# Linux Einführung

Vorlesung - Systeme I
WS 07/08

Lehrstuhl für Kommunikationssysteme

24. Oktober 2007

Autor: Dirk von Suchodoletz

mit Beiträgen von:
Frank Schwichtenberg
Daniel van Ross
Tim Oliver Kaiser
Steffen Wagner
Stefan Koospal

Korrekturgelesen und zusammengestellt von:
Antonia Blanke

# Inhaltsverzeichnis

# Kapitel 1

# Überblick

Die Unterlagen "Linux Einführung" wurden aus Anlass mehrerer Vorlesungen und Fortbildungskurse zum Thema Linux und Systemadministration zusammengestellt. Das Projekt startete vor ungefähr fünf Jahren und hat in der Zwischenzeit etliche Erweiterungen und Aktualisierungen erfahren. Diese Kursunterlagen sind inzwischen ein gemeinsames Projekt des Rechenzentrums der Universität Freiburg und der mathematischen Fakultät Göttingen.

## 1.1 Zu diesen Unterlagen

Die Vorlesung "Systeme" soll in die Begriffswelt der Betriebssysteme einführen. Es wird dabei in einzelnen Lehreinheiten auf die Aspekte verschiedener Prozessoren und Rechnerarchitekturen, verwendete Bussysteme, Speichermedien, Beispielbetriebssysteme und -applikationen eingegangen. In dieser Vorlesung geht es um Grundlagen, für fast alle angesprochenen Themengebiete findet der interessierte Hörer eine eigene Spezialvorlesung. Diese Veranstaltung liefert einen Einstieg für ein mehrsemestriges Studium der Informatik im Bachelor-Studiengang, kann aber auch im Rahmen eines Nebenfachstudiums gehört werden.

Die Vorlesung versteht sich nicht als Programmierkurs und leistet dies auch nicht - dieses bietet die Veranstaltung von Prof. Thiemann: "Informatik I". Programmieren lernt man überhaupt nicht durch Zuhören oder aus Büchern, wenn diese jedoch wichtige Hilfsmittel sind. Eigenes Probieren und eigene Anwendung sind der Schlüssel: z.B. in der zur Veranstaltung angebotenen Übung, in speziellen Kursen, und durch Versuch und Irrtum an der Maschine selbst.

Linux ist ein gutes Beispiel eines Betriebssystems, weshalb es in den praktischen Übungen zur Vorlesung eine prominente Rolle spielen wird. Es ist frei im Quellcode verfügbar und bietet nicht nur deshalb gute Anschauung und tiefere Einblicke in die Konzepte eines Betriebssystems. Da vermutlich noch nicht alle dieses System vorher benutzt haben, sollen sowohl der Einführungskurs als auch die praktischen Übungen dieses nachholen oder vertiefen.

Das Skript fällt deutlich umfangreicher aus, als der Stoff sein kann, der sich in insgesamt 15 - 20 Übungsstunden sinnvoll vermitteln läßt. Es werden nicht alle Kapitel im Kurs vorkommen und aus einigen Kapiteln spielen auch nur die ersten Abschnitte eine Rolle. Die restlichen Teile sind als Referenz und zum Weiterlesen gedacht. Auf der anderen Seite können in den Übungsstunden auch Aspekte vorkommen, die nicht durch diese Unterlagen abgedeckt sind.

Die meisten Kapitel enthalten am Ende eine ganze Reihe von Übungsaufgaben. Diese sollen einen Anhaltspunkt bilden, was in der Abschlussklausur des Kurses vorkommen

könnte. Zum Teil entstammen die Übungen einfach vorangegangenen Klausuren. Der Level der Übungsaufgaben fällt durchaus unterschiedlich aus. Nicht alle Aufgaben sind für den Einstieg geeignet. Die restlichen Aufgaben zeigen jedoch welche Fragestellungen es sonst noch so geben könnte :-)

Diese Unterlagen werden zwar in regelmäßigen Abständen aktualisiert, trotzdem sind Fehler nicht auszuschliessen. In diesem Fall mich einfach kurz benachtichtigen.[1] Wir hoffen, dass diese Kursunterlagen den Einstieg in das Betriebssystem Linux erleichtern. Die Beispiele haben wir in den meisten Fällen selbst ausprobiert und Angaben aus Konfigurationsdateien stammen aus der eigenen Praxis. Trotzdem können auch hier Fehler enthalten sein, die wir dann zu entschuldigen bitten. Dieses Skript ist nicht immer in allen Teile auf dem gleichen jüngsten Stand.

Die (Kurz-)beschreibung einiger Programme kommt durchaus mehrfach vor: Einmal im laufenden Text, wenn einzelne Bereiche eines Linux-Systems mehr oder weniger ausführlich behandelt werden und in der Liste wichtiger Kommandos am Ende dieses Skriptes. Das ist durchaus so beabsichtigt! Um aber zuviele Wiederholungen zu vermeiden, wurde versucht, jeweils verschiedene Beispiele zu wählen, wenn welche angegeben werden. Ebenso wiederholen sich zum Teil die Beschreibungen wichtiger Systemkomponenten: So spielt die Shell natürlich beim Erst-Login eine wesentliche Rolle und wird später nochmals ausführlich behandelt - trotzdem würden wir nicht auf die ersten einführenden Worte verzichten wollen :-))

Das LSfKS-Team!!

## 1.2 Geschichte

Damit nicht schon die erste Stunde in Stress ausartet, hat es sich eingebürgert, eine kurze Geschichte oder ähnliches vorzuschalten: 1991 kauft sich der finnische Student Linus Torvalds einen 386er PC, um mit den Möglichkeiten des 80386 Prozessors[2] zu experimentieren. Zunächst entwickelte er Linux unter Minix, einem Lehrbetriebssystem von A. Tanenbaum (von diesem stammt der Ausspruch: "Linux is obsolete". Sonst ist er aber eher für ein ordentliches Netzwerkbuch bekannt). Er schrieb hierzu in der Newsgroup `comp.os.minix`: *" ... 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."*

Aus den ersten Versuchen wird schnell ein einfacher, aber brauchbarer multitasking fähiger Betriebssystemkern. "Linux" ist genaugenommen ein Betriebssystemkern (engl. "kernel"). Der Name geht auf "Linus' Unix" zurück und zeigt damit an, dass Linux ein unix-artiger Kernel ist. 1992 stellt Linus Torvalds die Kernel-Version 0.12 über anonymes FTP ins Netz, womit eine breitere Zahl von Testern Zugriff erhält. Schnell wird die Zahl der per Mail kommunizierenden Tester und Anwender so gross, dass der nötige Austausch nicht mehr allein über diesen Weg zu bewältigen ist. Zur besseren Kommunikation wird

[1] beispielsweise per Mail an *semaan@... oder dsuchod@rz.uni-freiburg.de*

[2] Der 80386 ist der 32 bit Nachfolger des 16 bittigen 286ers. Er bringt wesentliche "Systemeigenschaften" mit, wie Speicherschutz gegen unberechtigte oder fehlerhafte Zugriffe im Multi-User-Betrieb, Systemschutz gegen Verändern von Programmen und Daten. Weiterhin kennt er einen Real(Address)Mode, in dem der Prozossor als schneller "32 bit 8086" arbeitet. Hinzu kommen Protected(Virtual Address) Mode mit Speicherverwaltung und 4-Ebenen Schutzmechanismus (Privilegien-Level der Programmausführung). Linux ist deshalb auch nicht ohne wesentliche Einschränkungen auf 286er CPUs lauffähig.

die Usenet-Gruppe `alt.os.linux` eingerichtet. Newsgroups waren vor der Ausbreitung des World Wide Web das dominierende Diskussions- und Kommunikationsmedium. Die eingerichtete Newsgroup schuf das Forum für eine explosionsartige Weiterentwicklung des Systems im ganzen Internet. Linus Torvalds koordiniert fortan die Weiterentwicklung des Kernels.

Nachdem der C-Compiler "gcc" und weitere wichtige Entwicklungswerkzeuge unter Linux liefen, konnte Linux unter Linux weiterentwickelt werden. Der Betriebssystemkern wurde von Grund auf neu programmiert und inzwischen modularisiert.

1994 wird die Version 1.0 des Kernels herausgebracht. Nun ist dieser bereits netzwerkfähig und die Zahl seiner Anwender steigt auf 100.000. Ein weiterer wichtiger Schritt geschieht ebenfalls in diesem Jahr: Die Anpassung einer grafischen Benutzerschnittstelle auf Linux. Diese wird von einer weiteren Non-Profit-Gruppe, dem XFree86-Projekt, geleistet. Torvalds stellt nun den Quelltext des Linux-Kernels offiziell unter die GPL. Die Marke "Linux" ist inzwischen in vielen Ländern eingetragenes Warenzeichen von Linus Torvalds. Somit ist die freie Existenz und Weiterentwicklung von Linux gesichert. In der Zwischenzeit erfolgt die Portierung auf die Plattformen Digital (DEC) und Sun Sparc.

Zwei Jahre nach dem Release der Version 1.0 wird die Majornumber um eins erhöht und die Version 2.0 des Linux-Kernels freigegeben. Dieser ist nun in der Lage mehrere Prozessoren gleichzeitig anzusprechen. Mit diesem Schritt verläßt Linux langsam den Experimentalstatus und wird zu einer ernstzunehmenden Betriebssystemalternative. Auf der PC-Plattform gehört es neben den verschiedenen BSDs zur Software mit der besten TCP/IP-Unterstützung. Erste Firmen beginnen ihre Produkte für Linux zu portieren, womit die Zahl der kommerziell verfügbaren Software-Pakete steigt.

Im Jahre 1998 startet ein wesentlicher Schritt zur Verbreitung von Linux für den Desktop: Das KDE-Projekt[3] wird aus der Taufe gehoben. Ein Jahr später kommt mit GNOME[4] ein weiteres ernstzunehmendes Desktop-Projekt hinzu. In diesem Jahr erscheint auch die Kernel-Version 2.2 mit verbessertem Support für symmetrisches Multiprozessing (SMP) und renoviertem Netzwerk-Code. Eine erste Soundunterstützung realisiert ein Jahr später das Open Sound System (OSS), außerdem wird Samba, zur Einbindung von Windows-Netzwerken, in einer neuen Version 2.0 veröffentlicht.

Das Jahr 2000 bringt Linux einen weiteren Schritt dem Desktop näher: XFree86 wird in der Version 4.0 veröffentlicht und KDE 2.0 erscheint. 2001 wird die derzeitig aktuelle Kernel-Linie 2.4.X eröffnet. Der Kernel kann nun bis zu 64 GByte RAM ansprechen und unterstützt 64 Bit-Dateisysteme. Ebenso sind USB-Unterstützung und Journaling Filesysteme realisiert; Samba erscheint in der Version 2.2. Die nächste Runde der Desktop-Entwicklung erfolgt 2002 mit der Verfügbarkeit von KDE 3.0 und GNOME 2.0. Die OpenSource-Projekte Mozilla und OpenOffice erscheinen in stabilen Releases.

Im Jahr 2004 wird die Entwicklung der Grafikserver wieder von X.org übernommen nachdem es einige Querelen im XFree86-Team gegeben hatte. Ungeachtet dessen steht KDE ab Mitte des Jahres in der Version 3.3 bereit. Es bietet nun die ganzen Annehmlichkeiten eingeschlossen Plug&Play von Speicherkarten, USB-Geräten und ähnlichen, wie man es von einer modernen grafischen Oberfläche erwartet. Samba gibts in der Version 3.0, die das Management der Backends vereinfacht und mit LDAP und AFS zusammenarbeiten kann. Der Linux-Kernel wird seit Anfang des Jahres von fast allen neu erschienenen Distributionen in der Version 2.6 ausgeliefert.

Zum jetzigen Zeitpunkt in 2005 ist die Kernel-Version 2.6.13 aktuell und KDE befindet

[3] Die Einstiegsseite in das KDE-Framework findet sich auf http://www.kde.org. Inzwischen gibt es eine ganze Reihe von Büchern zur Programmierung und Benutzung von KDE.

[4] Gnome Desktop Environment - http://www.gnome.org

sich auf dem besten Wege zu Version 4.0 und zieht damit dem zugrundeliegenden QT-Framework[5] hinterher.

Die Weiterentwicklung fand und findet unter Beteiligung von vielen interessierten Programmierern im Internet statt; Linus Torvalds, der inzwischen bei der Prozessorschmiede Transmeta arbeitet, ist und war nie der einzige Entwickler.

## 1.3 Linux/Unix

Im allgemeinen Sprachgebrauch wird nicht zwischen dem Betriebssystemkern, dem Betriebssytem oder den Linux-Distributionen[6] unterschieden, was manchmal für Verwirrung sorgt ... Der Betriebssystemkern von Linux ist in der Programmiersprache C geschrieben und liegt im Quelltext vor. Ohne zusätzliche Programme (Software) ist ein Betriebssystemkern ziemlich nutzlos. Um die Fähigkeiten des Betriebssystemkerns komfortabel und sinnvoll nutzen zu können, benötigt man mindestens:

- eine Reihe von Programmen für systemnahe Aufgaben (Systemsoftware)
- Programme zur Erkennung und Behebung von Fehlern
- den Zugriffsschutz, d.h. die Abfrage des Geheimwortes (Passwort)
- eine Befehlszeile zum Start weiterer Programme
- ein System zum automatischen Start der oben genannten Programme

Diese absolut notwendigen Komponenten bezeichnet man landläufig als "Betriebssystem". Derzeit wird in den meisten Fällen die Software des GNU-Projektes[7] in Verbindung mit dem Linux-Kernel verwendet. Das GNU-Projekt entwickelt bereits seit 1984 freie Software. Heute sind beide Komponenten, der Linux-Kernel und die GNU-Betriebssoftware, kaum noch voneinander zu trennen. Beide Teile haben sich gegenseitig vorangebracht und gegenseitig befruchtet. Korrekterweise müsste man also eigentlich, wenn man das Betriebssystem nennt, von "GNU/Linux" sprechen, da hier immer der Kernel (Linux) und die OS-Softwaretools (GNU) gemeinsam gemeint sind. Umgangssprachlich auch dem Drang nach Abkürzung folgend vereint dem Begriff "Linux" die Kombination aus Kernel und Betriebssystem. Dies soll keinesfalls eine Herabsetzung des GNU-Projektes sein, welches die wesentlichen Grundlagen überhaupt erst geliefert hat. Eher trägt die Abkürzung der Tatsache Rechnung, dass diese Benennung heutzutage von den meisten Anwendern und Journalisten verstanden und benutzt wird.

Die Installation des Linux-Betriebssystems ist weitgehend automatisiert und menügeführt. "Linux-Distributionen" sind inzwischen einfach zu installierende Sammlungen von Programmen "für Linux"; sie enthalten außerdem das Betriebssystem in der aktuellen Fassung. Linux-Distributionen werden von verschiedenen Firmen oder Interessengruppen in leicht unterschiedlicher und wechselnder Zusammensetzung, bzw. Qualität angeboten.

Es gibt keine "offizielle" Linux-Distribution und somit läßt sich von Linux nicht sprechen wie von MacOS, OS/2 oder den verschiedenen Windows-Versionen. Viele der Systemprogramme, die mit heutigen Linux-Distributionen ausgeliefert werden, gab es bereits vor

[5] QT ist ein Widget-Set für grafische Benutzeroberflächen. Es wird "cute" ausgesprochen und von der Firma Trolltech in Norwegen entwickelt: http://www.trolltech.no

[6] Hiervon gibt es einge ganze Menge - einige Zahlen sprechen von über 200. Zu den großen zählen sicherlich RedHat, SuSE, Debian, Gentoo, ...

[7] http://www.gnu.org

Linux. Sie wurden meistens von Systemadministratoren und -entwicklern geschrieben, um bestimmte Aufgaben zu erleichtern oder die mitgelieferten Systemprogramme der Unix-Hersteller zu verbessern und in ihrem Funktionsumfang zu erweitern. Diese Programme wurden zumeist im Quelltext (Source Code) weitergegeben und meist unter die GNU Public License gestellt.

Das Internet schaffte die Infrastruktur zur sekundenschnellen Kommunikation der Software-Entwickler; erst dadurch wurden so große Projekte wie Linux überhaupt möglich.

## 1.4 Linux im Vergleich zu Windows

Linux ist wie andere Unixe auch ein Multi-User-Betriebssystem. Dieses mag für den Einsatz am heimischen Rechner, wo man/frau sowieso die einzige Person am Einschalter ist, erstmal etwas übertrieben sein, aber wenn man sich das Sicherheitskonzept ansieht, wird man sehen, dass die Unterteilung in mehrere Benutzer durchaus Sinn macht. Inzwischen entwickelt sich auch das Microsoft-Universum immer stärker in Richtung Serverbetriebssystem, welches mit vielen Einschränkungen, die im Folgenden genannt werden, aufräumt. Trotzdem bleiben eine Reihe von Wünschen offen und sei es nur, für ein bestimmtes Problem einen Blick in den Sourcecode werfen zu können. Mit der Einführung der Serverfähigkeit in der Microsoft-Welt geht im Gegenzug jedoch ein Teil des "Komforts" verloren, der jedem Benutzer erlaubte, wirklich alles (inklusive des Total-GAUs) mit seinem Windows anzustellen. Nun bekommt man dafür schonmal schnell ein Problem, ein bekanntes Brennprogramm unter Windows XP für Normalbenutzer bereitzustellen.

Mit Windows in Form der Versionen 95/98/ME hat wahrscheinlich jede(r) schon einmal gearbeitet. Natürlich ist Linux ein ganz anderes System. Hier sollen einmal die wichtigsten Unterschiede und Besonderheiten erläutert werden.

**"Echtes" Multitasking**: Linux ist ein Mehrbenutzer System, das heißt es können sich mehrere Benutzer einen Computer teilen und das sogar gleichzeitig. Man kann nämlich an einen Computer viele Terminals anschließen oder auch über ein Netzwerk mit einer Terminalemulation vom PC aus auf diesen Computer zugreifen. Jeder Benutzer teilt sich CPU (Rechenleistung) und Arbeitsspeicher mit den anderen Benutzern und deren Programmen. Wobei eine ganze Reihe von Benutzern rein virtuelle Systembenutzer sind, die einfach nur für einen bestimmten Dienst "verantwortlich" zeichen. In Wirklichkeit sind es keine eigenen Identitäten. Die Dienste werden vom Systemadministrator verwaltet, laufen aber aus Sicherheitsgründen nicht unter der "ich-darf-alles" ID. Dieses Konzept findet sich ebenso bei den modernen Windows-Versionen, wie XP.

Das Risiko, jeden Benutzer jede Systemdatei ändern zu lassen, kann man sich an der seit etlichen Jahren ungelösten Virenfrage für DOS/Windowssysteme ansehen. Deshalb gibt es mindestens einen priviligierten Benutzer "root" auf einem Linuxsystem und weitere unpriviligierte Benutzer für normale User- bzw. Systemprozesse. So wird sichergestellt, dass nicht unbedachterweise Konfigurationen oder Systemdateien verändert, manipuliert oder gelöscht werden. Viren, Würmer und Trojaner sind jedoch nicht auf die Windowswelt beschränkt: Zur Zeit macht es einfach noch nicht viel Sinn einen Trojaner zu basteln, der eine Linux-Box zum Spam-Relay macht - das heisst aber nicht, dass es nicht möglich wäre. Geht die Popularisierung weiter, wird Linux auch in diesem Bereich der "Software-Versorgung" nachziehen, wenn auch die Rahmenbedingungen etwas anders stehen. Schlampig gewartete Linux-Systeme können als Angriffsziel sogar viel spannender sein: Hier geht vieles gleich, wo bei Windows erst noch die Software mühsam nachinstalliert werden muss.

**Dateisysteme und Speichermedien:** Festplatten, ihre Partitionen und eine Vielzahl von Wechselmedien existieren natürlich auch unter Linux. Es gibt nur keine Laufwerks-

buchstaben, sondern einen Verzeichnisbaum. Dieser Verzeichnisbaum fängt mit /, daher auch die Bezeichnung "root" (Wurzel). Eventuelle "Laufwerke" sind an den Übergängen zu Unterverzeichnissen zu finden.[8] Unter Unix könnten zum Beispiel */etc* und */usr* auf der ersten Festplatte sein und */home* könnte sich auf einer weiteren befinden. So lassen sich Probleme "überlaufender" Partitionen meistens schnell beheben, indem in stark belegte Bereiche einfach eine weitere Partition eingelinkt wird. Sollte generell die Festplatte zu klein werden, ist eine einfache Kopie des Dateisystems kein Problen: Einzig der Bootsektor muss anschliessend neu geschrieben werden.

Im eben Gezeigten wird auch schon deutlich, dass der DOS/Windows-gewohnte \ (meistens im allgemeinen Sprachgebrauch nur als "Backslash" bezeichnet) bei den meisten anderen Betriebssystemen ein / ("Slash") ist. Der Backslash besitzt in der Unix-Shell eine ganz andere Bedeutung.[9]

Dateisysteme gibt es viele. Linux kann auf DOS oder Windows-(VFAT-)formatierte Festplatten ebenso zugreifen wie auf Atari-, CD-ROM-, Apple- oder Windows-NT-Dateisysteme. Bei Linuxinstallationen wird jedoch hauptsächlich folgende Dateisystem verwendet: EXT3 (EXT2 mit Journaling-Erweiterung), ReiserFS, Reiser-4 oder XFS, welche lange Dateinamen und Zuordnung von Benutzern/Besitzern und Dateien ermöglicht.

Achtung Falle (oder Selbstverständlichkeit?!): Linux unterscheidet in Groß- und Kleinschreibung! */home/ich/Hallo* und */home/ich/hallo* sind zwei verschiedene Dateien! Probleme können auftreten, wenn unter Linux ein Dateisystem ohne diese Unterscheidung verfügbar gemacht wird, z.B. ein gemountet oder über das Netz eingebunden wird. Hier findet dann einstellbar eine generelle Umstellung auf Groß- oder Kleinschreibung statt.

Wichtiger Hinweis: Bevor der Computer ausgeschaltet wird, muss man ihm zuerst "gute Nacht" (oder weniger romantisch "reicht jetzt") sagen, d.h. das Sitzungsende signalisieren, damit Linux eventuelle im Arbeitsspeicher gepufferte Dateien auf die Festplatten schreibt und sich korrekt beendet[10]. Zum Teil werden dabei die ATX-Features der Maschine ausgenutzt, jedoch sind nicht immer alle Möglichkeiten unterstützt. Bei entsprechender Kernel-Unterstützung und passender Konfiguration sorgt ein Druck auf den Power-Off-Knopf des Rechners für eine passende Signalisierung zum Shutdown.

**Geräte und Treiber:** Unter Linux findet sich für jedes Gerät im Verzeichnis */dev* eine Datei. */dev/ttyS0* ist zum Beispiel die erste serielle Schnittstelle. */dev/hda* bezeichnet die Master-Festplatte am ersten IDE-Controller, */dev/hda1* verweist auf die erste Partition dieser Platte. Dies scheint auf den ersten Blick völliger Blödsinn zu sein, erleichtert aber die Programmierung wesentlich. Dem Programmierer kann es egal sein, ob die Ausgabe seines Programms auf den Bildschirm, in eine Datei oder auf ein Drucker ausgegeben wird. Auch Treiber gibt es unter Linux. Sie werden entweder fest in den Systemkern (Kernel) eincompiliert oder als Zusatzmodule geladen[11].

## 1.5 Anmerkungen

Alle Kommandos werden üblicherweise ohne Nachfrage ausgeführt, also sollte gewisse Vorsicht geboten oder Toleranz gegenüber verschwundenen Daten angebracht sein. Deshalb bietet es sich gerade für Neulinge an für die ersten Gehversuche CD-Linuxe, wie Knop-

[8] beispielsweise */media/cdrom* oder */media/floppy*, distributionsabhängig eventuell in anderen Unterverzeichnissen ...

[9] Er wird üblicherweise zum "Escapen" (Verstecken ihrer Bedeutung) von Zeichen verwendet, die sonst von der Shell speziell interpretiert werden würden.

[10] Dieses ist evtl. bereits von WindowsNT/2000/XP bekannt

[11] Dieses wird - wie auch vieles andere - in einem weiteren Abschnitt ausführlicher behandelt

pix[12] an. Eine andere Alternative ist die Benutzung von VMware[13] im nonpersistenten Modus, d.h. alle Daten werden in einen Zwischenpuffer geschrieben, statt direkt die virtuelle Festplatte zu verändern. Liegen wichtige Daten auf dem System, so sollten regelmässige Backups erfolgen (das gilt für alle Betriebssysteme). Viele Befehle zur Systemadministration sind dem Superuser ("root") vorbehalten.

Zur Erhöhung der Lesbarkeit werden alle ausführbaren Dateien oder Systembefehle durch **Fettdruck**, z.B. **ls** oder **dhcpd** hervorgehoben. Alle Konfigurationsdateien oder Verzeichnisse werden *italic* gesetzt, z.B. */home/name*. Dieses gilt ebenso zur Kennzeichnung von Rechnernamen und IP-Adressen. Beispiele für Kommandoeingaben, wie `last, sort, less` werden in Courier gesetzt.

Tasten, beispielsweise der einfache Druck auf das "d" ohne Modifier werden durch "[d]" dargestellt. Kombinationen durch "[Shift]-[d]", wenn sie gleichzeitig, "[y],[y]", wenn sie nacheinander gedrückt werden sollen.

Variablennamen, in erster Linie Umgebungsvariablen der Shell werden einfach nur in Kapitalien angegeben: $UID oder $HOME. Das Dollarzeichen $ kennzeichnet in der Shell und vielen Programmiersprachen eine Variable. Selbstdefinierte Variablen bestehen oft aus Kleinbuchstaben, z.B. $test.

Um den Lesefluss nicht stark zu stören, werden viele Erläuterungen als Fussnoten angefügt. Auch alle Links zu den entsprechenden Webseiten sind hier zu finden.

## 1.6 GNU und Free Software Foundation

GNU-Software unterliegt sämtlich dem GNU-Copyright! Dieses GNU-Copyright ist auch als COPYLEFT bekannt, da der Inhalt dieses Titels besagt, dass diese Software frei kopierbar ist, nicht verkauft werden darf, und dass entsprechender Quellcode veröffentlicht werden muss. Ferner ist alles, was mit GNU-Produkten erstellt wird (GNU-Teile enthält, z.B. Libraries nach einem Compilevorgang), selbst wieder ein GNU-Produkt. Ein Beispiel:

Meier schreibt ein nettes Programm. Er kompiliert es mit dem GNU-Compiler, den er kostenlos bekommen hat und der GNU-Libraries in das Compilat einbindet. Nun ist sein Programm (das Binary) ebenfalls GNU. Er darf es für sich behalten oder samt Quellcode weitergeben, ganz egal, er darf es nur nicht verkaufen und auch niemand sonst darf es. Bei einer (organisierten) Verteilung darf aber eine Gebühr für das Bereitstellen des Datenträgers erhoben werden. Das sind dann die Kosten einer Linux-Distribution.

Natürlich ist auf GNU/Linux-Systemen nicht nur Software lauffähig und einsetzbar, die unter der GPL steht; es gibt viele andere freie Lizenzen, unter denen Software stehen kann.

## 1.7 Begriffserklärungen - Allgemeine Grundlagen

Im folgenden werden einige Begriffe und Abkürzungen erläutert, die im Text häufig verwendet werden. Kurze Erläuterungen zu wichtigen Shell-Kommandos, Systembefehlen, Diensten und grafischen Programmen sind am Ende dieses Textes zu finden.

**Bibliothek** Eine Software-Bibliothek benennt eine Sammlung von wiederkehrenden Funktionen. Da viele Programme gleiche oder ähnliche Funktionen benötigen, wäre es sehr ineffektiv gleiche Teile immer wieder neu zu programmieren. Daher werden diese Teile in eine

[12] http://www.knopper.net - diese Spezialdistribution erregt regelmäßig viel Aufmerksamkeit und liegt recht oft den Heft-DVDs diverser Magazine bei. Es gibt inzwischen davon abgewandelte Distributionen, wie Kanotix, ...

[13] Virtueller PC, der 80386 Hardware in Software nachbildet

externe Datei ausgelagert. Da mehrere Programme darauf zugreifen, kann die Grösse des einzelnen Programms auf der Festplatte verkleinert werden. Fehlt die Bibliothek, funktioniert jedoch das gesamte Programm nicht mehr.

**Desktop** Der X-Server selbst bringt nur die Fähigkeit mit, Grafikausgaben (auch netzwerktransparent) zu realisieren. Es sind jedoch zusätzliche Programme notwendig, um den Arbeitskomfort zu realisieren. Dazu dient ein Desktop wie KDE oder GNOME. Dieser ermöglicht das Arbeiten, wie man es von Apple-OS oder Windows her kennt. Man verfügt über eine Arbeitsfläche mit Fenstern und kann Programme über das "Anklicken" von Icons starten.

**FTP** Das File Transfer Protocal ist eines der ältesten Möglichkeiten, mittels TCP/IP Dateien zwischen Rechnern zu kopieren. Es verwendet das verbindungsorientierte TCP und verwendet Port 23. Ein Nachteil von FTP ist die unverschlüsselte Übertragung sowohl der Dateien, als auch des Passworts.

**Linux** bezeichnet eigentlich nur den Kernel. Ein Kernel ist ein Stück Software, das die Kommunikation zwischen den einzelnen Hardwarekomponenten und den Anwenderprogrammen implementiert. Das mag trivial klingen, ist aber eine sehr komplexe Aufgabe. Jedes OS (Operating System) hat einen Kernel, nur werden die wenigsten nach dessen Namen benannt.

**NFS** Network File System. NFS ist ein UDP-basiertes Protokoll, das Dateisysteme über ein TCP/IP-Netzwerk zur Verfügung stellen kann.

**Perl** Perl ist eine freie interpretierte Skriptsprache, die sich im Bereich der Stringverarbeitung durchgesetzt hat. Sie steht unter allen gängigen Unix-Architekturen, aber auch unter Mac-OS und Windows zur Verfügung. Diese Programmiersprache kann durch Module erweitert werden. Inzwischen stehen Module für fast jeden Anwendungsfall[14] in weltweit zugänglichen Archiven[15] zur Verfügung.

**Server** Der Server ist in erster Linie ein Diensteanbieter im klassichen TCP/IP-Client-Server-Modell, d.h. er stellt, meistens zentral, bestimmte Funktionalitäten, wie Mail-, File- und Webdienste oder Applikationen zur Verfügung. Benutzer können sich an einem Server anmelden, werden aber nur in den seltensten Fällen physisch vor dem Gerät sitzen.

**Shell** (engl. für Muschel) Nach dem Einloggen befindet man sich in einer Shell. Dies ist ein Programm, das zwischen dem Benutzer und dem System arbeitet. Von dieser Aufgabe, dem Benutzer eine abgeschlossene Arbeitsumgebung zur Verfügung zu stellen, stammt der Name. Innerhalb der Shell hat man die Möglichkeit, Befehle und Programme aufzurufen. Zudem verfügt jede Shell über eine Programmiersprache, so dass Skripte zur Arbeitserleichterung geschrieben werden können.

**SSH** Secure Shell zur Verbindung zu einem anderen Rechner. Diese ist dem Telnet auf jeden Fall vorzuziehen, da sie verschlüsselt erfolgt. Das Programm auf der Serverseite heisst üblicherweise **sshd**, die Clientapplikation **ssh**.

[14] z.B. die Umsetzung von Netzwerkprotokollen oder die Schnittstellen zu bestimmten Anwendungen
[15] Das CPAN

**Telnet** Eines der ersten Protokolle der TCP/IP-Suite, um sich an entfernten Rechnern anmelden zu können. Telnet verwendet als Transportprotokoll TCP und arbeitet auf Port 21. Der Daemon, d.h. der Hintergrundprozess, der den Telnet-Dienst auf einem Rechner anbietet, heisst üblicherweise **(in.)telnetd** und wird meistens über den Internet-Super-Daemon **(x)inetd** gestartet. Die Clientapplikation heisst einfach **telnet**.

**X-Server** realisiert die Schnittstellen für das Graphical User Interface (GUI). Die grafische Oberfläche (unter Unix X-Server, X Window System oder X11 genannt) ist nicht Teil des Betriebssystems, sondern ein eigenständiges Programm.

**XDMCP** Das X display message control protocol steuert die Grafikschnittstelle auf Unix-Systemen. Diese Schnittstelle ist netzwerktransparent. Dabei erfolgt die Ausgabe der Grafikoberfläche des Servers lokal auf der Maschine. Die Benutzereingaben durch Tastatur und Maus werden über XDMCP an den Server weitergereicht.

# Kapitel 2

# Erste Schritte

Irgendwie muss es ja losgehen - wobei der Anfang von allem gerade in einem Unix/Linux-Kurs nicht fest bestimmt ist. Ausgehend vom Konzept des Persönlichen Computers (PC) geht ein Benutzer davon aus, dass man erstmal die Kiste einschalten muss, bevor es richtig losgehen kann. Bei Workstations - ein Rechnertyp mit dem die eher teure Klasse der Unix-Maschinen bezeichnet wurde - war es nicht üblich die Maschine erst starten zu müssen. Als klassische Multi-User-Rechner liefen sie 24 Stunden am Tag und standen mehr als einem Benutzer zur Verfügung.

Inzwischen kann jeder, gerade auch durch die Verbreitung von Linux und die Professionalisierung von Windows, seine eigene Workstation betreiben. Dann läuft die Kiste oft schon aus Lärm- und Energieverbrauchsgründen nicht mehr den ganzen Tag und schon beginnt der Start mit dem Druck auf den Power-On-Knopf.

Dieses Kapitel soll einen kurzen Überblick für einen ersten Einstieg nach Linux auf der Kommandozeile geben. Das Ganze geht natürlich auch über den grafischen Desktop - hier sind jedoch die Ähnlichkeiten zu anderen Desktopsystemen durchaus ausreichend, so dass es nicht so fürchterlich aufregend ist. Einige typische Konzepte von Linux-Systemen, wie die Benutzeranmeldung, die Multi-User-Fähigkeiten, die Prozess-Steuerung, das Rechtesystem werden erstmal nur am Rande gestreift. Tiefergehende Abhandlungen finden sich in den folgenden Kapiteln.

Irgendwie muss es ja losgehen - wobei der Anfang von allem gerade in einem Unix/Linux-Kurs nicht fest bestimmt ist. Ausgehend vom Konzept des Persönlichen Computers (PC) geht ein Benutzer davon aus, dass man erstmal die Kiste einschalten muss, bevor es richtig losgehen kann. Bei Workstations - ein Rechnertyp mit dem die eher teure Klasse der Unix-Maschinen bezeichnet wurde - war es nicht üblich die Maschine erst starten zu müssen. Als klassische Multi-User-Rechner liefen sie 24 Stunden am Tag und standen mehr als einem Benutzer zur Verfügung.

Inzwischen kann jeder, gerade auch durch die Verbreitung von Linux und die Professionalisierung von Windows, seine eigene Workstation betreiben. Dann läuft die Kiste oft schon aus Lärm- und Energieverbrauchsgründen nicht mehr den ganzen Tag und schon beginnt der Start mit dem Druck auf den Power-On-Knopf.

## 2.1 Hochfahren einer Linux-Box

Lange Zeit war es den Normalbenutzern[1] einer Unix-Maschine gar nicht möglich, das System zu starten oder herunterzufahren. Starten deshalb, weil die Rechner im Regelfall hinter verschlossenen Türen liefen und nur dem Systemverwalter physisch zugänglich waren. Und

[1] nichtprivilegierte Benutzer oder Nicht-Root-Benutzer

herunterfahren, weil es Unix (und auch Linux) seinen Benutzern verbietet, den Lauf des Systems ohne weiteres zu beenden. Auch dies bleibt dem Systemverwalter root vorbehalten.

Für ein Multiuser-Betriebssystem versteht sich diese Eigenschaft von selbst, schließlich arbeiten im Regelfall mehrere Benutzer auf einem Rechner. Hinzu kommt, dass der Rechner auch innerhalb eines Netzes für die Bereitstellung von Diensten zuständig sein kann, die natürlich ebenfalls beendet würden. Ein Benutzer kann oft gar nicht abschätzen, wieviele andere Benutzer von der Maschine abhängen, auf der er gerade arbeitet.

Inzwischen stehen Linux-Rechner gut erreichbar zu Hause oder unter dem eigenen Schreibtisch. Mit dem Zugriff spätestens auf die Stromzufuhr zum Rechner hat der Benutzer natürlich jede Möglichkeit des Eingriffs.

### 2.1.1 Standard-Bootvorgang

Üblicherweise arbeitet ein PC nach dem Start ersteinmal das BIOS ab, konfiguriert seine Komponenten und erkennt die bootfähigen Geräte. Dabei wird der Maschine vorgeschrieben in welcher Reihenfolge die Boot-Devices auf der Suche nach einer gültigen Bootsequenz abgeklappert werden sollen.

Hierzu gehört üblicherweise der Blick auf die Startsektoren von Festplatte, Diskettenlaufwerk, CD-Rom, DVD oder anderen Wechselmedien. Dort findet die Maschine hoffentlich eine Startsequenz vor, die oft zu einem Bootloader wie GRUB oder LILO[2] vor. Diese regeln dann das Weitere und laden den Kernel, der wiederum sein Rootdevice üblicherweise von einer Festplattenpartition einbindet. Dann kommt die Maschine meistens bis zur grafischen Login-Aufforderung hoch. Handelt es sich um einen Server oder eine Maschine, die eher nicht direkt sondern über das Netzwerk benutzt wird, erscheint ein Text-Login.

Alle Schritte die in der Zwischenzeit passieren, oft ist Geduld von durchaus bis zu zwei Minuten angesagt, haben mit der Initialisierung und den Runleveln der Maschine zu tun, welches ebenfalls einem eigenen Kapitel vorbehalten ist.

