurationsarbeit (oder sogar eine Neuinstallation) ersparen hilft. Es lohnt sich allemal, sich vor Beginn der eigentlich Installation (*vor dem Einschalten des PC!*) in Ruhe hinzusetzen und Bleistift und Papier zur Hand zu nehmen. Dies trifft vor allem dann zu, wenn eine spätere Neuinstallation dann wieder rasch von der Hand gehen soll, oder wenn der Installationsvorgang auf einer grösseren Menge PCs durchgeführt werden muss!

Als erste Entscheidung vor der Installation steht die *Wahl einer Linux-Distribution.* Wir haben bereits in Abschnitt 1.4 die Sinnlosigkeit der Frage nach der *besten* Distribution diskutiert. Die Entscheidung wird sinnvollerweise z.B. von folgenden Faktoren abhängen: (Haupt)-Aufgabenbereich des Systems, vorhandene Hardware, Anzahl und Erfahrenheit der

Folie 122

Benutzer, Verfügbarkeit von Dokumentation, Support, und Sicherheitsupdates, Vorkenntnisse und Vorlieben des Administrators, und einigen anderen mehr.

Hat man sich für eine Distribution entschieden, so stellt sich als nächste Aufgabe, die entsprechenden *Installationsmedien* zu beschaffen. Diese hängen allerdings von der Wahl der *Installationsvariante* ab. Im Wesentlichen gibt es zwei Möglichkeiten, Linux zu installieren; entweder lokal von einer CD-Rom oder einer Festplattenpartition oder über ein Netzwerk. Letzteres setzt allerdings eine adäquate Netzwerkanbindung voraus; eine Netzwerkinstallation über Modem ist nicht möglich, über Telekabelanschluß nur teilweise empfehlenswert. Allerdings kann man bei der Netzwerkinstallation davon ausgehen, daß man immer die aktuellste Variante der Software installiert; installiert man von CD oder DVD, so muss man nachher oft einiges an Updates einspielen, um die Software zu aktualisieren. Details über Installationsvarianten und die Beschaffung der Installationsmedien finden sich auf den entsprechenden Folien.

Als nächster wichtiger Schritt stellt sich die Aufgabe, die verwendete *Hardware genau zu kennen*. Das ist einerseits wichtig für die Planung des System-Layouts (z.B. sollte man vor der Planung der Partionierung der Festplatte über deren Größe Bescheid wissen), andererseits zur Optimierung der Konfiguration, sollte der Installer die Hardware nicht richtig erkennen. Insbesondere Kenntnis der Grafikkarte und der Spezifikation des Monitors sind sehr wichtig. Schließlich sollte man sich, am besten schon vor dem Kauf der Hardware, auf der Hardware-Compatibility-Liste (am besten der gewählten Distribution) der Unterstützung für das entsprechenden Produkt versichern.

Wie kommt man zur benötigten Information über die Hardware? Wichtigste Quelle ist die mitgelieferte Dokumentation. Ist diese entweder nicht (mehr) vorhanden oder unvollständig, so kann man für verschiedene Komponenten (Prozessor, Festplatte, ...) die nötige Information aus dem BIOS beziehen oder im Internet die Homepage des entsprechenden Herstellers besuchen. Zu bemerken ist hier, daß Produkte, die gleich heissen, nicht mit identischer Hardware bestückt sein müssen! Oft produziert ein Hersteller mehrere Serien „desselben“ Produkts, die aber mit unterschiedlichen Chips bestückt werden. Ob ein Treiber verfügbar ist oder nicht hängt aber natürlich ganz zentral von der tatsächlich verwendeten Hardware ab.

Ist bereits ein Betriebssystem vorinstalliert, so können die entsprechenden Tools verwendet werden (z.B. Systemsteuerung in MS-Windows). Selbstverständlich besteht auch immer die Möglichkeit, einen Schraubenzieher zur Hand zu nehmen, die entsprechenden

Installationsvarianten

- lokale Installation
  - CD-Rom
  - Festplatte
- Netzwerkinstallation
  - Nfs
  - Http
  - Anonymes Ftp
- Bootmedium
  - CD-Rom, DVD
  - Boot-Diskette (eventuell verschiedene je nach Installationsart)

Folie 123

Linux Quellen

- Homepage der Distribution (`www.linux.org/dist/index.html`, `www.distrowatch.com`)
- lokale Mirrorserver
  - `ftp.univie.ac.at` Ftp-Server der Uni Wien
  - `gd.tuwien.ac.at` Ftp-Server der TU Wien (Goodie Domain)
- CD-Rom kaufen/runterladen (Iso-Images) und brennen

Folie 124

Komponenten auszubauen und direkt auf der Platine (z.B. Grafikkarte, Netzwerkkarte) die Herstellerinformation zu lesen.

Kenne deine Hardware

- Prozessor und RAM (Swap-Partition)
- Festplatte (Platzbedarf, Partitionierung)
- SCSI-Adapter (Treiber)
- Netzwerkkarte (Treiber)
- Tastatur (dt./engl.)
- Maus (seriell, PS2, USB, 2/3 Tasten)
- Grafikkarte (Chip, Speicher)
- Monitor (vertikale und horizontale Bildwiederholraten)

Hardware-Compatibility-Liste!!!

Folie 125

8.2 Überblick über die Installation

Hier wird erklärt, welche Schritte prinzipiell bei der Installation eines Linux-Systems durchzuführen sind. Die genaue Reihenfolge hängt teilweise von der Distribution bzw. der gewählten Installationsmethode ab. Ein Überblick über die Installationsschritte befindet sich auf Folie 126.

Zunächst erklären wir den wichtigen Punkt der „Partitionierung“ der Festplatten des Systems. Auf Intel-basierten Systemen werden die Festplatten in verschiedene logische Teile, sogenannte *Partitionen*, geteilt; diese können dann vom Betriebssystem wie eigene Festplatten angesprochen werden.

Etwas detaillierter besteht jede Festplatte aus einem *Master Boot Record* (erster Sektor), einer *Partitionstabelle* (zweiter Sektor) und dem *Datenbereich*. Dieser Datenbereich kann in bis zu vier *primäre* Partitionen aufgeteilt sein. Eine davon kann eine sogenannte *erweiterte* (extended) Partition sein, die wiederum bis zu 12 *logische* Partitionen enthalten kann. Jede primäre, nicht erweiterte sowie jede logische Partition kann ein eigenes *Filesystem* enthalten, das jeweils von verschiedenem Typ (FAT16, FAT32, ext2, ...) sein kann. Es können sogar verschiedene Betriebssysteme auf den verschiedenen Filesystemen installiert sein. Die einzige Einschränkung ist, daß einige Betriebssysteme von primären Partitionen booten wollen.

Die erste IDE-Festplatte (oder das erste IDE-Gerät, es kann sich etwa auch um ein CD-Rom Laufwerk handeln), d.h. das Master-Device am ersten IDE-Controller wird unter Linux `/dev/hda` genannt, die zweite (Slave am ersten IDE-Controller) `/dev/hdb` usw, während die SCSI und SATA Gräte mit `/dev/sda`, `/dev/sdb` usw. bezeichnet werden. Die vier primären Partitionen auf einer Festplatte heißen `/dev/?d?[1-4]`, die logischen Partitionen werden mit `/dev/?d?5` usw. bezeichnet.

Unter Linux können Partitionen mit `fdisk`, einem sehr mächtigen und stabilen Werkzeug, erstellt werden. Ein etwas benutzerfreundlicheres Interface besitzt `cfdisk`; eine grafische Oberfläche (allerdings eine eingeschränkte Funktionalität) bietet der `Disk Druid`, der von vielen Installern verwendet wird. Die Linux-Partitionierungstools sind in der Lage, alle in der MS-Welt verbreiteten sowie viele Unix-übliche Partitionen zu erstellen. Als Faustregel gilt jedoch, daß man xy-Partitionen mit dem Partitionierungstool des xy-Betriebssystems

Installationsschritte

0. Repartitionieren der Festplatte(n)
   (freier Plattenplatz benötigt)
1. Installationsmedium booten
2. Installationsmenü/-programm durchlaufen
   (sehr distributionsspezifisch)
   - Auswahl Sprache, Tastatur, Maus
   - Linux-Partitionen erstellen
   - Dateisystem erzeugen
   - Auswahl der zu installierenden SW
   - Minimale Konfiguration
   - Bootloader konfigurieren/installieren
3. Booten des neuen Systems
4. Basiskonfiguration

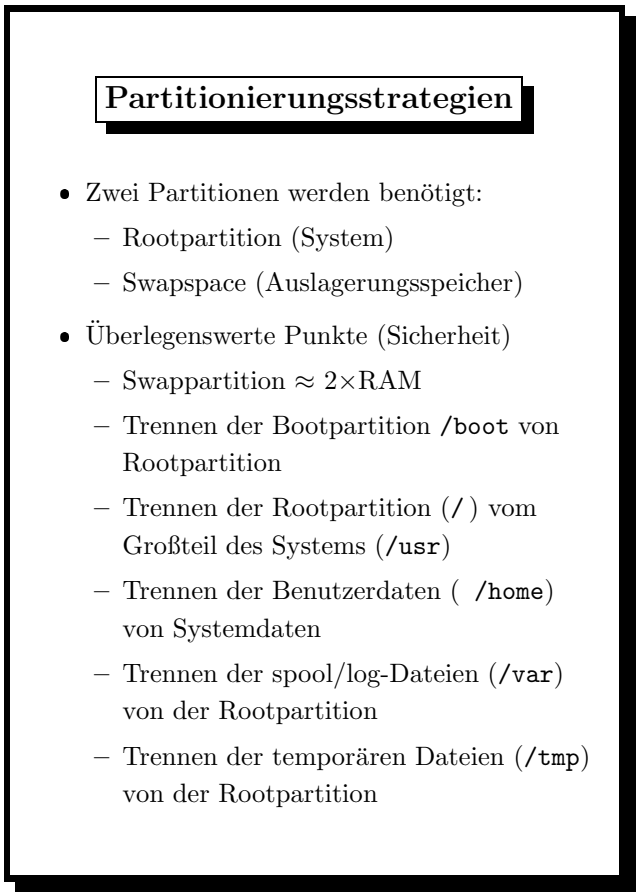
Folie 126

erstellen sollte. Insbesondere kann `fdisk` von DOS unter Linux erstellte FAT-Partitionen nicht löschen.

Um ein Linux-System zu installieren, benötigt man (üblicherweise) zumindest zwei Partitionen; eine *Rootpartition* für das System und eine *Swappartition* (Auslagerungsspeicher auf der Festplatte). Je nach Verwendung des Systems ist es jedoch günstig, Teile des Filesystems auf weitere Partitionen aufzuteilen (siehe auch 3.7). Wir geben auf Folie 127 einen Überblick über Partitionierungsstrategien und zeigen ein „echtes“ Beispiel auf den Folien 130 und 131. Auf Folien 128, 129 sind die Namen für Laufwerke unter Linux erklärt.

Wichtig ist, daß man die Partitionierung der Festplatten gut plant. Eine Änderung zu einem späteren Zeitpunkt ist—wenn überhaupt möglich—ein aufwändiges und eher riskantes Unternehmen, das man besser vermeiden sollte.

Im Lieferumfang (fast) aller Linux-Distributionen befindet sich das DOS-Tool `FIPS` zum Verkleinern bereits bestehender (nicht voller) FAT-Partitionen. Dies ermöglicht die Installation eines Linux-Systems, auch wenn bereits alle Festplatten des Systems partitioniert sind. Für NTFS-Partitionen verwendet man `ntfsresize` oder ein proprietäres Programm (wie z.B. PartitionMagic).



Folie 127

Als nächsten Punkt behandeln wir das Booten des Installationsmediums (der Bootvorgang wird im Detail in Kapitel 9.1 erklärt). Zunächst muß die Entscheidung für ein Bootmedium (Diskette, CD-Rom) getroffen werden. Die CD kann entweder in einem Fachgeschäft gekauft oder aus dem Internet heruntergeladen (sogenannte *Iso-Images*) und gebrannt werden. Diese CDs sind bootfähig, allerdings muß diese Funktion vom *BIOS* (Basic Input/Output System) unterstützt und aktiviert werden. Booten von der CD ist allerdings nur im Falle einer lokalen Installation sinnvoll. Anderenfalls sind Bootdisketten zu verwenden. Diese *Boot-Images* sind ebenfalls im Lieferumfang der Distribution enthalten, also auf der CD oder im Internet zu finden. Manche Distributionen verwenden (aufgrund der

Laufwerke unter Linux

Gerät	Name	
1. Floppy	/dev/fd0	A:
2. Floppy	/dev/fd1	B:
1. IDE-Festplatte		
(primary master)	/dev/hda	
1. Primäre Part.	/dev/hda1	C:
2. Primäre Part.	/dev/hda2	
3. Primäre Part.	/dev/hda3	
4. Primäre Part.	/dev/hda4	
Erweiterte Part.		
1. Logische Part.	/dev/hda5	E:
2. Logische Part.	/dev/hda6	G:
...		