Etwas anders verhält es sich noch mit dem geordneten Herunterfahren. Dies ist eigentlich nach wie vor, z.B. mittels der Kommandos **shutdown**, **init** oder **halt**, dem Administrator vorbehalten. Viele Distributionen bieten mittlerweile jedoch im Rahmen eines grafischen Login-Managers die Möglichkeit, das System nach dem Abmelden über eine Schaltfläche auf dem Desktop herunterzufahren. Des weiteren funktioniert üblicherweise die Tastenkombination [Strg]-[Alt]-[Entf], wenn man einen Reboot auslösen möchte. Netterweise hat jedoch der ATX-Standard dafür gesorgt, dass der Einschalter nicht mehr gleich den Strom trennt, sondern vorher noch das Shutdown-Kommando an das Betriebssystem weiterreicht. Die meisten Linux-Distributionen reagieren auf dieses schon in der Grundeinstellung.

Der ganze Zinnober ist wie inzwischen bei neueren Windows-Versionen auch angeraten. Linuxserver reagieren unter Umständen ausgesprochen empfindlich, wenn sie nicht ordnungsgemäss beendet werden.

### 2.1.2 Booten über das Netz

Gerade in Umgebungen mit sehr vielen gleichartigen Arbeitsplätzen erlaubt Linux wesentliche Vereinfachungen der Administration, indem es aus dem Netzwerk von einem oder wenigen zentralen Servern gebootet werden kann. Der Lehrstuhl für Kommunikationssysteme[3] entwickelt seit einiger Zeit Konzepte zum effizienten Betrieb vieler gleichartiger Linuxmaschinen.[4]

[2] die Bootloader werden in einem eigenen Kapitel gesondert behandelt

[3] Homepage: http://www.ks.uni-freiburg.de

[4] Siehe hierzu auch: http://www.ks.uni-freiburg.de/projekte/ldc

Dieses ist beispielsweise in den Kursraumumgebungen des Rechenzentrums der Fall. Es hat den angenehmen Nebeneffekt, dass nicht viel kaputt gehen kann: Das Betriebssystem liegt nur-lesbar auf dem Server, so dass lokale Manipulationen an der Festplatte der Maschine keine negativen Auswirkungen auf die Benutzbarkeit der Maschine haben. So kann man im Notfall, wenn man das Gefühl hat, dass nichts mehr geht, auch einfach mal den Ausschalter betätigen, ohne Inkonsistenzen des Systems zu befürchten.

Diese Betriebsart von Linux-Maschinen wird als eine Art von "Linux Diskless Clients" (LDC) oder auch als "Diskless X Station" (DXS) bezeichnet. Die Dokumentation hierzu findet sich auf der Homepage des Lehrstuhls.

## 2.2 Die erste Sitzung

Sitzung kommt von Sitzen. Jetzt kommt schon die erste Entscheidung: Entweder man setzt sich an die "Konsole", also direkt an den Bildschirm des Linux Computers (vor die reale Hardware). Oder man geht "per **telnet**" oder besser (weil verschlüsselt) "per **ssh**", also von irgendeinem entfernten PC aus über Netzwerk an die Sache heran. Dann sitzt man jedoch nicht mehr direkt an der Hardware, was aber kein Problem darstellt, solange man nicht an bestimmte Komponenten, wie z.B. das DVD-Rom-Laufwerk heran möchte. Sollte kein Netz zur Verfügung stehen, erübrigt sich diese Entscheidung. An dieser Stelle sollte darauf hingewiesen werden, dass sich eine Netzwerk-Sitzung immer über das sogenannte Loopback-Netzwerk-Interface realisieren lässt. Denn das Loopback-Interface ist auf jedem Linux-Rechner unabhängig von einer realen Internet- oder LAN-Anbindung eingerichet. Das Einloggen geschieht dann mittels `ssh localhost` oder `ssh -l user 127.0.0.1`. In diesem Beispiel sieht man bereits ein wichtiges Charakteristikum einer Linux-Maschine. Selbst wenn sie nicht an ein Netzwerk angeschlossen ist, steht das Netzwerkprotokoll TCP/IP zur Verfügung. Es bildet für viele Dienste überhaupt die Grundlage ihrer Funktion. Weiterhin kann ein Rechner fast immer auf zwei Wegen angesprochen werden: Die direkte Form geschieht über die IP-Adresse, im Beispiel über die *127.0.0.1*. Meistens kann der Rechner auch über seinen Namen adressiert werden, dieses setzt jedoch die Möglichkeit der Namensauflösung (DNS), d.h. der Zuordnung einer IP-Adresse zu einem Namen, voraus. Der Rechnername für die Maschine kann beliebig festgelegt werden. Er kann aber unter Umständen nur auf der Maschine selbst bekannt sein. Zusätzlich heisst jede Maschine selbst auch noch "localhost". Localhost entspricht fast immer der IP-Nummer *127.0.0.1*.

### 2.2.1 Die Konsole

An dieser sitzt man/frau üblicherweise, wenn der Rechner als Workstation eingerichtet ist. Einige Server hingegen werden vielleicht nur über das Netzwerk erreichbar sein, weil sie meistens an gesonderten Orten stehen. Hierin liegt ein Vorteil von Linux, denn es macht fast keinen Unterschied, ob man direkt am Rechner sitzt oder sich über das Netzwerk einloggt. So könnte der Login-Prompt an einer Linuxmaschine aussehen:

```
WELCOME TO

 __      __  ___  __  __   __  __   __     __
|  |    |  ||    |  ||  | |  ||  |_|  |   |  |
|  |    |  ||       ||  | |  |       /    |__|
|  |___ |  ||  |    ||  |_|  | /  _        __
|______||__||__| |__|  _____/ |__| |__|   |__|

hermes.test.site                              console:tty1
```

```
hermes login:
```

Der Text oberhalb der Login-Aufforderung wird in der Datei */etc/issue* eingetragen. Mittels spezieller Control-Sequenzen kann der Name des Rechners und der Konsole automatisch ermittelt und übernommen werden. Das Login auf einem anderen Rechner über ein Netzwerk sollte immer mit **ssh** (Secure Shell) erfolgen, damit die eingegebenen Passwörter nicht abgehört werden können. Der Befehl **telnet** sollte nur in äussersten Notfällen zur Anwendung kommen.[5] Ein Login über das Netzwerk stellt man in der nachstehenden Weise her:

```
dirk@linux:~/lak> ssh -l dsuchod login.gwdg.de
dsuchod@login.gwdg.de's password:
```

Soll auch für das Netzwerk-Login ein kurzer Text oder Infoscreen erscheinen, muss hierfür die Konfigurationsdatei des SSH-Servers geeignet angepasst werden. Meistens stehen über SSH- oder Telnetverbindungen nicht alle Control-Sequenzen der Shell zur Verfügung. Hierzu jedoch später mehr.

### 2.2.2 Wichtige Tastenkombinationen

Wenn man direkt vor der Maschine sitzt, stehen einige spezielle Tastenkombinationen zur Verfügung. Eine ganze Reihe von Funktionen realisiert die Shell, wie beispielsweise die Cursorbewegungen. Die Tastenfolgen zur Shell-Bedienung funktionen in den meisten Fällen auch noch bei Remote-Logins.

**[Alt]-[F1]** bis **[Alt]-[F6]** Umschalten der virtuellen Text-Konsolen. Man hat mehrere virtuelle Bildschirme (Konsolen) und kann sich mehrfach am System anmelden und parallel arbeiten. Die Zahl der Textkonsolen wird in der */etc/inittab* definiert.

**[Alt]-[F7]** Grafische Oberfläche (meistens die nächste Konsole nach der "letzten" Textkonsole. Der Standard sieht fast immer so aus. Jedoch kann jede Maschine individuell über die */etc/inittab* konfiguriert werden.

**[Alt]-[F10]** Hier landen üblicherweise die aktuellen Syslogmeldungen, wie sie auch in die Systemlog-Datei */var/log/messages* geschrieben werden. Konfiguriert wird dies über die Datei */etc/syslog.conf*.

**[Alt]-[b]** je ein Wort rückwärts ("backward") mit dem Cursor bewegen.

**[Alt]-[f]** je ein Wort vorwärts ("forward") mit dem Cursor bewegen.

**[Strg]-[Alt]-[F1]** Umschalten von X11 aus auf die erste virtuelle Konsole.

**[Strg]-[Alt]-[Backspace]** Beenden der grafischen Oberfläche ohne den Umweg des Sessionmanagers. Diese Tastenkombination spricht direkt den X-Server (das Programm zur Steuerung des grafischen Displays) an und kann in dessen Konfigurationsdatei: */etc/X11/XF86Config* ausgeschaltet werden.

[5] leider wird auch bei neuesten Windowssystemen keine SecureShell automatisch installiert. Eine sinnvolle Anwendung hat Telnet jedoch immer noch zum Testen der Erreichbarkeit von TCP-basierten Netzwerkdiensten, siehe dazu das Kapitel zur Netzwerkanalyse.

**[Strg]-[Alt]-[Entf]** Reboot des Rechners (steht nicht unter der grafischen Oberfläche zur Verfügung). Diese Funktionsweise wird in der */etc/inittab* ein- oder ausgeschaltet.

**[Strg]-[k]** löscht bis zum Ende der Zeile.

**[Strg]-[l]** löscht den Bildschirm. Dieses kann gleichfalls durch die Eingabe des Kommandos **clear** erreicht werden.

**[Cursor]-Tasten** Editieren in der Kommandozeile sowie Wiederholen von Befehlen (History-Funktion der meisten Unix-Kommandointerpreter mit Pfeil-nach-Oben).

**[Tab]** Die Tabulatortaste dient der Kommando- bzw. Dateinamen-Ergänzung (Typecompletion, spezielle Funktion des Kommandointerpreters **bash**). Wird der Anfang eines Befehls oder Dateinamens gegeben, so kann dieser damit automatisch vervollständigt werden.

**[Strg]-[a]** oder **[Pos1]** springt an den Anfang einer Kommandozeile. Dies ist nützlich, wenn man feststellt, dass man sich am Anfang vertippt hat und nicht den weiten Weg mit dem Cursor zurücklegen möchte.

**[Strg]-[c]** Bricht die meisten Kommandos in ihrer Ausführung ab, so dass der zuvor eingegebene Befehl nicht weiter ausgeführt wird. Ausnahmen sind Dateibetrachter, z.B. das Kommando **less** in bestimmten Situationen oder der Editor **vi** und natürlich Programme mit grafischen Oberflächen, außer man ist in der aufrufenden Shell.

**[Strg]-[d]** beendet die Bash. Ist diese die Haupt-Shell (Shell, die vom Login-Prozess gestartet wurde - siehe die Ausgabe von `echo $SHLVL` und `echo $SHELL`), so loggt man sich aus oder schliesst im grafischen Modus das Terminalfenster. Diese Tastenkombination entspricht den Befehlen **logout** oder **exit**.

**[Strg]-[e]** oder **[Ende]** springt analog zu [Strg]-[a] an das Ende einer Kommandozeile.

**[Strg]-[r]** Rückwärts suchen in der Liste der bis dahin eingegebenen Befehle. Hierbei erfolgt mit jedem danach eingegebenen Zeichen die Suche nach dem nächsten passenden Eintrag in der Liste der Befehle.

**[Strg]-[z]** Stoppt laufende Prozesse, die aus der Shell aufgerufen wurden und gibt den Prompt an den Benutzer zurück.

**[Shift]-[PgUP]** Erlaubt das “Hochschieben” des Terminalbildschirms, um bereits nach oben herausgeschobene Bildschirmbereiche wieder sichtbar zu machen.

**[Shift]-[PgDOWN]** Gegenstück zum eben Genannten: Erlaubt das Blättern nach unten.

**[~],[.]** Oft arbeitet man SSH-Verbindungen entfernt auf anderen Rechnern. Wenn eine solche Verbindung hängt, kann man sie mit dieser Tastenfolge beenden. Sonst kann es sein, dass die Shell sehr lange unbenutzbar ist, bevor die Verbindung endgültig zurückgesetzt wird.

[~],[.] Ist nur relevant für SSH. Diese Tastenkombination liefert das Kommando-Interface zu SSH, um beispielsweise Tunnel aufzusetzen.

### 2.2.3 Login

Jede(r) BenutzerIn meldet sich mit einer Kombination aus Benutzername und Passwort an der Maschine an und bekommt erst dann eine Umgebung (Shell, siehe Kapitel 2.2.4, S.16) zur Kommunikation mit dem System zur Verfügung gestellt. Wie auch für Datei- und Verzeichnisnamen, so gilt auch hier, dass nach Gross- und Kleinschreibung bei Benutzername[6] und Passwort unterschieden wird. Die Kommandozeile wird aus historischen Gründen als "shell" ("Muschel") bezeichnet; diese Bezeichnung wird im Folgenden verwendet. Nach dem Login wird üblicherweise die Datei */etc/motd* (Message of the Day) angezeigt.

Eine Shell wird ebenfalls gestartet, wenn man unter einer grafischen Benutzeroberfläche ein Kommandofenster startet: Der Aufruf von **xterm**, **konsole** oder **gnome-terminal** liefert einen bunten Rahmen um eine Shell herum. Vorteil der grafischen Oberfläche ist, dass quasi beliebig viele Shells gleichzeitig gestartet sein können oder im Blick sind.

### 2.2.4 Shell / Kommandointerpreter

Ist man auf der Kommandozeile angelangt, so findet man ein weisses bzw. schwarzes Kästchen oder einen nervös blinkenden Unterstrich vor; beide werden als "Cursor" bezeichnet und markieren die Stelle, an der die eingetippten Zeichen erscheinen.

Die Shell oder auch der Kommandointerpreter ist ein Prozess der nach dem Anmelden (Login) an einer Unix-Maschine gestartet wird und die Interaktion des Benutzers mit dem Betriebssystem erlaubt. Diese Shell bringt bereits etliche Funktionalitäten mit. Sie interpretiert die Kommandoaufrufe seitens der Benutzer und erlaubt einfache Skript- (bzw. Batch-)programmierung zur Erleichterung wiederkehrender Tätigkeiten. Sie stellt weiterhin über Umgebungsvariablen Programmen eine ganze Reihe von Systeminformationen, wie Benutzername, Home-Verzeichnis und Rechnername zur Verfügung.

Die Kommandozeile beginnt meistens etwa so:

```
meier@hermes:~/kursunterlagen >
```

Zu Anfang steht der Username, mit dem man/frau am System angemeldet ist. Dann folgt der Rechnername und anschliessend das Verzeichnis, in dem man sich befindet. All das, zusammen mit dem ">" bezeichnet man als Prompt. Das Aussehen des Prompts kann verändert werden, wenn die steuernde Umgebungsvariable "PS1" entsprechend angepasst wird. Dieses geschieht entweder systemweit in der Datei */etc/profile* bzw. in einer Datei, die aus dieser heraus aufgerufenen wird oder durch einen Eintrag in der *.profile* des Benutzers. Dieses Verfahren läßt sich bei vielen Unix-Programmen wiederfinden: Es gibt systemweite Konfigurationsdateien und die Möglichkeit für den Benutzer eigene Einstellungen vorzunehmen, die komplett unabhängig von anderen Benutzern oder von den Systemeinstellungen sind.

Die Shell dient zum Aufruf der Programme, mit denen man eigentlich arbeiten will. Durch Ausgabe der Eingabeaufforderung, des sogenannten Prompts zeigt die Shell an, dass sie bereit ist, Kommandos entgegenzunehmen. Ein typischer Programmaufruf sieht etwa so aus:

```
meier@hermes:~/kursunterlagen > less kurs01.txt
```

[6] Eine Ausnahme kann bestehen, wenn LDAP als zentrales Backend zur Benutzerauthentifizierung zum Einsatz kommt

Zuerst steht das Kommando, dann folgt nach einem Leerzeichen (white space) der Dateiname. Es ist dabei immer auf die Leerzeichen zu achten, da sonst die Shell die Kommandos nicht von ihren Optionen oder nachfolgenden Dateinamen unterscheiden kann. Beispielsweise hätte die Eingabe von `lesskurs01.txt` die Fehlermeldung "lesskurs01.txt: Command not found" verursacht, da die Shell nach einem (nicht vorhandenen) Befehl aus den Zeichen "lesskurs01.txt" am Stück sucht. Ausnahmen sind spezielle Zeichen, die von der Shell anders interpretiert werden. Dazu zählt zum Beispiel das Semikolon ";" für das Trennen von Kommandos in einer einzigen Zeile. Weitere Ausführungen hierzu finden sich in Kap. 5.1, S. 43.

Befehle bestehen aus Kommandonamen, Optionen und Argumenten, die an den Befehl angehängt werden. Beispiel: `ls -alh /home/test`. Hier heisst der Befehl **ls**, die Option ist "alh" und das Argument ist die Datei */home/test*. Optionen sind Anweisungen an den Befehl, seine Arbeitsweise gegenüber der Voreinstellung zu ändern. Optionen werden meist mit "-" eingeleitet. Für bestimmte Kommandos und Situationen findet man auch "--". Klassisches Beispiel ist der Aufruf der Kurz-Hilfe-Option, die mit "-h" oder auch "--help" erreichbar ist. Eine Option besteht gewöhnlich aus einem Buchstaben. Mehrere Optionen können aneinandergereiht werden, wie im obigen Beispiel die drei Optionen "a", "l" und "h". Jeder Befehl hat seine eigenen Optionen, die man in eigenen Hilfeseiten (mit "man befehlsname") nachschlagen kann. Argumente sind Zeichenketten, die vom Befehl interpretiert werden. Meist geben sie Dateien an, mit denen etwas gemacht werden soll.

Nach dem Drücken von [Enter][7] wird der Befehl ausgführt. Fehler in der Eingabezeile können mit Hilfe der Taste [Backspace][8] oder [Delete] korrigiert werden. Die meisten Shells erlauben die Benutzung der [Cursor]-Tasten (Pfeil-Tasten) zur Korrektur der Eingabezeile.

### 2.2.5 Ändern des Passworts

Normalerweise wird das Passwort bei der Einrichtung eines Accounts - des Benutzerkontos auf einem Rechner - vom Administrator oder automatisch vergeben. Dieses fällt entweder sehr kryptisch aus oder ist zu einfach. Deshalb sollte dieses beim Erst-Login neu gesetzt werden. Das Passwort kann mit dem Befehl **passwd** geändert werden. Dies gilt jedoch meist nur auf Maschinen, die nicht an eine zentrale Benutzerverwaltung angeschlossen sind. Das Kommando **passwd** wirkt auf die Datei */etc/shadow* in der auf einer Linuxmaschine die Passwörter aller Benutzer verschlüsselt gespeichert sind. Dabei wird man noch einmal nach dem alten Passwort gefragt, es sein denn man ist der Systemadministrator. Damit wird verhindert, dass nicht in einem unbeobachteten Moment ein Witzbold schnell das Passwort ändern kann. Danach muss man zweimal das neue Passwort eingeben - als Schutz vor Tippfehlern, da das Passwort wie beim Einloggen "blind" eingegeben wird.

Das erste Passwort, das bei der Einrichtung eines Accounts zugeteilt wird, sollte aus Sicherheitsgründen schnellstmöglich geändert werden. Aber auch später ist ein regelmäßiges Ändern des Passworts ratsam, um einen Missbrauch des eigenen Accounts zu erschweren. Man sollte bei der Auswahl eines Passworts eine Reihe von Regeln beachten:

- Eine ziemlich unclevere Idee ist die Benutzung von Namen, Geburtstage, Firmennamen, Telefon- und Autonummern. Sie sind leicht zu raten und daher ungeeignet. Gleiches gilt für Abwandlungen der User-ID.
- Die einzelnen Zeichen des Passworts sollten auf der Tastatur nicht direkt nebeneinanderliegen, damit das Passwort beim Eingeben nicht einfach mitgelesen werden kann.

[7] oder auch "Return"; Bezeichnung deshalb, weil der Prompt nach Ausführung des Kommandos wieder erscheint und eine neue Eingabe möglich wird

[8] Löschen eines Zeichens rückwärts

- Es sollte ein Passwort gewählt werden, das in keinem Wörterbuch steht. Es sollte daher mindestens ein nicht-alphanumerisches Zeichen, d.h. ein Sonderzeichen, nicht einen Buchstaben oder eine Zahl, enthalten und sowohl aus Gross- als auch Kleinbuchstaben bestehen.
- Ein Paßwort sollte aus mindestens acht Zeichen bestehen. Dieses ist die Maximalzahl von Zeichen eines klassischen Unixsystems. Viele Systeme erlauben durch Konfiguration meistens keine Passwörter, die kürzer sind als sechs Zeichen und melden bereits zu einfache Kombinationen.
- Beliebte Methoden, um auf ein einfach zu merkendes aber schwer zu ratendes Passwort zu kommen, sind das Ersetzen von Zeichen (etwa 's' durch '$', 'i' durch '!' etc.) oder Paßwörter aus Anfangsbuchstaben von Sätzen[9] zu bilden.
- Man sollte sein Passwort niemals aufschreiben oder weitersagen und vor dem Verlassen des Computers immer darauf achten, dass man ausgeloggt ist.

### 2.2.6 Umsehen am Kommando-Prompt

Für die ersten Schritte im System sollte man vielleicht einfach mal nacheinander folgende Kommandos ausprobieren, um sich mit der Funktionsweise von Kommandos, Shell und dem Linux-System vertraut zu machen:

- **ps** - Prozess-Status: Zeigt die gerade unter der eigenen User-ID laufenden Prozesse an. Mit `kill ProzessID` kann man einzelne Prozesse beenden. Eine ausführliche Beschreibung der Linux-Prozessverwaltung ist in Kap. 8.4 auf S. 89 zu finden.
- **w** - Eine Art "Who": Liefert eine Liste der gerade am System angemeldeter Benutzer mit zusätzlichen Informationen, welche Prozesse diese gerade am Laufen haben. Darüberhinaus erzählt einem das Kommando die Auslastung der Maschine, wie lange sie schon läuft und von woher die Benutzer angemeldet sind.
- **uptime** - Laufzeit der Maschine: Ausschliesslich Informationen zur aktuellen Last auf der Maschine und die vergangene Zeit nach dem letzten Neustart.
- **uname** - Name des Betriebssystems: Ausführlichere Informationen erhält man durch `uname -a`.
- **pwd** - Pfadangabe: Gibt an, wo man sich gerade aktuell im Dateisystembaum befindet. Dies kann nützlich sein, wenn diese Information nicht durch den Prompt bereitgestellt wird.
- **whoami** - Eigene User-ID: Vergessliche Benutzer können mit diesem Kommando ihren Account-Namen anzeigen lassen.
- **last** - Wer war als letztes da: Meldet alle zuletzt angemeldeten Benutzer. Diese Liste kann verdammt lang sein und sollte durch `last | less` besser lesbar gemacht werden.
- **chsh** - Shell ändern: Es sind meistens mehr als eine Shell installiert und jede(r) hat vielleicht andere Vorlieben, was die Features von Shells anbetrifft. Das Ändern der Default-Shell funktioniert so ähnlich wie beim **passwd**, nur wenn die Maschine nicht an zentralen Verwaltungs- und Authentifizierungsstrukturen hängt.

[9] z.B. "Ich stehe fast immer um acht Uhr auf" zu "!$fiu8Ua" abwandeln. Dieses Satz trifft zwar nicht auf den Autor dieses Textes zu, bietet aber ein besseres Beispiel als zweistellige Uhrzeiten :-))

- **chfn** - Name ändern: Hiermit kann man Daten zu seinem Account ändern, jedoch nicht seinen Accoutnamen selbst! Es gilt das zu **passwd** und **chsh** Gesagte.
- **ls** - Inhalt des Verzeichnisses: Listet alle Dateinamen im aktuellen Verzeichnis auf.
- **finger** - "Who" netzwerktransparent: Einfach aufgerufen liefert das Kommando ähnliche Informationen wie **w** oder **who**. Ein Aufruf von `finger @rechnername` liefert, wenn es nicht abgeschaltet ist,[10] die auf einem bestimmten Rechner angemeldeten Benutzer.
- **date** - Datum und Uhrzeit: Liefert die aktuelle Systemzeit mit Stunden, Minuten, Sekunden und das aktuelle Datum inklusive Information zur Zeitzone.
- **export** - Variablenliste: Zeigt den Inhalt aller Variablen, die an Programme oder weitere Shells weitergereicht werden können. Hier findet man z.B. die Variablen $USER für den eigenen Accountnamen, $UID für die eigene User-ID, die zum Accountnamen passt, $PATH für den Suchpfad nach ausführbaren Programmen und etliche weitere.
- **free** - Freier Speicher: Zeigt die Belegung des Arbeitsspeichers an. Die Menge des freien Speichers ist meistens recht gering, da Dateien für schnelleren Zugriff zwischengespeichert werden.
- **df** - Disk Free: Zeigt für die einzelnen Bereiche des Dateisystems die Belegung der Festplatte(n) an.
- **id** - Gibt Auskunft über die Benutzer-ID in der gerade laufenden Shell:
  `uid=500(dirk) gid=100(users) groups=100(users),`
  `14(uucp),16(dialout),17(audio),33(video).`

### 2.2.7 Ordnung in Verzeichnissen

Die meisten Aufgaben unter UNIX und Linux beinhalten das Anlegen, Löschen, kopieren und Umbenennen von Verzeichnissen und Dateien. An dieser Stelle werden die wichtigsten Befehle hierfür nur kurz genannt; eine ausführliche Beschreibung ist in Kapitel 7.2 auf S. 76 zu finden. Die wichtigsten Befehle sind:
**mkdir** "make directory", **rmdir** "remove directory", **cp** "copy", **mv** "move" und **rm** "remove". Im aktuellen Verzeichnis erzeugt man ein neues Unterverzeichnis mittels:

```
dirk@s02:~> mkdir name_des_neuen_verzeichnisses
dirk@s02:~> rmdir name_des_neuen_verzeichnisses
```

Dieses kann analog mit **rmdir** wieder entsorgt werden. Eine Datei zu kopieren geschieht mittels **cp** in den folgenden Ausprägungen:

```
user@s01:~> cp existierende_ datei neuer_dateiname
user@s01:~> cp existierende_datei existierendes_verzeichnis
user@s01:~> cp exist_datei exist_verzeichnis/neuer_dateiname
```

In vielen Aspekten analog arbeitet das Kommando **mv**, wobei die Quelldatei anschliessend nicht mehr unter ihrem ursprünglichen Namen existiert. Bei beiden Programmen wird eine eventuell existierende Zieldatei ohne Warnung überschrieben. Gelöscht werden kann eine Datei durch `rm dateiname`.

[10] wegen häufigen Missbrauchs eher noch ein Relikt aus Zeiten, als die Welt der Internets noch in Ordnung war :-)

Eine einfache Aufstellung des Verzeichnisinhalts liefert **ls** mit vielen Zusatzinformationen `ls -alh`, wobei zusätzlich die Dateigröße nicht einfach in Byte sondern "(h)uman readable" in Kilo-, Mega- oder GigaByte angegeben wird. Die aktuelle Belegung eines Verzeichnisses liefert **du** 'disk usage", wobei Unterverzeichnisse nochmal separat angezeigt werden.

### 2.2.8 Dateinamen

Die Benennung von Dateien und Verzeichnissen[11] besitzt fast keine Grenzen. Es gibt jedoch einige praktische Überlegungen, die einen auf den Einsatz eher unüblicher Zeichen verzichten lassen. Tabu ist das Trennzeichen "/" für Verzeichnisse. Nicht so sinnvoll sind Sonderzeichen, die von der Shell speziell interpretiert werden oder Umlaute. Leerzeichen in Dateinamen sind erlaubt, müssen aber in der Shell gesondert behandelt werden. Die Shell fasst Leerzeichen üblicherweise als Trennzeichen von Kommandos, Optionen und Argumenten auf. Size matters!! Klein- und Grossschreibung machen auf Linux-/Unixsystemen einen Unterschied: So sind *test*, *TEST* und *Test* drei unterschiedliche Dateien. Oft aber nicht immer gilt das auch für Benutzernamen.

### 2.2.9 Finden von Dateien

Um eine Datei zu suchen und eventuell zu finden, verwendet man das Kommando **find**. Abstrakt startet man die Suche so:

`find ''Startverzeichnis'' -name Dateiname.`

Man sollte das Startverzeichnis geeignet wählen, wenn man ungefähr weiss, wo die Datei grob liegen könnte. Sonst beginnt man am besten mit */*. Das ist nicht immer ratsam, da es sehr lange dauern kann und man Fehlermeldungen für jedes Verzeichnis erhält, auf das man keinen Lesezugriff hat.

Es muss nicht unbedingt der Dateiname bekannt sein, man kann auch nach völlig anderen Kriterien einen Dateibaum durchforsten:

```
find -type d -exec chmod a+x \;
```

Das Beispiel sucht nach allen Dateien vom Typ "Verzeichnis" und führt auf diesen eine Operation aus. Die Operation wird nach der Find-Option "-exec" angegeben (in diesem Beispiel werden die Zugriffsrechte aller Verzeichnisse gesetzt). Damit **find** weiss, wo die Operation endet, wird diese mit einem Semikolon beendet. Damit jedoch die Shell dieses nicht interpretiert (als Trennzeichen von Kommandos) muss es entsprechend "escaped" werden, was mit dem Backslash ("\") geschieht.

Zum Finden von Dateien, Verzeichnissen, Programmen, etc. ist auch der Befehl **locate** sehr hilfreich. Dieser Behehl sucht den Datei- oder Verzeichnisnamen in einer Datenbank; ob diese existiert und wie oft sie aktualisiert wird, hängt vom Systemadministrator ab. Beispiel: `locate XFree86`. Auf SuSE-Linux-Systemen findet man noch das kleine Skript **rpmlocate**, welches nicht ein **updatedb** benötigt, da es direkt auf die RPM-Datenbank zugreift.

### 2.2.10 Abmelden durch Ausloggen

Damit nicht nach dem Beenden einer Arbeitssitzung irgendjemand Blödsinn mit dem eigenen Account anstellen kann - was einem selbst in Rechnung gestellt werden könnte - sollte man sich unbedingt abmelden. Dies geschieht in Abhängigkeit von der Konfiguration und

[11] Maximal 255 Zeichen sind bei den meisten Dateisystemen erlaubt - das sollte aber schon ausreichenden Spielraum lassen, um erklärende Dateinamen zu vergeben.

der verwendeten Shell auf verschiedene Weise: Mit [Strg]-[d] lassen sich viele Shells schliessen. Handelt es sich dabei um die automatisch beim Login gestartete Shell, so loggt man sich damit automatisch aus.

Ähnliche Auswirkungen haben auch die Kommandos **logout** oder **exit**. Unter der grafischen Benutzeroberfläche genügt das einfache Schliessen der Shell nicht, sondern man verwendet entweder den geeigneten Menüpunkt des Windowmanagers oder beendet die X-Session mit [Strg]-[Alt]-[Backspace]. Man sollte auch immer daran denken, dass man an mehreren Konsolen gleichzeitig angemeldet sein kann und diese dann ebenfalls schliessen sollte.

### 2.2.11 Alternative zur Kommandozeile

Für Anfänger ist der **mc**[12] eine große Hilfe; er ersetzt die Kenntnis vieler Linux-Kommandos. Der **mc** läuft als eigenständiger Prozess in der Shell aus der er aufgerufen wurde - der Befehl hierfür ist einfach **mc**. Im Textmodus des **mc** findet man am unteren Bildschirmrand die Befehle mit jeweils einer Zahl davor. Drückt man die Funktionstaste mit der entsprechenden Zahl, z.B. [F4] zum Bearbeiten, so wird der entsprechende Befehl ausgeführt. Die restliche Bedienung ist recht schnell verstanden: Mit den Pfeiltasten navigiert man durch die Verzeichnislisten und durch die Menüs, mit [Enter] öffnet man Dateien bzw. aktiviert einen Menüpunkt. Die Tabulatortaste wechselt zwischen den beiden Fenstern.

Unter den grafischen Oberflächen, wie KDE, Gnome oder IceWM gibt es eingebaute Filemanager oder eine Reihe externer Applikationen, die diese Aufgabe übernehmen. Beim KDE ist der **konqueror** sowohl für das lokale File- und Resourcen-Browsing, als auch für den Webzugriff zuständig. Unter Gnome heisst der Filemanager **nautilus**. Die Bedienung dieser Applikationen orientiert sich am Gewohnten und ist erfreulich intuitiv, jedoch lassen sich hiermit schwer Automatisierungen von wiederkehrenden Abläufen vornehmen.

## 2.3 Suchen und Finden von Hilfe

Selbst erfahrene UNIX-/Linux-Anwender kennen nur einen Teil der Befehle mitsamt ihren Optionen. Daher gibt es auf jedem UNIX-System mehrere Möglichkeiten, Informationen über Kommandos abzurufen. Die klassischen Befehle hierfür sind **man**, **apropos** und **whatis**. Die meisten Programme bringen selbst eine eingebaute Mini-Hilfe mit, welche mit "-h" oder "--help" zu aktivieren ist: `ls --help` gibt zum Beispiel einen Kurzüberblick zu den Fähigkeiten des List-Kommandos.
Manualpages:

- enthalten die abstrakte Syntax eines Kommandos oder Anwendungsprogrammes.
- enthalten eine Liste aller Optionen und deren Wirkung.
- enthalten Querverweise zu anderen Befehlen/Manual Pages.
- enthalten manchmal Beispiele.
- sind nicht zum Lernen, sondern eher zum Nachschlagen geeignet.
- werden aufgerufen durch `man Kommandoname`

[12] "midnight commander", stark angelehnt an den Norton Commander, der aus der DOS-Welt noch bekannt sein könnte und die Vorlage für viele Dateimanager mit dem typischen senkrecht geteilten Fenster lieferte.

- werden seitenweise mithilfe eines Pagers (**less**, **more**) angezeigt

Das folgende Beispiel zeigt eine typische Man-Page für das Kommando **cp**. Defaultmäßig wird die Man-Page in Englisch angezeigt, ist die entsprechende Umgebungsvariable "LANG" entsprechend gesetzt, kann auch jede andere installierte Sprache angezeigt werden. Der Aufruf von `man cp` liefert:

```
CP(1)                          User Commands                          CP(1)

NAME
       cp - copy files and directories

SYNOPSIS
       cp [OPTION]... [-T] SOURCE DEST
       cp [OPTION]... SOURCE... DIRECTORY
       cp [OPTION]... -t DIRECTORY SOURCE...

DESCRIPTION
       Copy SOURCE to DEST, or multiple SOURCE(s) to DIRECTORY.

       Mandatory arguments to long options are mandatory for short options too.

       -a, --archive
              same as -dpR

       --backup[=CONTROL]
              make a backup of each existing destination file

[... eine Menge weiterer Optionen ...]
AUTHOR
       Written by Torbjorn Granlund, David MacKenzie, and Jim Meyering.

REPORTING BUGS
       Report bugs to <bug-coreutils@gnu.org>.

COPYRIGHT
       Copyright (C) 2005 Free Software Foundation, Inc.
       This is free software; see the source for copying conditions.  There is NO
       warranty; not even for MERCHANTABILITY or FITNESS FOR  A  PARTICULAR  PUR-
       POSE.

SEE ALSO
       The  full  documentation for cp is maintained as a Texinfo manual.  If the
       info and cp programs are properly installed at your site, the command

              info cp

       should give you access to the complete manual.

cp 5.3.0                         March 2005                          CP(1)
```

Das Kommando **apropos** liefert die Antwort auf die Frage "Was findet man alles zum Stichwort xyz?". Es gibt eine Liste der zu diesem Stichwort verfügbaren Manual Pages[13] aus. Aufgerufen wird es durch `apropos Stichwort`. Das Kommando **whatis** liefert eine Kurzinformation, wenn der Name des Kommandos bekannt ist:

[13] oder zu gut deutsch: Bedienungsanleitungen

```
dirk@test:~/kurs> whatis cp
cp (1)               - copy files and directories
cp (1p)              - copy files
```

Ein weiteres kurzes Kommando, `which Befehlname`, liefert den vollständigen Pfad zu einem ausführbaren Programm, wenn es sich im Suchpfad befindet. Der Suchpfad wird durch die Shell-Umgebungsvariable $PATH[14] bestimmt.

## 2.4 Remote Login, Execution

Mit dem Kommando **telnet** oder **rsh**[15] kann man sich auf einem anderen Rechner einloggen und den eigenen Rechner als "Terminal" benutzen. Das Einloggen mit **telnet** hat den gravierenden Nachteil, dass die Kommunikation zwischen den Rechnern und damit auch die Authentifizierung mit Login-ID und Passwort unverschlüsselt über das Netzwerk erfolgt und daher leicht mitgelesen werden kann.

### 2.4.1 Verbindung mit anderen Rechnern

Daher sollte man die Secure Shell (**ssh**) einsetzen, die eine verschlüsselte Verbindung mittels eines sogenannten Public Key-Kryptoverfahrens ermöglicht.

Zum Fernadministrieren einer Linux-Maschine kann man nicht nur das Text-Terminal der Secure Shell verwenden, sondern über Secure-Shell-Verbindungen den grafischen Output von Applikationen der Remote-Maschine auf den lokalen Desktop holen:

Der folgende Dialog zeigt den Aufbau einer Verbindung über die Secure Shell und den Start des grafischen Editors **kate**:

```
maier@rechner01:~ $ ssh -X -l root server02
Password:
Last login: Sun Sep 25 13:45:02 2005 from rechner01.localnet.private
Have a lot of fun...
server02:~ # kate /etc/resolv.conf &
server02:~ #
```

Diese kleine Befehlsabfolge kann man dazu nutzen, sich als Systemadministrator mit einer entfernten Linux-Box zu verbinden (hier: *server02*) und auf ihr einen Editor zum Bearbeiten beispielsweise der */etc/resolv.conf*[16] zu starten. Als Ergebnis zeigt der lokale Desktop die grafische Oberfläche des Editors als normales Fenster an auf dem Desktop an. Man kann nun mit **kate** von *server02* genauso arbeiten, als würde man direkt an dieser Maschine hocken. So kann man einen Rechner fernadministrieren, auch wenn er in einem gesicherten Serverraum steht. Wenn die Verbindung hängt, kann man sie durch die Eingabe von [~],[.] (Tilde Punkt) beenden. Die Eingabe von [~],[c] liefert das Konfigurationsinterface beispielsweise zum Aufsetzen von Tunneln.

Etwas abgewandelt kann man auch einfach direkt ein Kommando remote starten, ohne eine Shell auf der entfernten Maschine zu starten:

```
maier@rechner01:~ $ ssh -X -l root server02 kate /etc/resolv.conf
```

[14] diese Variable kann man sich durch `echo $PATH` ausgeben lassen. Achtung: Die Standardumgebungsvariablen der Shell sind immer gross geschrieben.

[15] für remote Shell - letzteres hat vielleicht nur noch die Bedeutung den das abgewandelte Kürzel "ssh" für die abgesichterte Variante zu liefern

[16] Diese Datei stellt die Namensauflösung ein.

### 2.4.2 Filetransfer per Secure Copy

Kopieren mit **ssh**; zum Beispiel:
`scp user@dozent:~/lak.pdf kurs/` kopiert die Datei *lak.pdf* aus dem Homeverzeichnis von user auf dem Rechner *dozent* in das Verzeichnis *kurs* auf dem lokalen Rechner. Umgekehrt kopiert:
`scp -r test user@rechner01:/tmp` das Verzeichnis *test* rekursiv[17] (d.h. mit allen Unterverzeichnissen und Dateien) vom lokalen Rechner in das Verzeichnis */tmp* auf der Maschine "Rechner01".

## 2.5 Aufgaben

### 2.5.1 Dokumentation/Hilfe

1. Man nenne das Kommando (und den Aufruf am Beispiel der Shell "bash") zum Anzeigen von Manpages!
2. Wo finden sich unter einer Suse-Linux-Installation üblicherweise weitere Hilfetexte, Howto's und Unterlagen?
3. Welche Möglichkeiten stehen alle zur Verfügung, um sich Hilfen, Informationen und Hinweise zu Programmen und Softwarepaketen anzeigen zu lassen?

### 2.5.2 Zentrale Kommandos

1. Man ermittele seine eigene UserID und GruppenID (numerisch und als Accountname) und die des Systemadministrators!
2. Wie lautet das Kommando zum Wechseln der Verzeichnisse? Wie kommt man in die oberste Verzeichnisebene?
3. Mit welchem Kommando legt man ein Verzeichnis an und wie lösche ich ein nichtleeres Verzeichnis komplett?
4. Lesen Sie Ihre aktuelle DISPLAY-Variable aus und geben Sie diese anschliessend an!
5. **who**: Wer ist da? Welche Befehle zeigen ähnliche Informationen an? Wie bekommt man heraus, wer sich in letzter Zeit an der Maschine eingeloggt hat? Wie kann man dafür sorgen, dass sich diese Liste bequem durchblättern läßt?
6. Wie kann ich meinen "Full Name" (Room Number, Work Phone, Home Phone) ändern? Wo wird festgelegt, ob normale User diese Felder wirklich ändern können? Weshalb klappt dieses in großen Umgebungen mit einer zentralen Passwortverwaltung meistens nicht mehr?
7. Welche(s) Kommando(s) liefern mir Datum und Systemzeit? Wie kann man diese (neu) setzen, welche Voraussetzungen sind dafür notwendig?
8. Wofür dienen die Kommandos **find** und **(rpm)locate**? Worin bestehen die Unterschiede zwischen ihnen?
9. Mit welchem Kommando erhält man die folgende Ausgabe?
   `6:39pm an 2 Tage 5:28, 9 Benutzer, Durchschnittslast: 0,01, 0,04, 0,23`

[17] Achtung: Symbolische Links werden nicht als solche kopiert, sondern die kompletten Dateien

10. Welche Kommandos (Mehrzahl!) zeigen mir an, wer gerade mit mir an einer Maschine eingeloggt ist?

### 2.5.3 Die Shell

1. Was sind Shell Prompt und Terminal-Fenster?

2. Wie setzt man eine Umgebungsvariable (und sorgt dafür, dass sie auch in abgeleiteten Shells vorhanden ist)? Wie zeigt man die Liste aller in der aktuellen Shell definierten Variablen an?

3. Wie "schneidet" man aus der */etc/passwd* die Zeile mit der eigenen UserID heraus?

4. Woher bezieht die Shell die Informationen zu Username, numerischer UserID und Home-Verzeichnis? Welcher Prozess ist bei der Anmeldung involviert?

5. Wie findet man heraus, in welchem Pfad man sich befindet? In welcher Shellvariablen kann diese Information angezeigt werden (damit man immer weiss, in welchem Verzeichnis man gerade ist)?

6. Welche Möglichkeiten der Abmeldung von einer Shell oder allgemeiner von einer Konsole stehen zur Verfügung?

7. Welche Shellvariablen sind nur lesbar und warum kann man sie nicht einfach überschreiben, z.B. mit `export VARIABLE=''neuer Wert''`?

8. Welche verschiedenen Möglichkeiten stehen zur Verfügung, um an (vor einiger Zeit) eingegebene Befehle und Befehlsfolgen wieder heranzukommen?

9. Wo werden Shellvariablen definiert? Wie kann ein Benutzer seine eigenen Definitionen speichern, damit sie bei jedem Login oder Shellaufruf automatisch aktiviert werden?

10. Wann sind ausführbare Programme, wie `ls -la /etc` einfach ausführbar? Wann muss der absolute Pfad, wie z.B. `/opt/mozilla/mozilla` eingegeben werden, um ein Programm zu starten?

11. Was verbindet das Kommando **which** mit der Shellvariablen $PATH?

12. Was verbindet das Kommando **cd** (ohne weitere Optionen und Argumente) mit der Shellvariablen $HOME?

### 2.5.4 Kommandos auf Dateien

1. Wie bekommt man heraus, von welchem Typ eine Datei ist? Spielen dabei Dateiendungen eine Rolle?

2. Man nenne die Befehle:
   a) Zum Kopieren
   b) Zum Umbenennen/Verschieben
   c) Zum Löschen von Dateien!
   d) Zum Löschen eines leeren Verzeichnisses!
   e) Was gibt man ein, wenn man sich die Anleitung zum Befehl **ls** ansehen will?
   f) Wo gibt es darüberhinaus Informationen wenn der Versuch aus e) fehlschlug?

3. Wie erzeuge ich in meinem Homeverzeichnis ein Verzeichnis *uebung* mit einem Unterverzeichnis *nummer01*, welches wiederum ein Unterverzeichnis *aufgabe01* enthält? Wie lösche ich diese Verzeichnisse wieder (alle auf einmal direkt auf oberster Ebene meines Homeverzeichnisses)?

### 2.5.5 Wichtige Tastaturkürzel

1. Wie wechselt man von der grafischen Konsole in den Textmodus (und umgekehrt)?
2. Wie beende ich X direkt per Tastatur? Welche Probleme handele ich mir damit evtl. ein?
3. In der Shell (Textmodus oder "xterm"): Wie kann man "hochblättern" um bereits aus dem Terminal "verschwundene" Zeilen wieder sichtbar zu machen?
4. Wie lässt sich während einer grafischen Sitzung die Bildschirmauflösung unter X (der Grafikserver unter Unix/Linux) verändern?
5. Wie lässt sich in der Shell ein Bereits vor kurzem ausgeführtes Kommando erneut aufrufen (History-Funktion)?
6. Wie komme ich aus einer hängenden SSH-Verbindung heraus?
7. Man verbinde die untenstehenden Blöcke geeignet: Links steht eine Tastenkombination und rechts wofür sie gut sein könnte!

| | |
|---|---|
| $[Cursor]$ nach oben | Grafische Oberfläche "hart" beenden |
| $[Strg] - [e]$ | Gerade in der Shell laufendes Programm abbrechen |
| $[Strg] - [d]$ | Auf die erste Textkonsole wechseln |
| $[Strg] - [c]$ | An den Anfang der Kommandozeile springen |
| $[Shift] - [PgUP]$ | Den Inhalt der Pfadvariablen wiederherstellen |
| $[Strg] - [Alt] - [F1]$ | Shell beenden / Ausloggen |
| $[Strg] - [Alt] - [Backspace]$ | An das Ende der Kommandozeile springen |
| $[Strg] - [a]$ | In der Command-History nach oben blättern |
| $[Pos1]$ | Nach oben "verschwundenen" Bildinhalt wieder sichtbar machen |

# Kapitel 3

# Die Shell

## 3.1 Einleitung

Sehr viele administrative Aufgaben laufen unter Unix/Linux durch Skripte. Diese Skripte sind üblicherweise Batches [1] von Programmabfolgen und Kontrollstrukturen, die von der Shell interpretiert werden. Möchte man überprüfen, welche Rechner gerade in einem bestimmten Netzwerkbereich erreichbar sind, kann das z.B. durch folgendes Skript geschehen:

```
declare -i i ; i=0
while [ $i -lt 255 ] ;
  do ping -c1 -w1 172.16.20.$i 1>/dev/null \
        && echo 172.16.20.$i is alive ; i=$i+1
done
```

Die Init-Skripte zum Starten und Stoppen von Diensten sind ebenfalls Batches von Kommandos, die im Shell-Kontext ausgeführt werden; hier exemplarisch ein Ausschnitt aus der */etc/init.d/nfsserver*:

```
[...]
case "$1" in
    start)
      PARAMS=3
      test "$USE_KERNEL_NFSD_NUMBER" -gt 0 && PARAMS="$USE_KERNEL_NFSD_\
NUMBER"

      echo -n "Starting kernel based NFS server"
      /usr/sbin/exportfs -r
      rc_status
      /usr/sbin/rpc.nfsd $PARAMS
      rc_status
      startproc /usr/sbin/rpc.mountd
      rc_status -v
      ;;
    stop)
      echo -n "Shutting down kernel based NFS server"
      /usr/sbin/exportfs -au
      killproc -n -KILL nfsd
```

[1] engl. Stapel, ursprünglich von der sequenziellen Programmabarbeitung mit Lochkarten

```
      rc_status
      killproc    -TERM /usr/sbin/rpc.mountd
      rc_status -v
      ;;
[... some more lines ...]
    *)
        echo "Usage: $0 {start|stop|status|try-restart|restart|\
 force-reload|reload}"
        exit 1
        ;;
esac
rc_exit
```

## 3.2 Einige Bash-Grundlagen

### 3.2.1 Die Standardshell

An dieser Stelle soll nur kurz auf die Linux-Standard-Shell, die Bourne Again Shell **bash**, eingegangen werden. Weitere Informationen findet man in der Man-Page (**man bash**). Die **bash** verfügt, neben der obligatorischen Manpage, auch über eine eingebaute Hilfe Funktion. Mit dem Kommando **help** wird eine Übersicht der **bash** eigenen Kommandos angezeigt. Shell-Skripte sind das A und O der UNIX-/Linux-Systemadministration. Fast alle Administrationsaufgaben werden automatisch von Skripten abgearbeitet. Für Administratoren ist es daher unerlässlich, wenigstens Grundzüge der Shell-Programmierung zu beherrschen. Shell-Skripte besitzen eine erweiterte Syntax: Es stehen mathematische Operationen und Vergleichsoperationen wie in anderen Programmiersprachen zur Verfügung. Shells kennen verschiedene Kontrollstrukturen.

Eine der nützlichsten Funktionen der **bash** ist die Möglichkeit, Programm- und Dateinamen zu vervollständigen. Dabei kann dieses zu jeder Zeit am Shell-Prompt geschehen: Man tippe ein paar Zeichen eines Befehls und betätige anschliessend die Tabulator-Taste. Wenn die Eingabe bis zu der Stelle, an der der Tabulator ausgelöst wurde, eindeutig war, es also kein weiteres ausführbares Programm gibt, das mit diesen Buchstaben beginnt, wird die Eingabe automatisch vervollständigt. Sollte es zwei oder mehr Mglichkeiten geben, so wird man mittels Signalton verständigt. Ein nochmaliges Drücken der besagten Taste zeigt nun alle erhältlichen Alternativen an. Man probiere es einfach an einem Beispiel aus: Nehmen wir an, man möchte sich die Datei */var/log/messages* ansehen. Dazu benutzt man z.B. das Programm **less**. Man tippe also einfach mal "le" und die Tabulator-Taste zweimal. Nun werden einige Programme angezeigt, welche mit der besagten Buchstabenkombination beginnen.

Nun tippe man so lange weitere Zeichen ein, bis die Eingabe eindeutig ist, in diesem Beispiel sollte ein zusätzliches "s" dem geforderten Zweck genügen. Jetzt tippe man nochmals die TAB-Taste, und der Befehl wird ergänzt. Was bei Befehlen vielleicht noch etwas banal aussieht, wird bei langen Dateinamen mit kryptischen Zeichen sehr schnell zu einer großen Erleichterung. Neben dieser Funktionalität beherrscht die Bash weitere Komplettierungen.

Wozu die **bash** den begonnenen Namen zu vervollständigen sucht, hängt von der getätigten Eingabe ab. Wenn die Eingabe mit einem $ beginnt, so versucht die Bash, einen Variablennamen daraus zu machen. Beginnt die Eingabe hingegen mit einer Tilde ($\tilde{}$) , so versucht sie einen Benutzernamen zu bilden. Beginnt sie mit @, versucht sie die Eingabe zu einem Hostnamen zu vervollständigen. Wenn keine dieser Bedingungen zutrifft, sucht

die bash nach einem Alias- oder Funktionsnamen. Selbstverständlich muss der Name, zu dem die **bash** vervollständigt - sei es nun eine Variable, ein Benutzername, ein Hostname, ein Alias, eine Funktion oder ein Pfad - auch wirklich existieren. Wenn alle Versuche, eine passende Vervollständigung zu erreichen, fehlschlagen, ertönt ein kurzer Signalton.

### 3.2.2 Aufbau der Kommandozeile

Der Bash-Prompt, vielfach auch als Eingabeaufforderung bezeichnet, kann zwar beliebig angepasst werden, doch auf den meisten Systemen ist es für normale Benutzer (also alle außer dem Systemadministrator) standardmäßig eine Zeichenfolge nach dem Schema:

```
benutzer@computer:verzeichnis>
```

Wenn man sich auf dem Computer im eigenen Heimatverzeichnis befindet, welches durch die Tilde (~) repräsentiert wird, lautet der Bash-Prompt entsprechend:

```
dirk@dozent:~>
```

Der Benutzername ist "dirk" (ein ganz normaler Linux-Benutzer ohne Administratorrechte) und der Rechnername lautet "dozent". Dahinter blinkt der Cursor, an dem die Eingabe erfolgen kann. Nachdem ein Kommando ausgeführt wurde, gelangt man wieder zum Prompt. Es gibt sehr viele Kommandos, wie man sich überzeugen kann, wenn man einfach zweimal auf die Tabulatortaste tippt. Einige sind in die Bash direkt eingebaut, z.B. das Kommando **echo** zur Ausgabe von Zeichenketten. Andere sind klassische ausführbare Programme oder Skripte, wie **mount** zum Einbinden von Dateisysteme, oder **updatedb** zum Erzeugen der Locate-Datenbank. Auch Programme für die grafische Benutzerschnittstelle, wie **openoffice** können von der Bash aus gestartet werden, wenn die Bash in einer Terminal-Emulation unter X11 läuft.

### 3.2.3 Die Kommando-Geschichte

Fast jede Linux-Shell verfügt über eine Liste der zuletzt abgesetzten Kommandos, eine sogenannte History. Selbst wenn man nur gelegentlich die Kommandozeile verwendet, erweist sich die History als ein ausgesprochen nützlicher Helfer. Das gilt umso mehr, wenn man ausgiebigen Gebrauch von der Shell macht. Die Möglichkeiten zur Nutzung der History entsprechen der Benutzung eines effizienten Editors und werden in ihrem vollen Umfang nur von den Wenigsten benutzt. Sie gehen weit über die Möglichkeiten beispielsweise von "doskey" hinaus, welches vielleicht noch aus DOS-Zeiten bekannt sein könnte.

Die zuletzt eingetippten Befehle können mit den Cursor-Tasten (Pfeil hoch und runter) ausgewählt und editiert werden. Eine Rückwärtssuche in allen gespeicherten Kommandoaufrufen kann mittels [STRG]-R erfolgen. Nach dem Abmelden von einer Shell wird die "Geschichte", die History, in der Datei *.bash_history* im Home-Verzeichnis des jeweiligen Benutzers abgelegt. Möchte man nicht, dass andere die Command History nachvollziehen können, hilft ein Link dieser Datei nach */dev/null*. Für eine Sitzung kann man die History durch die Eingabe von:
`export HISTFILE=/dev/null` ebenfalls abschalten. Mit `!!` wird der letzte Befehl ausgeführt, mit !-2 wird der vorletzte mit !-3 der vorvorletzte usw. ausgeführt. Mit "!string" wird der letzte Befehl, der mit "string" begonnen hat, ausgeführt. Der eingebaute Bash-Befehl **history** zeigt eine Liste der zuletzt abgesetzten Kommandos an.

### 3.2.4 Ein- und Ausgabe

Programme verhalten sich meist so, dass sie bestimmte Daten aufnehmen, diese Daten auf irgendeine Weise verwenden, um schließlich wieder Daten auszugeben. Dieses Schema verdeutlicht sich noch bei interaktiven Programmen, die immer wieder Informationen vom Benutzer annehmen und ihm andere Informationen zurückliefern. Eine Shell ist ein typisches interaktives Programm. Zu diesem Zweck muß sie über einen Eingabekanal verfügen, über den sie Information aufnehmen kann. Dieser Eingabekanal existiert tatsächlich und erhält unter Linux die Bezeichnung Standardeingabe.

Womit aber ist die Standardeingabe verbunden? Üblicherweise wird es die Tastatur sein: Die Shell nimmt Zeichen für Zeichen von der Tastatur entgegen und gibt diese Zeichen auch sofort auf dem Bildschirm aus. Die Ausgabeseite auf dem Bildschirm kann von der Standardeingabe erledigt werden. Die Shell verfügt also über einen weiteren Kanal, der folgerichtig als Standardausgabe bezeichnet wird. Die Standardausgabe der Shell ist üblicherweise mit einer Konsole[2] oder einem grafischen Pseudo-Terminal, wie **xterm** oder **konsole** verbunden, so dass eingetippte Zeichen sichtbar werden.

Darüberhinaus existiert ein dritter Kanal, der eine besondere Aufgabe zu erfüllen hat, der sogenannte Standardfehlerkanal. Wie der Name schon sagt, dient der Kanal zur Ausgabe von Fehlermeldungen, wenn der Programmlauf aus irgendeinem Grund nicht ordnungsgemäss fortgesetzt werden konnte. Üblicherweise ist Standardfehler ebenfalls mit dem Bildschirm verbunden und schreibt daher seine Meldungen zwischen die gewöhnliche Ausgabe. Es macht jedoch Sinn, Stardardausgabe und Standardfehler voneinander zu trennen, um die Möglichkeit zu haben, gewöhnliche Ausgaben und Fehlerausgaben getrennt zu verarbeiten. Beispielsweise könnte man die Fehlerausgabe in eine Datei umlenken, um sie später zu analysieren, während die gewöhnliche Ausgabe weiterhin über den Bildschirm läuft.

An dieser Stelle gilt es, etwas Wichtiges zu verstehen: Standardeingabe, Standardausgabe und Standardfehler sind lediglich Kanäle, die mit irgendeiner Quelle und irgendeinem Ziel verbunden sein können. Standardeingabe ist nicht gleich Tastatur. Und Standardausgabe ist nicht gleich Monitor. Es gibt viele andere Quellen und Ziele, mit denen diese Kanäle verbunden werden können, wie beispielsweise Dateien oder andere Programme. Bei einer Shell macht es jedoch Sinn, Tastatur und Monitor als Eingabe und Ausgabe zu verwenden, daher ist dies die Voreinstellung.

Die drei Standardkanäle werden von Linux wie Dateien behandelt. Für geöffnete Dateien verwaltet das System eine Liste von Dateideskriptoren, die mit fortlaufenden ganzen Zahlen bezeichnet werden. Die Zahlen von 0 bis 2 sind für die drei Standardkanäle vorbelegt:

| **Kanal** | **Bezeichnung** | **Nummer** |
|---|---|---|
| Standardeingabe | stdin | 0 |
| Standardausgabe | stdout | 1 |
| Standardfehlerkanal | stderr | 2 |

Tabelle 3.1: Standardkanäle der Shell

Bei der Umlenkung dieser Kanäle werden diese Bezeichnungen benötigt! Bereits sehr früh wurde das Prinzip der “Pipe” von den Mainframe-Betriebssystemen abgeschaut. Röhre ist hierfür keine schlechte Übersetzung. Man kann in diese etwas “hineinleiten” und am anderen Ende kommt etwas wieder heraus. Wie bereits beschrieben, gibt es sehr viele kleine, spezialisierte Programme unter Unix/Linux, die mit speziellen Parametern aufgerufen werden können. Daher entstand die Idee eine geeignete Schnittstelle zwischen diesen zu schaffen,

[2] Monitorausgabe im Textmodus

um eine geeignete Hintereinanderschaltung dieser Tools zu erreichen.

Sinnvoll ist daher eine Schnittstelle zwischen diesen Programmen, welche Daten austauscht oder die Ergebnisse eines Programmlaufs in einem weiteren Programm weiterverarbeitet. Diese Schnittstelle ist in Form von "Pipes" ( — ) realisiert. Bekannt ist sicherlich das System-Kommando **ls**, welches dazu dient sich die Dateien in einem Verzeichnis auflisten zu lassen. Wenn man sich aber in einem Verzeichnis mit einer sehr hohen Anzahl von Dateien befindet und die Liste durchblättern möchte, kann man die Ausgabe, z.B. an das Kommando **more** weiterleiten (z.B. durch `ls -la | more`).

Weiterhin möchte man evtl. die Zahl der Dateien ermitteln, welches durch einfaches Zählen leicht etwas umständlich wird. Um die Zahl von Zeichen, Wörtern oder Zeilen in einer Datei zu ermitteln, steht das Kommando **wc** (Word Count) zur Verfügung. Ein Weg besteht also darin, die Ausgabe von `ls -l` in eine Datei zu schreiben und mittels `wc -l` die Zahl der Zeilen ermitteln zu lassen. Der Umweg über eine Datei lässt sich jedoch mittels einer Pipe umgehen: `ls -l | wc -l` . Unix/Linux benutzt an dieser Stelle das Zeichen "—" (Pipe). Man verkünpfte nun einfach die beiden Kommandos mittels dieses Zeichens zu einer Zeile und erhält nun die Anzahl der Dateien im aktuellen Verzeichnis. Nur ein kleiner Haken an dieser Stelle: **ls** gibt als erste Zeile keinen Dateinamen aus, sondern eine Zeile, in der Informationen zum entsprechenden Verzeichnis angezeigt werden, man muss hier also vom Ergebnis eine Zeile subtrahieren, um auf das exakte Ergebnis zu kommen.

Statt der Tastatur als Standardeingabe kann man alternativ aus einer beliebigen Datei lesen, die mit absolutem oder relativem Pfad angegeben wird: `sort </etc/passwd`. Das Zeichen "¡" erzeugt die gewünschte Funktionalität. Dies mag an dieser Stelle vielleicht etwas sinnlos erscheinen, da der **sort**-Befehl einen Dateinamen als Argument übernehmen kann. Es gibt jedoch Situationen, in denen die Quellenauswahl nur auf diese Weise realisiert werden kann.

Nicht alle Ergebnisse möchte man sich direkt am Bildschirm ansehen, sondern sie für spätere Bearbeitung oder aufgrund sehr grosser Datenmengen in eine Datei schreiben. Dieses geschieht durch den Einsatz von "¿". Sollen die Daten an eine bereits existierende Datei angehängt werden, ohne den bestehenden Inhalt zu überschreiben setzt man das "¿" doppelt hintereinander: ">>". Gerade für Sachen, die einen nicht interessieren, bietet sich die Umleitung in die Spezialdatei */dev/null* an. Möchte man z.B. testen, ob ein bestimmtes Verzeichnis existiert, genügt der Return-Code, die Ausgabe des Kommandos selbst, kann hingegen entsorgt werden: `ls -al /usr/bin > /dev/null`. Da der Ausgabe- und der Fehlerkanal verschiedene Deskriptoren haben, lassen sich mit `ls -al /usr/bin 2> /dev/null` auch nur die Fehlermeldungen wegdrücken. Das davor gegebene Beispiel müsste formal eigentlich `ls -al /usr/bin 1> /dev/null` lauten, wenn aber keine Kanalnummer angegeben ist, wird die Standardausgabe als Defaultwert angenommen. Sollen einfach beide Ausgaben gemeinsam in eine Datei geschrieben werden, können beide Kanäle mit `ls -al /usr/bin &> /dev/null` (im Fall der **bash**) oder mit `ls -al /usr/bin 2 >&1> /dev/null` gemeinsam umgelenkt werden.

### 3.2.5 Abkürzen von Befehlen

Der Alias-Mechanismus dient der Ersparnis von Tipparbeit, macht Kommandos leichter erinnerbar, verschönert Kommandoausgaben und kann auch zur Absicherung gegen Tippfehler verwendet werden. Ein Alias ist eine definierte Zeichenfolge, die für eine andere Zeichenfolge steht. Welche Aliase in der aktuellen Shell definiert sind, kann mittels **alias** abgefragt werden:

```
dirk@hermes:~> alias
```

```
alias +='pushd .'
alias -='popd'
alias ..='cd ..'
[... weitere Alias-Deklarationen ...]
```

Die Alias-Definitionen muss man nicht jedesmal eingeben, sondern kann sie in den Setup-Dateien unterbringen. Listen solcher Abkürzungen der Form **alias ¡Abkürzung¿ ¡Befehl (mit Optionen)¿** trägt man am besten in die globale Datei */etc/profile* oder in benutzerlokale Dateien, wie *.profile* oder *.bashrc* ein. Der Befehl **alias** ohne Argumente zeigt alle bereits vergebenen Abkürzungen an. Mit dem Befehl **unalias** kann man Definitionen wieder aufheben.

### 3.2.6 Wildcards in Dateinamen

Viele Kommandos, zum Beispiel **ls** oder **cp** haben als Argumente die Namen von Dateien oder Verzeichnissen. Namen, die Wildcards enthalten, werden zu einer Liste von Namen expandiert. Wildcards sind nahe verwandt zu Regulären Ausdrücken, die von vielen Programmen verarbeitet werden können. Beispiele von Wildcards sind:

| **Wildcard** | **Bedeutung** |
|---|---|
| ? | beliebiges Zeichen |
| * | beliebige Zeichenkette, auch der Länge Null |
| $[abc]$ | die einzelnen Zeichen a, b oder c |
| $[a-dg-i]$ | die einzelnen Zeichen a, b, c, d, g, h oder i |
| $[\hat{}abd]$ | nicht die einzelnen Zeichen a, b oder d |
| $\{dies, das\}$ | optionale Zeichenketten |

Tabelle 3.2: Beispiele für Wildcards

### 3.2.7 Zeichen mit besonderer Bedeutung

Eine ganze Reihe von Zeichen haben für die Shell besondere Bedeutung, ein Teil hiervon wurde bereits in den vorherigen Abschnitten vorgestellt:

```
[SPACE], [TAB], [CR], $, *, [, ], ?, {, }, ~, -,
<, >,&, !, |, ;, (, ),\ , ', ``, `.
```

Um diese Zeichen benutzen zu können, ohne dass die Shell sie interpretiert, müssen diese quotiert werden. Ein einzelnes Sonderzeichen wird durch vorstellen eines Backslash ("\") quotiert:

```
echo Bitte Drücken Sie eine Taste\!
```

Wenn man doppelte Anführungsstriche ("Text"), so werden von der Shell nur noch !, $, und ` als Sonderzeichen behandelt. Man kann also z.B. noch Umgebungsvariablen verwenden: `echo ``$PRINTER''` gibt den Wert der Variablen "PRINTER" aus. (vgl. Kap. **??**, S. **??** Verwendet man einfache Anführungsstriche ('Text') [3], so werden von der Shell nur noch ! und ` als Sonderzeichen behandelt.

[3] nicht ` das sogenannte Backtick

### 3.2.8 Spezielle Escape-Sequenzen

Einige Zeichen haben in Zusammenhang mit dem Backslash in der Shell eine besondere Bedeutung und werden speziell interpretiert, bevor eine Ausgabe auf dem Bildschirm erscheint. Dies wird z.B. zum Aufbau der Prompt-Variablen[4] benutzt oder für besondere Ausgaben in der */etc/issue* oder */etc/motd*.

| **Escape-Sequenz** | **Bedeutung** |
|---|---|
| \h | repräsentiert den Hostnamen einer Maschine |
| \l | enthält die Angabe über das verwendete virtuelle Terminal Device, wie z.B. *tty2* |
| \n | liefert den Netzwerknamen der aktuellen Maschine |
| \r | zeigt das aktuelle Kernel-Release an |
| \u | meldet die Kennung des gerade aktiven Benutzers |
| \w | liefert das "working directory". Hier könnte analog auch das shellinterne Kommando **pwd** aufgerufen werden |

Tabelle 3.3: Beispiele für Escape-Sequenzen

### 3.2.9 Jobkontrolle

Wenn ein Kommando abgesetzt wurde, wartet die Shell normalerweise bis dieses ordnungsgemäß beendet wurde. Anschließend gibt sie wieder den Prompt aus, um auf das nächste Kommando zu warten. Manche Kommandos können jedoch viel Zeit benötigen oder gar während der kompletten Arbeitssitzung laufen. Damit man nicht für jedes Programm, das gestartet werden soll, eine eigene Shell öffnen muss, können Programme, wie man sagt, im Hintergrund gestartet werden. Das bedeutet nichts anderes, als dass die Shell nicht erst auf die Beendigung des abgesetzten Programmes wartet, sondern sofort wieder einen Prompt ausgibt, um ggf. ein weiteres Kommando entgegenzunehmen. Man kann ein Kommando auch direkt im Hintergrund starten, indem man "&"[5] an das Kommando anhängt. Weitere Kommandos können dann eingegeben werden, während das erste ausgeführt wird:

```
dirk@hermes:~> kdvi lak.dvi &
```

Unter der grafischen Oberfläche bedeutet das Anhängen von "&" im Terminalfenster, dass das Fenster, aus dem eine Anwendung gestartet wurde, weiter benutzt werden kann.

Der Sinn der Bezeichnungen Vordergrund und Hintergrund ist unmittelbar eingängig. In technischer Hinsicht sind Vordergrund und Hintergrund zwei Begriffe, die sich im Zusammenhang mit der Shell nur auf ein bestimmtes Terminal beziehen können. Ist die sogenannte Prozess-Gruppen-ID eines Prozesses identisch mit der eines Terminals, so kann der Prozess von diesem Terminal Signale empfangen. Solche Prozesse laufen im Vordergrund. Hintergrund-Prozesse sind solche, deren Prozess-Gruppen-Id von der des Terminals verschieden sind. Sie sind daher auch immun gegen irgendwelche Signale, die an der Tastatur eingegeben werden.

Der Begriff des Jobs ist eine Abstraktion, welche von der Shell zur Verwaltung eingesetzt wird. Als Job wird jede Pipeline bezeichnet, aus wievielen Kommandos oder Prozessen auch

[4] man sehe sich die Ausgabe von `echo $PS1` an

[5] das sogenannte "kaufmännische Und" bzw. "ampersand" im englischen

immer sie bestehen mag. Dem Job wird von der **bash** eine Jobnummer zugewiesen, unter welcher er angesprochen werden kann. Diese Jobnummer ist nicht mit der Prozess-ID zu verwechseln!

Jede Shell implementiert Funktionen zur Jobkontrolle. Ein laufendes Kommando kann mit [STRG]-[Z] angehalten werden. Nach dem Anhalten meldet sich die Shell mit ihrem Prompt wieder, neue Kommandos können eingegeben werden. Das unterbrochene Kommando kann mit dem Befehl **fg** (Foreground) wieder aktiviert werden. Mit dem Befehl **bg** (Background) kann das Kommando im Hintergrund fortgesetzt werden, sofern das Kommando keine Eingabe über die Tastatur (stdin) erfordert. Befinden sich mehrere Prozesse im Hintergrund kann dem **fg** die Jobnummer folgen, wie es weiter unten detailiert beschrieben wird. Die komplette Liste der in einer Shell laufenden Jobs kann mittels des Kommandos **jobs** angezeigt werden:

```
dirk@randy2:~> jobs
[2]-  Running                 konqueror lak.pdf &
[3]+  Running                 kdvi SharedFiles/tex/lak/lak.dvi &
```

In den eckigen Klammern wird die zugeteilte Jobnummer angezeigt. Sie unterscheidet sich von der sogenannten Prozessnummer, die hinter der Jobnummer angegeben wird. Das Plus-Zeichen bei der Ausgabe des **jobs**-Kommandos markiert den zuletzt gestarteten Job, das Minus-Zeichen den als vorletztes gestarteten Job.

Es gibt eine Reihe von Möglichkeiten, sich auf einen bestimmten Job zu beziehen. Das Zeichen % leitet einen Jobnamen ein. Jobnummer $n$ kann als %n angesprochen werden. Man kann sich auch auf einen Job beziehen, indem man dem % die ersten Buchstaben des Kommandos voranstellt, mit dem man den Job gestartet hat. Hat man z.B. Kommando gestartet, kann man sich darauf mittels %ko beziehen, falls kein weiterer laufender Job so beginnt. Auch eine Art von Wildcard ist erlaubt: %?ommando oder auch %?mmando bezieht sich ebenfalls auf den Job, der mittels **kommando** gestartet wurde. Wenn das angegebene Präfix oder Muster auf mehr als einen Job paßt, erfolgt eine Fehlermeldung. %% oder %+ bezieht sich immer auf den letzten Job. In den Begriffen der Shell ist das der zuletzt gestoppte Vordergrundprozess oder der zuletzt gestartete Hintergrundprozess. %- bezieht sich entsprechend auf den zuvorletzt gestarteten Job. Ein im Hintergrund laufendes oder unterbrochenes Programm benötigt unter Umständen die Tastatur bzw. den Bildschirm zur Ein- bzw. Ausgabe, deshalb kann die Shell dann nicht beendet werden, und es erscheint beim Versuch des Ausloggens die Fehlermeldung “There are suspended jobs”.

### 3.2.10 Skripte/Batches

Mehrere Kommandos können in einer Textdatei zusammengefasst werden, zum Beispiel, um immer wiederkehrende Kommandofolgen nicht jedesmal explizit eintippen zu müssen. Ein solcher Stapel von sequenziell auszuführenden Kommandos wird auch als Batch bezeichnet, deshalb findet man häufig auch diese Bezeichnung. Von Kommandostapeln wird exzessiv bei den Runlevel-Skripten Gebrauch gemacht, die eine Maschine in einen bestimmten Zustand versetzen. Diese Textdateien heißen Shell-Skripte und stellen neue Kommandos dar. (Es ist sinnvoll, solche Skripte mit dem emacs zu editieren!) In Shell-Skripten sind mit “#” beginnende Zeilen Kommentare, diese Zeilen werden nicht ausgeführt. Beginnt die erste Zeile mit “#!” so wird zur Ausführung des Shell-Skripts die danach angegebene Shell gestartet: `#!/bin/bash` ruft die Bourne Again Shell auf. Shell-Skripte müssen mit dem Befehl `chmod u+x Skriptname` als ausführbare Kommandos gekennzeichnet werden.

Es ist in der Shell auch möglich - ähnlich wie in einer “richtigen” Programmiersprache - Funktionen zu deklarieren und zu benutzen. Mit dem Kommando **return** hat man

die Möglichkeit, aus einer Funktion einen Wert zurückzugeben. Falls das Kommando **return** nicht eingesetzt wird, hat die Funktion als Rückgabewert den Standardoutput der eingesetzten Kommandos. Beispiel:

```
shadowfileexist() {
      if [ -f /etc/shadow ]
then
return 1 #shadow file exist
      fi
return 0 #does not exist
}
```

Beispiel:

```
count () {
      ls | wc -l # ls: Liste aller Dateien im Verzeichnis
      # wc: Word-Count, zählt Wörter
}
```

Die Funktion gibt die Anzahl der Dateien im aktuellen Verzeichnis zurück; aufgerufen wird diese Funktion wie ein Befehl, also einfach durch die Eingabe von **count**.

## 3.3 Aufgaben

### 3.3.1 Kommandozeile und Umleitung

1. Wie fasst man die beiden Standardausgabekanäle zusammen? Wie entsorgt man Ausgaben (auf irgendeinem Kanel) am besten rückstandsfrei?

2. Rufen Sie ein Kommando auf und zeigen Sie dessen Rückgabewert anschliessend an, jedoch keine seiner direkten Ausgaben (weder Fehler noch die Standardausgabe)!

3. Wie sorgt man dafür, dass der String (*dieses ist ein test*) als ein einziges Argument interpretiert wird?

4. Nennen Sie fünf wichtige Umgebungsvariablen!
   Welche verschiedenen Möglichkeiten bestehen, um an bereits eingegebene Kommandos wieder heranzukommen?

5. Was passiert, wenn ich das Kommando `export PATH=/tmp` eingebe?

# Kapitel 4

# Editoren und Dateibetrachter

Textorientierte Dateien sind ein wesentliches Element einer Linux-Installation. Viele Konfigurationsdateien, man werfe einfach nur einen Blick in das Verzeichnis */etc*, sind textorientiert. Hinzu kommen die Shellskripten[1] und Skripten anderer Interpretersprachen wie Perl oder Python.

Wenn man also wissen möchte, was auf einer Linux-Maschine warum passiert, kommt man auf die Dauer nicht um den Blick in verschiedene Dateien umhin. Zwar gibt es bei den meisten Distributionen grafische Frontends für alle möglichen Systemaufgaben. Diese decken jedoch oft nicht das ganze Spektrum der Systemadministration ab oder ignorieren bestimmte Aspekte ganz.

Wer zum Beispiel beim Login seiner Benutzer eine bestimmte Umgebungsvariable setzen möchte, fügt diese am besten der Datei */etc/profile* oder */etc/profile.d/profile.local* hinzu. Benutzer selbst können Shell- und Variableneinstellungen in den Dateien *.profile* oder *.bashrc* in ihrem Homeverzeichnis vornehmen. Oder ein Admin möchte ein eigenes Runlevelskript anlegen. Für alle diese Aufgaben benötigt man einen Texteditor. Hier ist die Auswahl ziemlich gross. Ein alter Veteran ist Vi, sehr bekannt sich auch **pico**, **nano**, **joe** oder **emacs**.