Folie 128

Gerät	Name	
2. IDE-Festplatte		
(primary slave)	/dev/hdb	
1. Primäre Part.	/dev/hdb1	D:
...		
1. Logische Part.	/dev/hdb5	F:
...		
3. IDE-Festplatte		
(secondary master)	/dev/hdc	
4. IDE-Festplatte		
(secondary slave)	/dev/hdd	
1. SCSI-Festplatte	/dev/sda	
1. Primäre Part.	/dev/sda1	
...		
2. SCSI-Festplatte	/dev/sdb	
...		

Folie 129

Ein „typisches" Beispiel

```
[root@banach /root]# df
 Filesystem   1024-blocks     Used   Available   Capacity   Mounted
 /dev/hda1         248847    90903      145094      39%     /
 /dev/hda5        2974519   502400     2318299      18%     /home
 /dev/hda7         248847      131      235866       0%     /tmp
 /dev/hda6        1981000   864498     1014090      46%     /usr
```

Welche Partition wohin gemountet wird: `/etc/fstab`

Folie 130

```
[root@banach /root]# /sbin/fdisk /dev/hda
Command (m for help): p

Disk /dev/hda: 255 heads, 63 sectors, 788 cylinders
Units = cylinders of 16065 * 512 bytes

 Device      Boot   Start   End    Blocks   Id   System
 /dev/hda1    *         1    32   257008+   83   Linux native
 /dev/hda2             33   788   6072570    5   Extended
 /dev/hda5             33   415   3076416   83   Linux native
 /dev/hda6            416   670   2048256   83   Linux native
 /dev/hda7            671   702   257008+   83   Linux native
 /dev/hda8            703   718   128488+   82   Linux swap
 /dev/hda9            719   734   128488+   82   Linux swap
```

Folie 131

Notwendigkeit, verschiedene Treiber zur Verfügung zu stellen) verschiedene Boot-Images für lokale oder Netzwerkinstallationen bzw. für Notebooks. Die Boot-Images können nicht „normal“ auf Diskette kopiert werden. Unter DOS muß das Tool `RAWRITE` verwendet werden, das mit jeder Linux-Distribution mitgeliefert wird (ACHTUNG: funktioniert nicht im DOS-Fenster unter WinNT). Unter Linux wird der Befehl `dd if=Inputfile of=Outputfile [Optionen]` (Data Dumper) verwendet, z.B.
`\$ dd if=bootnet.img of=/dev/fd0 bs=1440k`.

Das Bootmedium enthält einen minimalen Linux-Kernel, der zunächst in den Speicher geladen wird. Dann wird ein schlankes Linux-System gestartet, das während des weiteren Installationsvorgangs läuft. Manche Distributionen verwenden eine zweite Diskette, auf der dieses System gespeichert ist oder laden die entsprechenden Dateien von der CD oder über das Netzwerk. Da die Festplatten des PCs zu diesem Zeitpunkt noch nicht verwendet werden können, wird das System auf einer virtuellen Platte – die im RAM angelegt wird (RAM-Disk) – betrieben.

Nach der Erkennung und Initialisierung der Hardware wird das Installationsprogramm gestartet, das bei den (allermeisten) Distributionen menügesteuert ist. Meist hat man auch (auf einer Parallelkonsole) eine einfache Shell zur Verfügung. Fehlermeldungen und ein Protokoll der Installation werden ebenfalls vielfach auf einer Parallelkonsole ausgegeben.

Die Folien beschreiben die grundlegenden Schritte, die vom Installer jeder Distribution durchgeführt werden. Die genaue Reihenfolge ist jedoch sowohl von der verwendeten Distribution als auch der Installationsvariante abhängig. So muß die Netzwerkkonfiguration ganz am Anfang der Netzwerkinstallation stehen.

Schließlich wird nach Abschluß des Installationsprozesses das neue System gebootet. Einige Hinweise dazu finden sich auf Folie 136.

8.3 Installation eines Linux-Systems

In diesem Abschnitt wird in einer Serie von Folien der gesamte Verlauf einer Linux-Installation am Beispiel eines RedHat-Systems (Version 7.0) detailliert dargestellt. Die gewählte Installationsart ist lokal von einem CD-Rom-Laufwerk. Die CD-Rom dient auch als Bootmedium. Bei dem PC handelt es sich um einen Standard-PC (virtuell von VMware emuliert; Host-Betriebssystem ist RedHat Linux 6.1 auf einem ASUS-7400 Notebook mit einer Intel-Celeron-CPU mit 400MHz und 96MB RAM) mit folgenden Hardware-Spezifikationen:

1. Booten des Installationsmediums

- Bootdiskette; erstellen mittels
  - `RAWRITE` (DOS) oder
  - `dd if=image of=/dev/dfd0 bs=1440k` (Linux)

  eventuell verschieden für lokale, Netzwerk-Installation und PCMCIA
- CD-Rom (nur für lokale Installation)

Was genau passiert

- Kernel wird ins RAM geladen
- minimales Linux-System auf RAM-Disk eventuell als 2. Diskette, CD-Rom oder Netz (dann vorher Konfiguration der Netzwerkkarte)
- HW-Erkennung und Initialisierung
- Starten des Installationsprogramms

Folie 132

2. Installationsmenü/-programm

- Basiskonfiguration
  - Sprache
  - Tastatur
- Auswahl der Installationsmethode
  - lokal: CD-Rom oder Festplatte
  - Netz: Ftp, Http, Nfs
- **Netzwerkkonfiguration**
  - dynamisch mittels DHCP
  - statisch: benötigt wird
    - IP-Adresse
    - Netzmaske
    - Default Gateway
    - Primärer Nameserver

Folie 133

- Auswahl der Installationsart
  - vorgefertigte Installation (Server/Workstation)
  - Custom
- Partitionieren der Festplatte
- Mountpoints festlegen
- Formatieren der Festplatte
- Weitere Konfiguration
  - Hostname
  - Maus (USB/seriell/PS2, 2/3 Button)
  - Zeit (GMT+Zeitzone)
  - Rootpaßwort (!!!)
  - User-Accounts (optional)

Folie 134

- Konfiguration des X-Window-Systems
  - Grafikkarte (Type, Speicher)
  - Monitor (Type, vertikale und horizontale Bildwiederholfrequenz)
  - Bildschirmauflösung und Farbtiefe
- Konfiguration des Bootmanagers (`grub` oder `lilo`)
  - Master Boot Record oder Linux-Bootpartition
  - welche Betriebssysteme booten
- Auswählen der Software
  - langwieriger Schritt
  - Plattenplatz? Sicherheit!

Folie 135

3. Booten des neuen Systems

- Bootmanager-Prompt
  Menü bei `grub`; `lilo`: `TAB` zeigt eine Liste verfügbarer OS
- genaues Verfolgen der Bootmessages
  (später `dmesg` bzw. `/var/log/messages`)
- als root User-Accounts einrichten

  ROOT
  *** NUR Administration**
  *** NIEMALS Arbeiten**
  * NIE X starten!

  – `adduser name`
  – grafische Tools (`linuxconf`, `webmin`, ...)
- System stoppen (rebooten)
  – `shutdown -h(-r) now`
  – auf Textkonsole `<ctrl><alt><del>`
  – **NIEMALS ...**

Folie 136

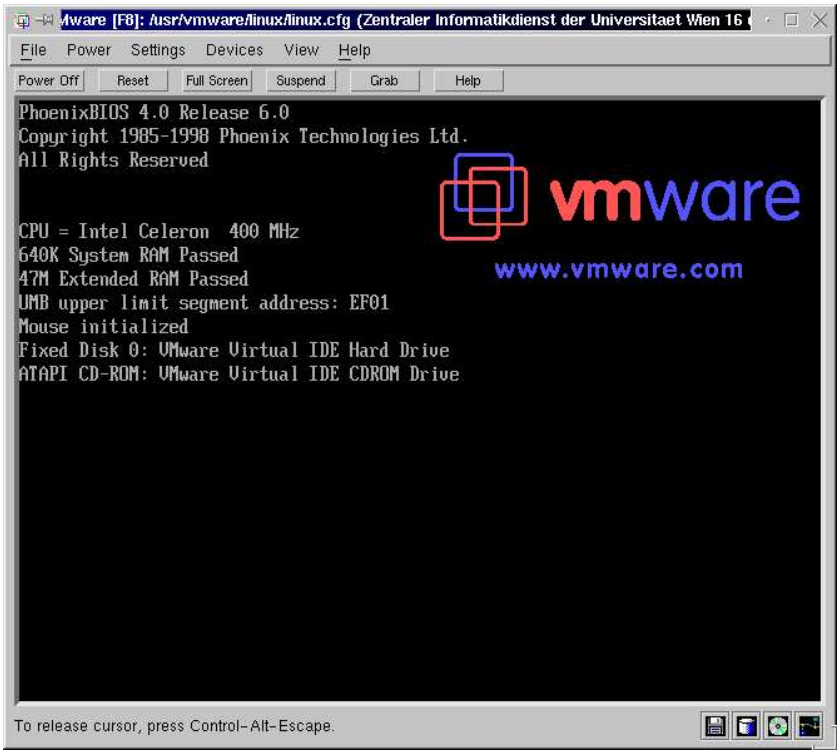


Abbildung 1: Booten des Rechners

- Intel Celeron, 400MHz
- 48MB RAM
- IDE-Festplatte, 1.5GB
- Standard-ATAPI-CD-Rom
- Standard-VGA-Grafikkarte
- LCD-Panel, 1024x768

Während der Vorlesung wird eine analoge Installation „live“ durchgeführt. Wir geben hier einen Index der einzelnen Schritte/Abbildungen.

1. Booten des Rechners
2. RedHat Bootup-Screen
3. Booten des Linux-Kernels
4. Sprachauswahl
5. Konfiguration der Tastatur
6. Konfiguration der Maus
7. RedHat Welcome-Screen
8. Installationsoptionen (wir wählen Custom-System)
9. Partitionierungswerkzeug wählen (wir wählen das mächtigere aber unelegantere Tool `fdisk`)
10. Partitionen erstellen (mit `fdisk`)

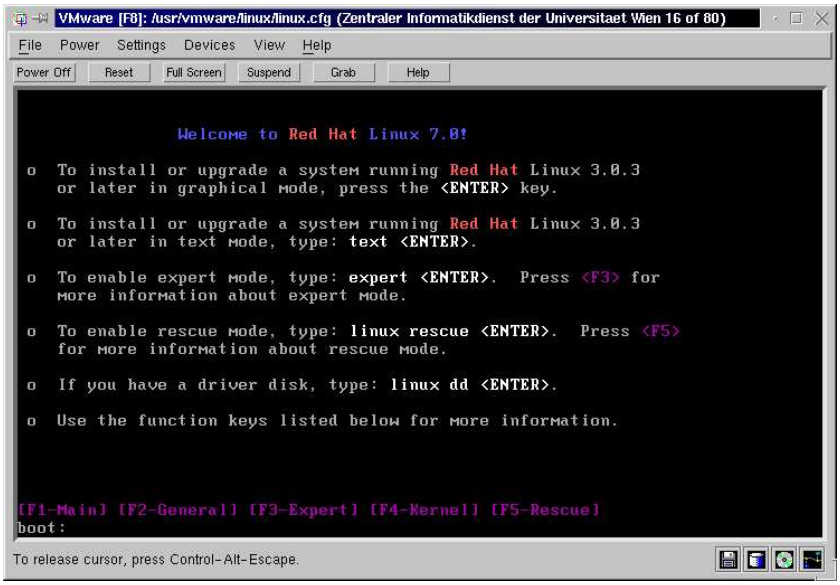


Abbildung 2: RedHat Bootup-Screen

11. Festlegen der Mountpoints
12. Auswählen der zu formatierenden Partitionen
13. Konfiguration des Boot-Loaders
14. Netzwerkkonfiguration
15. Konfiguration der Systemzeit
16. Rootpaßwort und Accountkonfiguration
17. Konfiguration der Authentifizierung
18. Paketauswahl (Übersicht)
19. Paketauswahl (Detail)
20. Monitorkonfiguration
21. Konfiguration des GUI (Grafikkarte, Auflösung,...)
22. Beginn der Paketinstallation
23. Formatieren der ausgewählten Partitionen
24. Paketinstallation
25. Postinstallations-Konfiguration
26. Erstellen der Bootdiskette
27. Ende der Installation

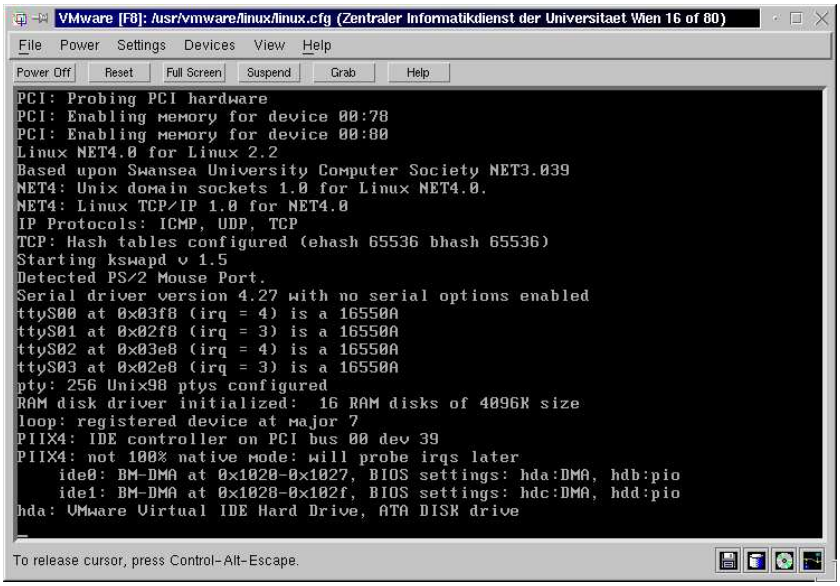


Abbildung 3: Booten des Linux-Kernels

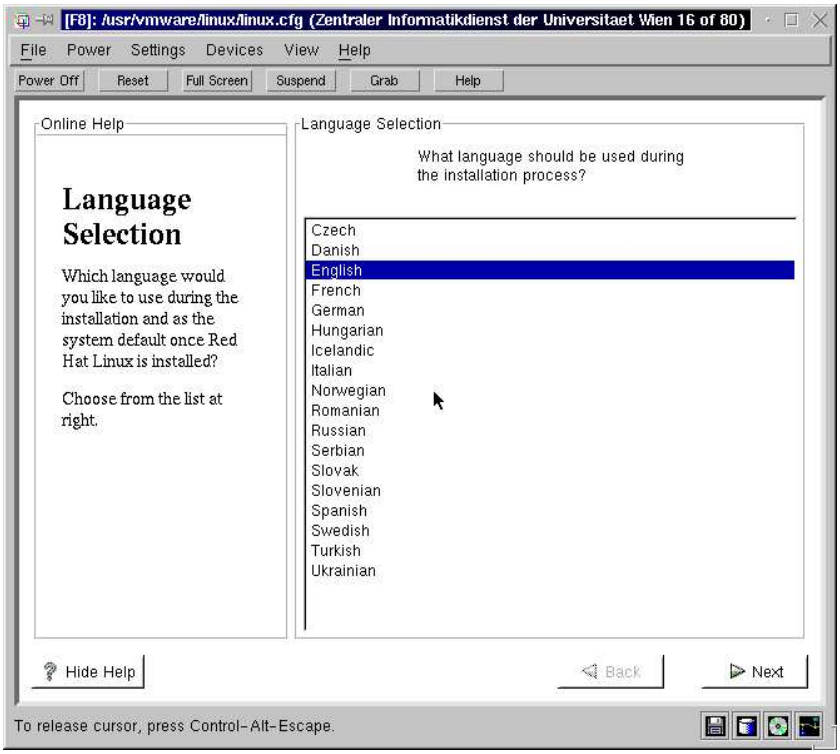


Abbildung 4: Sprachauswahl

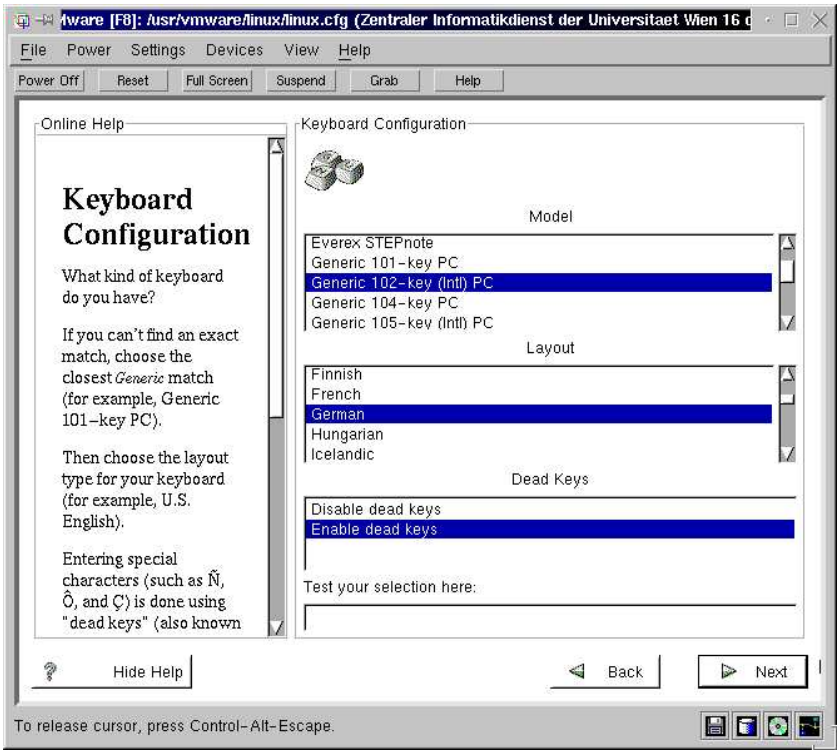


Abbildung 5: Konfiguration der Tastatur

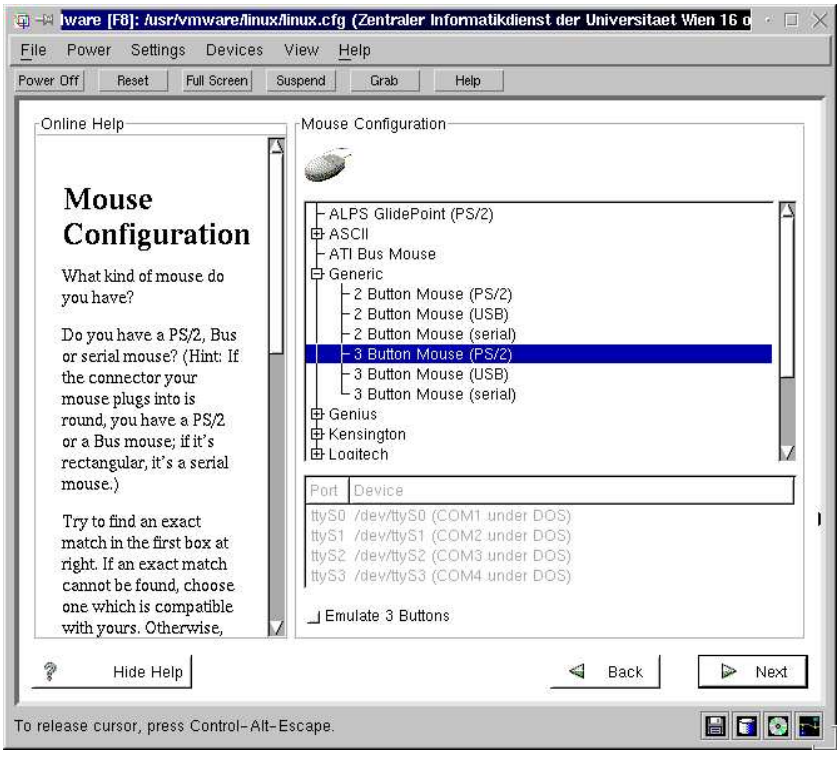


Abbildung 6: Konfiguration der Maus

Abbildung 7: RedHat Welcome-Screen

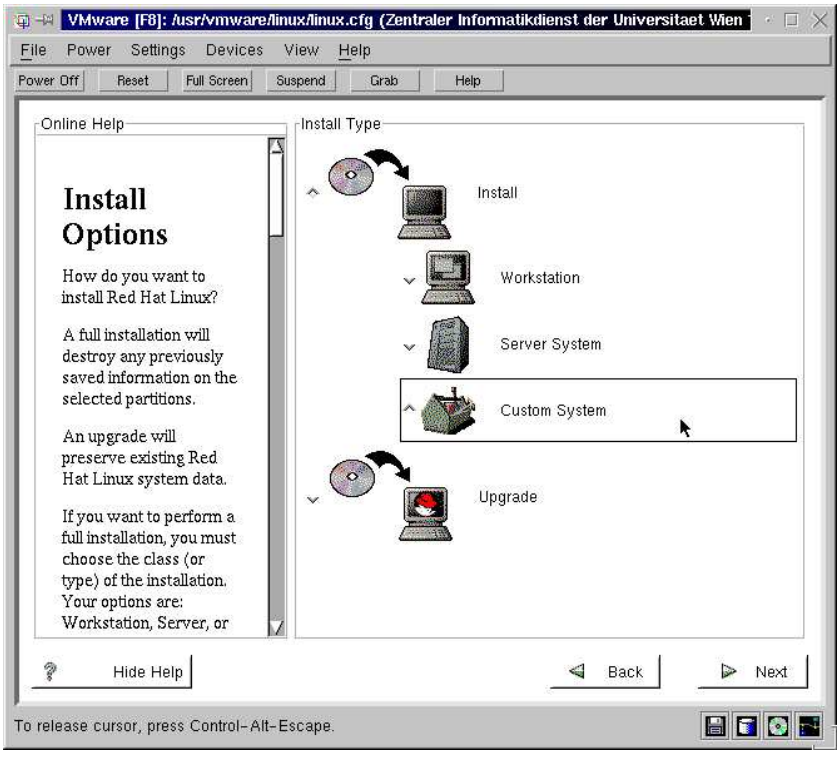


Abbildung 8: Installationsoptionen

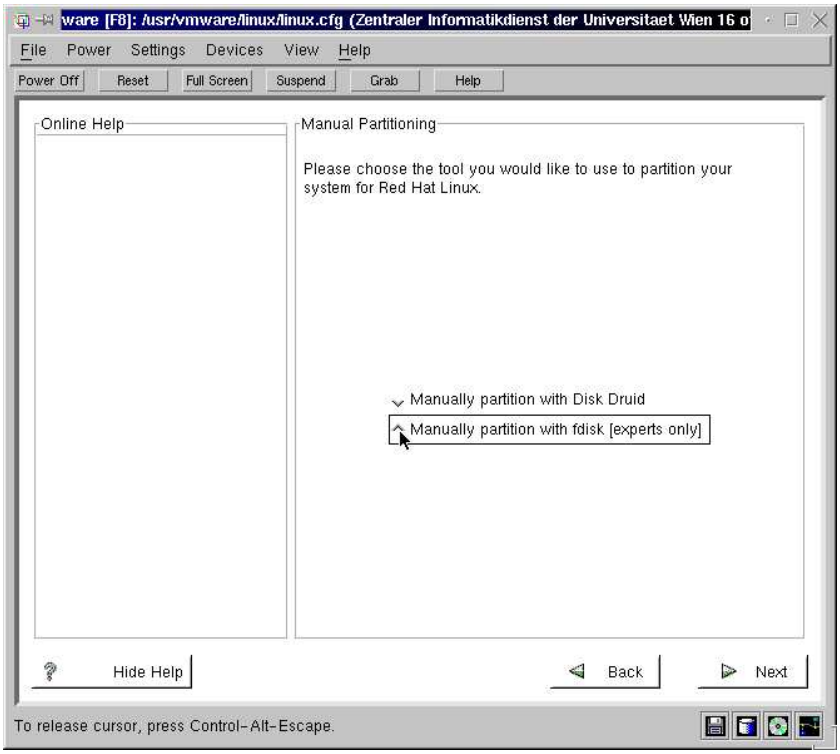


Abbildung 9: Partitionierungswerkzeug wählen

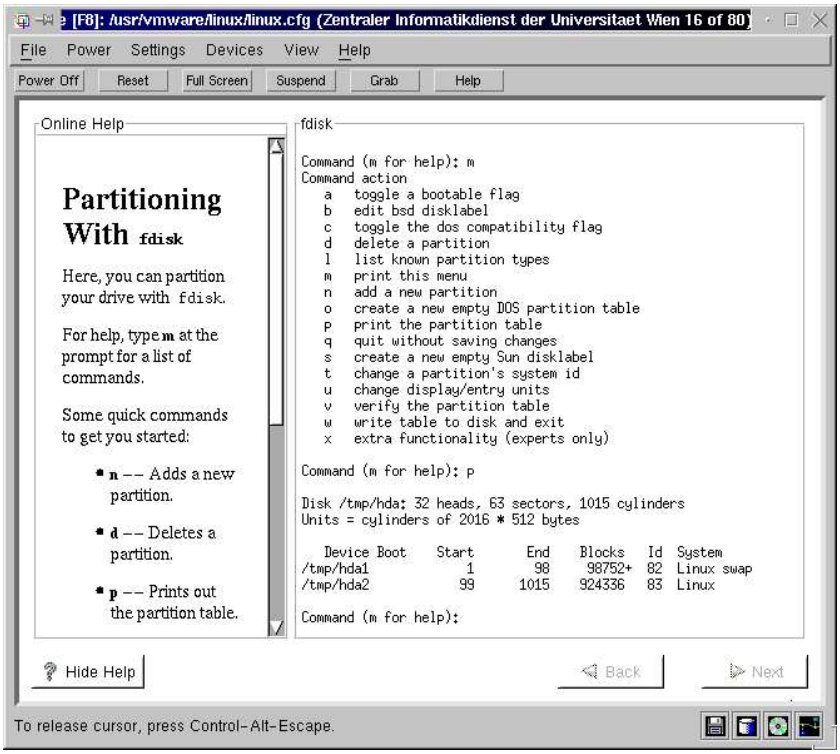


Abbildung 10: Partitionen erstellen (mit fdisk)