Einige der genannten Editoren haben auch grafische Brüder und Schwestern: Beispiele sind **gvim** für Vi oder **xemacs** für Emacs. Die verschiedenen Desktopsysteme unter Linux bringen jedoch meistens selbst noch weitere Texteditoren mit, die zum Teil von ihrem Funktionsumfang sehr mächtig sind. Unter KDE gibt es beispielsweise **kate** oder **kwrite**. Gnome-Benutzer hatten vielleicht schon mit **gedit** zu tun.

Darüberhinaus gibt es spezialisierte Editoren für besondere Anwendungen: Der Stream-Editor **sed** arbeitet nicht interaktiv und kann gut für Shell-Skripten verwendet werden. Binärdateien schaut man sich besser mit sogenannten Hexadezimal-Editoren an. **ghex2** ist ein Vertreter dieser Gruppe.

Nicht in jedem Fall braucht man gleich einen Editor. Wenn man nur schnell etwas nachschauen will oder sich ein Logfile anschaut, sind sogenannte Pager, Dateibetrachter, eine oft bessere und schnellere Wahl.

## 4.1 Texteditoren

Texteditoren spielen unter Linux eine grosse Rolle, da sie häufig zum Anlegen und Ändern von Dateien benötigt werden. Fast alle Konfigurationsdateien verschiedener Programme

[1] Eine gute Sammlung dieser findet sich unterhalb von */etc/init.d*: Die Start- und Stopskripten dienen zur anfänglichen Konfiguration der bootenden Maschine und der Kontrolle der diversen Dienste, wie beispielsweise ISDN, SSH oder NFS.

und Services sind Textdateien, welche auf die konkreten Bedürfnisse der User oder des Systems angepasst werden müssen oder können.

### 4.1.1 Joe

Der zu Beginn sicherlich am einfachsten zu bedienende Texteditor ist **joe**, der in seiner Kommandosyntax wordstar-kompatibel ist. Er eignet sich insbesondere für User, die sich partout nicht den **vi** aneignen wollen, aber dennoch eine Möglichkeit suchen, von der Konsole aus einfache Bearbeitungen an Textdateien vorzunehmen. Mit **vi** gemeinsam hat **joe** zum einen den Vorteil, auch dann zu laufen, wenn sonst nichts mehr geht, zum anderen die genügsamen Ansprüche an die Hardware. Im Gegensatz zum **vi**, dessen Bedienungskonzept manchem Neuling als recht kryptisch erscheint, gibt sich **joe** aber etwas konventioneller.

Der Editor wird einfach von der Konsole aus gestartet mit `joe <Dateiname>` Dann erscheint der Dateiinhalt im Editierbereich. Zum Eingeben einfach die Tastatur verwenden, der Cursor kann mit den Cursortasten bewegt werden. **joe** legt von jeder geänderten Datei eine Sicherungskopie an, die aus dem Dateinamen mit angehängtem ~(Tilde) besteht, wenn man es ihm nicht in der Konfigurationsdatei */etc/joerc* explizit verbietet.

| Tasten | Funktion | Tasten | Funktion |
|---|---|---|---|
| Strg-kh | Hilfefenster | Strg-kf | Suchen |
| Strg-kr | Datei einlesen | Strg-l | Weitersuchen |
| Strg-c | Beenden | Strg-kx | Speichern/Beenden |
| Strg-kd | Speichern unter | Strg-kb | Markierung Anfang |
| Strg-kc | Kopieren | Strg-kk | Markierung Ende |
| Strg-km | Verschieben | Strg-ky | Block löschen |
| Strg-ku | Dateianfang | Strg-kv | Dateiende |
| Strg-y | Zeile löschen | | |

Tabelle 4.1: Wichtige Tastenkürzel für **joe**

### 4.1.2 Pico und Nano

Beides sind wie die Namen schon andeuten sollen sehr kleine schlanke Editoren mit begrenztem Funktionsumfang. Ähnlich wie **joe** sind **pico** und **nano** recht leicht zu bedienen.

```
   UW PICO(tm) 4.8             File: /etc/resolv.conf
# /etc/resolv.conf
options timeout:1 attempts:1 rotate
nameserver 132.230.200.200
nameserver 132.230.200.201
search ruf.uni-freiburg.de uni-freiburg.de goe.net









                              [ Read 4 lines ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Pg   ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where is  ^V Next Pg   ^U UnCut Text^T To Spell
```

In den letzten zwei Zeilen sind die Kürzel für wichtige Kommandos angegeben.

## 4.2 Vi

Zu den ältesten Editoren, der eng mit den Anfängen der Unix-Ära verbunden ist, gehört der Vi. **vi** oder in der grafischen Version **gvim** ist für den Anfänger nicht so leicht zu bedienen, eignet sich aber besonders für die Arbeit über recht langsame Netzwerkverbindungen. Bis es soweit ist, liegt aber ein steiniger Weg vor einem. Vieles, was man bisher von Editoren wusste, gilt beim **vi** nicht mehr. Alles wird mit Hilfe der Tastatur gemacht, die Maus verliert im **vi** ihre Bedeutung als zentrales Hilfsmittel. Diesem Effekt ist es zu verdanken, dass der Vi immer zur Verfügung steht. Wenn nichts mehr geht, geht immer noch ein Vi. Daher ist er Teil der meisten Rettungssysteme. Da man in einem Notfall keine Zeit hat, sich in den Vi einzuarbeiten, sollte man sich schon vorher einmal mit den wichtigsten Grundfunktionen vertraut machen.

Damit der Umstieg zum Vi nicht zu schmerzhaft ist, gibt es eine Vielzahl von Vi-Clonen. Der oben genannte **gvim** ist eine angenehme Mischung aus der Stärke des Vi mit der Einfachheit von **gedit**. Gvim kann entweder wie Vi über die Tastatur bedient werden oder auch mit Hilfe der Maus. Hier erfolgt nur eine kurze Einführung, [2] die einige Grundlagen für die Bedienung von Rettungssystemen oder anderen Unixes legen soll.

Anders als die meisten Editoren unterscheidet Vi zwischen einem Kommando- und einem Eingabemodus. Durch Drücken der Taste [i] für "Input" erreicht man den Eingabemodus. Ein Druck auf die Taste [ESC] beendet den Eingabemodus, und man befindet sich wieder im Kommandomodus. Die wohl wichtigste Tastenkombination ist die zum Verlassen des Editors, falls man ihn nur versehentlich geöffnet haben sollte :-)) (ohne dabei die geöffnete Datei zu schreiben): Nacheinander sind [ESC],[:],[q],([!]) zu drücken.

| **Taste(n)** | **Dateioperation** | **Taste(n)** | **Funktion** |
|---|---|---|---|
| :w | Speichert geöffnete Datei | /Muster | Suche vorwärts |
| :w | Speichert nur lesbare Dateien | ?Muster | Suche rückwärts |
| :w d1.txt | Speichert unter *d1.txt* | x | Zeichen löschen |
| :q | »vi« beenden | dd | Zeile löschen |
| :e d2.txt | Öffnet *d2.txt* | i | Eingabe (insert) |
| :n | Gehe zur nächsten geladenen Datei | r | Eingabe (overwrite) |
| :f | Zeigt Namen der aktuellen Datei | u | Wiederherstellung (Undo) |

Tabelle 4.2: Wichtige Tastenkürzel des **vi** im Kommandomode

### 4.2.1 Kommandomodus - Kurzüberblick

Viele Aktionen erledigt man mit dem Kommando-Modus des Vi. Mit einzelnen Tasten lässt sich im Text navigieren oder auch Zeilen bzw. Zeichen löschen. Für erste Versuche lade man einfach eine Datei. Diese hat man besser vorher kopiert, damit nix schief geht.

```
dirk@s02:~> cp /etc/resolv.conf
dirk@s02:~> vi resolv.conf
```

Nach dem Aufruf des Editors befindet man sich im Kommandomodus. Wenn man nun ein einzelnes Zeichen löschen möchte, kann man das durch das Drücken der Taste [x] errei-

[2] für eine ausführliche Anleitung siehe: http://www.selflinux.org/selflinux-devel/html/vim.html

chen. Ein solcher Befehl, wie auch das Löschen einer Zeile durch [d],[d], kann mit der Taste [u] rückgängig gemacht werden.

Um den Cursor im Text zu bewegen, sollten in den meisten Fällen die Pfeiltasten funktionieren. Sollte dies aufgrund fehlerhafter Einstellungen, beispielsweise über eine SSH-Verbindung von einer Windows-Maschine aus, einmal nicht funktionieren, gibt es hierzu Tastenalternativen. Da die Buchstaben im Kommandomodus nicht im Text landen, kann man alternativ mit den Tasten [h] den Cursor ein Zeichen nach links setzen, mit [j] in die nächsten Zeile gehen, mit [k] eine Zeile nach oben springen oder durch das Drücken von [l] sich ein Zeichen nach rechts bewegen.

Seitenweises Blättern übernehmen die Kombinationen [Strg]+[u] statt der Bild-nach-oben-Taste und [Strg]+[d] statt Bild-nach-unten.

Eine ganze Reihe der Vi-Kommandos kann vervielfältigt werden. So löscht [9],[x] neun Zeichen ab der aktuellen Cursor-Position und [2],[d],[d] zwei Zeilen. Dies funktioniert mit den meisten anderen Kommandos analog.

Eine Standardaktion ist das Kopieren von ganzen Zeilen: Hierzu dient die Tastenkombination [y],[y]. Dieser speichert die aktuelle Zeile in einem Puffer. Die gespeicherten Daten lassen sich mit [p] wieder an einer anderen Stelle einfügen nach der gewählten Zeile einfügen. [Shift]+[p] macht es vor dieser Zeile. Analog zum bereits gesagten lassen sich mit [3],[y],[y] drei Zeilen kopieren.

Änderungen an einer Datei speichert man mit [:],[w]. Dateien unter einem anderen Namen legt man mit [:],[w] *neueDatei.txt* ab. Der Vi beendet, wenn man [:],[q] eingibt, wie in der Einführung schon kommentarlos angedeutet wurde. Auch hier sind Kombinationen möglich; so kann man eine Datei mit [:],[w],[q] sichern und dann den Editor verlassen.

### 4.2.2 Cursor-Navigation

Möchte man nicht stumpf in grossen Dateien stundenlang auf dem Cursor stehen, um beispielsweise an das Ende einer Datei zu kommen, helfen einige spezielle Tastenkürzel.

Einige werden vielleicht bestimmte Tastenkürzel wiedererkennen oder Ähnlichkeiten feststellen: So springt beispielsweise [Strg]+[e] an das Ende der Kommandozeile in der Bash.

### 4.2.3 Suchen, Ändern, Ersetzen & Co.

Die einfachen Löschbefehle für Zeichen und Zeilen wurden schon vorgestellt, leicht anders reagiert [Shift]+[x]. Hier wird das Zeichen vor dem Cursor entfernt. [Shift]+[d] löscht alles bis zum Zeilenende [d],[^] alles bis zum Zeilenanfang ausgehend von der Cursorposition. [d],[2],[w] löscht zwei Wörter ab Cursorposition, statt der "2" sind natürlich beliebige natürliche Zahlen erlaubt.

Neben dem [i] für den Editormodus vor dem Cursorzeichen, aktiviert [a] Eingaben nach der aktuellen Cursorposition. [Shift]+[a] setzt die Eingabeposition auf das Zeilenende. [o] beginnt eine neue Zeile nach der Zeile, auf der der Zeiger steht.

[s] ersetzt einzelne Zeichen, [Shift]+[s] komplette Zeilen. [Shift]+[j] ist oft sehr nützlich: Es entfernt den Zeilenumbruch am Ende einer Zeile und fügt damit die aktuelle und die darauffolgende Zeile zusammen.

Die Suche im Vi gestaltet sich analog zu **less**: `/muster` sucht vorwärts nach "muster", `?text` macht dieses einfach rückwärts. [n] sucht noch einmal in der bestehenden Richtung. [Shift]-[n] sucht noch einmal in der umgekehrten Richtung.

Etwas komplexer ist ein Befehl zum Suchen und Ersetzen von Zeichenketten aufgebaut: `[ addr ] s/muster/ersetzung/ [ g ]` tauscht einmalig "muster" durch "ersetzung" aus.

| Taste(n) | Cursorsprung | Taste(n) | Cursorsprung |
|---|---|---|---|
| Pfeiltasten | Navigation wie sonst auch | Strg+d | - Blättert halbe Seite nach unten |
| Strg+u | Blättert halbe Seite nach oben | Strg+f | Blättert komplette Seite nach unten |
| Strg+b | Blättert komplette Seite nach oben | :0 | Springt auf Dateianfang |
| :< $n$ > | Springt zur Zeile < $n$ > | :$ | Springt zum Dateiende |
| 0 | Springt zum Zeilenanfang | ^ | Springt zum ersten Nicht-Leerzeichen |
| $ | Springt zum Zeilenende | ENTER | Springt zum Beginn nächste Zeile |
| % | Zeigt zugehörige Klammer | Shift+g | Springt zur letzten Zeile |
| Shift+h | Springt zur ersten Zeile im aktuellen Fenster | Shift+l | Springt zur letzten Zeile im aktuellen Fenster |
| Shift+m | Springt in die Mitte des aktuellen Fensters | - | Springt zum ersten Nicht-Leerzeichen der Zeile drüber |
| + | Springt zum ersten Nicht-Leerzeichen der nächsten Zeile | j | Springt zur nächsten Zeile in der gleichen Spalte |
| k | Springt zur vorhergehenden Zeile in der gleichen Spalte | h | Springt ein Zeichen nach links |
| l | Springt ein Zeichen nach rechts | w | Springt ein Wort vorwärts |
| b | Springt ein Wort rückwärts | e | Springt zum Ende des Wortes |
| ( | Springt zum vorhergehenden Satz | ) | Springt zum nachfolgenden Satz |
| { | Springt zum vorhergehenden Absatz | { | Springt zum nachfolgenden Absatz |

Tabelle 4.3: Sprungbefehle im **vi** zur Cursor-Navigation

Mit "addr" kann ein Bereich angegeben werden, in dem die Aktion durchgeführt werden soll. Die Zeilennummern sind durch ein Komma zu trennen. "g" führt die Aktion an jeder gefundenen Stelle durch. Beispiel: `:4,14s/bla/blub/g` ersetzt in den Zeilen von 4 bis 15 alle Zeichenketten "bla" durch "blub".

## 4.3 Emacs

Emacs ist wohl einer der flexibelsten Editoren unter Linux/Unix. Es gibt umfangreiche Unterstützungen für die verschiedensten Programmiersprachen und Formatierungen. Der Editor emacs verfügt über eine mächtige Lisp-Unterstützung für weitestgehende Konfiguration. Funktionen lassen sich entweder durch Benutzung der Maus oder durch Tastenkombinationen aktivieren. In einer Emacs-Sitzung kann man viele Dinge parallel erledigen: Dateien eines Verzeichnisses umbenennen, Dateien bearbeiten, die eigentliche Aufgabe eines Editors, den Compiler für eine Programmiersprache starten ... Jede dieser Aufgaben erledigt Emacs in einem eigenen (Daten-)Puffer. Ein spezieller Puffer ist "scratch", der für Notizen gedacht ist, die nicht gespeichert werden sollen. Ruft man den Emacs ohne Dateinamen auf, wird per default der Puffer Scratch gezeigt.

| Tasten | Funktion | Tasten | Funktion |
|---|---|---|---|
| Strg-g | Vorgang abbrechen | Strg-x Strg-s | Buffer speichern |
| Strg-s | Wort suchen | Strg-x Strg-c | Beenden/Verlassen |
| Strg-_ | Undo | Strg-x Strg-f | Datei laden |
| Strg-w | Text ausschneiden | Strg-x Strg-i | Datei einfügen |
| Strg-SPC | Marke setzen | Strg-x 1 | Fenster schliessen |
| Strg-y | Text einfügen | Strg-h m | Bearbeitungsmodus |
| F10 | Menü (versionsabhängig) | | anzeigen |

Tabelle 4.4: Wichtige Tastenkürzel für **emacs**

## 4.4 Aufgaben

1. Wie heisst Ihr/Dein Lieblings-Texteditor (verordnete Standardeditor)? Wie kann man in diesem:
   a) Ihn verlassen ohne zu speichern?
   b) Verlassen und dabei speichern?
   c) Einen Block kopieren,
   d) verschieben,
   e) löschen?

2. Wie lösche ich eine ganze Zeile in einem Schritt?

3. Editieren Sie eine Datei mit dem Texteditor Ihrer Wahl: Nennen Sie sie *uebung1.txt* und schreiben Sie in diese Datei: Ihren Usernamen (Zeile 1), Vollständigen Namen (Zeile 2)!

4. Tauschen Sie in einem Übungstext (z.B. diesem Skript) die Umlaute gegen ihre HTML-Umschrift (ä, ü, ...) aus!

5. Man hänge an die Übungsdatei den veränderten Übungstext (in Teilen) an!

6. Man schreibe einen Textblock aus zwei Zeilen und kopiere diesen zehn mal!

7. Wie hängt man am besten die ersten zehn der */etc/passwd* an den Anfang der Datei *uebung1.txt*?

# Kapitel 5

# Linux-Administration mittels Shell

Nach einem kurzen Ritt durch die Anmeldung an eine Linux-Maschine, die Benutzung der Kommandozeile, bestimmter Tastenkombinationen und Kommandos soll es nun um tiefergehende Fragestellungen gehen, die sich eher in der Domäne des Administrators bewegen.

Für die Systemadministration von Linux-Systemen existieren inzwischen auch eine ganze Reihe grafischer Tools. Viele Aufgaben lassen sich mit diesen jedoch nicht sonderlich bequem über schmalbandige Fernverbindungen abwickeln oder schwer mit diesen automatisieren. Auch Microsoft hat für seine neueren Windows-Systeme erkannt, dass inzwischen kein Weg mehr an der Kommandozeile vorbeiführt. Auch hier gibt es bestimmte Aufgabenstellungen, die sich nicht mehr durch das Ausfüllen bunter Karteikarten erledigen lassen.

Die Kommandozeile, gesteuert durch die Shell, bildet die Schnittstelle zwischen Benutzer und Betriebssystem. Die Shell ist darüberhinaus das Standardwerkzeug im Systembetrieb. Gegenüber dem "Normalbenutzer" ist der Administrator häufig eher gezwungen oder gewillt sich die Kommandozeile und ihre innewohnenden Möglichkeiten zunutze zu machen.

Dieser Abschnitt ist mehrgeteilt. Er vertieft die einzelnen Aspekte und Themen, die in einem früheren Kapitel vorkamen. Zuerst geht es um die Standard-Shell, ihre Anwendung als Skriptinterpreter und die Kommandozeile, die Verknüpfung von Befehlen, weiterhin um spezielle Tastenkombinationen, die eine direkte Kontrolle von Kernel-Aspekten erlauben, gefolgt von Abschnitten zur Remote-Administration mit SSH.

## 5.1 Einige Bash-Grundlagen

An dieser Stelle soll nur kurz auf die Linux-Standard-Shell, die Bourne Again Shell **bash**, eingegangen werden. Weitere Informationen findet man in der Man-Page (`man bash`). Die **bash** verfügt, neben der obligatorischen Manpage, auch über eine eingebaute Hilfe Funktion. Mit dem Kommando **help** wird eine Übersicht der **bash** eigenen Kommandos angezeigt. Shell-Skripte sind das A und O der UNIX-/Linux-Systemadministration. Fast alle Administrationsaufgaben werden automatisch von Skripten abgearbeitet. Für Administratoren ist es daher unerlässlich, wenigstens Grundzüge der Shell-Programmierung zu beherrschen. Shell-Skripte besitzen eine erweiterte Syntax: Es stehen mathematische Operationen und Vergleichsoperationen wie in anderen Programmiersprachen zur Verfügung. Shells kennen verschiedene Kontrollstrukturen.

Eine der nützlichsten Funktionen der **bash** ist die Möglichkeit, Programm- und Dateinamen zu vervollständigen. Dabei kann dieses zu jeder Zeit am Shell-Prompt geschehen: Man tippe ein paar Zeichen eines Befehls und betätige anschliessend die Tabulator-Taste. Wenn die Eingabe bis zu der Stelle, an der der Tabulator ausgelöst wurde, eindeutig war, es also kein weiteres ausführbares Programm gibt, das mit diesen Buchstaben beginnt, wird die

Eingabe automatisch vervollständigt. Sollte es zwei oder mehr Mglichkeiten geben, so wird man mittels Signalton verständigt. Ein nochmaliges Drücken der besagten Taste zeigt nun alle erhältlichen Alternativen an. Man probiere es einfach an einem Beispiel aus: Nimmt man an, ein Admin möchte sich die Datei */var/log/messages* ansehen. Dazu wird beispielsweise das Programm **less** benutzt. Man tippe also einfach mal "le" und die Tabulator-Taste zweimal. Nun werden einige Programme angezeigt, welche mit der besagten Buchstabenkombination beginnen.

Nun tippe man so lange weitere Zeichen ein, bis die Eingabe eindeutig ist, in diesem Beispiel sollte ein zusätzliches "s" dem geforderten Zweck genügen. Jetzt tippe man nochmals die [Tab]-Taste, und der Befehl wird ergänzt. Was bei Befehlen vielleicht noch etwas banal aussieht, wird bei langen Dateinamen mit kryptischen Zeichen sehr schnell zu einer großen Erleichterung. Neben dieser Funktionalität beherrscht die Bash weitere Komplettierungen.

Wozu die **bash** den begonnenen Namen zu vervollständigen sucht, hängt von der getätigten Eingabe ab. Wenn die Eingabe mit einem $ beginnt, so versucht die Bash, einen Variablennamen daraus zu machen. Beginnt die Eingabe hingegen mit einer Tilde (~) , so versucht sie einen Benutzernamen zu bilden. Fängt sie mit @ an, versucht sie die Eingabe zu einem Hostnamen zu vervollständigen. Wenn keine dieser Bedingungen zutrifft, sucht die Bash nach einem Alias- oder Funktionsnamen. Selbstverständlich muss der Name, zu dem die **bash** vervollständigt - sei es nun eine Variable, ein Benutzername, ein Hostname, ein Alias, eine Funktion oder ein Pfad - auch wirklich existieren. Wenn alle Versuche, eine passende Vervollständigung zu erreichen, fehlschlagen, ertönt ein kurzer Signalton.

### 5.1.1 Aufbau der Kommandozeile

Der Bash-Prompt, vielfach auch als Eingabeaufforderung bezeichnet, kann zwar beliebig angepasst werden, doch auf den meisten Systemen ist es für normale Benutzer (also alle außer dem Systemadministrator) standardmäßig eine Zeichenfolge nach dem Schema:

```
benutzer@computer:verzeichnis>
```

Wenn man sich auf dem Computer im eigenen Heimatverzeichnis befindet, welches durch die Tilde (~) repräsentiert wird, lautet der Bash-Prompt entsprechend:

```
dirk@dozent:~>
```

Der Benutzername ist "dirk" (ein ganz normaler Linux-Benutzer ohne Administratorrechte) und der Rechnername lautet "dozent". Dahinter blinkt der Cursor, an dem die Eingabe erfolgen kann. Nachdem ein Kommando ausgeführt wurde, gelangt man wieder zum Prompt. Es gibt sehr viele Kommandos, wie man sich überzeugen kann, wenn man einfach zweimal auf die Tabulatortaste tippt. Einige sind in die Bash direkt eingebaut, z.B. das Kommando **echo** zur Ausgabe von Zeichenketten. Andere sind klassische ausführbare Programme oder Skripte, wie **mount** zum Einbinden von Dateisysteme, oder **updatedb** zum Erzeugen der Locate-Datenbank. Auch Programme für die grafische Benutzerschnittstelle, wie **firefox** können von der Bash aus gestartet werden, wenn die Bash in einer Terminal-Emulation unter X11 läuft.

### 5.1.2 Die Kommando-Geschichte

Fast jede Linux-Shell verfügt über eine Liste der zuletzt abgesetzten Kommandos, eine sogenannte "History". Selbst wenn man nur gelegentlich die Kommandozeile verwendet, erweist sich die History als ein ausgesprochen nützlicher Helfer. Das gilt umso mehr, wenn man ausgiebigen Gebrauch von der Shell macht. Die Möglichkeiten zur Nutzung der History

entsprechen der Benutzung eines effizienten Editors und werden in ihrem vollen Umfang nur von den Wenigsten benutzt. Sie gehen weit über die Möglichkeiten beispielsweise von "doskey" hinaus, welches vielleicht noch aus Windows/DOS-Zeiten bekannt sein könnte.

Die zuletzt eingetippten Befehle können mit den Cursor-Tasten (Pfeil hoch und runter) ausgewählt und editiert werden. Eine Rückwärtssuche in allen gespeicherten Kommandoaufrufen kann mittels [Strg]-[r] erfolgen. Nach dem Abmelden von einer Shell wird die "Geschichte", die History, in der Datei *.bash_history* im Home-Verzeichnis des jeweiligen Benutzers abgelegt. Das Aussehen und die Länge der Bash-History werden durch Umgebungsvariable definiert:

```
export HISTTIMEFORMAT="%m/%d %H:%M:%S "
export HISTSIZE=1000
export HISTFILE=/dev/null
```

Möchte man nicht, dass andere die Command History nachvollziehen können, hilft ein Link dieser Datei nach */dev/null*. Für eine Sitzung kann man die History durch die Eingabe von: `export HISTFILE=/dev/null` ebenfalls abschalten. Mit `!!` wird der letzte Befehl ausgeführt, mit `!-2` wird der vorletzte mit `!-3` der vorvorletzte usw. ausgeführt. Mit "!string" wird der letzte Befehl, der mit "string" begonnen hat, aufgerufen. Der eingebaute Bash-Befehl **history** zeigt eine Liste der zuletzt abgesetzten Kommandos an.

### 5.1.3 Abkürzen von Befehlen

Der Alias-Mechanismus dient der Ersparnis von Tipparbeit, macht Kommandos leichter erinnerbar, verschönert Kommandoausgaben und kann auch zur Absicherung gegen Tippfehler verwendet werden. Ein Alias ist eine definierte Zeichenfolge, die für eine andere Zeichenfolge steht. Welche Aliase in der aktuellen Shell definiert sind, kann mittels **alias** abgefragt werden:

```
dirk@hermes:~> alias
alias +='pushd .'
alias -='popd'
alias ..='cd ..'
[... weitere Alias-Deklarationen ...]
```

Die Alias-Definitionen muss man nicht jedesmal eingeben, sondern kann sie in den Setup-Dateien unterbringen. Listen solcher Abkürzungen der Form `alias <Abkürzung> <Befehl (mit Optionen)>` trägt man am besten in die globale Datei */etc/profile* oder in benutzerlokale Dateien, wie *.profile* oder *.bashrc* ein. Der Befehl **alias** ohne Argumente zeigt alle bereits vergebenen Abkürzungen an. Mit dem Befehl **unalias** kann man Definitionen wieder aufheben.

### 5.1.4 Wildcards in Dateinamen

Viele Kommandos, zum Beispiel **ls** oder **cp** haben als Argumente die Namen von Dateien oder Verzeichnissen. Namen, die Wildcards enthalten, werden zu einer Liste von Namen expandiert. Wildcards sind nahe verwandt zu Regulären Ausdrücken, die von vielen Programmen verarbeitet werden können. Beispiele von Wildcards sind:

### 5.1.5 Zeichen mit besonderer Bedeutung

Eine ganze Reihe von Zeichen haben für die Shell besondere Bedeutung, ein Teil hiervon wurde bereits in den vorherigen Abschnitten vorgestellt:

| Wildcard | Bedeutung |
|---|---|
| ? | beliebiges Zeichen |
| * | beliebige Zeichenkette, auch der Länge Null |
| $[abc]$ | die einzelnen Zeichen a, b oder c |
| $[a-dg-i]$ | die einzelnen Zeichen a, b, c, d, g, h oder i |
| $[\hat{}abd]$ | nicht die einzelnen Zeichen a, b oder d |
| $\{dies, das\}$ | optionale Zeichenketten |

Tabelle 5.1: Beispiele für Wildcards

```
[Space], [Tab], [CR], $, *, [, ], ?, {, }, ~, -,
<, >,&, !, |, ;, (, ),\ , ', ``, `.
```

Um diese Zeichen benutzen zu können, ohne dass die Shell sie interpretiert, müssen diese quotiert werden. Ein einzelnes Sonderzeichen wird durch vorstellen eines Backslash ("\") quotiert:

```
echo Bitte Drücken Sie eine Taste\!
```

Wenn man doppelte Anführungsstriche ("Text"), so werden von der Shell nur noch !, $, und ` als Sonderzeichen behandelt. Man kann also z.B. noch Umgebungsvariablen verwenden: `echo ``$PRINTER''` gibt den Wert der Variablen "PRINTER" aus. (vgl. Kap. **??**, S. **??** Verwendet man einfache Anführungsstriche[1] ('Text'), so werden von der Shell nur noch ! und ` als Sonderzeichen behandelt.

Eine vergleichbare Rolle spielen eine ganze Reihe von Zeichen im Zusammenhang mit sogenannten "Regulären Ausdrücken".[2] Diese werden von einer ganzen Reihe von Kommandozeilenprogrammen verstanden.

### 5.1.6 Spezielle Escape-Sequenzen

Einige Zeichen haben in Zusammenhang mit dem Backslash in der Shell eine besondere Bedeutung und werden speziell interpretiert, bevor eine Ausgabe auf dem Bildschirm erscheint. Dies wird z.B. zum Aufbau der Prompt-Variablen[3] benutzt oder für besondere Ausgaben in der */etc/issue* oder */etc/motd*.

Der Standardpromt der meisten Distributionen sieht oft so aus: `PS1=''\u@\h \w \$ ''`. Wenn man das Ganze nun etwas bunter haben will, kann auch dafür gesorgt werden. Das Ganze sieht dann zwar etwas kryptischer aus, man muss es aber ja nicht ständig eingeben:

```
PS1="\[\033[0;32;40m\u@\h:\w\$ \]"
```

Im Beispiel wird der komplette Prompt in grün dargestellt. \033 leitet die Escape Sequenz ein, [ eröffnet die Farbangabe. Die anschließende 0 gibt an, dass eine Normaldarstellung benutzt wird. Welche anderen Möglichkeiten man an dieser Stelle hat, wird später erwähnt. Die gesamte Sequenz ist in \[ und \] eingeschlossen, damit sie nicht in die Ausgabe mit hinein kommen und Platz auf der Shell wegnehmen.

Als nächstes wird die Vordergrundfarbe gewählt (in diesem Fall 32, das entspricht Grün). Die Hintergrundfarbe 40 steht für die Farbe Schwarz. Möchte man in unserem Beispiel nicht, dass die Schrift nach dem Prompt auch grün ist, so fügt man an den Schluss die

[1] nicht ` das sogenannte Backtick

[2] engl. regular expressions, siehe hierzu Kap. 5.4.1 S. 54

[3] man sehe sich die Ausgabe von `echo $PS1` an

| Sequenz | Bedeutung |
|---|---|
| \h | repräsentiert den Hostnamen einer Maschine |
| \l | enthält die Angabe über das verwendete virtuelle Terminal Device, wie z.B. *tty2* |
| \n | liefert den Netzwerknamen der aktuellen Maschine |
| \r | zeigt das aktuelle Kernel-Release an |
| \s | gibt den Namen der Shell aus |
| \u | meldet die Kennung des gerade aktiven Benutzers |
| \v | meldet die altuelle Version der Shell |
| \w | liefert das "working directory". Hier könnte analog auch das shellinterne Kommando **pwd** aufgerufen werden |

Tabelle 5.2: Beispiele für Escape-Sequenzen

Escape Sequenz \033[0m an. Dies ist die Voreinstellung für die Shellfarbe. Sowohl für den Vordergrund als auch für den Hintergrund stehen 8 Farben zur Verfügung. Das Setzen der Hintergrundfarbe verläuft genauso, der einzige Unterschied ist die führende 4.

| Farbe | Code | Farbe | Code |
|---|---|---|---|
| schwarz | 30 | rot | 31 |
| grün | 32 | gelb | 33 |
| blau | 34 | magenta | 35 |
| cyan | 36 | weiß | 37 |

Tabelle 5.3: Farbcodes für Konsolenschrift

Wie schon gesagt, ist die 0 direkt nach der ersten Escape Sequenz die Voreinstellung für die Schrift des Shell-Promptes. Für die Schrifteigenschaft sind die folgenden Werte sinnvoll: 0, 1, 22, 4, 24, 5, 25, 7, 27. Sie bedeuten: Standard, dick, nicht dick, unterstrichen, nicht unterstrichen, blinkend, nicht blinkend, invers, nicht invers.

### 5.1.7 Jobkontrolle

Wenn ein Kommando abgesetzt wurde, wartet die Shell normalerweise bis dieses ordnungsgemäß beendet wurde. Anschließend gibt sie wieder den Prompt aus, um auf das nächste Kommando zu warten. Manche Kommandos können jedoch viel Zeit benötigen oder gar während der kompletten Arbeitssitzung laufen. Damit man nicht für jedes Programm, das gestartet werden soll, eine eigene Shell öffnen muss, können Programme, wie man sagt, im Hintergrund gestartet werden. Das bedeutet nichts anderes, als dass die Shell nicht erst auf die Beendigung des abgesetzten Programmes wartet, sondern sofort wieder einen Prompt ausgibt, um ggf. ein weiteres Kommando entgegenzunehmen. Man kann ein Kommando auch direkt im Hintergrund starten, indem man "&"[4] an das Kommando anhängt. Weitere Kommandos können dann eingegeben werden, während das erste ausgeführt wird:

```
dirk@hermes:~> kdvi lak.dvi &
```

Unter der grafischen Oberfläche bedeutet das Anhängen von "&" im Terminalfenster, dass das Fenster, aus dem eine Anwendung gestartet wurde, weiter benutzt werden kann.

[4]das sogenannte "kaufmännische Und" bzw. "ampersand" im englischen

Der Sinn der Bezeichnungen Vordergrund und Hintergrund ist unmittelbar eingängig. In technischer Hinsicht sind Vordergrund und Hintergrund zwei Begriffe, die sich im Zusammenhang mit der Shell nur auf ein bestimmtes Terminal beziehen können. Ist die sogenannte Prozess-Gruppen-ID eines Prozesses identisch mit der eines Terminals, so kann der Prozess von diesem Terminal Signale empfangen. Solche Prozesse laufen im Vordergrund. Hintergrund-Prozesse sind solche, deren Prozess-Gruppen-Id von der des Terminals verschieden sind. Sie sind daher auch immun gegen irgendwelche Signale, die an der Tastatur eingegeben werden.

Der Begriff des Jobs ist eine Abstraktion, welche von der Shell zur Verwaltung eingesetzt wird. Als Job wird jede Pipeline bezeichnet, aus wievielen Kommandos oder Prozessen auch immer sie bestehen mag. Dem Job wird von der **bash** eine Jobnummer zugewiesen, unter welcher er angesprochen werden kann. Diese Jobnummer ist nicht mit der Prozess-ID[5] zu verwechseln!

Jede Shell implementiert Funktionen zur Jobkontrolle. Ein laufendes Kommando kann mit [Strg]-[z] angehalten werden. Nach dem Anhalten meldet sich die Shell mit ihrem Prompt wieder, neue Kommandos können eingegeben werden. Das unterbrochene Kommando kann mit dem Befehl **fg** (Foreground) wieder aktiviert werden. Mit dem Befehl **bg** (Background) kann das Kommando im Hintergrund fortgesetzt werden, sofern das Kommando keine Eingabe über die Tastatur (stdin) erfordert. Befinden sich mehrere Prozesse im Hintergrund kann dem **fg** die Jobnummer folgen, wie es weiter unten detailiert beschrieben wird. Die komplette Liste der in einer Shell laufenden Jobs kann mittels des Kommandos **jobs** angezeigt werden:

```
dirk@linux02:~> jobs
[2]-  Running                 konqueror lak.pdf &
[3]+  Running                 kdvi SharedFiles/tex/lak/lak.dvi &
```

In den eckigen Klammern wird die zugeteilte Jobnummer angezeigt. Sie unterscheidet sich von der sogenannten Prozessnummer, die hinter der Jobnummer angegeben wird. Das Plus-Zeichen bei der Ausgabe des **jobs**-Kommandos markiert den zuletzt gestarteten Job, das Minus-Zeichen den als vorletztes gestarteten Job.

Es gibt eine Reihe von Möglichkeiten, sich auf einen bestimmten Job zu beziehen. Das Zeichen % leitet einen Jobnamen ein. Jobnummer $n$ kann als %n angesprochen werden. Man kann sich auch auf einen Job beziehen, indem man dem % die ersten Buchstaben des Kommandos voranstellt, mit dem man den Job gestartet hat. Hat man z.B. Kommando gestartet, kann man sich darauf mittels %ko beziehen, falls kein weiterer laufender Job so beginnt. Auch eine Art von Wildcard ist erlaubt: %?ommando oder auch %?mmando bezieht sich ebenfalls auf den Job, der mittels **kommando** gestartet wurde. Wenn das angegebene Präfix oder Muster auf mehr als einen Job paßt, erfolgt eine Fehlermeldung. %% oder %+ bezieht sich immer auf den letzten Job. In den Begriffen der Shell ist das der zuletzt gestoppte Vordergrundprozess oder der zuletzt gestartete Hintergrundprozess. %- bezieht sich entsprechend auf den zuvorletzt gestarteten Job. Ein im Hintergrund laufendes oder unterbrochenes Programm benötigt unter Umständen die Tastatur bzw. den Bildschirm zur Ein- bzw. Ausgabe, deshalb kann die Shell dann nicht beendet werden, und es erscheint beim Versuch des Ausloggens die Fehlermeldung “There are suspended jobs”.