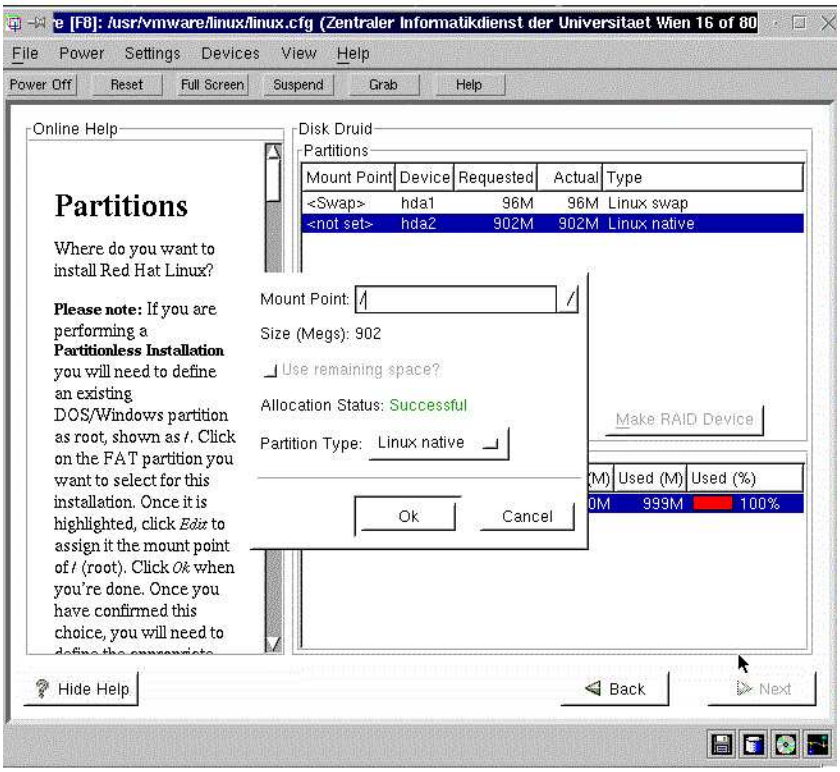


Abbildung 11: Festlegen der Mountpoints

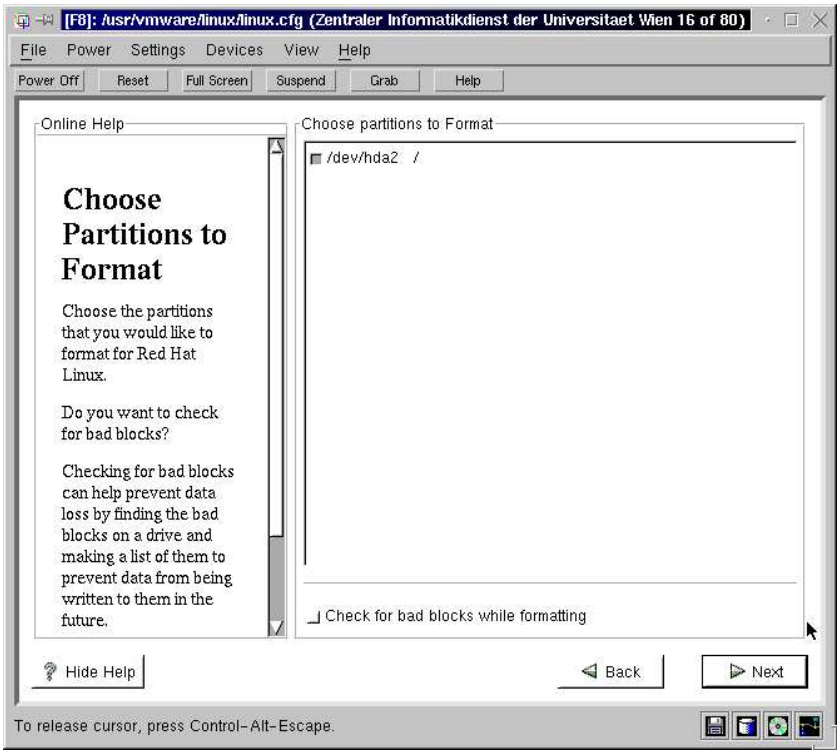


Abbildung 12: Auswählen der zu formatierenden Partitionen

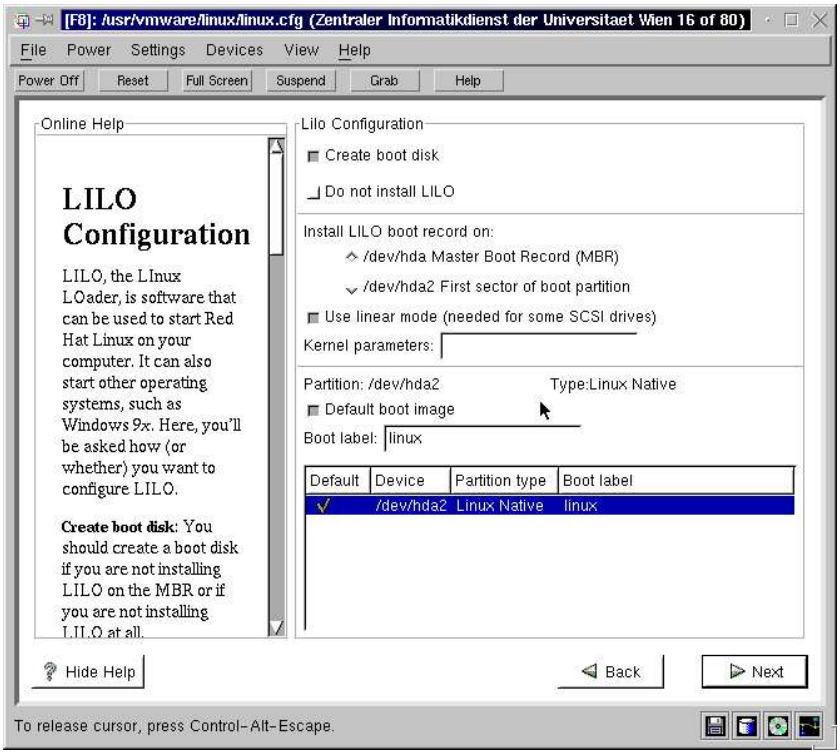


Abbildung 13: Konfiguration des Boot-Loaders

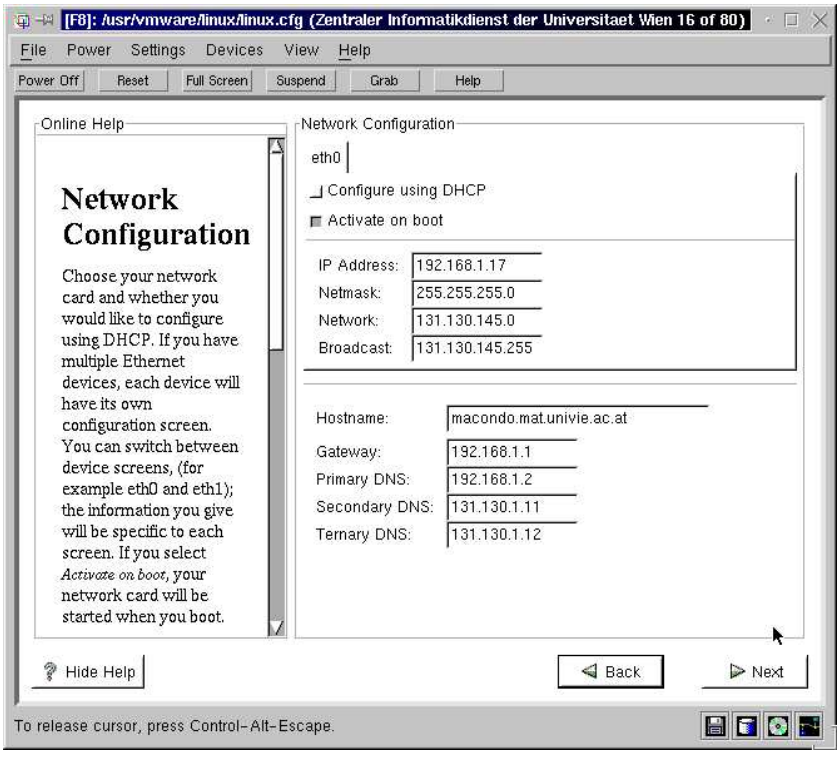


Abbildung 14: Netzwerkkonfiguration

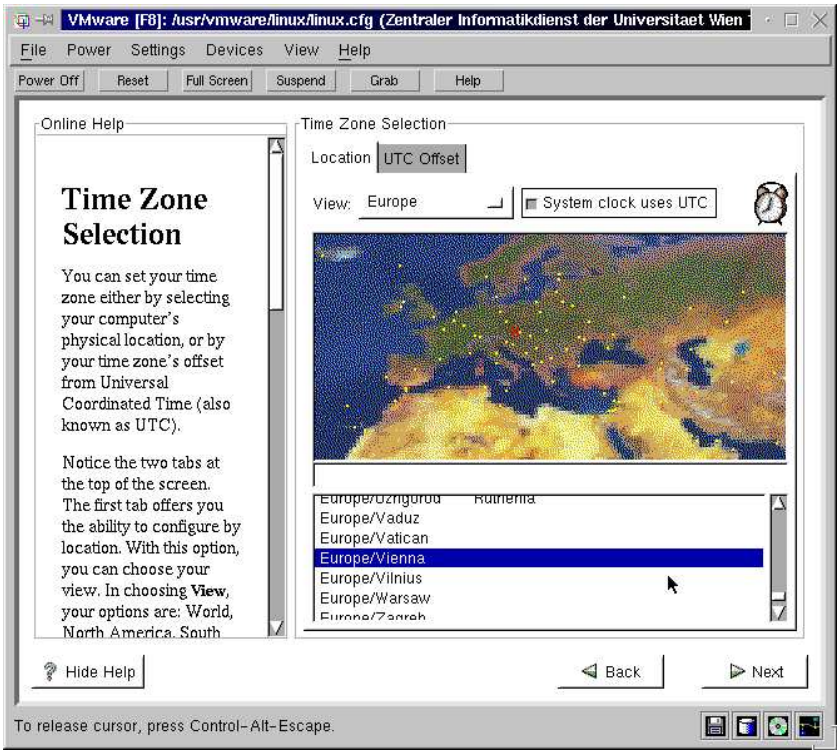


Abbildung 15: Konfiguration der Systemzeit

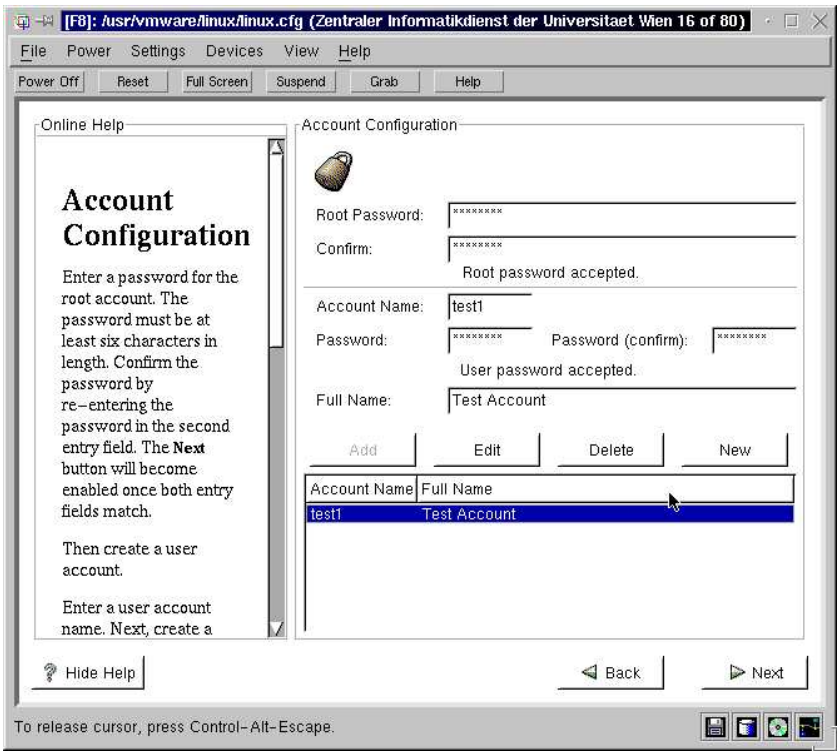


Abbildung 16: Rootpaßwort und Accountkonfiguration

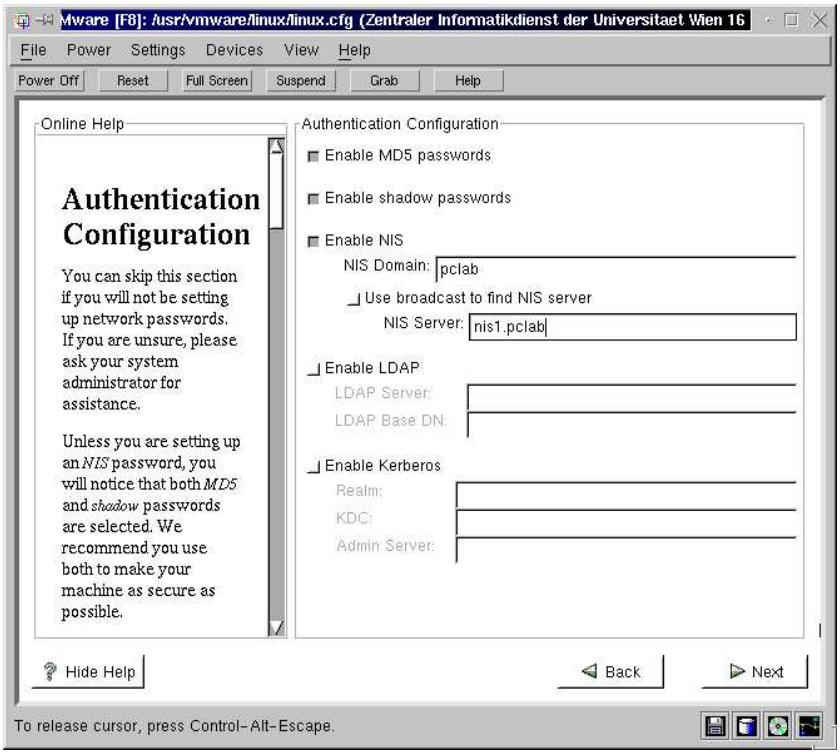


Abbildung 17: Konfiguration der Authentifizierung

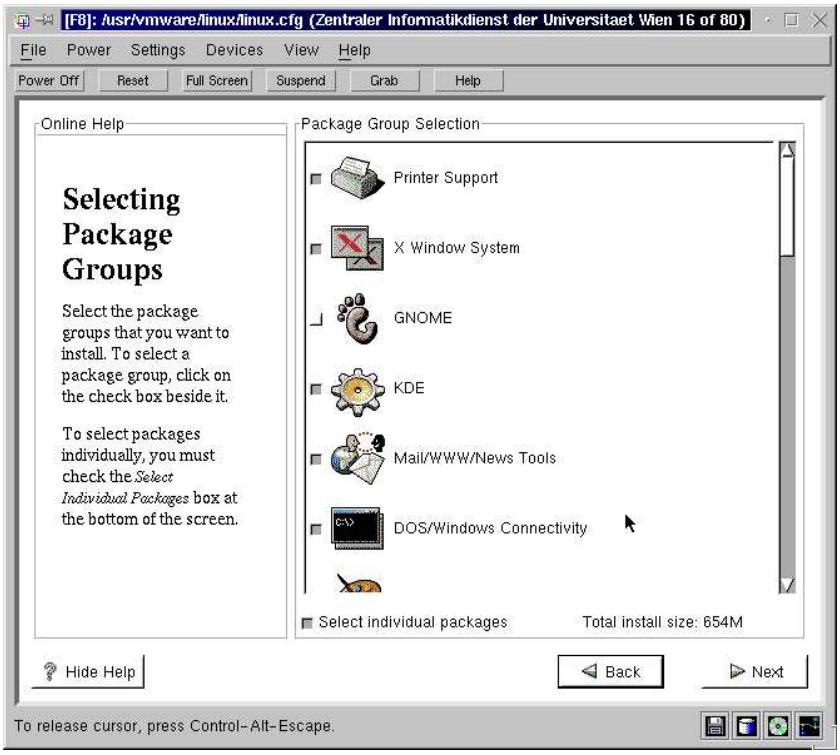


Abbildung 18: Paketauswahl (Übersicht)

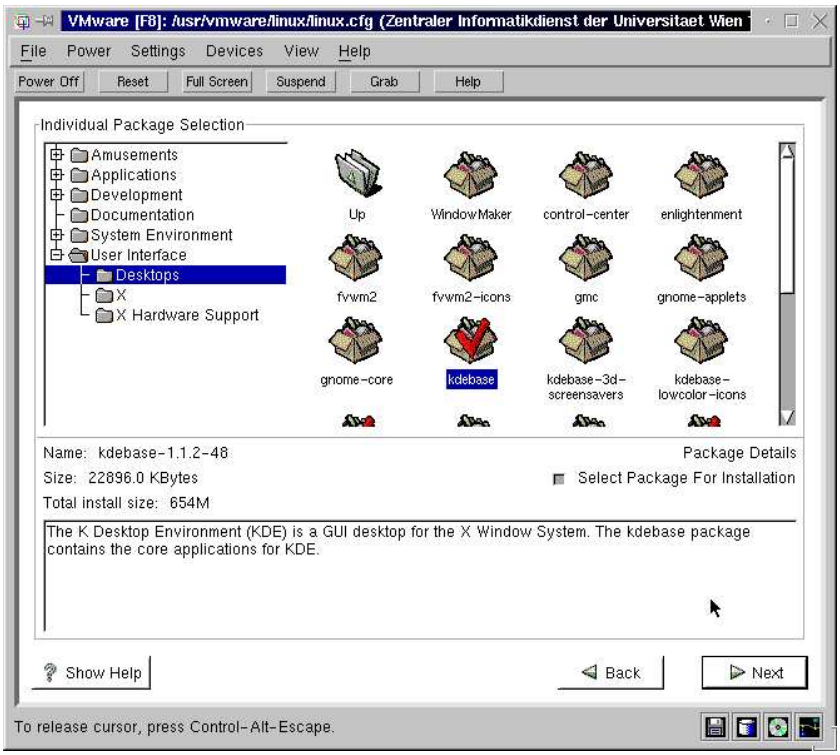


Abbildung 19: Paketauswahl (Detail)