[5] Dem Konzept und Management von Prozessen ist ein eigenes Kapitel gewidmet

### 5.1.8 Skripte/Batches

Sehr viele administrative Aufgaben laufen unter Unix/Linux durch Skripte. Diese Skripte[6] sind üblicherweise Batches[7] von Programmabfolgen und Kontrollstrukturen, die von der Shell interpretiert werden. Möchte man überprüfen, welche Rechner gerade in einem bestimmten Netzwerkbereich erreichbar sind, kann das z.B. durch folgendes Skript geschehen:

```
declare -i i ; i=0
while [ $i -lt 255 ] ;
  do ping -c1 -w1 172.16.20.$i 1>/dev/null \
        && echo 172.16.20.$i is alive ; i=$i+1
done
```

Die Init-Skripte zum Starten und Stoppen von Diensten sind ebenfalls Batches von Kommandos, die im Shell-Kontext ausgeführt werden; hier exemplarisch ein Ausschnitt aus der */etc/init.d/nfsserver*:

```
[...]
case "$1" in
    start)
      PARAMS=3
      test "$USE_KERNEL_NFSD_NUMBER" -gt 0 && PARAMS="$USE_KERNEL_NFSD_NUMBER"

      echo -n "Starting kernel based NFS server"
      /usr/sbin/exportfs -r
      rc_status
      /usr/sbin/rpc.nfsd $PARAMS
      rc_status
      startproc /usr/sbin/rpc.mountd
      rc_status -v
      ;;
    stop)
      echo -n "Shutting down kernel based NFS server"
      /usr/sbin/exportfs -au
      killproc -n -KILL nfsd
      rc_status
      killproc    -TERM /usr/sbin/rpc.mountd
      rc_status -v
      ;;
[... some more lines ...]
    *)
      echo "Usage: $0 {start|stop|status|try-restart|restart|force-reload|reload}"
      exit 1
      ;;
esac
rc_exit
```

Mehrere Kommandos können in einer Textdatei zusammengefasst werden, zum Beispiel, um immer wiederkehrende Kommandofolgen nicht jedesmal explizit eintippen zu müssen. Von Kommandostapeln wird exzessiv bei den Runlevel-Skripten Gebrauch gemacht, die eine Maschine in einen bestimmten Zustand versetzen. Diese Textdateien heißen Shell-Skripte und stellen neue Kommandos dar.[8]

[6] Der Shellprogrammierung ist ein eigenes Kapitel gewidmet, so dass hier nur ein kurzes Beispiel zu Illustration angegeben wird.

[7] engl. Stapel, ursprünglich von der sequenziellen Programmabarbeitung mit Lochkarten

[8] Es ist sinnvoll, solche Skripte mit einem Texteditor zu bearbeiten. Einige Texteditoren werden in diesem Skript vorgestellt.

In Shell-Skripten sind mit "#" beginnende Zeilen Kommentare, diese Zeilen werden nicht ausgeführt. Beginnt die erste Zeile mit "#!" so wird zur Ausführung des Shell-Skripts die danach angegebene Shell gestartet: `#!/bin/bash` ruft die Bourne Again Shell auf. Shell-Skripte müssen mit dem Befehl `chmod u+x Skriptname` als ausführbare Kommandos gekennzeichnet werden.

Es ist in der Shell auch möglich - ähnlich wie in einer "richtigen" Programmiersprache - Funktionen zu deklarieren und zu benutzen. Mit dem Kommando **return** hat man die Möglichkeit, aus einer Funktion einen Wert zurückzugeben. Falls das Kommando **return** nicht eingesetzt wird, hat die Funktion als Rückgabewert den Standardoutput der eingesetzten Kommandos. Beispiel:

```
shadowfileexist() {
      if [ -f /etc/shadow ]
then
return 1 #shadow file exist
      fi
return 0 #does not exist
}
```

Beispiel:

```
count () {
      ls | wc -l # ls: Liste aller Dateien im Verzeichnis
      # wc: Word-Count, zählt Wörter
}
```

Die Funktion gibt die Anzahl der Dateien im aktuellen Verzeichnis zurück; aufgerufen wird diese Funktion wie ein Befehl, also einfach durch die Eingabe von **count**.

## 5.2 Kommandos, Pipes und Dateien

Unix/Linux besteht aus vielen kleinen Programmen, die meistens genau eine Aufgabe oder Aufgabenklasse lösen können. So sucht beispielsweise **grep** nach Strings in Textdateien und **find** nach Dateien im Dateisystem.

Es gibt aber kein Kommando, was beide Aufgaben gleich in einem vereinigt, da man dieses zwar manchmal aber nicht oft braucht. Die Zusammenarbeit erfolgt stattdessen durch die Verknüpfung der einzelnen Programme, wofür die Shell das perfekte Werkzeug liefert.

### 5.2.1 Ein- und Ausgabe

Linux-Kommandos arbeiten sich meist so, dass sie bestimmte Daten aufnehmen, diese Daten auf irgendeine Weise verwenden, um schließlich wieder Daten auszugeben. Dieses Schema verdeutlicht sich noch bei interaktiven Programmen, die immer wieder Informationen vom Benutzer annehmen und ihm andere Informationen zurückliefern. Eine Shell ist ein typisches interaktives Programm. Zu diesem Zweck muss sie über einen Eingabekanal verfügen, über den sie Information aufnehmen kann. Dieser Eingabekanal existiert tatsächlich und erhält unter Linux die Bezeichnung Standardeingabe.

Womit aber ist die Standardeingabe verbunden? Üblicherweise wird es die Tastatur sein: Die Shell nimmt Zeichen für Zeichen von der Tastatur entgegen und gibt diese Zeichen auch sofort auf dem Bildschirm aus. Die Ausgabeseite auf dem Bildschirm kann von der Standardeingabe erledigt werden. Die Shell verfügt also über einen weiteren Kanal, der folgerichtig als Standardausgabe bezeichnet wird. Die Standardausgabe der Shell ist

üblicherweise mit einer Konsole[9] oder einem grafischen Pseudo-Terminal, wie **xterm** oder **konsole** verbunden, so dass eingetippte Zeichen sichtbar werden.

Darüberhinaus existiert ein dritter Kanal, der eine besondere Aufgabe zu erfüllen hat, der sogenannte Standardfehlerkanal. Wie der Name schon sagt, dient der Kanal zur Ausgabe von Fehlermeldungen, wenn der Programmlauf aus irgendeinem Grund nicht ordnungsgemäss fortgesetzt werden konnte. Üblicherweise ist Standardfehler ebenfalls mit dem Bildschirm verbunden und schreibt daher seine Meldungen zwischen die gewöhnliche Ausgabe. Es macht jedoch Sinn, Stardardausgabe und Standardfehler voneinander zu trennen, um die Möglichkeit zu haben, gewöhnliche Ausgaben und Fehlerausgaben getrennt zu verarbeiten. Beispielsweise könnte man die Fehlerausgabe in eine Datei umlenken, um sie später zu analysieren, während die gewöhnliche Ausgabe weiterhin über den Bildschirm läuft.

An dieser Stelle gilt es, etwas Wichtiges zu verstehen: Standardeingabe, Standardausgabe und Standardfehler sind lediglich Kanäle, die mit irgendeiner Quelle und irgendeinem Ziel verbunden sein können. Standardeingabe ist nicht gleich Tastatur. Und Standardausgabe ist nicht gleich Monitor. Es gibt viele andere Quellen und Ziele, mit denen diese Kanäle verbunden werden können, wie beispielsweise Dateien oder andere Programme. Bei einer Shell macht es jedoch Sinn, Tastatur und Monitor als Eingabe und Ausgabe zu verwenden, daher ist dies die Voreinstellung.

Die drei Standardkanäle werden von Linux wie Dateien behandelt. Für geöffnete Dateien verwaltet das System eine Liste von Dateideskriptoren, die mit fortlaufenden ganzen Zahlen bezeichnet werden. Die Zahlen von 0 bis 2 sind für die drei Standardkanäle vorbelegt:

| **Kanal** | **Bezeichnung** | **Nummer** |
|---|---|---|
| Standardeingabe | stdin | 0 |
| Standardausgabe | stdout | 1 |
| Standardfehlerkanal | stderr | 2 |

Tabelle 5.4: Standardkanäle der Shell

Bei der Umlenkung dieser Kanäle werden diese Bezeichnungen benötigt! Bereits sehr früh wurde das Prinzip der “Pipe” von den Mainframe-Betriebssystemen abgeschaut. Röhre ist hierfür keine schlechte Übersetzung. Man kann in diese etwas “hineinleiten” und am anderen Ende kommt etwas wieder heraus. Wie bereits beschrieben, gibt es sehr viele kleine, spezialisierte Programme unter Unix/Linux, die mit speziellen Parametern aufgerufen werden können. Daher entstand die Idee eine geeignete Schnittstelle zwischen diesen zu schaffen, um eine geeignete Hintereinanderschaltung dieser Tools zu erreichen.

### 5.2.2 Röhren zwischen Kommandos

Der zentrale Mechanismus beruht auf der Verknüpfung der Ein- und Ausgabekanäle von Programmen. Es erfolgt die Umlenkung der Standardausgabe eines Kommandos in die Standardeingabe eines anderen Kommandos. Dieses erreicht die Verwendung des "Pipe"-Symbols |. Auf diese Weise lassen sich beliebig viele Kommandos zusammenfügen, so dass sich häufig auch sehr komplizierte Aufgabenstellungen durch eine einzige Kommandozeile bewältigen lassen.

Sinnvoll ist daher eine Schnittstelle zwischen diesen Programmen, welche Daten austauscht oder die Ergebnisse eines Programmlaufs in einem weiteren Programm weiterverarbeitet. Diese Schnittstelle ist in Form von Pipes, der eben beschriebenen "Röhren" realisiert.

[9] Monitorausgabe im Textmodus

Bekannt ist sicherlich das System-Kommando **ls**, welches dazu dient sich die Dateien in einem Verzeichnis auflisten zu lassen. Wenn man sich aber in einem Verzeichnis mit einer sehr hohen Anzahl von Dateien befindet und die Liste durchblättern möchte, kann man die Ausgabe, z.B. an das Kommando **more** weiterleiten (z.B. durch `ls -la | more`).

Weiterhin möchte man evtl. die Zahl der Dateien ermitteln, welches durch einfaches Zählen leicht etwas umständlich wird. Um die Zahl von Zeichen, Wörtern oder Zeilen in einer Datei zu ermitteln, steht das Kommando **wc** (Word Count) zur Verfügung. Ein Weg besteht also darin, die Ausgabe von `ls -l` in eine Datei zu schreiben und mittels `wc -l` die Zahl der Zeilen ermitteln zu lassen. Der Umweg über eine Datei lässt sich jedoch mittels einer Pipe umgehen: `ls -l | wc -l` . Man verknüpft nun einfach die beiden Kommandos mittels des Pipe-Zeichens zu einer Zeile und erhält nun die Anzahl der Dateien im aktuellen Verzeichnis. Nur ein kleiner Haken an dieser Stelle: **ls** gibt als erste Zeile keinen Dateinamen aus, sondern eine Zeile, in der Informationen zum entsprechenden Verzeichnis angezeigt werden, man muss hier also vom Ergebnis eine Zeile subtrahieren, um auf das exakte Ergebnis zu kommen.

### 5.2.3 Aus Dateien lesen und in andere schreiben

Statt der Tastatur als Standardeingabe kann man alternativ aus einer beliebigen Datei lesen, die mit absolutem oder relativem Pfad angegeben wird: `sort </etc/passwd`. Das Zeichen "<" erzeugt die gewünschte Funktionalität. Dies mag an dieser Stelle vielleicht etwas sinnlos erscheinen, da der **sort**-Befehl einen Dateinamen als Argument übernehmen kann. Es gibt jedoch Situationen, in denen die Quellenauswahl nur auf diese Weise realisiert werden kann.

Nicht alle Ergebnisse möchte man sich direkt am Bildschirm ansehen, sondern sie für spätere Bearbeitung oder aufgrund sehr grosser Datenmengen in eine Datei schreiben. Dieses geschieht durch den Einsatz von ">". Sollen die Daten an eine bereits existierende Datei angehängt werden, ohne den bestehenden Inhalt zu überschreiben setzt man das ">" doppelt hintereinander: ">>". Gerade für Sachen, die einen nicht interessieren, bietet sich die Umleitung in die Spezialdatei */dev/null* an. Möchte man z.B. testen, ob ein bestimmtes Verzeichnis existiert, genügt der Return-Code, die Ausgabe des Kommandos selbst, kann hingegen entsorgt werden: `ls -al /usr/bin > /dev/null`. Da der Ausgabe- und der Fehlerkanal verschiedene Deskriptoren haben, lassen sich mit `ls -al /usr/bin 2> /dev/null` auch nur die Fehlermeldungen wegdrücken. Das davor gegebene Beispiel müsste formal eigentlich `ls -al /usr/bin 1> /dev/null` lauten, wenn aber keine Kanalnummer angegeben ist, wird die Standardausgabe als Defaultwert angenommen. Sollen einfach beide Ausgaben gemeinsam in eine Datei geschrieben werden, können beide Kanäle mit `ls -al /usr/bin &> /dev/null` (im Fall der **bash**) oder mit `ls -al /usr/bin 2 >&1> /dev/null` gemeinsam umgelenkt werden.

## 5.3 System-Tastenkombinationen

Eher für Not- und Administrationsfälle gedacht, kann der Kernel bestimmte Tastenkombinationen abfangen und direkt auswerten. Diese sind in der Gruppe der "Magic-Sysrequests" zusammengefasst. Der Name rührt von der Tastenbezeichnung im Englischen, wo die [Druck/S-Abf]-Taste als [SysRQ]-Taste beschriftet ist. Diese Funktion kann ein Administrator über das */proc*-Interface des Kernels einschalten: `echo ''1'' > /proc/sys/kernel/sysrq` aktiviert die Benutzung der nachstehend genannten Kombinationen. Alle Kombinationen beginnen mit [Alt]-[SysRQ]

[**r**] - Schaltet den RAW-Tastaturmode aus und setzt es auf XLATE.

[**k**] - Secure Access Key (SAK) Beendet alle Programme auf der gerade benutzten virtuellen Konsole.

[**b**] - Bootet die Maschine sofort und ohne jede Warnung sofort neu. Es findet kein Sync evtl. zwischengespeicherter Blöcke auf die Festplatte oder ein Unmount der Dateisysteme statt.

[**o**] - Analog zu "b", nur dass die Maschine abgeschaltet wird (Power-Off), soweit vom Rechner unterstützt.

[**s**] - Versucht ein Sync alle gemounteten Dateisysteme, d.h. zwischengespeicherte Blöcke werden auf die Datenträger herausgeschrieben.

[**u**] - Versucht ein Remount aller gemounteten Dateisysteme in den Nur-Lese-Status.

[**p**] - Schreibt ein Abbild der aktuellen Register und Flags auf die Konsole

[**t**] - Gibt eine Liste der aktiven Tasks und ihre Information in die Konsole aus.

[**m**] - Gibt Auskünfte über die Speicherbelegung/-auslastung auf der Konsole aus.

[**0**], ... [**9**] - Sets the console log level, controlling which kernel messages will be printed to your console. ('0', for example would make it so that only emergency messages like PANICs or OOPSes would make it to your console.)

[**e**] - Schicken eines SIGTERM an alle Prozesse, ausgenommen **init**, der erste Prozess auf einem jeden System.

[**i**] - Send a SIGKILL to all processes, except for init.

[**l**] - Send a SIGKILL to all processes, INCLUDING init. (Your system will be non-functional after this.)

[**h**] - Zeigt eine Kurzhilfe an (derzeit passiert das auf jeder nicht belegten Taste, die nicht oben in der Liste auftaucht)

Es gibt sogar eine Möglichkeit diese Eingabe entfernt auf einem Rechner vorzunehmen. Dazu kann ein Admin den jeweiligen Buchstaben nach */proc/sysrq-trigger* schreiben, beispielsweise so die Maschine einfach ausschalten: `echo o >/proc/sysrq-trigger`.

## 5.4 Filter

Filter sind Programme, die von der Standardeingabe lesen und zur Standardausgabe schreiben und dabei mit dem Datenstrom nach bestimmten Kriterien etwas anstellen, insgesamt also filtern. Wichtige Filter sind:

- sed - Stream Editor

- awk - Muster-Such- und Verarbeitungssprache
- grep - Durchsucht Texte nach Mustern

Filter kann man kombinieren, z.B. durchsucht:
`cat *.txt | grep Unix | sed s/alt/neu/g` alle mit *.txt* endenden Dateien nach dem Wort Unix und ersetzt das Wort "alt" durch "neu". Die Ausgabe von **sed** erfolgt auf den Bildschirm.

Die Kommandozeile stellt dem Administrator und Anwender mit den genannten Tools mächtige Helfer zur Seite, die das Leben gerade bei komplexeren Aufgaben und sich wiederholenden Tätigkeiten stark vereinfachen können. Die Hürde für das Verständnis, beispielsweise von **sed** und **awk** ist leider nicht zu vernachlässigen. Da genügt oft nicht der Blick in die sonst sehr hilfreichen Man Pages allein. Wenn man aber mal verstanden hat, was beispielsweise der Stream-Editor zuwege bringt, fragt man sich später schon, warum man früher so viele stumpfe Standardtätigkeiten im normalen Texteditor unter grafischer Oberfläche ausgeführt hat. Gerade wo vieles im Hintergrund eines Linuxsystems durch Textdateien erledigt wird, macht die Beschäftigung mit Regulären Ausdrücken Sinn.

### 5.4.1 Reguläre Ausdrücke

Linux hält mit den oben genannten Programmen **sed**, **awk**, **grep** und weiteren sehr mächtige Tools zum Durchsuchen von Texten nach bestimmten Mustern bereit. Will man diese Programme wirklich sinnvoll nutzen, kommt man irgendwann nicht mehr an den sogenannten regulären Ausdrücken vorbei. Diese beschreiben eine Menge von Zeichenfolgen. Sie können aus Textzeichen (Buchstaben, Ziffern, Sonderzeichen) oder Metazeichen mit erweiterter Bedeutung bestehen. Metazeichen sind Spezialzeichen, die Operatoren darstellen, mit deren Hilfe sich komplexe Textmuster beschreiben lassen. Mustersuchen werden durch das Zeilenende begrenzt. Damit kann man keine regulären Ausdrücke definieren, die über das Zeilenende hinaus arbeiten.

| **Ausdruck** | **Bedeutung** |
|---|---|
| . | Wildcard, bezeichnet jedes einzelne Zeichen außer das Zeilenende |
| $[xyz\$]$ | passt auf alle aufgeführten Zeichen |
| d-g | bezeichnet alle Zeichen im angegebenen Limit |
| $[\hat{}nix]$ | passt auf alle Zeichen außer den angegebenen |
| $\hat{}xyz$ | passt auf das angegebene Muster, wenn es am Zeilenanfang steht |
| $xyz\$$ | passt auf das angegebene Muster, wenn es am Zeilenende steht |
| {n,m} | passt auf ein Muster mindestens n-mal und höchstens m-mal |
| $\backslash < xyz \backslash >$ | passt auf das Muster nur, wenn es ein eigenes Wort ist |
| $\backslash(xyz\backslash)$ | die Klammern fassen Ausdrücke zusammen. Jede Zeile wird nach "xyz" durchsucht und jeder Treffer wird in einem Puffer gespeichert (max. 9 dieser Muster sind pro Befehl erlaubt) |
| \ | Maskierung (Escapen) des folgenden Zeichens |
| * | passt auf den vorherigen Ausdruck kein- oder mehrmals |
| + | passt auf den vorherigen Ausdruckt ein oder mehrmals |
| ? | passt auf den vorherigen Ausdruck 0 oder einmal |
| \| | ist ein Trennzeichen. Passt entweder auf den nachfolgenden oder vorherigen Ausdruck |
| (...) | bildet eine Gruppe von regulären Ausdrücken |

Tabelle 5.5: Zeichenklassen und Wiederholungsoperatoren

Reguläre Ausdrücke werden von links nach rechts aufgelöst. Operatoren werden in der folgenden Reihenfolge interpretiert.

| Ausdruck | Bedeutung |
|---|---|
| [: *alpha* :] | alle Gross- und Kleinbuchstaben, entspricht $[A-Za-z]$ |
| [: *lower* :] | alle Kleinbuchstaben $[a-z]$ |
| [: *upper* :] | alle Großbuchstaben $[A-Z]$ |
| [: *blank* :] | ein oder mehrere Leerzeichen und Tabulator |
| [: *space* :] | ein Leerzeichen oder Tabulator |
| [: *cntrl* :] | Kontrollzeichen wie <newline> |
| [: *digit* :] | alle dezimalen Zahlen, entspricht $[0-9]$ |
| [: *xdigit* :] | alle hexadezimalen Zahlen $[0-9A-Fa-f]$ |
| [: *alnum* :] | alle alphanumerischen Zeichen, übersetzt $[A-Za-z0-9]$ |

Tabelle 5.6: Alternative Darstellung bestimmter Zeichenklassen

Mit komplexeren Regulären Ausdrücken lassen sich solche Dinge wie MAC-Adressen oder ISBNs beschreiben:

```
dirk@linux02:~> echo "3-89842-329-8" | \
        grep -E "(?=.{13}$)\d{1,5}( |\-)\d{1,7}\1\d{1,6}\1(\d|X)" \
        && echo " ... ist eine ISBN"
dirk@linux02:~> /sbin/ip addr show | grep -E \
        "([0-9a-fA-F][0-9a-fA-F]:){5}([0-9a-fA-F][0-9a-fA-F])" && \
        echo "... sind MAC Adressen"
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    link/ether 00:e0:18:bd:cb:d4 brd ff:ff:ff:ff:ff:ff
    link/ether 00:50:56:c0:00:01 brd ff:ff:ff:ff:ff:ff
... sind MAC Adressen
```

### 5.4.2 Der Streameditor sed

Besonders wenn es darum geht, in mehreren Dateien immer die gleichen Vorgänge zu wiederholen, hat man zum interaktiven Editieren schnell keine Lust mehr. An dieser Stelle übernimmt **sed**. Das ist ein spezieller Editor zum zeilenweisen Bearbeiten von Strings, z.B. zum Austauschen des Trennzeichens ":" gegen ein Leerzeichen: `cat /etc/passwd|sed -e ‘‘s,:, ,g’` Hat man beispielsweise gerade eine Maschine geklont und möchte nun die IP-Nummern austauschen, spielt sed seine Stärken aus:

```
for i in ‘grep -rlIs "10.30.14.71" /etc‘ ; do
    echo "changing file $i"
    sed ’s/10.30.14.71/10.30.4.33/’ $i  >sed.tmp
    mv sed.tmp $i
done
```

Vielfach möchte man wie im Beispiel direkt eine Datei ändern ohne erst eine Kopie anzulegen. Das geschieht durch die Option "-i": `sed -e ’s/suchen/ersetzen/’ -i file.txt`.

Der Streameditor erschöpft sich nicht im klassische Suchen und Ersetzen. Mit `sed -n ’’5,$ p’’ datei.txt` wird der Inhalt der Datei ohne die ersten fünf Zeilen ausgegeben. Das $-Zeichen steht ähnlich zu seiner Bedeutung in den Regulären Ausdrücken für das Dateiende. Durch "10,20 p" würde man die Zeilen 10 bis 20 herausschneiden. Ähnlich funktioniert `sed ’’5q’’`, welches die ersten fünf Zeilen aus einer Datei anzeigt. Mit dem Parameter "-n" kann man die Funktion von **grep** nachbilden: `sed -n -e ’’/suchmuster/p’’`

`datei.txt` Auch das Kommando **tr**[10] kann durch geeignete Sed-Aufrufe zu Teilen ersetzt werden. So sorgt `sed -e ''y/'KGWXDM'/'kgwxdm'/''` dafür, dass alle genannten Großbuchstaben in Kleinbuchstaben übersetzt werden. Eine Übersetzung des Alphabets wird entsprechend länger. Hier ist sicherlich **tr** die schnellere Lösung, welches die gleiche Aufgabe mit  erledigt.

Das Kürzel "d" steht für "delete", womit das Konstrukt "4d" die vierte Zeile und "/$/d" alle Leerzeilen aus einer Datei löscht. Es funktioniert auch mit Mustern: "/muster/d" entfernt alle Zeilen aus einer Datei, in denen "muster" vorkommt. Das Konstrukt "/Anfang/,/Ende/d" löscht alle Zeilen, die zwischen den Zeilen, in denen "Anfang" gefolgt mit Zeilen, in denen "Ende" vorkommt. Es kann auch dazu genutzt werden, nur in diesem Bereich Ersetzungen vorzunehmen: "/Anfang/,/Ende/s/eins/zwei/".

Bisher wurden Zeilen immer nur für sich betrachtet. Will man nun eine Zeile über einem bestimmten Suchmuster einfügen, dann erreicht man das durch folgendes Konstrukt:

```
echo "Das ist ein Text\nmit einem SUCHMUSTER" | \
sed -e '/SUCHMUSTER/{;i\' -e 'der in einer Datei stehen koennte' -e ' }'
```

Analog zum "i" für insert funktioniert "a" für das Anfügen nach der Zeile mit dem Suchmuster. Folgender Aufruf `sed ''/`$[a-z]/s/\#/$`'' datei.txt` kommentiert in *datei.txt* alle Zeilen, die mit Kleinbuchstaben anfangen mit einem Doppelkreuz aus.

### 5.4.3 Die Programmiersprache awk

Die Shell kann schon eine ganze Menge, jedoch gibt es Aufgaben der String-Verarbeitung, wo sich spezialisierte Programme besser schlagen. Eines davon ist **awk**. Es ist eine eigene Programmiersprache für eine zeilenorientierte Textbearbeitung, Wenn man beispielsweise die einzelnen Zeilen in der */etc/passwd* auftrennen möchte, kann das wie folgt geschehen:

```
cat /etc/passwd|awk -F : '{print $1 `` ''$2 `` ''$3 `` ''$4}'
netname=`route -n | grep -m 1 eth0 | grep -v "UG" | awk '{ print $1 }'`
```

Das zweite Beispiel wertet die Ausgabe des **route** Kommandos aus und schreibt den Netzwerknamen in die Variable $netname. Im Beispiel kam das Kommando grep vor, welches ein mächtiges Kommando zum Durchsuchen von Zeichenketten nach Mustern auch als Regular Expressions ist.

### 5.4.4 String-Sucher grep

Der Aufruf `grep Suchmuster Datei(en)` durchsucht die angegebenen Dateien (oder die Daten aus der Standardeingabe) nach einem Ausdruck und gibt die entsprechenden Zeilen aus. Der Status von **grep** ist 0, wenn der Ausdruck gefunden wurde und sonst 1. Grep mit "-E" aufgerufen kann Reguläre Ausdrücke verarbeiten, mit "-i" ignoriert es Gross- und Kleinschreibung und "-v" invertiert den Suchausdruck.

## 5.5 Entferntes Arbeiten

```
cat datei1 | ssh rechner "cat > /tmp/datei2"

mkisofs -R -J /ordner | ssh rechner "cdrecord dev=1,0,0 -"
```

[10] Translate ...

| Option [Parameter] | Bedeutung |
|---|---|
| $-A < n >$ | gib $< n >$ Zeilen nach der passenden Zeile zusätzlich aus |
| $-B < n >$ | gib $< n >$ Zeilen vor der passenden Zeile zusätzlich aus |

Tabelle 5.7: Optionen (mit Parameter) von grep

```
dd if=/dev/hda | ssh rechner "cat > /tmp/hda.image"

ssh rechner "tar -cpzf - /etc" > /tmp/etc.rechner.tgz
```

## 5.6 Aufgaben

### 5.6.1 Kommandozeile und Umleitung

1. Wie fasst man die beiden Standardausgabekanäle zusammen? Wie entsorgt man Ausgaben (auf irgendeinem Kanel) am besten rückstandsfrei?

2. Man rufe ein Kommando auf und zeige dessen Rückgabewert anschliessend an, jedoch keine seiner direkten Ausgaben (weder Fehler noch die Standardausgabe)!

3. Wie sorgt man dafür, dass der String (*dieses ist ein test*) als ein einziges Argument interpretiert wird?

4. Nennen Sie fünf wichtige Umgebungsvariablen!

5. Welche verschiedenen Möglichkeiten bestehen, um an bereits eingegebene Kommandos wieder heranzukommen?

6. Was passiert, wenn ich das Kommando `export PATH=/tmp` eingebe?

### 5.6.2 Tastatur-Kürzel

1. Was erreicht die Tastenkombination [Strg]-[Alt]-[Backspace] wenn gerade die grafische Oberfläche angezeigt wird?

2. Mit welcher Magic-Sysrequest-Taste kann ich den Abschuss (kill) aller Prozesse bis auf den Init-Prozess erreichen?? Wo wird es gesteuert, ob es funktioniert oder nicht? Wo findet man die Informationen hierzu?

3. Wie kann ich einen PC mit ATX-Netzteil per Magic-Sysrequest ausschalten?

# Kapitel 6

# Filesysteme

## 6.1 Aufbau

Die Linux-Verzeichnisstruktur verteilt sich über eine einzige hierarchische Baumstruktur. Die Nahtstellen in dieser Hierarchie (Festplattenpartionen, Netz-Filesysteme, Übergang auf CD-Rom ...) lassen sich nur mit speziellen Tools (z.B. **mount**) sichtbar machen. Den "Baum" muss man sich umgedreht vorstellen: Die Wurzel (root, dargestellt durch /) befindet sich in der obersten Hierarchie, danach erfolgt die Verzweigung in weitere Ebenen. In diesem Baum kann man mit dem Kommando **cd** navigieren. Es ist immer möglich, zur Navigation und zum Aufruf von Programmen, den Pfad zu einem Verzeichnis oder für eine Datei absolut oder relativ anzugeben. Absolut bedeutet von root ( / ) aus gesehen und relativ ist immer in Bezug auf die aktuelle Position.

In dieser Baumstruktur kann man sich die Blätter als die Dateien und die Äste und Zweige als Verzeichnisse vorstellen. Es gibt keine Laufwerksbuchstaben, wie sie evtl. von anderen OS bekannt sind. So "wissen" Kernel (das eigentliche Betriebssystem) und die Programme immer, wo bestimmte Verzeichnisse (und damit die benötigten Dateien, wie Konfigurationsdateien, Bibliotheken, Programmodule ...) und damit bestimmte Resourcen zu finden sind.

Der Kernel und die Anwendungsprogramme sind auf diese Struktur angewiesen. Deshalb lassen sich Teilbereiche der Verzeichnisstruktur nicht beliebig ein- oder aushängen. Dies geschieht durch den Vorgang des Mountens bzw. Unmountens. Durch diesen Anmelde- und Abmeldevorgang kann das Kernel-Caching für Dateien und Dateiteile recht effektiv wirken.

## 6.2 Einhängen und Aushängen

Das Kommando zum Aufbau der Verzeichnisstruktur bzw. dem Einhängen von Filesystemen heisst **mount**; das Aushängen geschieht mit **umount**. Wegen der oben angesprochenen Problematik darf normalerweise nur der Superuser (User "root", UID 0, GID 0) dieses Kommando ausführen. Zusätzliche Filesysteme von weiteren Festplatten, CD-Rom, DVD, Floppy, ZIP ... werden an Verzeichnisse des Dateibaumes angehangen. Standen bereits Dateien in diesen Verzeichnissen, so werden diese für den Zeitraum des Mountens "verdeckt". Hier ist mit Vorsicht vorzugehen, damit keine systemwichtigen Daten (Devices, Programme, Konfigurationsdateien ...) auf diese Weise unsichtbar werden.

Beim Aushängen von Filesystemen ist zu beachten, dass diese nicht mehr in Benutzung sein dürfen. Schon wenn man sich in einem Verzeichnis des gemounteten Gerätes oder Netzwerkfilesystems befindet, ist dieses in Benutzung. CD-Laufwerke oder ZIP-Drives lassen sich im gemounteten Falle nicht aus dem Laufwerk entfernen, um einen undefinierten

Zustand der Daten/Dateien zu vermeiden. Aus diesem Grund sollte man auch Disketten, Firewire- oder USB-Datenträger (USB-Stick, Firewire-Festplatte, Flash-Karten etc.) ausmounten oder auswerfen (mit **eject**), bevor man sie aus dem Laufwerk entfernt.

Beispiel zum Ein- und Ausmounten des CD-Laufwerkes (hier als Secondary Master am ATAPI-Bus): `mount -t iso9660 -o ro /dev/hdc /mnt/cdrom` Die Optionen -t und -o können gegebenenfalls weggelassen werden. Wenn ein entsprechender Eintrag in der Datei */etc/fstab* vorhanden ist, reicht ein einfaches `mount /mnt/cdrom` Zum Ausmounten: `umount /dev/hdc` oder `umount /mnt/cdrom`. Man kann das Ausmounten und Auswerfen eines Datenträgers durch das Kommando **eject** kombinieren: `eject dvd` wirft den Datenträger mit dem String "dvd" im Device-Namen soweit vorhanden aus. Eine Voraussetzung besteht auch hier - Das Gerät darf sich nicht mehr in irgendeiner Form in Benutzung befinden.

Linux ist für die Ablage von Dateien nicht auf ein bestimmtes Filesystem festgelegt, dieses muss aber bestimmte Eigenschaften mitbringen, welche sich z.B. aus dem Zugriffsrechtesystem von Linux/Unix ergeben.

Seit den Kernelversionen 2.4.X besteht die Möglichkeit mittels **mount** eine Art Link im Dateisystem zu etablieren: So kann man z.B. die beiden Verzeichnisse */tmp* und */var/tmp* auf eines zusammenfassen, ohne symbolische Links zu verwenden. Mit `mount --bind /tmp /var/tmp` macht man */tmp* zusätzlich auf */var/tmp* verfügbar. Dieses Verfahren arbeitet lokal; bei einem Transport über NFS[1] funktioniert diese Verknüpfung nicht.

## 6.3 Die Datei */etc/fstab*

In der */etc/fstab* sind alle wichtigen benötigten Bereiche des Filesystems aufgeführt, die während des Betriebes zu Verfügung stehen (sollen). Sie beschreibt quasi die Bastelanleitung der Linux-Verzeichnisstruktur.

```
# /etc/fstab
#  Device    Mountpoint   FS-Type   Options                Dump Fsckorder
/dev/hda1    /            ext3      defaults                  1 1
/dev/hda3    /tmp         ext3      defaults                  1 2
/dev/hda2    swap         swap      defaults                  0 0
/dev/fd0     /misc/floppy auto      noauto                    0 0
/dev/cdrom   /misc/cdrom  iso9660   noauto,ro,noexec          0 0
none         /proc        proc      defaults                  0 0
/dev/hda4    /dos/c       vfat      user,noexec,nosuid,nodev 0 2
testlin:/export /mnt      nfs       rw,addr=144.41.13.150     0 0
#
# noauto = Do not try to mount at boot time
#
```

Für jeden Eintrag, der einem Teilbaum der Gesamtstruktur entspricht, muss eine eigene Zeile mit vier bis sechs Einträgen angelegt werden. Die Einträge werden durch "white spaces" (Leerzeichen oder Tabulator) voneinander getrennt. Der erste Eintrag bezeichnet die zu der Partition gehörende Gerätedatei im */dev* Verzeichnis oder den Pfad eines im Netzwerk liegenden Bereiches (z.B. zur Einbindung per NFS). Der Name dieser Datei wird mit absolutem Pfadnamen angegeben. Im Beispiel wird die erste Partition der ersten IDE-Festplatte */dev/hda1* in der ersten nichtkommentierten Zeile angegeben. Für spezielle Dateisysteme, wie das Prozeß- oder USB-Dateisystem steht anstelle einer Gerätedatei oder eines Netzwerkpfades das Schlüsselwort "none".

[1] mit dem Kernel-NFSD, da dieser sonst durch gleiche Inode-Nummern verwirrt werden könnte

Der zweite Eintrag bezeichnet das Verzeichnis, an welches das Teilverzeichnissystem angehängt werden soll. Durch die Reihenfolge der Zeilen in der Datei */etc/fstab* muss sichergestellt sein, dass das Verzeichnis, auf dem ein Teilsystem aufgesetzt wird, auch tatsächlich zu diesem Zeitpunkt bereits existiert.

Die Art des Dateisystems, das zum Einbinden in der *fstab* angegeben werden kann, wird in der dritten Spalte eingetragen. Wenn hier "auto" eingetragen ist, wird versucht das Filesystem vor dem Einhängen zu erkennen. Diese Angabe wird üblicherweise dem Mount-Kommando mit der Option "-t" (Type) mitgegeben.

| Option | Bedeutung |
|---|---|
| async | erzwingt asynchrone IO-Operationen |
| auto | erlaubt das automatische Einbinden eines Eintrages |
| defaults | entspricht normalerweise der Kombination "suid", "rw" |
| gid=Wert | bei DOS/VFAT und HPFS Dateisystemen wird allen Dateien die angegebene Gruppen-ID zugeordnet |
| noauto | verhindert das automatische Einbinden der Partition |
| nodev | die zeichen- und blockorientierten Gerätedateien in dieser Partition werden nicht angesprochen |
| noexec | verbietet die Ausführung jedes (binären) Programms dieses Bereiches |
| norock | nur ISO9660, schaltet die Rock-Ridge Erweiterung ab |
| nosuid | unterdrückt die Wirkung der SUID und SGID Bits bei der Ausführung von Programmen |
| nouser | verbietet ausdrücklich die Benutzung von **mount** durch unprivilegierte Systembenutzer |
| remount | veranlaßt mount ein bereits aufgesetztes Dateisystem ab- und sofort wieder aufzusetzen. Auf diese Weise können die Parameter eines bereits aufgesetzten Dateisystems geänderet werden |
| ro | read only, verbietet das Schreiben auf diese Partition |
| rw | read write, erlaubt das Lesen und das Schreiben (muss vom Filesystem unterstützt werden) |
| swap | kennzeichnet eine Swappartition |
| sync | die Metadaten (Superblock, Inode, Verzeichnisdaten) werden ungepuffert (synchron) auf das Speichermedium geschrieben |
| umask=Wert | läßt die Zugriffsrechte für Dateien und Verzeichnisse im DOS/VFAT oder HPFS Dateisystem durch die Maske Wert erscheinen. Der Wert wird als Oktalzahl eingegeben und interpretiert wie beim Shellkommando **umask** beschrieben |
| uid=Wert | mappen der UserID aller Dateien auf die angegebene UserID |
| user | kann das Einbinden von Dateisystemen durch normale Systembenutzer erlauben. Die Benutzer können dann die Kommandos **mount** und **umount** benutzen, denen sie entweder die Gerätedatei oder das Verzeichnis zum Aufsetzen als Parameter übergeben können |

Tabelle 6.1: Auswahl einiger Optionen in der vierten *fstab*-Spalte

Das Kommando **mount** macht exzessiven Gebrauch von dieser Datei. Wenn es z.B. am Anfang des Bootvorganges einer Maschine mit der Option "-a" aufgerufen wird, werden alle Einträge der *fstab* automatisch in das Dateisystem eingebunden, die mit der Option

"auto" (vierte Spalte) gekennzeichnet sind. Hinter der Option "defaults" verbergen sich eine ganze Reihe von Standardeinstellungen, wozu z.B. "auto" und "nouser" zählen. Wenn ein Dateisystem von jedem Systembenutzer eingebunden werden kann, durch die Option "user", entspricht "default" für nicht privilegierte Benutzer der Kombination "nosuid", "noexec", "nodev" und "rw". Es dürfen mehrere Optionen in der vierten Spalte in einer durch Kommata getrennten Liste angegeben werden. Die Optionen, die mit einem "no" beginnen, können auch ohne die Vorsilbe eingesetzt werden, womit sich ihre Bedeutung erwartungsgemäß umkehrt.

Die Tabelle 6.3 auf Seite 61 bietet eine Übersicht über die Optionen der vierten Spalte der fstab.

## 6.4 Filesysteme

### 6.4.1 Überblick

Die Linux-Verzeichnisstruktur kann sich durchaus auf sehr verschiedene Filesysteme verteilen, wobei zumindest für einen Teil der Verzeichnisse diese bestimmte Eigenschaften aufweisen müssen. Als Rootfilesystem (notwendige Basishierarchie) kommen z.B. Ext3, zur Zeit eines der Standard-FS, ReiserFS oder Reiser4, ein weiteres modernes Journaling-FS, XFS, UMSDOS (Filesystem auf Basis einer DOS bzw. Windows95/98/ME-Installation), MINIX (Mini-FS), NFS (Networkfilesystem, für Diskless Clients) in Frage. Für kleine Embedded Devices gibt es weitere Filesysteme wie jffs2 oder squashfs.

Linux versteht (teilweise im Nur-Lese-Modus) weitere Filesysteme: DOS, VFAT, NTFS, HFS, ISO9660, Rom-FS, UFS, ...

Alle Dateisysteme werden auf ein übergeordnetes virtuelles Filesystem abgebildet, welches eine Reihe von Standarddateisystemfunktionen verfügt. Dieses übersetzt die Anforderungen seitens der Programme über den Kernel in die entpsrechenden Funktionen des gerade zur Verfügung stehenden Filesystems. Dabei kann es durchaus vorkommen, dass bestimmte Funktionen nicht zur Verfügung stehen und dann in irgendeiner mehr oder weniger passenden Form emuliert werden.

Die Filesysteme, die vom laufenden Linux-Kernel einer Maschine angesprochen werden können (incl. der aktuell geladenen Module für die FS-Unterstützung) kann man durch das Auslesen der Datei */proc/filesystems* ermitteln. Möchte man bei der "auto"-Option des Mountkommandos (z.B. `mount -t auto /dev/fd0 /floppy`, wenn man nicht genau weiss, ob die Diskette DOS-, Minix-, HFS- ... formatiert ist) weitere Dateisysteme unterstützen, deren Module evtl. noch nicht geladen sind, trägt man diese in der */etc/filesystems* ein. Die Module werden dann zum Test auf das entsprechende Format geladen und die Filesysteme in der Reihenfolge in dieser Datei durchprobiert.

### 6.4.2 Ext2 und Ext3-Filesysteme

Das Extended Secondary Filesystem hat jahrelang als Linux-Standard-Filesystem dominiert. Es ist schnell und effizient, hat aber den gravierenden Nachteil, dass wenn es nicht regulär abgeschlossen wird (z.B. durch **umount**), beim nächsten Mounten eine Konsistenzprüfung durchgeführt wird. Diese wird durch das Programm **fsck** angestossen. Es stellt sicher, dass unvollständige Einträge im Dateisystem repariert werden. Hierin liegt jedoch ein gravierender Nachteil dieses Konzepts: Bei den heute üblichen Festplattengrößen, kann dieser Vorgang durchaus Stunden in Anspruch nehmen, was für Produktionssysteme nicht akzeptabel ist. Während des Checks ist ausschliesslich ein lesender Zugriff auf das Dateisystem erlaubt.

Ein weiterer Nachteil war, dass Dateigrößen größer als 2 GByte nicht unterstützt waren. Als diese Nachteile immer stärker zu Tage traten, wurden Anstrengungen von eine Reihe von Seiten unternommen, diese auszugleichen. Fast zeitgleich wurden von verschiedenen Parteien Dateisysteme mit Journalfähigkeiten für Linux programmiert oder portiert. Diese Fähigkeit bedeutet, dass ähnlich zu transaktionsbasierten Datenbanken, vor jeder Änderung ein Protokolleintrag erfolgt und erst anschliessend diese ausgeführt wird. Dadurch wird sichergestellt, dass im Fall eines Crashes noch ausstehende Vorgänge anhand des Protokolls identifiziert und sauber abgeschlossen werden können.

### 6.4.3 Dateisystemüberprüfung

Bei jedem **mount** wird geprüft, ob das zu mountende Verzeichnis in Ordnung ist. Wird ein Verzeichnis nicht ordnungsgemäß (durch **umount**) ausgehängt, so wird beim ersten nächsten mounten automatisch ein Filesystem-Check durchgeführt. Außerdem findet ein solcher umfangreicher Check in gewissen Abständen (zum Beispiel aller 30 mounts) statt. Dieses läßt sich mit dem Kommando **tunefs** konfigurieren. Damit wird sichergestellt, dass die Informationen auf dem Dateisystem immer konsistent sind. Der Systemadministrator kann einen Filesystemcheck mit Hilfe des Programms **fsck** durchführen.

Um Inkonsistenzen im Dateisystem zu vermeiden, ist es erforderlich, den Rechner vor jedem Abschalten herunterzufahren. Dies erledigt der Systemverwalter mit dem Programm **shutdown**.

## 6.5 Journaling FS

Die Dateisysteme XFS, ReiserFS und JFS sind im Zuge der Suche nach einem Ext2-Ersatz für Linux entwickelt oder portiert worden. Das ReiserFS ist eine gesponsorte komplette Neuentwicklung, deren Version 4 gerade freigegeben wurde, XFS und JFS sind Portierungen von anderen Unix-Systemen. Die wohl wichtigste Eigenschaft dieser neueren Dateisysteme ist das Journaling. Ein Journal ermöglicht es, ein Dateisystem nach einem plötzlichen Systemausfall in einem konsistenten Zustand zu erhalten. Damit sind langwierige Dateisystem-Tests nach einem solchen Ausfall nicht mehr notwendig oder können im Hintergrund durchgeführt werden.

### 6.5.1 Inkonsistente Daten

Daten werden nicht in jedem Fall[2] sofort auf den Datenträger geschrieben, sondern aus Performance-Gründen zunächst im Arbeitsspeicher gehalten. Für die Anwendungen gelten die Daten aber schon in diesem Zustand als gespeichert, damit diese zügig weiterarbeiten können. Im Arbeitsspeicher wird zusätzlich die Reihenfolge der Schreibzugriffe so umgestellt, dass möglichst viele Schreibzugriffe auf einmal durchgeführt werden können. Durch die Firmware der Festplatten wird die nochmals optimiert, so dasse Kopfbewegungen der Festplatte erheblich reduziert werden.

Die Daten werden anschließend, mit einer gewissen zeitlichen Verzögerung, in einem Rutsch auf die Festplatte geschrieben. Dieses als Caching bezeichnete Verfahren ermöglicht ein wesentlich schnelleres Arbeiten.

Fällt nun aber plötzlich der Strom aus, ist nicht klar, in welchem Zustand die Daten gerade waren. Sind sie auf die Platte geschrieben oder waren sie noch im Arbeitsspeicher?

[2]Das Verhalten lässt sich über Mount-Optionen steuern

Deshalb ist in solch einem Fall ohne Journaling eine Prüfung aller Dateien notwendig, was bei größeren Festplatten sehr lange, bis zu mehrere Stunden, dauern kann. Dies ist für Produktiv-Systeme in der Regel nicht akzeptabel.

Darüber hinaus kann bei Inkonsistenzen ein manueller Eingriff notwendig werden, schlimmstenfalls lässt sich das Dateisystem nicht mehr reparieren, was allerdings sehr selten vorkommt.

Das Journaling-Dateisystem vermeidet derartig lange Dateisystem-Prüfungen. Darüber hinaus werden die genannten Inkonsistenzen, die in seltenen Fällen das Dateisystem zerstören können, meist vermieden.

### 6.5.2 Aufbau von Journalaling FS

**Metadaten** Ein Dateisystem benötigt interne Verwaltungs-Strukturen, welche die eigentlichen Daten der Festplatte organisieren und griffbereit halten. Solche internen Strukturen werden Metadaten genannt und sind sozusagen die Daten über die Daten. Die Metadaten definieren beispielsweise, wo die Datenblöcke einer Datei zu finden sind, wer Besitzer ist, die Rechte, die letzten Zugriffszeitpunkte und anderes mehr.

Alle Verwaltungsdaten müssen unbedingt konsistent gehalten werden. So kann auf eine Datei nicht zugegriffen werden, wenn die Datenblöcke nicht dort liegen, wo sie laut Metadaten erwartet werden. Oder es könnte passieren, dass bestimmte Datenblöcke als nicht belegt definiert sind, obwohl dort Daten abgelegt sind, die somit überschrieben werden könnten.

Wird eine Datei neu angelegt, so werden in mindestens fünf verschiedenen Strukturen der Metadaten Änderungen vorgenommen. Gibt es während dieser Änderungen einen Systemausfall, ist das Dateisystem inkonsistent - es sei denn, es gibt ein Journal.

**Journal** Bevor eine Änderung an den Metadaten vorgenommen wird, wie durch das Anlegen einer neuen Datei, werden die dafür nötigen Metadaten-Änderungen zunächst ausschließlich in das Journal geschrieben, welches eine Art Logfile darstellt. Diese Einträge im Journal gelten solange nicht für das Dateisystem, bis die Journal-Einträge mit einem commit abgeschlossen werden. Erst dann werden die neuen Metadaten auf die Festplatte geschrieben.

Wie soll dies nun vor Inkonsistenzen nach einem Systemabsturz schützen? Nach einem Neustart zieht das Dateisystem als erstes das Journal zu Rate. Sind die Einträge im Journal schon mit einem commit abgeschlossen, sind die Metadaten gültig und die Einträge werden auf die Festplatte übertragen. Fehlt das commit als abschließender Eintrag, werden die Metadaten nicht von dem Journal auf die Festplatte geschrieben, sondern verworfen.

Bei Dateisystemen ohne Journal, wie Ext2, müssen dagegen alle Metadaten überprüft werden, ob sie konsistent sind, was die erwähnten langen Wartezeiten bewirkt.

Was ist nun mit den eigentlichen Daten, wann werden diese auf die Festplatte gespeichert? Das ist bei den verschiedenen Dateisystemen verschieden implementiert. Bei Ext3 werden zunächst die eigentlichen Daten auf die Festplatte geschrieben, erst anschließend wird das abschließende commit im Journal gesetzt. Bei den anderen Journaling Dateisystemen können dagegen die Metadaten schon auf die Festplatte geschrieben werden, bevor die Daten komplett auf der Festplatte sind, was zu Problemen führen kann, aber schneller ist. Hier hat Ext3 in Sachen Sicherheit die Nase vorn.

## 6.6 Schicht- oder Overlay-Dateisysteme

Im folgenden geht es um einen Überblick zu einer neuen Klasse von Dateisystemen, mit denen sich viele Fragestellungen des Linuxbetriebes deutlich vereinfachen können. So verwendet beispielsweise das Live-CD-Linux Knoppix seit der Version 3.8 das stapelbare Dateisystem UnionFS, welches dem Anwender nun erlaubt Dateien virtuell auf der CD zu verändern.

In dieser Kategorie ist UnionFS nicht das einzige Dateisystem. Parallel dazu wurden eine ganze Reihe verschiedener Ansätze entwickelt, die aber hier nicht näher beleuchtet werden sollen. Mit dem Einsatz unter Knoppix sind die Einsatzgebiete eines stapelbaren Dateisystems noch nicht erschöpft: Immer wo Dateisysteme mit verschiedenen Eigenschaften in einem einzigen Verzeichnis vereinigt werden sollen, spielt UnionFS seine Stärken aus.

Dateisysteme für Linux gibt es auch ohne UnionFS schon eine ganze Menge. Fast für jeden Zweck stehen oft ähnliche Implementierungen für die gleichen Aufgaben zur Verfügung. Neben den meist wohl bekannten Implementierungen für den Einsatz auf Festplatten, optischen Datenträgern oder auch den netzwerkbasierten Zugriff gibt es einige exotische Dateisysteme, die in der Regel für spezielle Zwecke eingesetzt werden.

Zu letzterem gehört dabei die besondere Art der stapelbaren Dateisysteme, die sich oft hinter englischen Bezeichnungen wie "union", "overlay", "copy-on-write", "translucent" oder "multi-layer filesystems" verstecken. Diese füllen eine Lücke, die bei anderen Unixen schon seit einiger Zeit geschlossen wurde: Man kann mindestens zwei Dateisysteme übereinanderlegen und Dateien durch darüber liegende Dateisysteme hindurchscheinen lassen.

Ein besondere Eigenschaft ist das Schreibverhalten auf den einzelnen Schichten, das bei UnionFS beliebig auf nur lesbaren oder auch schreibbaren Zugriff festgelegt werden kann. Bei einer Live-CD kann das nur lesbare Dateisystem einer CD im ISO-Format mit dem schreibbaren "tmpfs" einer Ramdisk erweitert werden. Zu verändernde Dateien auf dem nur lesbaren Medium werden in die Ramdisk kopiert und können nun problemlos modifiziert werden. Diese copy-on-write-Technik findet man übrigens auch bei der virtuellen Speicherverwaltung im Linux Kernel.

### 6.6.1 UnionFS im Einsatz

UnionFS wurde an der Stony Brook University in New York entwickelt und basiert auf FiST[3] – welches Schnittstellen zur vereinfachten Entwicklung stapelbarer Dateisysteme bereitstellt. UnionFS gehört bisher nicht zum Standardumfang des Kernels, steht aber bei einigen Distributionen schon als Source-Paket zur Verfügung.

Hat das Kompilieren des Moduls geklappt, sollte man nun ausprobieren, ob sich das Modul mit `modprobe unionfs` erfolgreich laden läßt. UnionFS lässt sich nun wie jedes andere Dateisystem mit dem Mount-Befehl in die Verzeichnishierarchie einhängen.

Der wohl einfachste Fall ist die Vereinigung zweier Verzeichnisse. Für ein erstes Beispiel legt man jeweils zwei Verzeichnisse mit jeweils einer Datei an. Anschliessend erzeugt man ein drittes Verzeichnis, welches die Inhalte vereinigt:

```
lp-srv01a:~ # mkdir layer1 layer2 merged
lp-srv01a:~ # touch layer1/file01 layer2/file02
```

Zur Sicherheit sollen beim ersten Aufruf von **mount** die Verzeichnisse ausschließlich mit nur lesenden Zugriffrechten (ro=read-only) zusammengefasst werden. Mit der Angabe der

[3] File System Translator

Option "-t" weiss Mount, dass es "unionfs" verwenden soll. Mit der Option "-o" weist man das stapelbaren Dateisystem an, wie die Schichtung genau aussehen soll und welche Verzeichnisse an der Aktion teilnehmen. Die Verzeichnisse mit der Art ihres Zugriffs werden durch Doppelpunkte voneinander getrennt. Der Aufruf von mount zum Einrichten eines UnionFS ist aus Sicherheitsgründen dem Administrator vorbehalten. Ein anschliessendes `ls merged` zeigt die vereinigten Inhalte an.

```
lp-srv01a:~ # mount -t unionfs -o dirs=layer1=ro:layer2=ro none merged
lp-srv01a:~ # ls merged
file01 file02
```

Spannender wird dieses Szenario, wenn eines der beiden Verzeichnisse mit schreibbarem Zugriff eingebunden wird. So lassen sich mit `mount -t unionfs -o dirs=layer1=rw: layer2=ro none merged` auch neue Dateien unter *merged* erzeugen, die in Wirklichkeit dann unter *layer1* abgelegt werden.

```
lp-srv01a:~ # touch merged/file03
lp-srv01a:~ # ls merged
file01 file02 file03
lp-srv01a:~ # ls layer1
file01 file03
lp-srv01a:~ # ls layer2
file02
```

Diese Art des Dateisystemstapels hilft zum Beispiel für das schmerzfreie Ausprobieren einer neuen Software, die in das Dateisystem einer Linuxmaschine installiert werden soll. Möchte etwa ein Admin das neue Xorg mit 3D-Unterstützung für einen speziellen Grafikchip testen, ohne die ganze Installation in Gefahr zu bringen, bietet sich die Verwendung von UnionFS an. Xorg installiert sich mit seinen Programmen und Bibliotheken unterhalb von */usr*. Wenn man auch die notwendigen Konfigurationsdateien mitbetrachtet, sollte ebenfalls */etc* als weitere Vereinigung angelegt werden.

```
lp-srv01a:~ # mkdir /tmp/union
lp-srv01a:~ # mount -t unionfs -o dirs=/tmp/union=rw:/usr=ro none /usr
lp-srv01a:~ # mount
/dev/hda2 on / type xfs (rw,noatime)
none on /proc type proc (rw)
none on /sys type sysfs (rw)
none on /dev type ramfs (rw)
none on /dev/pts type devpts (rw)
none on /dev/shm type tmpfs (rw)
none on /proc/bus/usb type usbfs (rw)
none on /usr type unionfs (rw,dirs=/tmp/union=rw:/usr=ro)
lp-srv01a:~ # touch /usr/X11R6/test
lp-srv01a:~ # ls /tmp/union/X11R6/test
/tmp/union/X11R6/test
lp-srv01a:~ # umount /usr
lp-srv01a:~ # ls /usr/X11R6/test
ls: /usr/X11R6/test: No such file or directory
```

Das Beispiel zeigt, dass die beiden verbundenen Verzeichnisse und das Mount-Ziel nicht unbedingt alle voneinander verschieden sein müssen. Das Verzeichnis */usr* wurde nur lesbar eingebunden und mit einem schreibbaren Teil überlagert. Deshalb kann man nicht mehr in das "alte" */usr* hineinschauen, sondern sieht nur die Vereinigung unterhalb von */usr*. Möchten man wissen, was verändert wurde, schaut man unterhalb von */tmp/union* nach. Mit einem einfachen `umount /usr`[4] ist der Spuk vorbei und die zwischenzeitlich angelegte Datei nicht mehr zu sehen.

[4] sofern das Verzeichnis nicht in Benutzung ist

Nun verschwinden die Dateien jedoch nicht automatisch aus */tmp/union*. Wenn ein Admin nun wieder den letzten Stand seiner Experimente wiederherstellen will, wiederholt er einfach das UnionFS-Mount und schon sieht */usr* wieder so aus wie vorher. So lässt sich unproblematisch in mehreren Schritten mit einer neuen Xorg-Version herumprobieren ohne dabei die Maschine bei Fehlversuchen unbrauchbar zu machen. Nebenbei kann man auch sehen, welche Dateien wohin installiert wurden. Ist alles zur vollen Zufriedenheit eingerichtet, hebt man einfach die Union auf und synchronisiert die Inhalte aus dem RW-Bereich */tmp/union* an ihre "richtige" Position. Das stellt keinerlei Problem dar, da alle Pfade zu allen dort angelegten Dateien vollständig eingetragen sind.

### 6.6.2 Variationen des Themas

Nach dem gleichen Schema lassen sich aber nicht nur zwei, sondern gleich mehrere Verzeichnisse vereinen. Dies macht beispielsweise Sinn, wenn man zu einem unveränderlichen Basissystem Rechner spezifische Anpassungen durchführen will und zusätzliche Änderungen in einer RAM-basierten Dateisystem verbleiben sollen: In einem Pool mit Diskless Clients bekommt so jeder Rechner über ein Netzwerkdateisystem wie NFS dasselbe Basissystem, ebenso werden Host spezifische Anpassungen lokal auf Festplatte oder ebenso über das Netz bereitgestellt; Dateien, die während des Betriebs verändert oder erstellt werden, landen in einer Ramdisk und gehen bei einem Neustart verloren[5].

## 6.7 Netzwerkdateisysteme

Fast alle modernen Rechnerarbeitsplätze sind inzwischen miteinander vernetzt. Die Bandbreiten im Local Area Network (LAN) liegen dabei meistens bei 100 Mbits und mehr. Sie erlauben den schnellen Zugang zu zentralen und oft gemeinsam genutzten Ressourcen, wie dem E-Mail-Server, den Home-Verzeichnissen oder dem Redaktionsverzeichnis für den Webserver. Viele Anwendungen nutzen Konzepte der zentralen Datenorganisation und können so durch einfachere Administration Zeit und Kosten sparen. Die klassischen Variante ist der Einsatz eines Netzwerkdateisystems.

Der Linuxkernel wurde bereits in einem sehr frühen Entwicklungsstadium mit Netzwerkfähigkeiten ausgestattet. Der Kernel unterstützt daher inzwischen eine ganze Reihe von Dateisystemen, die nicht klassischerweise auf eine lokalen Datenträger liegen, sondern über das Netzwerk verfügbar sind. Dabei kann die Linuxmaschine sowohl als Server, d.h. Anbieter von solchen Dateisystemen, als auch Client, d.h. Bezieher dieser auftreten.

Als Standard zur Verteilung von Filesystemen über TCP/IP-Netze gilt das zu Beginn der 90er Jahre von Sun Microsystems entwickelte und früh auf Linux portierte Network File System (NFS). Derzeit werden die Versionen 2 und 3 von NFS genutzt. Der neue Linux-Kernel 2.6 enthält bereits die Version 4.

Neben NFS gibt es weitere Dateisysteme, die zum Teil zum Standardlieferumfang einer Linux-Distribution zählen. Das SMB-Protokoll[6] unter Linux auch als "Samba" bezeichnet, bietet neben der Option Teile von Dateisystemen zu im- oder exportieren, weitere Steuerungsaufgaben zu übernehmen. Dieses Protokoll wird in Windowsnetzen auch zur Anmeldung an und Verwaltung von sogannten Domänen eingesetzt.

Das Andrew Filesystem ist ursprünglich von Transarc/IBM für das Betriebsystem AIX entwickelt worden. Inzwischen heisst es Open-AFS und steht auch für die Linux-Plattform

[5] um das System im einem sauberen Zustand zu halten

[6] Server Message Block

zur Verfügung. Es arbeitet anders als das alte NFS mit Verschlüsselung und Kerberos-Authentifizierung.

Samba kann Linux mit der Windowswelt verbinden. Solche Beziehungen gibt es auch zu anderen Betriebssystemen. Eine Linux-Maschine kann ebenfalls als Server in Apple-Netzwerken eingesetzt werden bzw. Novellserver oder -client sein.

Da Netzwerkdateisysteme und Linux-Fileserver inzwischen eine immense Bedeutung erlangt haben, werden sie in eigenen Kapiteln oder eigenen Abschnitten behandelt.

## 6.8 Automatisches Einhängen von Filesystemen

Vielfach wird es von den Benutzern aber auch den Administratoren eines Unix-Systems als störend empfunden die Mount-Vorgänge manuell ausführen zu müssen, besonders wenn ein häufiger Datenträgerwechsel anliegt, so muss zum Beispiel beim Ein- und Ausmounten des CD-Laufwerkes (hier als erstes SCSI-CDROM in der Maschine): `mount -t iso9660 -o ro /dev/sr0 /mnt/cdrom` und anschliessend `umount /mnt/cdrom` oder `eject sr0` in der Kommandozeile eingegeben werden. Zwar bringen die grafischen Benutzeroberflächen KDE und GNOME inzwischen eine starke Erleichterung für diese Vorgänge, können jedoch nichts auf der Kommandozeile oder im SSH-Remote-Betrieb ausrichten.

Hierzu bietet sich das Dienstprogramm "Automounter" an. Dieser Dienst wird über den klassischen Runlevel-Mechanismus gestartet. Die Konfiguration, welche Verzeichnisse vom Automounter beobachtet werden sollen, erfolgt in */etc/auto.master*:

```
# $Id: 0500-automount.tex,v 1.1 2006/04/24 21:08:29 dirk Exp $
# Sample auto.master file
# Format of this file:
# mountpoint map options
# Also see variable AUTOFS_OPTIONS in /etc/sysconfig/autofs
# For details of the format look at autofs(8).
/misc          /etc/auto.misc
/home          /etc/auto.home
```

An das Verzeichnis */misc* hängt sich ein Automounter-Prozess, der den Zugriff auf klassische Removable-Devices, wie Diskettenlaufwerk, CD-Rom oder DVD-Laufwerk realisiert.

```
# $Id: 0500-automount.tex,v 1.1 2006/04/24 21:08:29 dirk Exp $
# This is an automounter map and it has the following format
# key [ -mount-options-separated-by-comma ] location
# details may be found in the autofs(5) manpage
cdrom          -fstype=auto,ro         :/dev/cdrom
floppy         -fstype=auto,sync       :/dev/fd0
```

Wenn die Sicherheit der Benutzerdaten nicht im Vordergrund steht, kann einfach das Network Filesystem (NFS) für die Home-Verzeichnisse eingesetzt werden. Die Freigabe erfolgt in der */etc/exports* des entsprechenden Servers. Der Automounter bindet die Homeverzeichnisse nur bei Bedarf ein und ist in der Lage das nur jeweils gebrauchte Verzeichnis zu mounten.

```
# /etc/auto.home
*       -rsize=8192,wsize=8192,rw    192.168.1.1:/home/&
```

Das Beispiel für die Home-Verzeichnisse illustriert den Einsatz des Automounters über NFS und zeigt die Möglichkeiten der Ersetzung: "*" steht für das beliebig angeforderte Verzeichnis, welches am Ende des Pfades anstelle des "&" wieder ersetzt wird. Ein Export für ein statisches NFS-Verzeichnis sieht entsprechend einfacher aus: `/mnt/nfs -rsize=8192,ro 192.168.1.2:/some/path`. Nun ist NFS aber nicht die beste Wahl für den Export eventuell sensibler Daten, wie Benutzerverzeichnisse über ein Netzwerk, da Unberechtigte im Besitz von Root-Rechten auf Maschinen mit Mount-Erlaubnis Zugriff auf fremde Benutzerdaten erhalten könnten.

Der Automounter ist nicht darüberhinaus auch in der Lage Skripten statt nur einfacher Kommandos[7] auszuführen.

## 6.9 Dateiarten

Linux kennt[8] sechs Arten von Dateien auf Filesystem-Ebene. s. Tabelle 6.2, S. 69

| **Art** | **Beschreibung** | **Kennung** |
|---|---|---|
| "normale" Datei | Klassische Dateien, wie ausführbare Programme, Porgrammbibliotheken, Konfigurations-, Druck-, Textdateien ... | - |
| Verzeichnisse | "Containerdatei" in der wieder alle sechs Dateitypen vorkommen können. | d |
| Links | Soft- oder Hardlinks. Softlinks sind Spezialdateien in denen ein Zeiger auf eine andere Datei steht. Diese können im Gegensatz zu Hardlinks auch über physikalische Grenzen von Datenträgern hinweg existieren. Hardlinks sind weitere Einträge in die Inode. | l |
| Sockets | Erlauben die Kommunikation zwischen Prozessen über das Filesystem analog zu TCP/IP. | s |
| FIFOs (named pipes) | Einfachere Version (als Sockets) zur Kommunikation zwischen Prozessen über das Filesystem. | p |
| Gerätedateien | Schnittstellen zum Kernel. Blockorientiert oder Zeichenorientiert. | b (block) od. c (character) |

Tabelle 6.2: Dateiarten

### 6.9.1 Typ einer Datei ermitteln

Das Konzept von Dateiendungen ist eine Sache für sich: Viele Windowsbenutzer haben sich schon gefragt, weshalb plötzlich aus einem Textdokument eine Excel-Tabelle werden soll, einfach nur durch die Änderung der Endung. Deshalb ist den meisten Linux-Programmen, die Endung auch herzlich egal. Programme[9] selbst haben oft überhaupt keine Endung

[7] Der Automounter greift direkt auf das klassische Mount-Kommando zurück.
[8] verschiedene Filesysteme mögen einige Typen davon nicht unterstützen
[9] sog. Executables oder ausführbare Dateien

- vielleicht abgesehen von *.sh* für Shell- oder *.pl* für Perl-Skripten. Programme sind am sogenannten Execute-Bit bei den Zugriffsrechten zu erkennen:

```
-rwxr-xr-x  1 root root 78136 2005-03-19 21:28 /bin/ls
-rwxr-xr-x  1 root root 44616 2005-03-19 21:28 /usr/bin/du
```

Das "x" ist eine Voraussetzung für die Ausführbarkeit - bei anderen Dateiarten macht das X-Bit jedoch keinen Sinn. GNU/Linux stellt mit dem Befehl **file** ein sehr mächtiges Tool zum Feststellen des Types einer Datei zur Verfügung.

```
linux02:/tmp/file # file *
dump:       tcpdump capture file (little-endian) - version 2.4 (Ethernet, \
  capture length 96)
files.sxw:  setgid sticky empty
ls:         ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), for \
  GNU/Linux 2.2.5, dynamically linked (uses shared libs), stripped
man-df.png: PNG image data, 667 x 463, 8-bit/color RGB, non-interlaced
nnn.dvi:    setgid sticky TeX DVI file (TeX output 2005.05.04:1720\213)
nnn.tex:    setgid sticky ISO-8859 text
shell.sxw:  setgid sticky Zip archive data, at least v2.0 to extract
```

Der Befehl **file** kennt den "Fingerabdruck" von sehr vielen verschiedenen Programmen und Dateitypen und kann deshalb auch ohne, dass das passende Programm installiert ist, feststellen, worum es sich jeweils handelt. Wer beispielsweise Filme auf seinem System vor allzu Neugierigen verstecken will, sollte mehr als nur reine Umbenennung vornehmen, **file** lässt sich nicht so einfach täuschen:

```
dirk@linux02:~/SharedFiles/TV/Filme> file *
Film.avi: setgid sticky RIFF (little-endian) data, AVI, 640 x 272, \
  25.00 fps, video: XviD, audio: MPEG-1 Layer 3 (stereo, 48000 Hz)
Text.doc: setgid sticky RIFF (little-endian) data, AVI, 640 x 272, \
  25.00 fps, video: XviD, audio: MPEG-1 Layer 3 (stereo, 48000 Hz)
```

File bezieht die Daten zur Erkennung verschiedener Dateiarten aus */etc/magic*.

### 6.9.2 Textdateien und Kodierung

Derzeit findet auf Linux-Systemen die Miration von den länderspezifischen alten Kodierungen auf das übergreifende UTF8 statt. Früher waren Zeichen einheitlich 8bit lang und Dateien nach bestimmten Codepages, wie ISO-8859-1 oder ISO-8859-15 für westeuropäische Zeichensätze kodiert, UTF8 kennt variable Zeichengrößen. Kommt ein Kommando mit einer Kodierung nicht klar, dann sieht die Darstellung von bestimmten Zeichen nicht korrekt aus. Für die Umwandlung gibt es **iconv**. Dieser ist ein einfacher Umkodierer. Das Umwandeln einer UTF-8-Datei - als Beispiel *utf8.txt* - in eine Datei *iso-alt.txt* nach dem alten ISO8859-15-Standard geschieht so: `iconv -f UTF-8 -t ISO-8859-15 -o iso-alt.txt utf8.txt`.

Das Ergebnis kann man mit `file -i iso-alt.txt` einfach überprüfen. Ein einfacheres Tool ist **recode**. Mit dem folgenden Beispiel wird die Datei *test.txt* vom alten ISO8859-1 Zeichensatz nach UTF8 umgewandelt: `recode latin1.utf-8 test.txt` Einige Text-Editoren bieten es dem Benutzer an, anzugeben in welchem Zeichensatz eine Datei dargestellt und gespeichert wird.

## 6.10 SquashFS - ein komprimiertes, nur-lesbares Dateisystem

SquashFS ist ein Spezialdateisystem eigentlich entwickelt für Embedded Devices, welches nur lesend genutzt werden kann. Es kann aber für einige Anwendungen auch interessant sein,

es auf einer normalen Workstation eizusetzen. So könnte man beispielsweise eine Standard-Installation ähnlich wie ein CD-Linux so komprimieren, dass es auf einen relativ kleinen USB-Stick passt und bequem an verschiedenen Maschinen beispielsweise für Wartung und Recovery benutzt werden kann. Ebenso könnte man ein Linux-Systems mit Hilfe eines Netzwerkblockgeräts auf einem anderen Rechner verfügbar machen. Das Besondere daran ist, dass die Daten in kompakter Form auf dem Server liegen und somit der Server weniger belastet wird, da auch viel weniger Daten übers Netz übertragen werden müssen.

In jedem Fall muss die Maschine die Daten wieder dekomprimieren, was allerdings angesichts der heutigen Rechenleistungen kein Problem mehr darstellt. In einem weiteren Schritt könnte das nur-lesbare SquashFS dann wieder mit Hilfe von UnionFS[10] schreibbar gemacht werden.

SquashFS ist bisher nicht Bestandteil des Kernels und muss in diesen hinein gepatcht werden, bevor es beispielsweise als Modul gebaut werden kann:

```
linux02:/usr/src/linux # patch -p1 <
~/fs/squashfs3.0/linux-2.6.15/squashfs3.0-patch
patching file fs/Kconfig
Hunk #1 succeeded at 1248 (offset 97 lines).
patching file fs/Makefile
Hunk #1 succeeded at 58 (offset 3 lines).
patching file fs/squashfs/inode.c
patching file fs/squashfs/Makefile
patching file fs/squashfs/squashfs2_0.c
patching file fs/squashfs/squashfs.h
patching file include/linux/squashfs_fs.h
patching file include/linux/squashfs_fs_i.h
patching file include/linux/squashfs_fs_sb.h
patching file init/do_mounts_rd.c
```

Nach dem Patchen findet man im Kernel-Konfigurationsdialog unterhalb des Menüpunktes "Filesystems - Miscellaneous" am Ende der Liste "SquashFS 3.0", welches man am besten als Modul anwählt und anschliessend mit `make prepare modules` baut.

Das Programm **mksquashfs** wird separat kompiliert. Das hat den Vorteil, dass man auf der Servermaschine auch nur das Packprogramm erzeugt, welches einen SquashFS-Container generiert. Wenn man diesen Container hier nicht einmounten will, braucht man auch keine passenden Kernelmodule. Will man nun ein SquashFS generieren[11], sieht das so aus:

```
mksquashfs /Quellverzeichnis01 /Quellverzeichnis02 ... Zieldatei -e \
Datei_Exclude
```

Das komprimierte Ergebnisdateisystem kann eine maximale Grösse von 4 GByte erreichen. Das erscheint ersteinmal nicht viel, jedoch erlaubt die erreichte Kompression schnell 10 GByte und mehr für das Ausgangs-Filesystem. Damit lässt sich schon eine fast komplette Installation einer Distribution abdecken.

Das Erzeugen des komprimierten Dateisystems dauert je nach Ausgangsdatenmenge und Performance des Servers ziemlich lange. So wurden auf einem normal belasteten Opteron mit 2 GHz 8,4 GByte in 32 Minuten auf 2,9 GByte komprimiert. Ein anderes Ausgangs-Filesystem von 5,6 GByte Umfang brauchte auf derselben Maschine im unbelasteten Zustand zur Schrumpfung auf 1,9 GByte immerhin noch 26 Minuten. Der ganze Aufwand ist es jedoch Wert, wenn man sich das Ergebnis ansieht.

[10] spezielles Overlay-Dateisystem, welches an anderer Stelle beschrieben wird

[11] am besten eine Datei, eine Partition oder LVM-Share ginge auch

Das "Einpacken" eines Client-Rootfilesystems findet eher selten statt, im Gegensatz dazu stehen eine Reduktion der Netzwerkbelastung gegenüber ext2 auf demselben Blockdevice oder NFS. Zudem wird das Block-Caching auf dem Client schätzungsweise effizienter, da pro Block nun eine größere Datenmenge untergebracht wird. Angesichts der Leistungen moderner CPUs kann der Dekompressionsaufwand auf Clientseite vernachlässigt werden, besonders wo SquashFS auf Systeme mit um Größenordnungen schwächeren CPUs abzielt.

## 6.11 Aufgaben

### 6.11.1 Filesystem - Aufteilung

1. 1) Wo liegen in der Linux-Verzeichnisstruktur üblicherweise die Bibliotheken und ausführbare Progamme? Nennen Sie wichtige Kommandos und ihre Lage im Dateisystem!

2. Warum verwendet man nicht das Konzept der Laufwerksbuchstaben, wie unter DOS oder den verschiedenen Windowsversionen bekannt?

3. Wo findet man Dateien, wie z.B. **kmail**, **mount** oder **ls**?

4. Wo befinden sich die Konfigurationsdateien zum Einrichten einer Modemverbindung? Wo liegen im Linux-Dateisystem üblicherweise die Konfigurationsdateien? Nennen Sie wichtige Konfigurationsdateien, die im Kurs vorgekommen sind!

5. Weshalb darf der Befehl `mv *.txt *.html` so nicht angewendet werden, was passiert da?

6. Woran erkennt man den Dateityp "Verzeichnis" bzw. "Link"?

7. Wie bestimmt man generell den Belegungsgrad der gemounteten Festplatten?

8. In welcher Datei sind die Benutzer und in welcher ihre Passwörter eingetragen? Wie verhält es sich bei einer zentralen Authentifizierung?