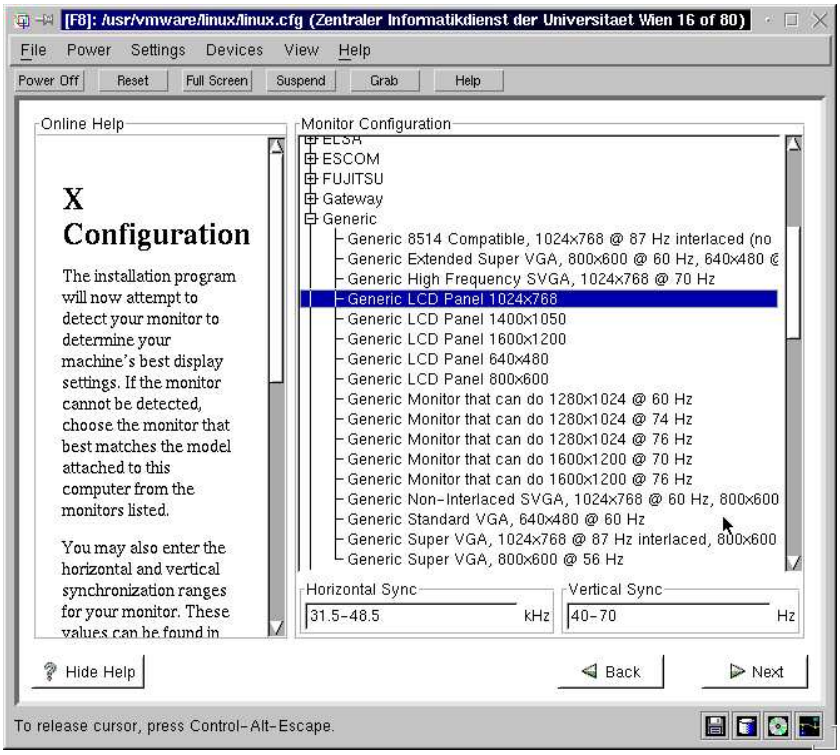


Abbildung 20: Monitorkonfiguration

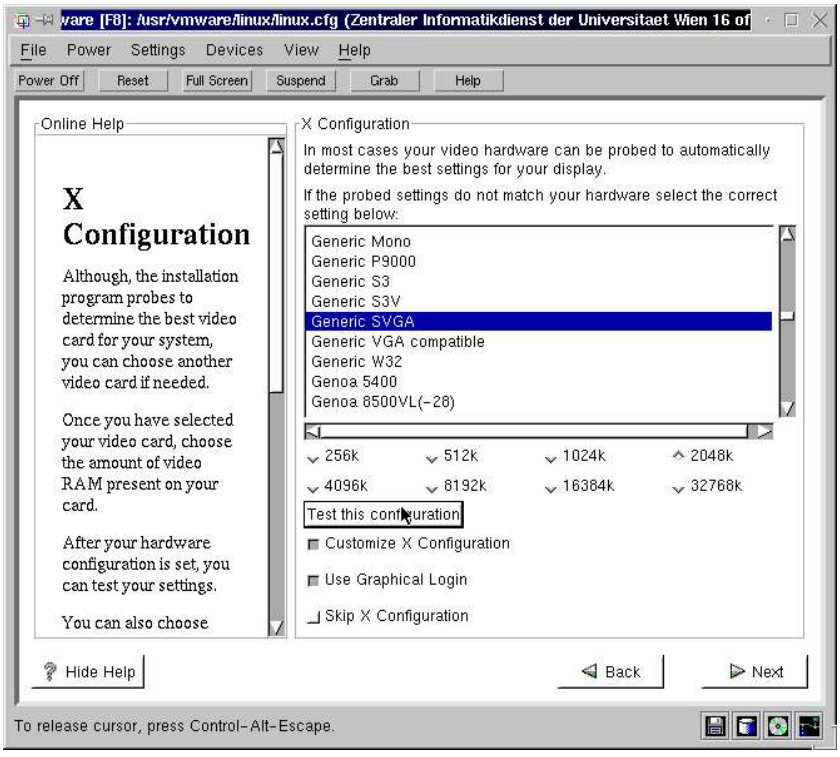


Abbildung 21: Konfiguration des GUI

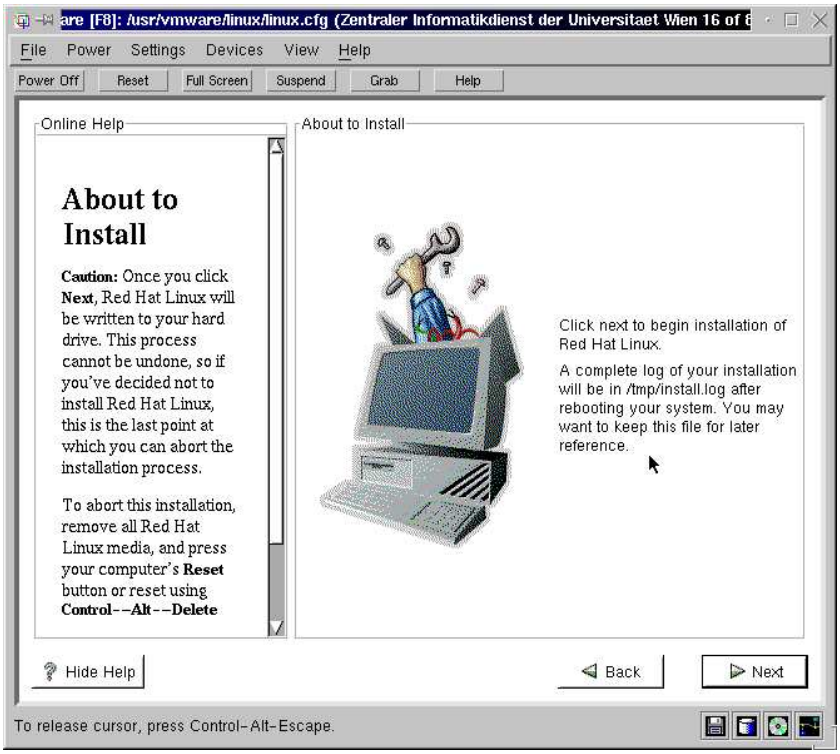


Abbildung 22: Beginn der Paketinstallation

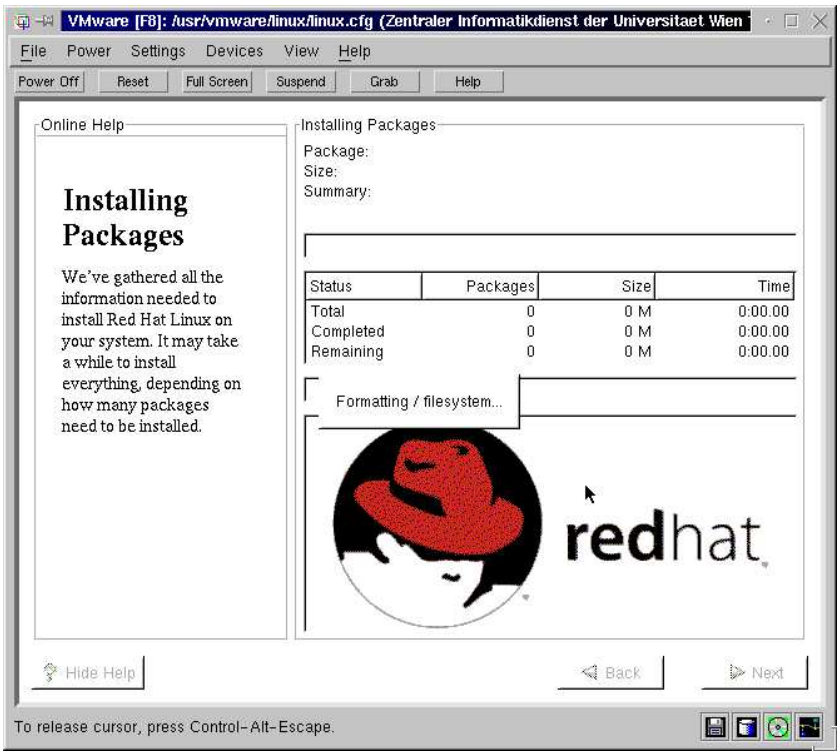


Abbildung 23: Formatieren der ausgewählten Partitionen

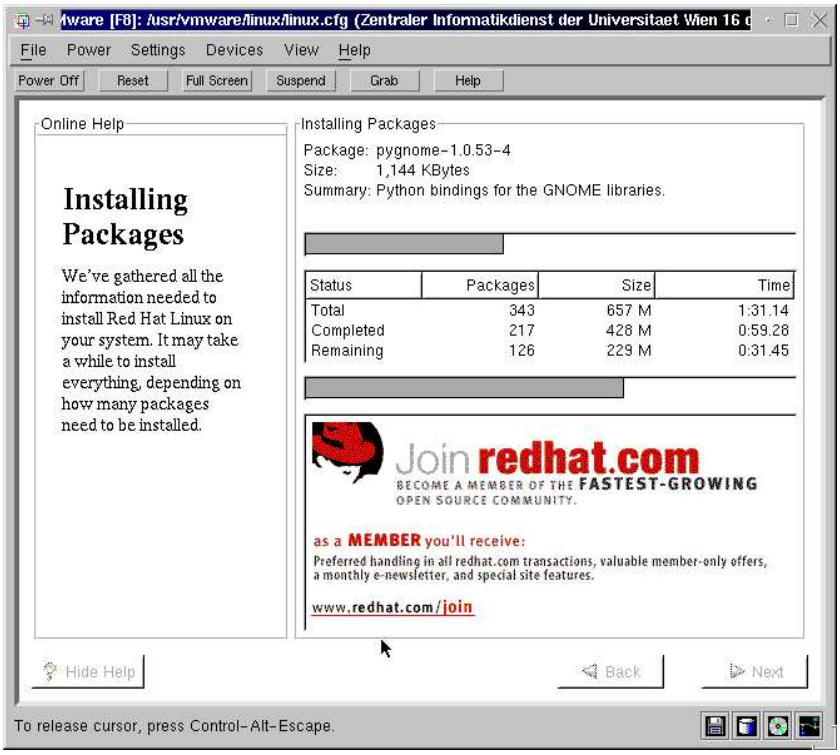


Abbildung 24: Paketinstallation

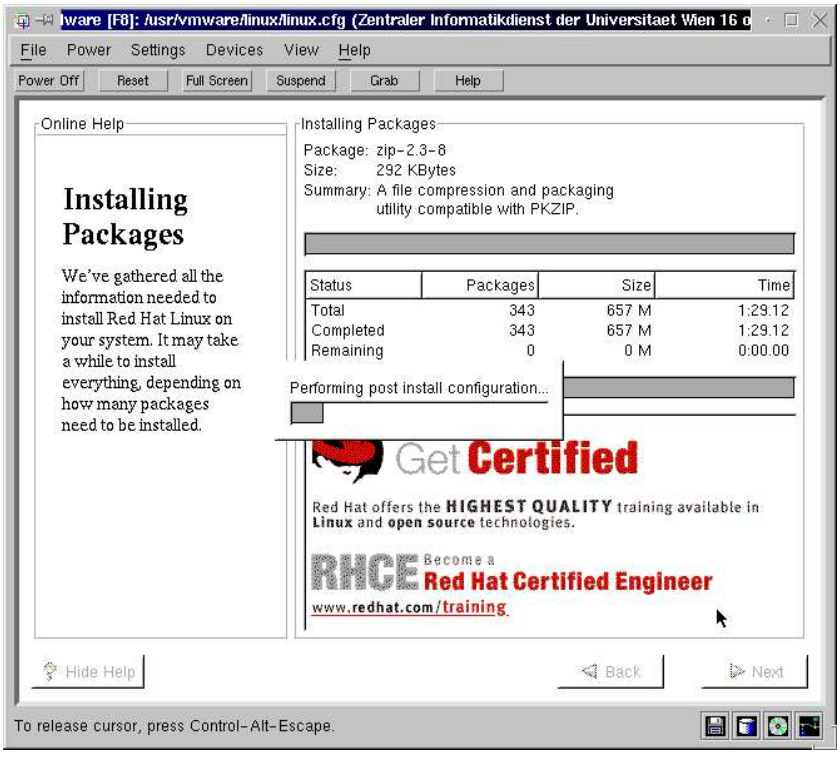


Abbildung 25: Postinstallations-Konfiguration

Abbildung 26: Erstellen der Bootdiskette

Abbildung 27: Ende der Installation

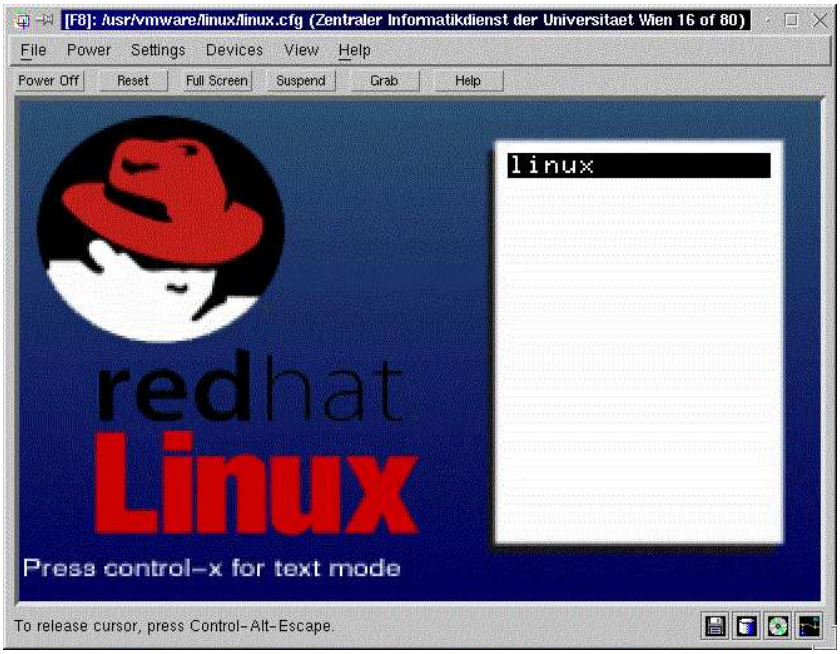


Abbildung 28: lilo-Bootmanager

Hier ist nun die eigentliche Installation abgeschlossen! Wir können nun das neue System booten. (Die CD muß natürlich aus dem Laufwerk entfernt werden.)