9. Wofür dienen die Kommandos **find** und **locate**? Worin bestehen die Unterschiede zwischen ihnen?

### 6.11.2 (Un-)Mounten

1. Man gebe die Kommandos ein, um festzustellen, welches Device das CD-Rom oder DVD-Laufwerk ist (wichtig beim Mounten), welche Grösse die eingebaute Festplatte hat und wie diese partitioniert ist!

2. Mounte das DVD-Laufwerk und anschliessend das einen USB-Stick (oder Diskettenlaufwerk) nach */mnt*!

3. In welcher Konfigurationsdatei wird festgelegt welche Bereiche und Devices normale User mounten dürfen? Welche Dateisysteme werden beim Booten automatisch gemountet und welche nicht? Wie legt man das fest?

4. Welche Vereinfachungen gibt es zum vollständig ausgeschriebenen Mountkommando?

5. Mit welchem Dateisystem ist die Root Partition formatiert bzw. wo liegt das Root-Filesystem?

6. Welche Partitionen sind auf dem Rechner wohin gemountet? Wieviel Speicherplatz ist auf ihnen noch vorhanden.

7. Man nenne Dateisysteme, welche als Root-Filesystem für eine Linuxmaschine in Frage kommen. Welche weiteren gibt es und warum können diese nicht für den genannten Zweck eingesetzt werden?

8. Welche Besonderheit(en) hat UnionFS gegenüber klassischen Dateisystemen, wie Ext3, ReiserFS oder NFS?

### 6.11.3 Speicherplatz auf der Festplatte

1. Bestimmen Sie die Menge an Speicherplatz die das Verzeichnis */usr/share* auf Ihrem Computer belegt! Wie erhält man eine Ausgabe in Mega- oder Gigabyte-Angabe?

2. Wie lässt sich generell der Belegungsgrad der gemounteten Festplatten bestimmen?

# Kapitel 7

# Zugriffsrechte und Verzeichnisstruktur

Die Nutzung von Informationssystemen ist üblicherweise mit einem Zugangssystem verbunden, welches die Verwendung des Systems auf eine bekannte Benutzergruppe beschränkt, Daten über die registrierten Benutzer speichert und die Verteilung der Ressourcen auf die Benutzer steuert. Häufig ist die Konzeption des Zugangssystems für den einzelnen Benutzer transparent - außer seinem Benutzernamen und einem Passwort benötigt der Benutzer kaum weitere Kenntnisse, um das System in Anspruch zu nehmen. Für die Arbeit mit einem Linux-System sollte man sich deshalb einige elementare Kenntnisse über dessen Benutzer- und Berechtigungskonzept aneignen.

Die Notwendigkeit für diese Konzepte ergibt sich für Linux aus seiner Mehrbenutzerfähigkeit. Ein erster wichtiger Aspekt ist der Schutz des Systems vor den Handlungen seiner Benutzer. Weiterhin müssen auch die einen Benutzer vor den Handlungen der anderen geschützt werden. Und schließlich darf bei allem Schutz des Systems und der Benutzer voreinander das Miteinander-Arbeiten nicht allzusehr erschwert werden. Um all dies zu gewährleisten, bedarf es eines feinkörnigen Systems der Einschränkungen und Erlaubnisse. Dieses System ideal an die jeweiligen Gegebenheiten anzupassen, ist die Aufgabe des Systemverwalters.

## 7.1 Zugriffsrechte

Das Sicherheitskonzept von Unix basiert stark auf den Zugriffsrechten für Dateien. Deshalb hat in der Unix/Linux-Welt jedes Verzeichnis und jede Datei IndexZugriffsrechte für einen Besitzer ("user", Kürzel **u**, z.B. die User ftp, mysql, testuser ...), für die gesamte Gruppe ("group", Kürzel **g**, z.B. bin, named, users) zu der dieser Besitzer gehört und schließlich für alle anderen, den "Rest der Welt" ("other", Kürzel **o**). Diese sind im Ganzen drei Benutzergruppen, die beim Setzen der Rechte auch zusammen angegeben werden können ("all", Kürzel **a**). Es ist z.B. bei der Einrichtung einer Homepage im eigenen Verzeichnis darauf zu achten, dass den HTML-Dateien die Leserechte für "other", bzw. für "all" gesetzt wurden und die darüberliegenden Verzeichnissebenen das Suchrecht für "all" besitzen.

Weiterhin gibt es drei Arten des Dateizugriffs: Schreiben, Lesen, Ausführen (bzw. Suchen in Verzeichnissen). Das Kommando zum Setzen der Zugriffsrechte ist **chmod**. Dabei gibt es zwei Möglichkeiten es aufzurufen:

```
mayer@hermes:~/kursunterlagen >chmod o+r kurs01.txt
```

Setzt für alle anderen Benutzer (other) , die nicht der eigenen Gruppe (z.B. "users") angehören, das Leserecht (read) auf die Datei kurs01.txt. Sollen die Rechte entfernt werden, geschieht dieses durch die Angabe des Minuszeichens anstelle des Plus, welches für das Hinzufügen der Rechte steht.

Eine andere Möglichkeit besteht darin die Zugriffsrechte über die sie repräsentierenden Oktalwerte zu setzen, wie sie in der untenstehenden Tabelle gezeigt werden.

| **Oktalwert** | **Beschreibung** | **Benutzergruppe** |
|---:|---|---|
| 4000 | Set-User-Id-Bit | |
| 2000 | Set-Group-Id-Bit | |
| 1000 | Sticky-Bit | |
| 400 | Lesezugriff | |
| 200 | Schreibzugriff | Dateibesitzer |
| 100 | Ausführungs- / Suchzugriff | |
| 40 | Lesezugriff | |
| 20 | Schreibzugriff | Benutzergruppe |
| 10 | Ausführungs- / Suchzugriff | |
| 4 | Lesezugriff | |
| 2 | Schreibzugriff | Alle anderen |
| 1 | Ausführungs- / Suchzugriff | |

Tabelle 7.1: Die traditionellen Zugriffsrechte unter Linux

Der Aufruf des Kommandos **chmod** sieht dann z.B. so aus:

```
meier@hermes:~/kursunterlagen >chmod 0700 kurs01.txt
```

Darüberhinausgehende Rechtesysteme lassen sich mit den Posix-ACL's (Access Control Lists) implementieren. Dafür muss jedoch eine Unterstützung seitens des Dateisystems (z.B. xfs) vorhanden sein.

## 7.2 Systembefehle zur Arbeit mit Dateien

Folgende Basisbefehle stehen neben einer ganzen Reihe weiteren unter Linux zur Verfügung. Die spitzen Klammern sollen bloß anzeigen, dass dieser Bestandteil des Befehls - von den Leerzeichen abgesehen - variiert, die tippt man in Wirklichkeit nicht mit. Diese Befehle sind immer nach dem Schema: Befehl, Leerzeichen, Name, ev. Leerzeichen, ev. Name aufgebaut. (Nach der Eingabe des Befehls diesen immer mit [Enter] abschicken!)

**mkdir <Verzeichnis>** erstellt ein neues Verzeichnis (im Windows-Jargon: einen neuen Ordner), mit dem von euch gewählten Namen ¡Verzeichni¿. "mkdir" steht für "make directory" Bsp.: `mkdir test`. Das neue Verzeichnis wird immer als Unterverzeichnis des aktuellen Verzeichnisses ( = in dem man sich gerade befindet) eingerichtet.

**cd** führt zum Verzeichniswechsel, dahinter kann man den Namen desjenigen Verzeichnisses angeben, in das man wechseln möchtet (z.B. `cd test`). **cd** steht für "change directory" Wenn man keinen Verzeichnisnamen angibt, bringt einen dieser Befehl immer in das Homeverzeichnis des angemeldeten Users. Denkt daran, dass man sich so immer nur von einer

Ebene in die nächste vorhangeln kann (relative Pfadangabe). Um in das nächsthöhere Verzeichnis zurückzuwechseln, könnt man **cd ..** benutzen (beachtet das Leerzeichen vor den Punkten). Wenn man Verzeichnisebenen überspringen möchtet, muss man den ganzen Pfad eingeben (z.B. */test/texte* - ohne Leerzeichen ). Ein solcher Pfad bezeichnet den genauen Standort im Verzeichnisbaum. Das Home-Verzeichnis kürzt man mit ˜/ ab. Das aktuelle Verzeichnis gibt man mit ./ an.

**pwd** Diese Eingabe liefert den vollständigen Pfad zu dem Verzeichnis, in dem man sich gerade befindet. pwd steht für "print working directory"

**ls** zeigt in Inhalt von Verzeichnissen an. **ls** steht für "list". Man könnte auch alternativ `ls <Verzeichnis>` eingeben, ohne vorher `cd <Verzeichnis>` ausgeführt zu haben, aber immer nur von einer Ebene in die nächste, oder mit dem vollständigen Pfadnamen. Wenn man `ls -l` eingibt, erfahrt man sogar noch mehr über die Dateien als den Namen (z.B. die Größe etc.)

**rmdir <Verzeichnis>** löscht leere Verzeichnisse. **rmdir** steht für "remove directory". Wieder kann man nur Unterverzeichnisse des aktuellen Verzeichnisses oder den ganzen Pfad angeben. Wenn sich in dem zu löschenden Verzeichnis noch etwas befindet, löscht der Computer sie nicht. Dann muss man erst mit

**rm <Dateiname>** die Dateien darin löschen. Mit `rm *` kann man sie alle auf einmal beseitigen. Bestimmte Dateitypen löscht man mit `rm *.<extension>`, also z.B. `rm *.jpg`, um alle Bilddateien des Typs "jpg" zu löschen.

**mv <Datei> <Verzeichnis>** verschiebt eine Datei von einem Verzeichnis in ein anderes - natürlich nur, wenn die Datei in dem aktuellen Verzeichnis steht, oder man den gesamten Pfad angibt. **mv** funktioniert (eingeschränkt) über die Grenzen von Filesystemen hinweg, da nur Änderungen in den Inode-Einträgen erfolgen und die Datei nicht physikalisch "bewegt" wird. **mv** steht für "move". Wenn man den Namen einer Datei ändern möchte, gibt man `mv <alteDatei> <neueDatei>` ein.

**cp <Datei> <Verzeichni>** kopiert tatsächlich eine Datei. Diese Datei existiert dann zweimal (im Gegensatz zu **mv**). Natürlich kann man auch einfach den Befehl `cp <alteDatei> <neueDatei>` ausführen, wenn man eine Datei zweimal in einem Verzeichnis haben möchte.

## 7.3 Dateiablagestandards

Es gibt nicht "das" klassische Konzept für eine Filehierarchie der freien Unixe, früher orientierte sich die Struktur am System V Unix Release 4. Seit einiger Zeit gibt es einen Filesystemhierarchiestandard (FSHS), der Mitte diesen Jahres in einer aktualisierten Version veröffentlicht wurde. Die meisten neueren Linuxdistributionen halten sich an diesen.

Zwei Hauptkriterien der Verteilung der Dateien spielen im FSHS eine Rolle: verteilbare vs. lokale und variable vs. statische Daten. Die Idee dieser Einordnung entstammt dem Wunsch, Teile des Filesystems zentral auf einem Server im Netzwerk zur Verfügung stellen zu können.

| | **Verteilbar** | **Lokal** |
|---|---|---|
| **Statisch** | /usr, /opt | /etc, /boot |
| **Variabel** | /var/mail, /var/spool | /var/log, /var/run |

Tabelle 7.2: Aufteilung des Dateisystems

## 7.4 Dämonen

Im Unterschied zu anderweitigen Prozessen, die stets an ein Dialogstation (tty) und Benutzer gekoppelt sind und je nach Aufruf im Vorder- oder Hintergrund laufen, werden Dämonen meistens automatisch beim Hochfahren des Systems über die Runlevelskripte gestartet. Sie sind nicht an eine Konsole gebunden und schreiben deshalb ihre Meldungen üblicherweise über das Syslog oder in spezielle Logdateien und -verzeichnisse.

## 7.5 Aufbau einiger wichtiger Verzeichnisse

Unter Linux sind Dateien im Gegensatz zu Windows in einem großen zusammenhängenden Verzeichnisbaum gespeichert, der mit dem Wurzelverzeichnis / beginnt und sich schnell weit verzweigt.

```
dirk@linux01:~> ls /
bin   data01  dev  home  lost+found  mnt  proc  sbin  sys  usr
boot  data02  etc  lib   media       opt  root  srv   tmp  var
```

Dateien werden unter Linux geordnet nach ihrer Funktion abgelegt. So ergeben sich viele Vereinfachungen: Konfigurationsdateien liegen in einem gemeinsamen Verzeichnis, Programmdateien verteilen sich je nach Funktion über wenige Verzeichnisse, was den Suchpfad vereinfacht und die Angabe des absoluten Pfades vermeidet. Bibliotheken und Module liegen wieder in speziellen Verzeichnissen ...

In einigen Hierachiestufen wiederholen sich bestimmte Verzeichnisse: “lib”, “bin” findet man z.B. in */*, */usr*, */usr/local*, */usr/X11R6*, */opt/kde3* ...

## 7.6 Konfigurationsdateien

Konfigurationsdateien liegen üblicherweise im Systemkonfigurationsverzeichnis */etc*. Für bestimmte Programme (noKDE, Gnome) findet man diese jedoch in den entsprechenden Unterverzeichnissen. Im folgenden werden jedoch nur einige der Dateien in */etc* näher erläutert.

### 7.6.1 Allgemein

Die Dateien */etc/passwd* und */etc/shadow* verwalten die User des lokalen Systems, speichern deren Passwörter enthalten evtl. zusätzliche Informationen und legen das Home-Verzeichnis sowie die Login-Shell fest. Die Datei hat folgendes Aussehen:

```
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/bin/bash
daemon:x:2:2:Daemon:/sbin:/bin/bash
lp:x:4:7:Printing daemon:/var/spool/lpd:/bin/bash
mail:x:8:12:Mailer daemon:/var/spool/clientmqueue:/bin/false
news:x:9:13:News system:/etc/news:/bin/bash
uucp:x:10:14:Unix-to-Unix CoPy system:/etc/uucp:/bin/bash
```

```
games:x:12:100:Games account:/var/games:/bin/bash
man:x:13:62:Manual pages viewer:/var/cache/man:/bin/bash
at:x:25:25:Batch jobs daemon:/var/spool/atjobs:/bin/bash
wwwrun:x:30:8:WWW daemon apache:/var/lib/wwwrun:/bin/false
ftp:x:40:49:FTP account:/srv/ftp:/bin/bash
named:x:44:44:Nameserver daemon:/var/lib/named:/bin/false
postfix:x:51:51:Postfix Daemon:/var/spool/postfix:/bin/false
sshd:x:71:65:SSH daemon:/var/lib/sshd:/bin/false
ntp:x:74:65534:NTP daemon:/var/lib/ntp:/bin/false
ldap:x:76:70:User for OpenLDAP:/var/lib/ldap:/bin/bash
gdm:x:100:100::/home/gdm:/bin/bash
vdr:x:101:33:Video Disk Recorder:/var/spool/video:/bin/false
quagga:x:102:101:Quagga routing daemon:/var/run/quagga:/bin/false
dhcpd:x:103:65534:DHCP server daemon:/var/lib/dhcp:/bin/false
messagebus:x:104:102:User for D-BUS:/var/run/dbus:/bin/false
haldaemon:x:105:103:User for haldaemon:/var/run/hal:/bin/false
nobody:x:65534:65533:nobody:/var/lib/nobody:/bin/bash
dirk:x:500:100:Dirk von Suchodoletz:/home/dirk:/bin/bash
```

Nacheinander werden folgende Informationen durch Doppelpunkte getrennt aufgelistet: Der Username (als String), der Platzhalter für das Passwort ("x"), die numerische User-ID, die numerische Gruppen-ID, eine Beschreibung des Benutzers (die kommasepariert weitere Informationen enthalten kann), das Arbeitsverezeichnis und die Standard-Shell. Diese Datei ist die zentrale Benutzerdatenbank, es sei denn es werden zentrale Authentifizierungs- und Konfigurationsdienste (wie z.B. LDAP) verwendet. Über die Reihenfolge der Einbindung zur Authentifizierung entscheidet die */etc/nsswitch.conf*.

Die */etc/shadow* ist die zur */etc/passwd* korrespondierende Datei. Sie verfügt über stark beschnittene Zugriffsrechte, damit nur der Systemadministrator in diese Datei hineinsehen kann. Zwar sind die Passwörter nicht als Klartextvariablen abgelegt, dennoch besteht die Möglichkeit, das gecryptete Passwort mittels Brute-Force-Attacke auf einem anderen Rechner zu knacken. Die */etc/shadow* sieht folgendermaßen aus:

```
at:!:11962:0:99999:7:::
bin:*:8902:0:10000::::
daemon:*:8902:0:10000::::
dhcpd:!:12625:0:99999:7:::
ftp:*:8902:0:10000::::
games:*:8902:0:10000::::
gdm:!:11988:0:99999:7:::
haldaemon:!:12906:0:99999:7:::
ldap:!:12285:0:99999:7:::
lp:*:8902:0:10000::::
mail:*:8902:0:10000::::
man:*:8902:0:10000::::
messagebus:!:12906:0:99999:7:::
named:*:8902:0:10000::::
news:*:8902:0:10000::::
nobody:4qCm64P9en1is:11984:0:10000::::
ntp:!:11962:0:99999:7:::
postfix:!:11962:0:99999:7:::
quagga:!:11797:0:99999:7:::
root:$2a$10$6zzscSISd8OMUMsyRvSjzenYD.fh.02sb9.i0XmO3usgtdu/M3tFW:\
```

```
  12548:0:10000::::
sshd:!:11962:0:99999:7:::
uucp:*:8902:0:10000::::
vdr:!:12391:0:99999:7:::
wwwrun:*:8902:0:10000::::
dirk:$2a$10$/n1GWnEVKd.140ANahfv3uK87OadkP277/6hvDgp39X9ycinUZmhO:\
  13052:0:99999:7:::
```

*/etc/login.defs* ist eng mit dem **login**-Prozess verknüpft. Dieses liest daraus die Parameter mit denen beispielsweise eingestellt wird, wie oft ein fehlgeschlagenes Login wiederholt werden darf und ob und wo die Meldungen über solche Fehlschläge festgehalten werden. Hier können auch einige Umgebungsvariablen für alle Prozesse festgelegt werden, die in der vom **login** erzeugten Prozessfamilie gestartet werden.

*/etc/issue(.net)* wird beim Konsolen-Login und bei "telnet"-Verbindungen vor dem Login-Prompt angezeigt.

*/etc/rc.config* ist die zentrale Konfigurationsdatei der SuSE-Linuxdistribution.

### 7.6.2 Shell

*/etc/skel* ist ein Skeleton-Verzeichnis, wo die Konfigurationsdateien für Userprogramme liegen, welche beim Anlegen des Accounts in das Verzeichnis des entsprechenden Benutzers eingespielt werden und so das Tool zur Benutzerverwaltung unterstützt wird.

*/etc/securetty* gibt die Ports (Terminals) an, von denen aus sich der Systemadministrator einloggen darf. Diese Datei wird vom **login**-Kommando gelesen und ausgewertet. Die Verwendung dieser Datei wird in der *login.defs* spezifiziert.

*/etc/shells* listet alle verfügbaren (zugelassenen und uneingeschränkten) Loginshells auf. Diese Datei wird vom **chsh**-Kommando ausgewertet. Dem Anwender wird damit die Möglichkeit gegeben, die in dieser Datei zeilenweise aufgelisteten Programme als Loginshell in der Datei **/etc/passwd** einzutragen, wenn es sich um einen lokalen Account handelt.

*/etc/profile* setzt wichtige Umgebungsvariablen für die Shell (Suchpfad, Manpath ...). Diese Datei wird immer beim Login aufgerufen.

*/etc/motd* Die "Message of the day" wird nach dem Login angezeigt. Es gibt einige Escapesequenzen der Shell, welche z.B. den Hostnamen, Uhrzeit, verwendete Konsole etc. anzeigen können.

### 7.6.3 Netzwerk

*/etc/hosts* enthält eine Liste von IP's und dazugehörigem Rechnernamen. Sie wird verwendet, wenn z.B. kein DNS zur Verfügung steht. Dies ist wärend des Hochfahrens der Maschine üblicherweise der Fall.

*/etc/resolv.conf* In dieser Datei erfolgt die Konfiguration des Clients für die Namensauflösung (DNS).

*/etc/nscd.conf* Hier wird die Konfiguration des Name Service Caching Dienstes hinterlegt.

*/etc/nsswitch.conf* Der Nameservice-Switch regelt, wie bestimmte Informationen, wie Benutzernamen, UserID, Passwörter ermittelt werden.

```
[...]
passwd:     compat ldap
group:      compat
hosts:      files dns
networks:   files dns
[...]
```

Die gezeigten Einstellungen im Ausschnitt aus der Datei legen fest, dass für die Passwortüberprüfung und UserID zuerst in den klassischen Dateien */etc/passwd* und */etc/shadow* nachgesehen werden soll, bevor ein LDAP-Server um diese Daten befragt wird. Für die Auflösung von Rechnernamen wird zuerst die Datei */etc/hosts* bemüht, bevor eine Anfrage an den DNS-Server gestellt wird.

## 7.7 Das umfangreichste Verzeichnis /usr

*/usr* enthält je nach Linux-Distribution die umfangreichste Verzeichnisstruktur des Systems. Hier liegt der größte Teil der installierten Software.[1] Auf vielen Systemen befinden sich in und unterhalb von */usr* mehr Daten als in allen anderen Dateien zusammen. Die Programmdateien sind meist in */usr/bin*, die Dienste in */usr/sbin* abgelegt.

```
X11    bin    i586-suse-linux  lib      local  sbin   src
X11R6  games  include          libexec  man    share  tmp
```

Im gezeigten Beispiel ist der Inhalt von */usr* auf einem SuSE-Linux wiedergegeben. */tmp* ist aus historischen Gründen vorhanden und zeigt auf *../var/tmp*, *X11* auf *X11R6*.

In Netzwerken, an die viele gleichartige Systeme angeschlossen sind, wird dieses Verzeichnis häufig auf einem zentralen Server gespeichert, und alle anderen Computer greifen über das Netzwerk darauf zu.

## 7.8 Optionale Software

Nicht alle Linux-Distributionen machen von diesem Verzeichnis Gebrauch. Unterhalb von */opt* sollte kommerzielle Software oder sehr große Programme, die nicht unmittelbar zum System gehören, wie etwa KDE, Gnome, OpenOffice, Mozilla, ihren Platz finden.

```
MozillaFirefox  cisco-vpnclient  kde3     novell
OpenOffice.org  gnome            mozilla
```

## 7.9 Die Schnittstelle zum Kernel /proc

## 7.10 Gerätedateien

Dieses Verzeichnis enthält nur Spezialdateien, sogenannte Gerätedateien. Diese stellen eine einfach zu nutzende Schnittstelle zur Hardware dar. Hier finden sich auch Einträge

[1] wenn sie nicht, wie beispielsweise bei SuSE nach teilweise nach */opt* verlagert ist

für alle Festplatten und ihre Partitionen: */dev/hda* ist die erste ATA-, */dev/sda* die erste SCSI-Festplatte[2] im System. Höhere Buchstaben (hdb, hdc) stellen weitere Festplatten dar, Zahlen am Ende (sda1, sda2) sind die Partitionen der Festplatten.

Da auf einer Festplatte nur vier primäre Partitionen möglich sind, wird häufig eine erweiterte Partition angelegt, die den größten Teil der Festplatte umfasst. In der erweiterten Partition können dann "logische Laufwerke" angelegt werden. Diese erhalten grundsätzlich die Partitionsnummern ab 5. Enthält eine Festplatte also eine primäre und eine erweiterte Partition, in der sich wiederum zwei logische Laufwerke befinden, gibt es auf dieser Platte die Partitionen 1, 2, 5 und 6. Die primäre Partition ist 1, die erweiterte ist 2, und die beiden logischen Laufwerke sind 5 und 6.

### 7.10.1 Probleme statischer Namensgebung

Die meisten wissen, was eine Geräte(device)datei ist. Aus dem oben genannten weiss man auch, warum Gerätedateien besondere Nummern haben. Aber was die meisten für gegeben ansehen, ist, dass die Primary Master IDE Festplatte als */dev/hda* bezeichnet wird. Das mögen am Anfang viele vielleicht nicht so sehen, aber das ist ein grundlegender Designfehler.

Wenn es an den bunten Pool von Hotplug-Geräten geht, wie sie am USB, IEE1394, hot-swappable PCI... werden können, stellt sich schnell die Fragen: "Was ist denn nun das erste Gerät? Und für wie lange? Wie werden die anderen Geräte benannt, wenn das erste verschwindet? Wie wird das laufende Transaktionen beeinflussen?"

Was würde passieren, wenn ein USB-Stick, eingebunden als erste (virtuelle) SCSI-Platte, plötzlich gezogen würde, aber noch weitere USB-Speichergeräte angeschlossen sind? Würde es Sinn machen bei jedem Anstecken einfach den Gerätebuchstaben hochzuzählen, um Verwechslungen auszuschliessen - die Buchstaben wären irgendwann am Ende etc.

### 7.10.2 Dynamische Devices mit udev

Das udev-Projekt addressiert die auftretenden Probleme und übernimmt dabei die folgenden Aufgaben:

- Läuft im userspace
- Erstellt/entfernt dynamisch Gerätedateien
- Liefert konsequente Benennung
- Liefert ein user-space API

Um diese Funktionen zu liefern wird udev in drei unterschiedlichen Teilprojekten entwickelt: udev, namedev und libsysfs.

**udev** Jedes Mal, wenn der Kernel ein Update in der Gerätestruktur feststellt, ruft er das **hotplug** Programm im User-Space auf. Hotplug führt die Anwendung aus, welche im */etc/hotplug.d/default* Verzeichnis, wo man auch einen symlink zum udef Programm findet, verlinkt ist. Hotplug übergibt die Informationen vom Kernel an udev, welches die notwendingen Aktionen - Erstellen oder Entfernen von Gerätedateien - an der */dev* Struktur ausführt.

[2] "SCSI"-Platten können auch USB-Festplatten, SATA-Platten und weitere Devices sein

**Libsysfs und /sys** udev interagiert mit dem Kernel durch das sysfs Pseudodateisystem. Das libsysfs Projekt liefert ein Standard API um auf die Informationen gegeben durch das sysfs Dateisystem in einem gängigen Verfahren zuzugreifen. Dies erlaubt eine Abfrage von aller Art von Hardware, ohne dass man Vermutungen über die Art der Hardware anstellen muss.

**namedev** Namedev gestattet es, Geräte seperat vom udev Programm zu bezeichnen. Dies erlaubt flexible Benennungsrichtlinien und Namensschemata, entwickelt von verschiedenen Körperschaften. Dieses Subsystem zur Gerätebenennung liefert ein Standardinterface, das udev benutzen kann.

Momentan wird nur ein einzelnes Benennungsschema von namedev geliefert, und zwar jenes, welches von LANANA geliefert wird. Dieses wird von der Mehrheit der Linux Systeme momentan verwendet und ist daher für die Mehrheit der Linuxanwender sehr brauchbar.

Namedev verwendet eine fünfstufige Prozedur um den Namen eines bestimmten Gerätes herauszufinden. Wenn in einem dieser Schritte der Gerätename gefunden wird, wird dieser Name verwendet. Diese Schritte sind:

- Beschriftung oder Seriennummer
- Bus Gerätenummer
- Bus Topologie
- Statisch vergebener Name
- Vom Kernel gelieferter Name

Der Beschriftung oder Seriennummer Schritt überprüft, ob das Gerät ein einzigartiges Identifikationsmerkmal hat. Zum Beispiel haben USB Geräte eine einzigartige USB Seriennummer und SCSI Geräte eine einzigartige UUID. Wenn Namedev eine Übereinstimmung zwischen dieser einzigartigen Nummer und einer gegebenen Konfigurationsdatei findet, dann wird der von der Konfigurationsdatei gelieferte Name verwendet.

Der Bus Gerätenummer Schritt überprüft die Bus Gerätenummer. Für nicht-hot-swapable Umgebungen ist diese Prozedur ausreichend, um ein Hardwaregerät zu identifizieren. Zum Beispiel verändern sich PCI Busnummern selten in der Lebenszeit eines Systems. Auch hier wird, wenn namedev eine Übereinstimmung mit dieser Position und einer gegeben Konfigurationsdatei findet, der Name verwendet, der von der Konfigurationsdatei geliefert wird.

Genauso ist auch die Bus Topologie ein eher statischer Weg zur Definition von Geräten solange die Benutzer nicht Geräte auswechseln. Wenn die Position des Gerätes zu einer vom Benutzer gelieferten Einstellung passt wird der beiliegende Name verwendet.

Der vierte Schritt Statisch vergebener Name ist ein simpler String Ersatz. Wenn der Kernelname (der Standardname) zu einem gegebenen Ersatzstring passt, wird der Ersatzname stattdessen verwendet.

Der letzte Schritt (Vom Kernel gelieferter Name) ist ein "Allesfänger": Dieser nimmt den vom Kernel gelieferten Standardnamen. In den meisten Fällen ist dies ausreichend, da es zu der Gerätebenennung, welche auf momentanen Linuxsystem verwendet wird, passt.

### 7.10.3 Beteiligte Prozesse und Dienste

## 7.11 Binärdateien (ausführbare Dateien)

in */bin* befinden sich wichtige Programme für Anwender, die immer verfügbar sein müssen, beispielsweise die Shells und das Mount-Kommando. Ähnlich wie /bin enthält auch /sbin wichtige Programme. Diese sind jedoch hauptsächlich für den Systemverwalter gedacht, da sie Funktionen erfüllen, auf die ein normaler Benutzer keinen Zugriff hat.

In */usr/bin* liegen alle ausführbaren Dateien, die nicht zur absoluten Basis- bzw. Bootausstattung gehören. Unterhalb von */usr/sbin* findet man die Systemdienste, wie **sshd**, **rpc.mountd**, **xntpd** und die meisten weiteren Standarddienste.

Weitere "bin"-Verzeichnisse gibt es unter */usr/X11R6*, */usr/local*, */opt/kde*, */opt/gnome* je nach Softwareausstattung, Distribution und Konfiguration der Maschine. Die Shellumgebungsvariable PATH erleichtert das Ausführen von Programmen, da nicht der absolute Pfad angegeben werden muss. Ausführbare Programme müssen entweder für die entsprechende Hardware programmiert und übersetzt (kompiliert) worden sein oder als Skripten installierter Interpretersprachen vorliegen.

Die Standardausstattung im Verzeichnis */bin* sieht ungefähr so aus:

```
dirk@s02:/bin> ls
arch           dumpkeys     ln             ps                  showkey
ash            echo         loadkeys       psfaddtable         sleep
ash.static     ed           loadunimap     psfgettable         sort
awk            egrep        logger         psfstriptable       stty
basename       eject        login          psfxtable           su
bash           false        ls             pwd                 sync
bluepincat     fgconsole    lsmod          rescan-scsi-bus.sh  tar
cat            fgrep        lsmod.static   resizecons          tcsh
chgrp          fillup       mail           rm                  testutf8
chmod          fuser        mapscrn        rmdir               touch
chown          gawk         mkdir          rpm                 true
chvt           getkeycodes  mknod          sash                umount
cp             grep         mktemp         scsidev             uname
cpio           guessfstype  more           sed                 unicode_start
csh            gunzip       mount          setfont             unicode_stop
date           gzip         mv             setkeycodes         usleep
dd             hostname     netstat        setleds             vi
deallocvt      initviocons  nisdomainname  setmetamode         vim
df             ipg          openvt         setserial           vitmp
dmesg          kbd_mode     pidof          sg_start            ypdomainname
dnsdomainname  kbdrate      ping           sh                  zcat
domainname     kill         ping6          showconsolefont     zsh
```

Diese Kommandos genügen zur einfachen Systemadministration. Programme, wie die Shell, diverse Filter und Parser, wie **sed**, **awk**, **grep**, Packprogramme **tar** und **gzip** sind hier zu finden. Im */usr/bin*-Verzeichnis sieht es je nach Zweck der Maschine um einiges voller aus, so dass sie hier nicht dargestellt werden soll.

## 7.12 Bibliotheken

Programmbibliotheken (shared libraries) und Programmteile, Skriptsprachenelemente, der Compiler etc. liegen üblicherweise in den Verzeichnissen:

*(/usr)/lib, /usr/X11R6/lib, ...* Der "Suchpfad" für Bibliotheken wird in einem Hashfile abgelegt */etc/ld.so.cache*. Die Erzeugung dieser Datei erfolgt über **ldconfig** wobei der Suchpfad für dieses Programm in der */etc/ld.so.conf* definiert wird. **ldconfig** wird automatisch beim Systemstart ausgeführt und sollte nach der Installation bzw. Update von dynamisch gelinkten Bibliotheken aufgerufen werden. Die Basisbiblotheken, die für die Standardprogramme aus */(s)bin* benötigt werden, finden sich unterhalb von */lib*

```
YaST                    libevms.so              libpam_misc.so.0
bootsplash              libext2fs.so.2          libpam_misc.so.0.78
cpp                     libext2fs.so.2.4        libpamc.so.0
evms                    libgcc_s.so.1           libpamc.so.0.78
firmware                libgetconfig.a          libpcprofile.so
klibc                   libgetconfig.la         libpthread.so.0
ld-2.3.4.so             libgetconfig.so         libreadline.so.5
ld-linux.so.2           libgetconfig.so.1       libreadline.so.5.0
ld-lsb.so.2             libgetconfig.so.1.1.0   libresmgr.so.0.9.8
libBrokenLocale.so.1    libhandle.so.1          libresolv.so.2
libNoVersion.so.1       libhandle.so.1.0.3      librt.so.1
libSegFault.so          libhd.so.10             libscpm.so
libacl.so.1             libhd.so.10.16          libscpm.so.1
libacl.so.1.1.0         libhistory.so.5         libscpm.so.1.1
libanl.so.1             libhistory.so.5.0       libselinux.so.1
libattr.so.1            libm.so.6               libss.so.2
libattr.so.1.1.0        libmemusage.so          libss.so.2.0
libblkid.so.1           libncurses.so.5         libsysfs.so.1
libblkid.so.1.0         libncurses.so.5.4       libsysfs.so.1.0.2
libc.so.6               libnscd.so.1            libthread_db.so.1
libcap.so               libnscd.so.1.0.0        libutil.so.1
libcap.so.1             libnsl.so.1             libuuid.so.1
libcap.so.1.92          libnss_compat.so.2      libuuid.so.1.2
libcidn.so.1            libnss_dns.so.2         libwrap.so.0
libcom_err.so.2         libnss_files.so.2       libwrap.so.0.7.6
libcom_err.so.2.1       libnss_hesiod.so.2      libxcrypt.so.1
libcrypt.so.1           libnss_ldap.so.2        libxcrypt.so.1.2.2
libdevmapper.so         libnss_mdns-0.2.so      libz.so.1
libdevmapper.so.1.01    libnss_mdns.so.2        libz.so.1.2.2
libdl.so.2              libnss_nis.so.2         lsb
libe2p.so.2             libnss_nisplus.so.2     modules
libe2p.so.2.3           libnss_winbind.so.2     scpm
libevms-2.3.so.0        libnss_wins.so.2        security
libevms-2.3.so.0.3      libpam.so.0             tls
libevms.a               libpam.so.0.78
```

Unterhalb von */lib* finden sich (distributionsspezifisch) einige weitere wichtige Verzeichnisse, wie *modules* für die Kernelmodule, *security* für die PAM-Bibliotheken und Module.

## 7.13 Variable Daten

Unterhalb des Verzeichnisses */var* werden alle vom System wärend der Laufzeit generierten Daten, wie z.B. Logfiles, Caches, Locks ... abgelegt. Hier liegen auch bestimmte Laufzeitdaten, wie z.B. die Datenbankfiles von MySQL und LDAP oder die Tabellen des Nameservers.

Die SMTP-Programme (Mailagents) legen unterhalb von */var* üblicherweise ihre Queues und teilweise die Mail der Systembenutzer ab. Dies geschieht auch durch den Serverdienst für Usenet-News. Laufen viele und häufig benutzte Dienste, so wird in diesem Bereich

Mail gespeichert ..., dann sollte für dieses Verzeichnis ausreichend Platz für dynamisches Wachstum eingeplant werden.

## 7.14 Das Temporärverzeichnis

Im Verzeichnis */tmp* legen einige Programmen temporäre Dateien zur Zwischenspeicherung von Laufzeitdaten oder zur Prozess-Interaktion an. Zum Erzeugen einer Datei in diesem Verzeichnis braucht ein Prozess keine besonderen Rechte. Das Löschen von Dateien ist nur den Prozessen des Dateieigentümers erlaubt, wenn das Stickybit für das */tmp* Verzeichnis gesetzt ist.

```
user@linux:~/skripten/grundlagen> ls -ld /tmp/
drwxrwxrwt   37 root     root         4096 2003-08-20 14:15 /tmp
```

Weil in */tmp* Dateien unvorhersehbarer Größe zu unberechenbaren Zeitpunkten von allen Benutzern eines Systems erzeugt werden können, legt man dieses Verzeichnis am besten in eine eigene Partition. Der Cron-Dienst sollte dazu eingesetzt werden, regelmäßig in bestimmten Abständen das Verzeichnis aufzuräumen und die Dateien zu entsorgen. Sonst kann es schnell passieren, dass normale Benutzer durch großzügiges Auslagern ihrer Überbestände den Temporärbereich überlaufen lassen.

## 7.15 Literatur

- S. Hetze, D. Hohndel, M. Müller, O. Kirch, *Anwenderhandbuch -LINUX-, Leitfaden zur Systemverwaltung*, LunetIX, 1997
- Fred Hantelmann, *LINUX für Durchstarter*, Springer, 1997

Beide Bücher geben eine gute Einführung in den Umgang mit Linux, das Login, die Shell (Kommandoeingabe), das Dateisystem und wichtige Systembefehle. Erklärungen zu Netzwerken erfolgen nicht oder nur sehr am Rande und sind in Anbetracht des Druckdatums nicht mehr aktuell.