Die nächsten Abbildungen zeigen diesen Vorgang:

1. `lilo`-Bootmanager
2. Booten des neuinstallierten Systems
3. Grafisches Login

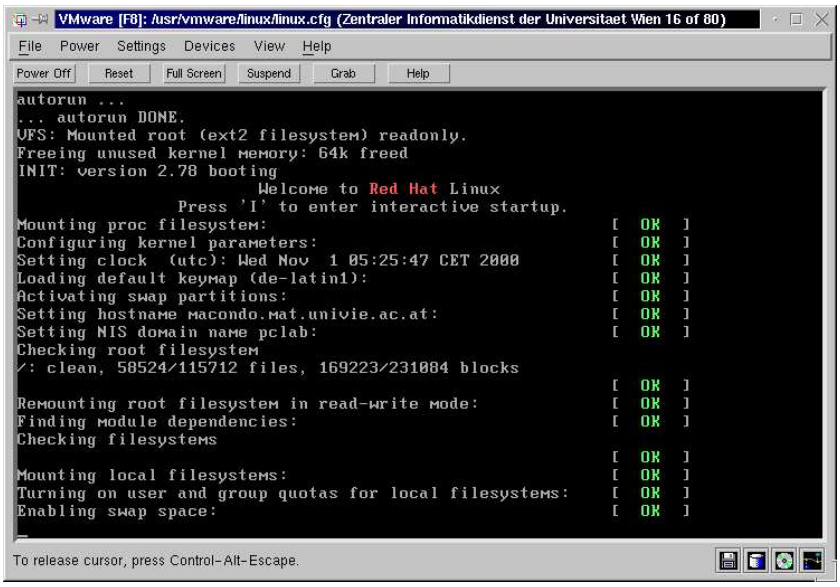


Abbildung 29: Booten des neuinstallierten Systems

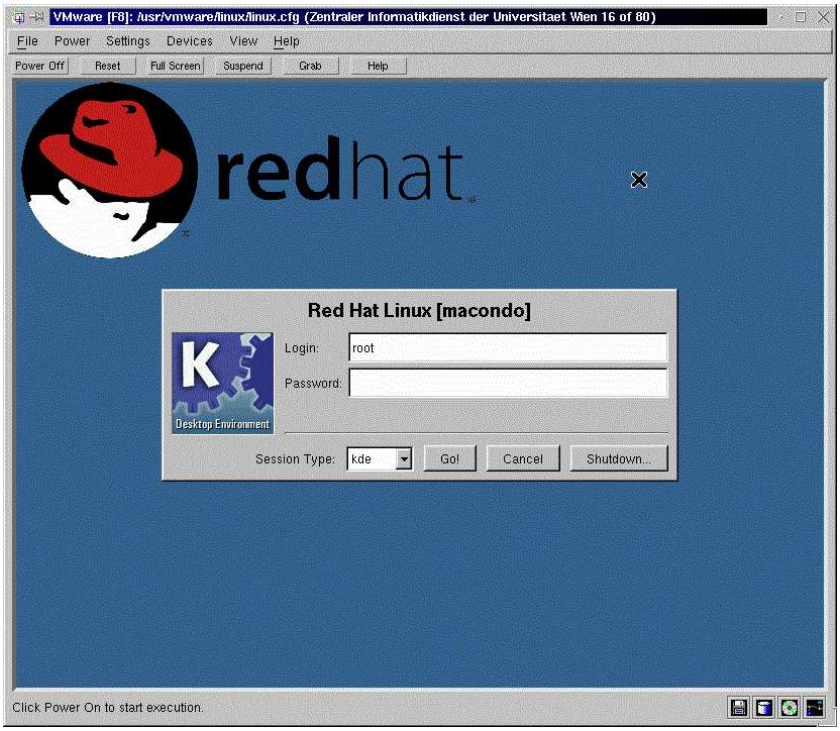


Abbildung 30: Grafisches Login

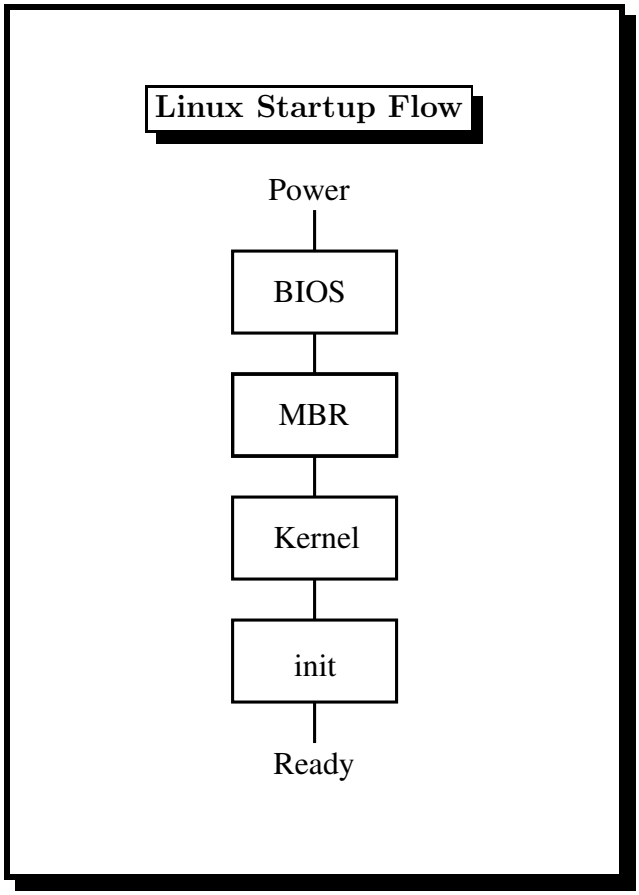
9 Grundlegende Systemkonfiguration

In diesem abschließenden Kapitel wird die grundlegende Kofiguration der Systemdienste eines Linux-Systems erklärt. Nach einem Abschnitt über das Starten und Stoppen des Systems erklären wir insbesondere die ersten Konfigurationsschritte nach der (Neu-)Installation. Besonderes Augenmerk legen wir auf elementare Sicherheit (Abschalten nicht benötigter Services) und das händische Nachinstallieren einzelner (Update-)Pakete.

9.1 Startup und Shutdown

Nachdem wir bereits mehrmals erklärt haben, wie man ein Multitasking- und Multiuser-System *nicht* auschaltet, erläutern wir hier, *wie* man dies tatsächlich tut und vor allem, was während Systemstart und -stop passiert und wie man diese Vorgänge beeinflussen kann. Wir beginnen mit dem Systemstart.

Nach dem Einschalten des Computers wird zunächst das *BIOS* (Basic Input/Output System) aktiviert. Dieses kleine Programm ist auf einem *EEPROM* (Electrically Erasable Programmable Read-Only Memory) auf dem Mainboard gespeichert. Es führt einige grundlegende Hardwarechecks durch, setzt einige Parameter und prüft die vorhandenen Boot-Geräte (Floppy, Festplatte, CD-Rom). Das BIOS ist über ein Setup-Menü, das während des Bootens meist mittels `<del>`-Taste aktiviert werden kann, konfigurierbar. Schließlich lädt das BIOS den MBR (Master Boot Record) des festgelegten Boot-Gerätes.



Folie 137

Der *Master Boot Record* ist der erste Sektor des entsprechenden Massenspeichergeräts und enthält Software, um das entsprechende Betriebssystem zu booten. Im Falle von Linux ist dies meist `grub` oder `lilo`. Er gibt einen *Bootprompt* aus, an dem das gewünschte zu bootende Betriebssystem gewählt werden kann und diesem zusätzliche Parameter übergeben

BIOS

- Basic Input/Output System
- in EEPROM am Mainboard gespeichert
- Setup mittels <DEL> beim Booten
- lädt Optionen (Bootdevices)
- minimaler HW Check
- lädt Master Boot Record

Folie 138

Master Boot Record

- erster Sektor des Bootgeräts
- liest Partitionstabelle
- lädt aktive Partition (default)
- Boot-Loader des OS (`grub`, `lilo`, `NTLDR`)

Folie 139

werden können. Der Default-Master-Boot-Record hingegen liest nur die Partitionstabelle des Geräts ein und übergibt die Kontrolle an die *aktive* Partition, von dieser wird dann das entspechende System gebootet.

Die Bootmanager verfügen über weitreichende Konfigurationsmöglichkeiten; so ist es möglich, den Zugriff (auch zu bestimmten Konfigurationen) über ein Passwort zu schützen, und das Verhalten beim Booten anderer Betriebssysteme zu beeinflussen. `grub` wird in dem (je nach Distribution an verschiedener Stelle zu findenden) Konfigurationsfile (`grub/menu.lst`) konfiguriert, `lilo` in `lilo.conf`. Wichtig ist es, daß der Bootmanager das Filesystem auf dem sich der zu ladende Betriebssystemkern befindet, verwenden kann (hier hat `grub` eindeutig die Oberhand), und es unter Umständen auch entsprechend dekomprimieren kann. Einzelheiten findet man in der Dokumentation der Bootmanager.

Der Boot-Loader beendet seine Arbeit mit dem Einlesen der gewählten Boot-Partition in den Speicher. Im Falle von Linux wird dabei der (komprimierte) Kernel (`/boot/vmlinuz`) geladen, entkomprimiert sich selbst und beginnt mit der Arbeit. Zunächst erkennt und initialisiert er die Hardware des Systems, schaltet die CPU in den Multitasking-Mode, startet (Low-Level) das Netzwerk und mountet die Rootpartition des Systems. Dann startet er die Mutter aller Prozesse `init` mit PID=1 sowie einige weiter spontane Prozesse. Danach startet der Kernel nie wieder einen Prozeß, sondern ist in der in Kapitel 2 beschriebenen Funktionsweise für das „Management“ des Systems verantwortlich. Während des Bootens schreibt der Kernel Meldungen auf die Konsole, die über Erfolg oder Fehler Auskunft geben; diese können später (im laufenden Betrieb) mittels `dmesg`, oder in `/var/log/messages`, manchmal auch in `/var/log/boot`, erneut angesehen werden.

Der weitere Startup-Fluß läuft somit über `init`. Dieser Prozeß liest zunächst das Konfigurationsfile `/etc/inittab` ein. Es wird das Default-Runlevel festgelegt, wobei unter *Runlevel* ein Betriebszustand des Systems verstanden wird, der jeweils eine bestimmte Funktionsweise definiert. Die Nummerierung der Runlevels kann von der Distribution abhängen; Im Falle von RedHat bedeutet Runlevel 3 normaler Multiuserbetrieb, Runlevel 5 Multiuserbetrieb mit grafischem Login, Runlevel 0 bzw. 6 ist der Shutdown bzw. Reboot und Runlevel 1 ist der sogenannte *Single User Mode*, der dem Administrator den Betrieb des Systems ohne Multiuser-Umgebung ermöglicht; dieser wird vor allem bei grundlegenden Eingriffen in die Konfiguration des Systems verwendet, z.B. ist es sinnvoll, daß kein Benutzer einloggen kann, während root nach einem Festplattencrash die Benutzerdaten vom Backup auf eine neue Platte einspielt. Statt in den Default-Runlevel zu starten, kann man durch den Bootmanager auch in andere Runlevel starten; übergibt man in den Kernel-Optionen ein `single`, so wird in den Single-User-Mode gestartet. Auch kann man das verwendete `init` überschreiben, zB durch `init=/bin/bash` (noch ein guter Grund, den Bootmanager durch ein Passwort zu schützen, ist aber für ein genügend „zerschossenes“System manchmal die letzte Rettung).

`init` startet alle für das gesetzte Runlevel in `/etc/inittab` vorgesehenen Programme und Systemdienste. Zunächst wird üblicherweise das für alle Runlevels vorgesehen Skript `/etc/rc.sysinit` oder `/etc/rc.d/rc.sysinit` abgearbeitet (Setzen des Hostnamens, Nis-Domainnames, Netzwerkkonfiguration, Festplattencheck, Mounten der lokalen Festplattenpartitionen, ...). Dann werden der Reihe nach die verschiedenen Systemdienste gestartet. Die meisten Linux-Systeme folgen hierbei dem System-V-Standard, der vorsieht, daß alle Dienste über eigene Scripts gestartet werden; diese befinden sich in `/etc/rc.d/init.d/` oder `/etc/init.d/`. In den Directories `/etc/rc.d/rc?.d/` bzw. `/etc/rc?.d/`, wobei „?“ für die Nummer des Runlevels steht, befinden sich jeweils für die in diesem Runlevel zu aktivierenden Dienste Links auf die Scripts in `/etc/(rc.d/)init.d/`. Die Namen der Links lauten `SnnDienst`, wobei „S“ für „Start“ steht, die zweistellige Zahl „nn“ die Reihenfolge festlegt und Dienst das jeweilige Programm meint, z.B.

```
$ ls -l /etc/rc.d/rc3.d
    S30sendmail -> ../init.d/sendmail
    K35atalk -> ../init.d/atalk
```

Analog dazu werden die Dienste beim Verlassen des entsprechenden Runlevels mit Links der Form `KnnDienst` gestoppt. Die einzelnen Systemdienste und wie man sie in verschie-

Booten des Kernels

- (komprimierter) Kernel geladen
  `/boot/vmlinuz`
- Hardware erkennen und aktivieren
- schaltet in Multitasking-Betrieb
- mountet Rootpartition
- startet `init` mit PID 1
- Meldungen auf Konsole; `dmesg`

Folie 140

`init`

- Mutter aller Prozesse
- liest Konfiguration aus `/etc/inittab`
- startet alle Programme fürs Runlevel
  - `/etc/rc.d/rc.sysinit`
  - `/etc/rc.d/rc?.d/`
  - `/etc/rc.d/rc.local`
- startet Login-Programme

Folie 141

/etc/inittab (1)

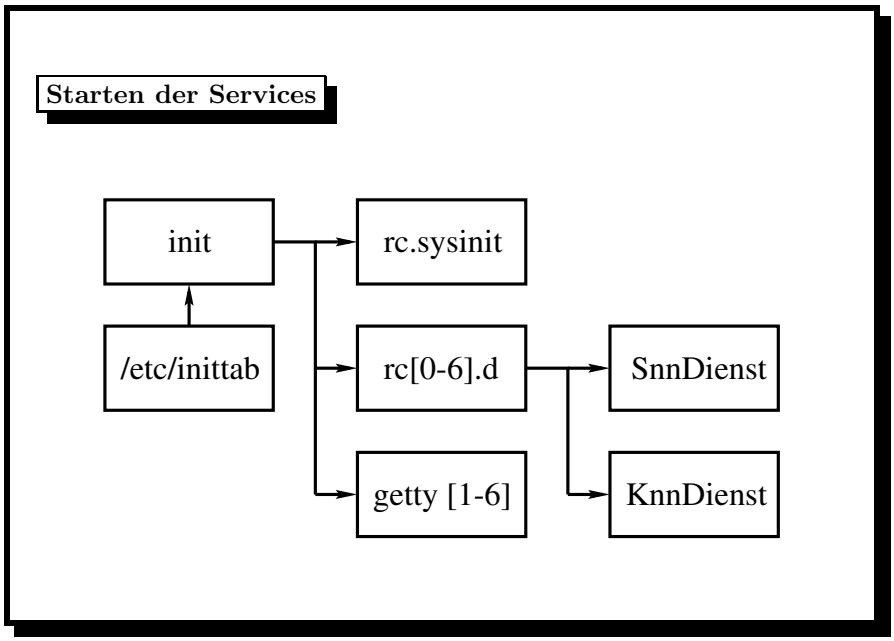
```
# Default runlevel. The runlevels used by RHS are:
#   0 - halt (Do NOT set initdefault to this)
#   1 - Single user mode
#   2 - Multiuser, without NFS
        (The same as 3, if you do not have networking)
#   3 - Full multiuser mode
#   4 - unused
#   5 - X11
#   6 - reboot (Do NOT set initdefault to this)
#
id:5:initdefault:
```

Folie 142

```
# System initialization.
si::sysinit:/etc/rc.d/rc.sysinit

l0:0:wait:/etc/rc.d/rc 0
.
.
l6:6:wait:/etc/rc.d/rc 6

# Trap CTRL-ALT-DELETE
ca::ctrlaltdel:/sbin/shutdown -t3 -r now

# Run gettys in standard runlevels
1:2345:respawn:/sbin/mingetty tty1
2:2345:respawn:/sbin/mingetty tty2
3:2345:respawn:/sbin/mingetty tty3
4:2345:respawn:/sbin/mingetty tty4
5:2345:respawn:/sbin/mingetty tty5
6:2345:respawn:/sbin/mingetty tty6

# Run xdm in runlevel 5
# xdm is now a separate service
x:5:respawn:/etc/X11/prefdm -nodaemon
```

Folie 143

denen Runlevels aktiviert bzw. deaktiviert, werden im Abschnitt 9.2 genauer besprochen. Schließlich wird als letztes Startup-Skript `/etc/rc.d/rc.local` ausgeführt, das dem Administrator die Möglichkeit bietet, eigene Konfigurationen übersichtlich ins allgemeine Schema einzubringen.



Folie 144

`init` spannt nach dieser Initialisierung der Systemdienste die `getty`-Prozesse auf, die auf das Login der User auf der Konsole warten; gegebenenfalls wird auch der grafische Login `xdm` aufgespannt. „Aufspannen“ bedeutet hier, daß diese Prozesse gestartet werden und `init` ihre Existenz überwacht. Stirbt der Prozeß, so wird er neu gestartet. Das garantiert, daß ein neues Login zur Verfügung steht, selbst wenn ein Benutzer es schafft, alle „seine“ Prozesse zu killen.

Wie bereits angedeutet ist der System-Shutdown durch einen Wechsel des Runlevels auf 0 (Halt) oder 6 (Reboot) zu bewerkstelligen. Dies kann man entweder direkt durch Senden des entsprechenden Signals an den `init`-Prozeß (`init 0` oder `telinit 0` bzw. 6) oder komfortabler durch den `shutdown`-Befehl. Seine Syntax ist:

```
shutdown [-rh] time [warning-message],
```

wobei die Option `-h` bzw `-r` für „halt“ bzw. „reboot“ stehen. „`time`“ ist der Zeitpunkt des Shutdowns, der im Format hh:mm absolut oder +m angegeben werden muß; „now“ ist ein alias für +0. Der Shutdown bzw. Reboot kann auch durch die Befehle `halt` bzw. `reboot` eingeleitet werden. Eine weitere Möglichkeit ist oft beim grafischen Login durch ein Menü gegeben. Je nach Konfiguration des Systems ist der Shutdown nur für root oder für eine Gruppe von Benutzern oder auch alle Benutzer erlaubt. Gemäß dem Eintrag in `/etc/inittab` ist ein Reboot auch durch den 3-Finger-Griff (Affengriff) `<ctrl><alt><del>` auf einer Textkonsole (nicht im GUI) möglich.

9.2 Systemdienste

In diesem Abschnitt beschreiben wir die wichtigsten Systemdienste und erklären, wie sie im laufenden Betrieb und in der Konfiguration der einzelnen Runlevels gestartet und gestoppt werden können. Details über die einzelnen Dienste (Daemonen) und ihre Konfiguration werden im zweiten Teil der Vorlesung behandelt.

Der wichtigste aller Dienste ist natürlich `init`, den wir bereits ausführlich behandelt haben. Weitere wichtige Dienste sind:

- **getty, xdm**: Auch diese Dienste haben wir schon kennengelernt; hier die etwas vollständigere Beschreibung. `getty` ist für das Login auf der Konsole verantwortlich. `init` startet für jede Konsole ein eigenes `getty`. Dieses liest den Benutzernamen und startet das Programm `login`, das dann das Paßwort einliest. Stimmt alles, so startet `login` die Shell (Eingabeaufforderung). Stirbt `login` (wenn sich der Benutzer wieder ausloggt, oder das Passwort falsch eingegeben wurde), so startet `init getty` erneut. `xdm` ist in analoger Weise für das grafische Login verantwortlich.

- **syslogd, klogd**: Der Kernel und andere Prozesse produzieren eine Vielzahl an Fehler-, Warn- und weiteren Meldungen, die teilweise recht wichtig sein können. Damit diese später gelesen werden können (vielleicht sogar viel später) schreibt der Daemon `syslogd` diese Meldungen in die entsprechenden Logfiles; das wichtigste ist `/var/log/messages`. `syslogd` schreibt aber nicht alle Meldungen in eine Datei, sondern sortiert diese nach Art und Gewicht gemäß seiner Konfiguration in verschiedene Dateien. `klogd` ist speziell für Kernel-Meldungen zuständig.

- **cron, at**: Viele Benutzer wollen gewisse Aufgaben/Programme regelmäßig ausführen. Dafür ist `cron` zuständig. Dieser Dienst wacht einmal in der Minute auf, um nachzuschauen, ob ein Auftrag auszuführen ist. Diese Programme arbeiten selbständig (ohne Benutzereingabe). Die Ausgabe erfolgt mittels E-Mail an den Benutzer, der den Auftrag erstellt hat. Die Aufträge können mittels `crontab` manipuliert werden. `at` funktioniert ähnlich, nur daß er die Aktion genau einmal ausführt.

- **lpd, cupsd**: sind Drucker-Daemonen (je nach verwendeten Drucksystem, normalerweise nur einer!). Er überwacht die Druckerwarteschlange und ist für das Ausführen und Queuen der Druckafträge zuständig. Er kann auch als Druckserver über ein Netzwerk agieren.

- **(x)inetd**: steht für Internetdaemon. Er überwacht viele der über das Netzwerk einlaufenden Anfragen, stellt gemäß seiner Konfiguration ihre Rechtmäßigkeit fest und startet den entsprechenden Server, der die Anfrage übernimmt, z.B. `in.telnetd`, `in.ftpd`, `in.talkd` für eingehende `telnet-`, `ftp-` und `talk`-Anfragen. Einige größere Netzwerkdienste laufen nicht über den `inetd`, sondern sind als Stand-Alone-Server konzipert. Beispiele dafür sind

  - **httpd**, der WWW-Server,
  - **sshd**, der Secure Shell Server,
  - **sendmail**, der BSD E-Mail Transport Agent (Mailserver),
  - **smbd, nmbd**, die Samba-Daemonen,
  - **portmap**, der RPC-Portmaper (C-Remote Procedure Call), grundlegend für NFS- und nis-Dienste
  - **rpc.mountd, rpc.statd, nfsd, rpc.lockd, rpc.quotad, ...**, Services für das Network File System (NFS),
  - **ypbind, ypserv** Client und Server für das Network Information Service.

- **kswapd**: managt die Benutzung des Swapspeichers.

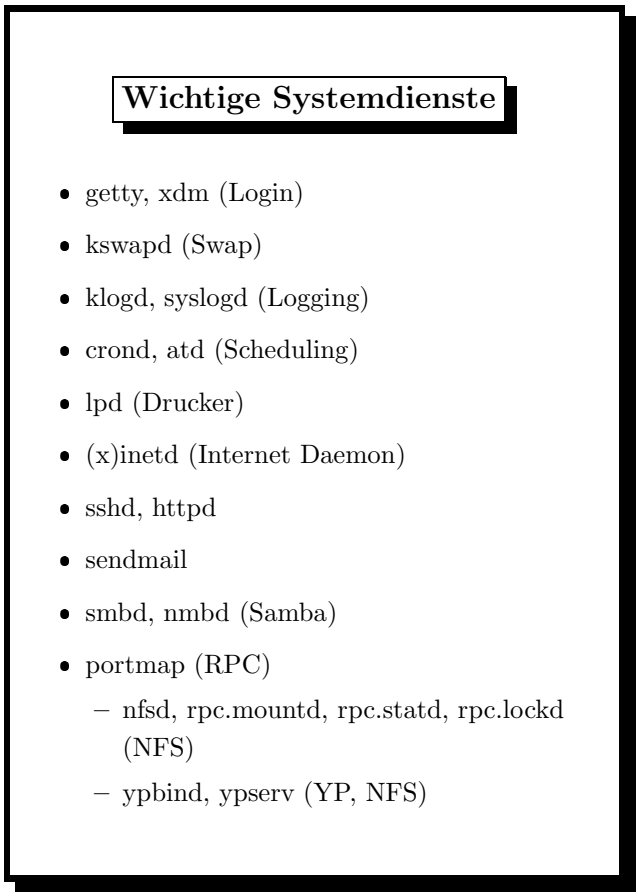
- **autofs**: ist ein Dienst, um lokale (Floppy, CD-Rom) oder Netzwerk-Laufwerke automatisch zu mounten, wenn sie benötigt werden.

- **apmd, acpid**
  sind die Daemonen, um über Advanced Power Management bzw. das Advanced Configuration and Power Interface verschiedenes zu automatisieren. Darunter fällt auf Laptops die Überwachung der Stromversorgung.

Alle oben angeführten Systemdienste und Daemonen werden am saubersten und komfortabelsten über die entsprechenden Startup- und Shutdown-Skripts in `/etc/(rc.d/)init.d` gesteuert. Die meisten Skripts können (von root) mit den Optionen `start, stop, restart`

sowie manchmal `status` oder `reload` aufgerufen werden; die Bedeutungen sind selbsterklärend (siehe auch Folie 146). Einer Reihe von Daemonen können mittels `kill` verschiedene Signale übermittelt werden. Detaillierte Auskunft darüber findet man auf den entsprechenden Manpages.

Welche Prozesse und Daemonen laufen, kann mittels `ps aux(ww)` abgefragt werden (siehe dazu auch Abschnitt 6.3).



Folie 145

Zum Konfigurieren der einzelnen Runlevels ist es im Prinzip notwendig, entsprechende Links in den Directories anzulegen. Um das nicht händisch machen zu müssen, stehen dem Administrator eine Reihe von Tools zur Verfügung. Diese und ihre Syntax werden auf Folie 147 vorgestellt.

Bei einem neu installierten System ist aus Sicherheitsgründen vor allem darauf zu achten, daß keine unbenötigten (und hier vor allem Netzwerk-) Dienste mit unzureichender Konfiguration laufen. Installiert man z.B. den WWW-Server, so wird er per default in den Runlevels 3 und 5 aktiviert. Er läuft dann mit einer Defaultkonfiguration, die oft sicherheitsrelevante Fehler oder Lücken aufweist. Dies gilt vor allem auch für den Daemon `sendmail`, den `inetd` und die `nfs`-Dienste. Es ist unbedingt zu empfehlen, nur jene Dienste laufen zu lassen, die tatsächlich auch benötigt werden, das minimiert einerseits das Risiko erfolgreicher Hackerattacken und erhöht andererseits die Performance des Systems.

Abschließend erwähnen wir noch, daß für die allermeisten Konfigurationsarbeiten vor allem dem unerfahrenen Benutzer eine Reihe von grafischen Tools zur Verfügung stehen; S.u.S.E.-Systeme werden vollständig über `yast` konfiguriert.

Systemdienste kontrollieren (1)

- Starten, Stoppen,...
  - `/etc/(rc.d/)init.d/dienst` mit Optionen `start`, `stop`, `restart`, `reload` oder `status` z.B. `/etc/rc.d/init.d/sendmail restart`
  - `kill [signal]` z.B. HUP, SIGHUP, ...
- Anzeigen
  `ps aux, ps auxww | less`

Folie 146

Systemdienste kontrollieren (1)

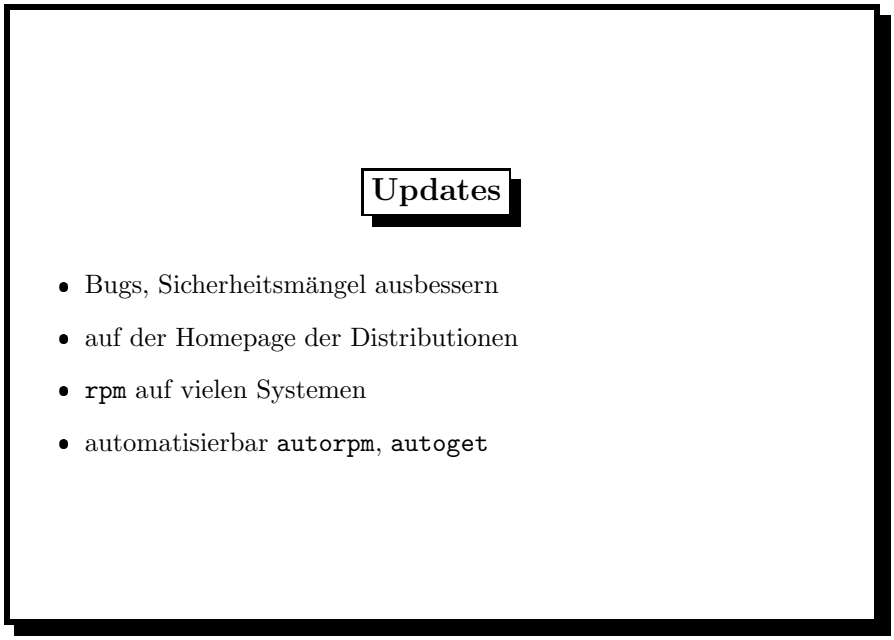
- Textkonsole (bei Fedora)
  - `chkconfig --level 123456 Dienst on|off`
  - `chkconfig --list Dienst`
- oder grafisch; viele Tools, Desktopumgebungen haben meist eines integriert.

Folie 147

9.3 Softwareinstallation

Die meisten Linux-Distributionen verfügen über ein Tool, das es erlaubt auf einfache und überschaubare Art und Weise Softwarepakete zu installieren, zu deinstallieren und upzugraden (man Spricht auch von *Software-Paketmanagement*). Ein weit verbreitetes Tool ist der ursprünglich von RedHat entwickelte *rpm Package Manager* `rpm`, der aber auch von vielen anderen Distributionen verwendet wird. Wir geben hier eine Kurzeinführung in die Benutzung des `rpm`; einige der fortgeschrittenen Funktionsweisen und Optionen behandeln wir im zweiten Teil der Vorlesung. Hier geht es vor allem um das Upgrade von Softwarepaketen, das aus folgendem Grund auch bei neuinstallierten Systemen von großer Wichtigkeit ist:

Da Linux ein Open-Source-Betriebssystem ist und der Quellcode ständig von einer Vielzahl vom Programmierern weiterentwickelt wird, werden in relativ kurzen Abständen vor allem sicherheitsrelevante Bugs bekannt. Die Distributoren geben dann in der Regel innerhalb weniger Tage neue Softwarepakete (sogenannte *Updates*) heraus, die diese Fehler korrigieren. Diese können von der Homepage der entsprechenden Distribution heruntergeladen werden. Es ist von großer Wichtigkeit, zumindest grobe Bugs und Sicherheitsmängel durch Einspielen der Updates auszuräumen. Diese Prozedur, die auch automatisiert werden kann, wird durch das Verwenden des `rpm` wesentlich erleichert. Dieser verwendet ein eigenes (überraschenderweise `rpm` genanntes) Format der Softwarepakete, das neben der leichten Installation auch ermöglicht, Abhängigkeiten zwischen einzelnen Paketen zu berücksichtigen. Darüberhinaus wird am System eine `rpm`-Datenbank angelegt, in der Informationen über die installierten Pakete gespeichert werden; das Speichern der Versionsnummer eines `rpm`-Pakets, die auch am Dateinamen ersichtlich ist, ermöglicht ein einfaches Upgrade der Pakete. Die Basis-Syntax von `rpm` wird auf Folie 149 erklärt.



Folie 148

RedHat Package Manager-Basics

- `rpm -i|U|F[hv] Paket [Optionen]`
  - `-i|U|F` install, upgrade, freshen
  - `Paket` z.B. `initscripts-4.70-1math`
  - Optionen: `--test`, `--nodeps` `--force`
- Files z.B. `apache-1.3.12-2.i386.rpm`

Folie 149