## 7.16 Aufgaben

### 7.16.1 Rechtesystem

1. Wie lässt sich das Rechte-Muster -rw-r- -r- - einer Datei erzeugen und was sagt es aus?
2. Mit welchem Kommando lässt sich der Dateibesitzer/die Gruppe ändern? Wie kann man die Rechte-Maske auch für Unterverzeichnisse geltend machen?
3. Wozu dient das SUID-Bit und wie kann es gesetzt werden?
4. Welche Aufgabe hat das Sticky-Bit und wo findet man es im Einsatz?

# Kapitel 8

# Prozessmanagement

## 8.1 Einführung

In Unix-Systemen können mehrere Programme gleichzeitig gestartet werden. Ein Programm, das gerade ausgeführt wird, trägt die Bezeichnung Prozess. Das Betriebssystem verwaltet mehrere Prozesse mit einem Scheduler, der die Zuteilung der CPU an die Prozesse steuert,in dem er jedem ausführbereitem Prozess die CPU eine gewisse Zeit zuteilt und sie ihm nach Ablauf dieser Zeit wieder entzieht. Dieses Prinzip wird präemptives Multitasking genannt, im Unterschied zum kooperativen Multitasking unter älteren Windows-Versionen, in denen ein Prozess selbst entscheiden konnte, ob er die CPU-Zuteilung an einen anderen Prozess erlaubt. Alle Prozesse erhalten eine eindeutige Nummer (PID), angefangen wird mit der "1", es geht bis "32767", dann startet die Vergabe wieder von vorne, wobei bereits belegte Zahlen ausgelassen werden. PID steht für Process Identifier. Somit sind für das System Verwechslungen ausgeschlossen. Logischerweise können Prozesse über diese Zahl auch kontrolliert und gegebenenfalls beendet werden.

Nach dem Starten des (Linux-)Kernels werden eine Reihe von Programmen und Hintergrundsdiensten hochgefahren. Dieses lässt sich in definierten Abstufungen steuern. Vieles wird dabei von speziellen Skripten erledigt. Linux bietet vielfältige Möglichkeiten, sich über den Zustand des Systems zu informieren.

## 8.2 Systemstart/Runlevel

Es gibt unter Unix/Linux zwei Konzepte: Je nachdem, welches Konzept ein System verfolgt, läuft der bootstrap ein wenig anders ab. Die beiden Familien sind BSD und SystemV, sprich System5. Eines haben beide Familien beim Bootstrap gemeinsam: Die Datei */etc/inittab*. Diese Datei ist der Anfang des Unix-Lebens. Das erste Skript, das von */etc/inittab* aus ausgeführt wird, rangiert unter dem Namen sysinit (Abkürzung si). In diesem Eintrag steht meistens */sbin/init.d/boot*. In dem Sysinit-Skript werden Dinge wie **fsck** und das Einbinden von Filesystemen, Installation von Swap-Partitionen sowie weitere Basisaufgaben erledigt. Bis hierher verhalten sich alle Systeme gleich. Zurück zur */etc/inittab*:

In der */etc/inittab* wird die erste Möglichkeit geschaffen, dass sich jemand einloggen kann, um das System zu managen. Von hier aus werden auch alle anderen Konfigurationsdateien/Startupdateien (indirekt) angestoßen. In dieser Datei wird der Default-/Runlevel festgelegt, und je nach Runlevel auch Konfigurations-Shellskripte wie */etc/rc.d/rc.multi* angestoßen. In dieser und in einer handvoll anderer Dateien wird das gesamte System gestartet. Der Bereich /etc/rc.d ist typisch für BSD-artige Systeme. Die meisten Linuxdistributionen verwenden jedoch das "System V - Init".

SystemV hat eine andere Methode für die Start-Skripte. Hier wird intensiv von Softlinks Gebrauch gemacht. In SystemV gibt es pro Dienst (bzw. Daemon) genau ein Start-/Stop-Skript. Die Start-/Stop-Skripte in SystemV akzeptieren genau einen Parameter; dieser muss entweder "start", "stop", "restart" , "status" ... sein. Ruft man ein solches Skript mit "start" auf, so wird der Dienst hochgefahren.

Entsprechend wird er bei dem Parameter "stop" angehalten. All diese Dateien befinden sich in einem einzigen Verzeichnis, üblicherweise in *(/etc)/init.d* bzw. */etc/rc.d* (Vor der Festlegung der File System Hierarchy Standard fand man diese Unterverzeichnisse unterhalb */sbin.*)

Außer diesem Verzeichnis gibt es auch noch die Unterverzeichnisse:

- /etc/init.d/rc0.d
- /etc/init.d/rc1.d
- /etc/init.d/rc2.d
- /etc/init.d/rc3.d
- /etc/init.d/rc4.d
- /etc/init.d/rc5.d
- /etc/init.d/rc6.d

In den Verzeichnissen befinden sich Links der Form

- /etc(/init.d)/rc?.d/S??dienstname –> /etc/init.d/dienstname
- /etc(/init.d)/rc?.d/K??dienstname –> /etc/init.d/dienstname

Wobei nicht für jedes Skript unbedingt eine Start und eine Stopverknüpfung existieren müssen. Für das automatische Hochfahren von Diensten gibt es zwei Möglichkeiten der Steuerung: Entweder werden die o.g. Links nur bei Bedarf angelegt oder es existieren für jedes Skript die notwendigen Einträge. Dieses ist bei SuSE-Linux der Fall. Die Steuerung ob ein Dienst gestartet werden soll oder nicht, findet hier in der zentralen Konfigurationsdatei */etc/rc.config* statt.

Möchte das System nun aus irgendeinem Grund in einen Runlevel kommen, so schaut es (auch wieder ein Skript) in dem entsprechenden Verzeichnis nach Dateien (die in Wirklichkeit Links nach (/etc)/init.d sind), die mit "S" beginnen, und führt diese in alphabetischer/numerischer Reihenfolge mit dem Parameter "start" aus. Konventionsgemäß kommt nach dem "S" eine zweistellige Zahl, mit der man festlegen kann, in welcher Reihenfolge ein Dienst gestartet werden soll. Soll ein Dienst in einem Runlevel nicht verfügbar sein, so gibt es einen Softlink auf das entsprechende Skript, der mit einem K beginnt. Diese Skripte (K*) werden mit dem Parameter "stop" ausgeführt - der Dienst wird dann gestopt. Unter den SuSE-Distributionen wird der Eintrag von Diensten in die entsprechenden Runlevel mit dem Programm **insserv** erledigt.

SuSE-Linux verwendet nicht alle Runlevel, wobei die folgenden Runlevel standardmässig zur Verfügung stehen und nachstehende Bedeutung haben (dieses kann von anderen Linuxdistributionen abweichen):

| Runlevel | Funktion |
|---|---|
| S | Single-User-Mode |
| 0 | Halt des Systems |
| 1 | Single-User-Mode ohne Netzwerk ohne X |
| 2 | Multiusermode ohne Netzwerk ohne X |
| 3 | Multiusermode mit Netzwerk ohne X |
| 5 | Multiusermode mit Netzwerk und grafischer Oberfläche |
| 6 | Reboot |

Tabelle 8.1: Runlevel

Den aktuellen Runlevel kann man sich mit dem Befehl **runlevel** anzeigen lassen. Wobei nur der Init-Prozess dafür sorgt, dass die Programme, die in der */etc/inittab* angegeben sind, permanent laufen, wenn sie über eine gültige Konfiguration verfügen. Ist man erstmal in einem bestimmten Runlevel, so können bereits weitere Dienste von Hand nachgestartet oder gestoppt werden, hier reagiert das System nicht automatisch.

## 8.3 Dämonen

Im Unterschied zu anderweitigen Prozessen, die stets an ein Dialogstation (tty) und Benutzer gekoppelt sind und je nach Aufruf im Vorder- oder Hintergrund laufen, werden Dämonen meistens automatisch beim Hochfahren des Systems über die Runlevelskripte gestartet. Sie sind nicht an eine Konsole gebunden und schreiben deshalb ihre Meldungen üblicherweise über das Syslog oder in spezielle Logdateien und -verzeichnisse.

## 8.4 System- oder Resourcen-Auslastung

Prozesse können im System in verschiedenen Zuständen auftreten. Die naheliegensten sind “in Ausführung befindlich” und “auf die Zuteilung von CPU-Zeit wartend”. Meistens wird aber noch feiner klassifiziert, wobei die Einteilung für verschiedene Unix-Systeme abweicht. Unter Linux gibt es folgende Einteilung:

- **R - runnable** (ablaufbereit) - Der Prozess wird gerade auf der CPU abgearbeitet oder steht zur Abarbeitung bereit.
- **S - sleeping** (schlafend) - Der Prozess wartet auf das Eintreten eines Ereignisses, zum Beispiel das Ablaufen eines Timers, das Ende einer Interaktion mit Festplatten, dem Nutzer oder einem anderen Prozess.
- **D - uninterruptibly sleeping** (Schlaf ohne Unterbrechungsmöglichkeit) - Der Prozess wartet auf einen bestimmten Hardwarezustand.
- **T - traced** (gestoppt) - Der Prozess befindet sich im Einzelschrittlauf (Tracing).
- **Z - zombie** - Der Prozess wurde beendet, der Elternprozess hat jedoch den Rückkehrstatus noch nicht geprüft.

Informationen über den Zustand der momentan im System ablaufenden Prozesse kann sich der Administrator auf verschiedene Weise beschaffen. Dazu fallen einem zuerst die folgenden Tools ein: **ps** und **top**. Sie zeigen laufende Prozesse und ihren Status an.

### 8.4.1 ps

Der Befehl **ps** liefert einen "snapshot" der laufenden Prozesse in der gerade aktiven Shell bis zum kompletten Abbild des Systemzustandes. Was die einzelnen Spalten bedeuten, steht in der Manpage. Jeder Prozess im Unix-System wird von einem anderen Prozess mit Hilfe eines Systemaufrufs gestartet. Dadurch entsteht eine Eltern-Kind-Beziehung zwischen den Prozessen. Diese Abhängigkeiten von Prozessen lassen sich mit dem Kommando **pstree** oder auch durch `ps auxf` darstellen. Mit einer bestimmten Option wird neben der Prozess-ID (PID) auch die Prozess-ID des Elternprozesses (PPID - parental PID) angezeigt. Der Vaterprozeß wird vom "Ableben" (der Beendigung) seines Kindprozesses durch einen speziellen Mechanismus, der Signal genannt wird, benachrichtigt.

Jeder Prozess hinterläßt bei der Beendigung einen sogenannten Exitstatus, den der Vaterprozeß auswerten kann. Der Exitstatus enthält Informationen darüber, ob und durch welches Signal der Prozess beendet wurde und einen vom Programmierer selbst bestimmten numerischen Wert, mit dem er dem Nutzer weitere Informationen über das Ende des Prozesses geben kann.

Solange der Vaterprozeß die Statusinformation seines beendeten Kindprozesses noch nicht abgeholt hat, existiert dieser in Form eines Zombieprozesses. Ein Zombieprozeß nimmt weder Rechenzeit noch Hauptspeicher in Anspruch, sondern existiert nur als Eintrag in der Prozesstabelle, die das Betriebssystem führt. Unter Umständen kann es dazu kommen, dass der Elternprozeß vor dem Kindprozeß beendet wird. Damit kann dieser die Statusinformation beim Beenden des Kindprozesses nicht mehr abfragen. In diesem Fall übernimmt **init** diese Statusmeldung.

Jeder Prozess besitzt einen Standardeingabe-, Standardausgabe- und Standardfehlerkanal. Diese sind normalerweise mit der Tastatur (Standardeingabe) bzw. dem Bildschirm (Standardausgabe, Standardfehler) verbunden. Wie diese Kanäle umgebogen werden können, so dass Fehlerausgaben zum Beispiel in eine Datei geschrieben werden, wird im Rahmen der Betrachtungen zur Shell-Kommandozeile behandelt. (s. 5.2.1, S. 50)

### 8.4.2 top

Dieses Kommando greift in regelmässigen Abständen auf das Procfilesystem zu und gibt eine gute (sortierte) Übersicht darüber, welche Task auf einem Rechner wieviel Resourcen ( i/o , Memory , Prozessor) schluckt. Die Zeit zwischen den Updates kann bereits beim Aufruf über die Kommandozeile eingestellt werden. **top** wird interaktiv bedient. Das einfache Drücken einer Taste genügt also, um dem Programm mitzuteilen, was Sie wünschen. Die wichtigsten Tasten sind:

h : zeigt die Hilfe mit allen Tastenkürzeln an.

M : sortiert die Liste nach dem Speicherverbrauch der Programme.

P : sortiert die Liste wieder nach der Prozessorbelastung der Programme .

k : killt einen Prozess. Fragt nach der PID und dem gewünschten Signal.

q : beendet top

Nun soll kurz auf ein paar der Spalten eingegangen werden, die **top** in seiner tabellarischen Ausgabe anzeigt:

**top** kann in vielen Fällen ein hilfreiches Tool sein. Kommt Ihnen Ihr Computer z.B. viel zu langsam vor, so kann es sein, dass ein oder mehrere Programme die Ressourcen

| | |
|---|---|
| PID | Logischerweise der PID des angezeigten Programms |
| USER | Der Benutzer, der das Programm aufgerufen hat |
| SIZE | Die Größe des Programms im Arbeitsspeicher |
| SHARE | Menge des Speichers, den das Programm mit anderen gemeinsam nutzt |
| %CPU | Anteil an der Prozessorauslastung durch das Programm |
| %MEM | Anteil an der Auslastung des Arbeitsspeichers durch das Programm |

Tabelle 8.2: Spaltenüberschrift und deren Bedeutung

übermäßig beanspruchen. Diese Programme können Sie mit **top** leicht ausfindig machen und ggf. beenden. Desweiteren ist es sehr interessant zu sehen, was das System am meisten belastet.

### 8.4.3 uptime

**uptime** zeigt die Systemzeit, die Zeitspanne in der das System bereits läuft, die Zahl der Benutzer(Innen) und die durchschnittliche Systemlast der letzten Minute, fünf Minuten und 15 Minuten an:

```
dirk@shuttle:~ > uptime
  7:39pm  up 6 days  6:43,  15 users,  load average: 1.17, 1.23, 1.20
```

### 8.4.4 time

Manchmal kann es recht nützlich sein, die Zeit zu ermitteln, die einzelne Prozesse zur Abarbeitung benötigen: **time kommando** (z.B. um einen einfachen Bench zur Systemleistung zu haben, kann man die Kernelkompilationszeit mit **time make bzImage** ermitteln).

### 8.4.5 nice und renice

Mit **nice** kann man beim Start eines Prozesses bestimmen, wie kooperativ (freundlich) er mit der Systemresource "Prozessor" umgehen soll. Gerade für nicht-interaktive Prozesse (Langläufer) sollte man die Freundlichkeit erhöhen. Mit **renice** geht das auch im Nachhinein.

### 8.4.6 kill, killall -9

Frißt ein Prozess gar zu viele Resourcen, so bestraft man ihn am besten mit `kill PID` oder `kill -9 PID` - um ihn zu beenden. Aber Achtung, wenn der Prozess gerade im IO steckt, so klappt das eventuell nicht. Das Kommando `killall programmname` wirkt auf alle Prozesse mit einem bestimmten Programmnamen.

Ein Signal wird durch einen numerischen Wert repräsentiert (`kill -N`). Jedes Programm kann sogenannte Signalhandler für die meisten Signale anbieten, in denen dann das Signal ignoriert oder in angemessener Weise darauf reagiert werden kann. Obwohl Signale allgemein vom Betriebssystem verwendet werden, um dem Programm schwerwiegende Fehler anzuzeigen (zum Beispiel Überschreitungen des Prozess-Speicherbereichs, Divisionen durch Null, ...), kann der Nutzer den von ihm selbst gestarteten Prozessen auch Signale schicken und die Prozesse damit steuern.

Dabei legt der Programmierer fest, wie auf bestimmte Signale reagiert wird: SIGTERM soll zwar die Ausführung eines Prozesses beenden, kann auch ignoriert werden, sodass der Prozess dann nicht auf die Beendigungsaufforderung reagiert. Will man als Nutzer oder Administrator einen laufenden Prozess zerstören, so sollte zunächst das Signal SIGTERM

| Signal | Nummer | Beschreibung |
|---|---|---|
| SIGHUP | 01 | Hangup (Auflegen bzw. erneutes Lesen der Konfiguration) |
| SIGINT | 02 | Interrupt (Unterbrechung z.B. durch Strg-[c]) |
| SIGQUIT | 03 | Quit (Beenden) |
| SIGALL | 04 | illegale Instruktion |
| SIGTRAP | 05 | Trace Trap (-¿ Debugger) |
| ... | | |
| SIGKILL | 09 | Unbedingte Beendigung des Prozesses durch das Betriebssystem |
| ... | | |
| SIGSEGV | 11 | Prozess hat den Speicherschutz verletzt |
| ... | | |
| SIGTERM | 15 | Prozess soll Ausführung beenden |

Tabelle 8.3: Prozess-Signale

benutzt werden. Viele Prozesse sichern daraufhin für ihre weitere Arbeit wichtige Daten und beenden sich dann selbst. Erst wenn auf SIGTERM hin keine Reaktion erfolgt, sollte SIGKILL verwendet werden.

SIGSEGV hat eher Informationscharakter: Ein Prozess hat den Speicherschutz verletzt und versucht, außerhalb des für ihn vorgesehenen Speicherbereiches zu schreiben. Dieses Signal ähnelt von der Bedeutung her der "Allgemeinen Schutzverletzung"[1] unter älteren Windows-Systemen. Man beachte jedoch, dass unter Unix-Systemen ein Prozess keine Systemdatenstrukturen beschädigen kann. Daher ist dieses Signal mehr als Information über den Grund des Programmabsturzes zu betrachten. Das System wird auch nach dem SIGSEGV eines Prozesses stabil bleiben.

## 8.5 Selbständige Prozesse

Normalerweise führt das Beenden eines Elternprozesses zur Terminierung aller von diesem Prozess abgeleiteten Kindprozesse (Kann man sich mit `ps auxf` anzeigen lassen). Im Einzelnen zieht ein **logout**, das Abmelden vom System, ein automatisches Löschen aller noch anhängigen Jobs nach sich, die wärend dieser Sitzung gestartet wurden.

Abhilfe schafft hier der Befehl `nohup Kommando &`. Denn **nohup** startet den Prozess in der Weise, dass er gegen ein Beenden der Shell immun ist und schreibt die evtl. anfallenden Ausgaben des Befehls in eine Datei (üblicherweise *nohup.out*). Ein Nebeneffekt der Verwendung von **nohup** ist, dass die Prozesse mit einem um fünf höheren Nicelevel gestartet werden.

## 8.6 Zeitsteuerung

Linux bietet etliche Programme für das Automatisieren von Aufgaben. Ein Beispiel ist **cron**. Dieser ist ein Systemdienst und sollte beim Booten automatisch gestartet und im Normalbetrieb nicht beendet werden. Cron ist für sich wiederholende Aufgaben zuständig, die automatisch zu bestimmten Zeiten stattfinden sollen.

[1] dem berühmtesten Windows-Fenster, dem Blue-Screen

Für die Verwaltung seiner Aufgaben liest **cron** die *crontab*-Datei. Das System und jeder User haben ihre eigenen *cron*-Dateien. Die des Systems befindet sich in der Datei */etc/crontab*. (Sie sollte möglichst nicht geändert werden!) Auch der Benutzer "root" sollte seine eigene Datei erzeugen.

Das Kommando **crontab /etc/crontab** wird eine crontab-Datei erzeugen, die eine Kopie der System crontab-Datei ist. Diese Datei kann nun mit **crontab -e** editiert werden. Beachten Sie, dass *crontab* sowohl der Name der Datei, als auch der Name des ausführbaren Programmes ist - ähnlich wie *passwd*. Die erzeugte Datei wird in ihrer Grundstruktur in etwa wie diese Konfiguration eines SuSE-Systems aussehen:

```
SHELL=/bin/sh****
#dhcpcd: if not then delete /var/run//dhcpcd-eth0.pid file

PATH=/usr/bin:/usr/sbin:/sbin:/bin:/usr/lib/news/bin
MAILTO=root
#
# check scripts in cron.hourly, cron.daily,
#  cron.weekly, and cron.monthly
#
-*/15 * * * *   root  test -x /usr/lib/cron/run-crons && /usr/lib/\
cron/run-crons >/dev/null 2>&1
59 *  * * *     root  rm -f /var/spool/cron/lastrun/cron.hourly
15 4  * * *     root  rm -f /var/spool/cron/lastrun/cron.daily
29 5  * * 6     root  rm -f /var/spool/cron/lastrun/cron.weekly
44 4  1 * *     root  rm -f /var/spool/cron/lastrun/cron.monthly

# reinitialize adsl connection
30 4  * * *     root (rcnetwork stop dsl0; sleep 5; rcnetwork start\
dsl0;)
```

Die Zeilen zur Steuerung bestimmter Abläufe haben ein spezielles Format. Fünf Zeitfelder, gefolgt von dem auszuführenden Programm. Die systemweite *crontab* besitzt ein weiteres Feld, das **cron** anweist, das Programm als spezieller User auszuführen (z. B. "root"). In einem User-crontab wird dieses Feld ignoriert. Die fünf Zeitfelder sind: Minuten, Stunden, Tag-des-Monats, Monat, Wochentag.

| **Zeit-/Datumsfeld** | **Definitionsbereich** |
|---|---|
| Minute | 0-59 |
| Stunde | 0-23 |
| Tag-des-Monats | 1-31 |
| Monat | 1-12 (oder Namen, siehe unten) |
| Wochentag | 0-7 (0 oder 7 ist Sonntag oder Namen) |

Tabelle 8.4: Definitionsbereiche der Zeit- und Datumsfelder

Ein Feld kann auch ein Stern (*) sein, was immer für "Erster-Letzter" steht. Zahlenbereiche sind erlaubt. Bereiche sind zwei Zahlen, getrennt durch einen Bindestrich. Die angegebenen Grenzen sind inklusive. Beispielsweise: 8-11 in "Stunde" bewirkt die Ausführung um 8, 9, 10, 11 Uhr. Listen sind ebenfalls erlaubt. Eine Liste ist eine Menge von Nummern (oder Bereichen), getrennt durch Kommata. Beispiele: 1,2,5,9 oder 0-4,8-2.

Schrittweiten können in Verbindung mit Bereichen genutzt werden. Hinter einem Bereich mit "/¡Schrittweite¿" angegeben, bestimmt die Schrittweite, ob Werte innerhalb des Bereiches übersprungen werden. Beispiel: "0-23/2" kann unter Stunden benutzt werden, um ein spezielles Kommando alle zwei Stunden auszuführen. Die Alternative wäre: "0,2,4,6,8,10,12,

14,16,18,20,22". Schrittweiten sind auch nach Sternen (*) erlaubt, "alle zwei Stunden" lässt sich auch durch "*/2" beschreiben.

Namen können für "Monat" und "Wochentag" benutzt werden. Benutzen Sie die ersten drei Buchstaben des entsprechenden Tages oder Monats (Gross-/Kleinschreibung wird nicht beachtet). Bereiche oder Listen sind mit Namen nicht erlaubt.

## 8.7 Systeminformation- und Überwachung

### 8.7.1 System-Log

Der Syslog-Daemon ist vor allem bei der Fehlersuche und Sicherheitsprüfung ein wichtiger Helfer. Programme können an ihn Meldungen schicken (mit Hilfe spezieller Befehle), die dann - je nach Einstellung - in verschiedenen Dateien und/oder auf Konsolen auch auf anderen Rechnern geschrieben werden. Viele Daemons und der Kernel benutzen diese speziellen Befehle und geben ihre Ausgaben und Meldungen an den **syslogd** weiter, da sie sonst keine Ausgabemöglichkeit haben.

| **Typ** | | **Priorität** | |
|---|---|---|---|
| auth | Authentifizierung | debug | Debugmeldungen |
| cron | Meldungen des cron-Daemon | info | Informationen |
| daemon | Meldungen von Daemons | warn | Warnungen |
| kern | Kernelmitteilungen | crit | kritische Fehlermeldungen |
| syslog | Meldungen des Logsystems | ... | weitere ... |
| mail | Meldungen des Mail-Subsystems | | |
| ... | weitere ... | | |

Tabelle 8.5: Unterteilung der Meldungen nach zwei Eigenschaften

Als einer der ersten Prozesse wird beim Hochfahren von Linux **syslog** gestartet. Dieser Dienst nimmt die System-, Fehler- und Warnmeldungen des Kernels und der einzelnen Dienste auf und schreibt sie je nach Konfiguration (*/etc/syslog.conf* an einen Logserver im Netz, in die Datei(en) */var/log/messages* bzw. */var/log/warn* (und weitere) und/oder auf die zehnte Textkonsole. Wenn viele Log-Informationen geschrieben werden, kann es unterhalb von */var/log* voll werden, was vielleicht zum Überlauf der entsprechenden Partition führt. Je nach Log-Einstellungen muss also dieser Bereich regelmäßig kontrolliert werden.

Jeder Autor eines Programmes, das Meldungen an den **syslogd** weitergibt, ist angehalten, sich an diese Konventionen zu halten; die Schlüsselwörter also im richtigen Sinne zu gebrauchen.

```
# /etc/syslog.conf - Configuration file for syslogd(8)
#
# einige Standardlogfiles
#
auth,authpriv.*                 /var/log/auth.log
*.*;auth,authpriv.none          -/var/log/syslog
cron.*                          /var/log/cron.log
daemon.*                        -/var/log/daemon.log
kern.*                          -/var/log/kern.log
user.*                          -/var/log/user.log
#
#
```

```
# alle DEBUG-Meldungen in diese Datei
#
*.=debug;\
        auth,authpriv.none;\
        news.none;mail.none     -/var/log/debug
#
#
# alle oben nicht gelogten Meldungen in diese Datei
#
*.=info;*.=notice;*.=warn;\
        auth,authpriv.none;\
        cron,daemon.none;\
        mail,news.none          -/var/log/messages
#
#
# alle Meldungen auf die Konsole 10 ...
#
daemon,mail.*;\
        news.=crit;news.=err;news.=notice;\
        *.=debug;*.=info;\
        *.=notice;*.=warn       /dev/tty10
#
#
# ... und die XKonsole
#
daemon.*,mail.*;\
        news.crit;news.err;news.notice;\
        *.=debug;*.=info;\
        *.=notice;*.=warn       |/dev/xconsole
```

### 8.7.2 Boot-Log

Für einen Administrator ist es sinnvoll zu wissen, was beim Booten vor sich ging. Diese Information erhält man aus den Logfile(s) */var/log/boot.msg* (in Abhängigkeit von der Distribution), */var/log/messages*, sofern der **syslogd** läuft und aus dem Befehl **dmesg**.

### 8.7.3 Belegung des Plattenspeichers

Auskunft über den verfügbaren Festplattenplatz erhält man mit dem Kommando **df** (Disk Free). Dieses zeigt die einzelnen gemounteten Bereiche des Filesystems auf. Mit **du** (DiskUsage) lässt sich der belegte Platz in einem Verzeichnis bestimmen. Dieses kann jedoch je nach Grösse des Bereichs recht lange dauern. Die Kommandos **df**, **du** und **ls** kennen den gemeinsamen die Option "-h", die aus der Zahlenausgabe in Byte etwas "menschlich lesbares" (von human readable) generiert.

### 8.7.4 Offene Dateien und Netzwerkverbindungen

Das Kommando **lsof** (ListOpenFiles) zeigt an, welche Prozesse welche Dateien (zum Lesen oder Schreiben) geöffnet haben. Weiterhin werden offene Sockets (Netzwerkverbindung mit Portadresse), sowie verwendete Pipes und Filesockets angezeigt.

### 8.7.5 Das Kommando "netstat"

**netstat** hat sich auf das Anzeigen offener Sockets, sowohl des Filesystems, als auch von Netzwerkverbindungen spezialisiert. Die Ausgabe von `netstat -a` hat folgendes Aussehen, wobei die Protokolle TCP, UDP und unix (Socket im Filesystem) in dieser Reihenfolge sortiert aufgeführt werden.

```
dirk@shuttle:~/ > netstat -a |more
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 *:silc                  *:*                     LISTEN
tcp        0      0 *:sunrpc                *:*                     LISTEN
tcp        0      0 *:ipp                   *:*                     LISTEN
tcp        0      0 localhost:smtp          *:*                     LISTEN
tcp        0      0 linux2.rz.uni-fr:45256  114-0.ruf.:microsoft-ds ESTABLISHED
tcp        0      0 linux2.rz.uni-fr:40796  login2.ruf.uni-frei:ssh ESTABLISHED
...
udp        0      0 192.168.100.:netbios-ns *:*
udp        0      0 randy2.ruf.u:netbios-ns *:*
udp        0      0 *:netbios-ns            *:*
udp        0      0 192.168.100:netbios-dgm *:*
udp        0      0 randy2.ruf.:netbios-dgm *:*
udp        0      0 *:netbios-dgm           *:*
Active UNIX domain sockets (servers and established)
unix  2      [ ACC ]     STREAM     LISTENING     814283 /tmp/.ICE-unix/dcop...
unix  2      [ ACC ]     STREAM     LISTENING     21171  /tmp/orbit-dirk/linc-...
unix  2      [ ACC ]     STREAM     LISTENING     18666  private/defer
unix  2      [ ACC ]     STREAM     LISTENING     18674  private/trace
unix  2      [ ACC ]     STREAM     LISTENING     18678  private/verify
...
```

Mit `netstat -M` kann man sich einen Überblick verschaffen, welche Netzwerkverbindungen gerade maskiert werden:

```
root@server:/ # netstat -M
IP masquerading entries
prot  expire  source          destination   ports
tcp   0:40.46 1234.test.local 210.24.30.216 3592 -> 2183 (64155)
tcp   0:45.64 1234.test.local 210.24.30.216 3593 -> 2187 (64156)
tcp   0:49.43 1234.test.local 210.24.30.216 3594 -> 2188 (64157)
tcp   1:03.12 1234.test.local 210.24.30.216 3596 -> 2195 (64159)
tcp   0:54.57 1234.test.local 210.24.30.216 3595 -> 2192 (64158)
...
```

Wird die Option "-n" beim Aufruf des Kommandos gesetzt, wird keine Namensauflösung durchgeführt, was die Anzeige stark beschleunigen kann, gerade bei IP- Nummern, die nicht im Nameserver bekannt sind.

**netstat -r** liefert die Kernel-Routingtabelle in ähnlicher Weise, wie sie auch vom Kommando **route** angezeigt wird. Wenn Sie netstat mit dem Flag -i aufrufen, gibt es die Statistiken für die gerade aktiven Netzwerk-Schnittstellen aus. Geben Sie außerdem das Flag -a mit an, werden alle im Kernel vorhandenen Schnittstellen ausgegeben, nicht nur die konfigurierten. An der Konsole produziert **netstat -i** in etwa folgendes:

```
root@server:/ # netstat -i
Kernel Interface table
Iface   MTU Met   RX-OK RX-ERR RX-DRP RX-OVR   TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0   1500   0 4413866      0      0      0 3421721      0      0      0 BMRU
```

```
lo     16436   0 1946052      0      0      0 1946052      0      0      0 LRU
vmnet   1500   0       0      0      0      0    1549      0      0      0 BMRU
```

Die Spalten MTU und Met geben die aktuelle MTU (Maximal Transfer Unit) und Metrik des Interface an. Die mit RX bzw. TX überschriebenen Spalten geben an, wie viele Pakete fehlerfrei empfangen bzw. gesendet wurden (RX-OK/TX-OK), wie viele beschädigt waren (RX-ERR/TX-ERR), wie viele wegeworfen werden mußten (RX-DRP/TX-DRP) und wie viele aufgrund eines Overruns verlorengingen (RX-OVR/TX-OVR). Die letzte Spalte zeigt wieder die Liste der Flags, die für die Schnittstelle gesetzt sind. Das sind einbuchstabige Versionen der langen Flagnamen, die **ifconfig** ausgibt. Siehe Tabelle 8.7.5 auf S. 97

| **Buchstabe** | **Bedeutung** |
|---|---|
| B | BROADCAST |
| L | LOOPBACK |
| M | PROMISC |
| O | NOARP |
| P | POINTOPOINT |
| R | RUNNING |
| U | UP |

Tabelle 8.6: Flags

## 8.8 Aufgaben

### 8.8.1 Runlevel

1. Welchen Sinn haben Runlevel?

2. Ab welchem Runlevel einer SuSE-Distribution habe ich üblicherweise eine a) Netzwerkverbindung, b) die grafische Oberfläche?

3. Man starte den Rechner manuell im Runlevel 1! Was hat dieser zu bedeuten? Kann man nun andere Rechner im Netz erreichen? Wie erreicht man, dass die Maschine sofort in einen bestimmten (anderen als den Defaultrunlevel) startet?

4. Wechseln Sie nach dem Hochfahren von Linux auf Runlevel 1 und anschliessend auf Runlevel 3!

5. Laufen in einem bestimmten Runlevel immer die entsprechenden Programme oder kann dieses abweichen? Wie wird die Reihenfolge festgelegt, in der bestimmte Prozesse gestartet werden, welchen Sinn macht das?

6. Wie sorge ich dafür, dass bestimmte Skripte beim Wechsel des Runlevels ausgeführt werden? Wie regelt das Suse, um etwas Aufwand zu ersparen? Wie lege ich fest, dass ein bestimmter Daemon etc. gestartet oder nicht mehr gestartet wird?

7. Wie wird die Reihenfolge festgelegt, in der bestimmte Prozesse gestartet werden, welchen Sinn macht das?

8. Man schreibe ein Runlevelskript (die meisten Linux-Distributionen bieten hierzu Vorlagen), welches die LocateDB beim Start des Rechners aktualisiert. Dabei soll bei der Übergabe der Option “start” das Kommando “updatedb” ausgeführt werden und

bei “status” zurückgemeldet werden, wieviele Stunden es her ist, dass dieses Update stattfand. Bei “stop” soll nur ausgegeben werden, dass nix passiert.

9. Wo würde man dieses Skript hinkopieren und wo würde man es am sinnvollsten in der Hierarchie einordnen? Weiterhin soll es nur in Runlevel 3 und 5 ausgeführt werden, wie erreicht man das?

10. Wie ändert man den “Default-Runlevel”, so dass man im netzwerkfähigen Multiusermodus ohne X startet? In welcher Datei geschieht dieses? Wie sorgt man dafür, dass man nur mit 2 Konsolen startet. (Für kleine Maschienen mit wenig Speicher sinnvoll.)

11. Wenn man beim Booten bestimmte Befehle ausführen will, ohne ein Startskript zu schreiben, was kann man dann im einfachsten Fall tun? (In welche Datei sollte die Befehlsabfolge kopiert werden?)

12. Was kann man machen, wenn die grafische Ausgabe nicht funktioniert (wildes Blinken des Bildschirmes, kurzes Aufflackern des typischen grauen Hintergrundes von XFree86, ... ) und man sich nicht einloggen kann?

13. Welche Möglichkeiten (grafisch bzw. kommandozeilenorientiert) bietet SuSE-Linux, um bestimmte Dienste einem Runlevel hinzuzufügen oder diese aus einem zu entfernen?

### 8.8.2 Prozesse

1. Wie bekomme ich heraus, welche Prozesse bestimmte Verzeichnisse noch benötigen (wenn sich z.B. das CD-Rom nicht ausmounten lässt)?

2. Welche Prozesse werden im Runlevel nach Beenden immer wieder neu gestartet und wo wird dieses konfiguriert?

3. Bestimmen Sie den Prozess, welcher die Datei */var/log/messages*, */tmp/.X11...* benutzt!

4. Sortieren Sie die Prozessliste nach Benutzername, Startzeit des Prozesses bzw. Speicherbelegung. Welches andere Kommando kommt dafür in Frage?

5. Welche Möglichkeiten gibt es z.B. den Dienst **dhcpd** zu beenden und sofort wieder neu zu starten? Wo muss bei SuSE was eingetragen sein, damit der **dhcpd** automatisch beim Booten der Maschine gestartet wird? In welche Kategorie von Programmen sind **dhcpd**, **named**, **smbd**, **proftpd**, ... einzuordnen?

6. Wie heisst das Kommando (unter Suse-Linux 9.X), um bestimmte Prozesse (bzw. deren Runlevelskripten) in den automatischen Bootvorgang (bzw. Wechsel der Runlevel) zu integrieren? Was macht dieses Programm genau?

# Kapitel 9

# Grafische Oberfläche

## 9.1 Einführung

Die Basisinstallation eines Linux-Rechners muss nicht zwingend eine grafische Benutzeroberfläche (GUI) vorsehen. Aus Sicht des Linux-Einsteigers macht das die Installation und die ersten Schritte an der Maschine nicht gerade leichter, da heute jeder mit der Handhabung einer Maus und dem Konzept grafischer Oberflächen vertraut ist. Wie man gesehen hat, stellt ein textbasiertes System in vielen Anwendungen einschliesslich der Erstinstallation kein wirkliches Hindernis dar.

Linux "an sich", der Kernel, weiss nichts von Grafik - im Gegensatz zu weitverbreiteten anderen Betriebssystemen, die die grafische Oberfläche fest in ihrem Kern integriert haben. Bei UNIXen ist das nicht so, vielmehr läuft das X Window System völlig unabhängig vom Kernel; es ist sogar völlig unabhängig davon entstanden.

Linux, eigentlich jeder Distribution, verwendet, das X-Window-System zur Darstellung der grafischen Benutzeroberfläche. Die Treiber für die verschiedenen Grafikkarten werden vom XFree86-Team entwickelt [1] und stellen neben dem Kernel die zweite wesentliche Hardware-Software-Schnittstelle dar. XFree86 ist eine freie Implementation des X-Window-Systems, welches auf allen Unix-ähnlichen Betriebssystemen bis hin zu Mac-OS-X läuft. Die "86" im Namen läßt vielleicht vermuten, dass diese Software nur auf 3/4/5 86er-Prozessoren lauffähig ist. Ursprünglich wurde XFree86 auch dafür entwickelt, mittlerweile ist XFree86 aber auch auf anderen Prozessor-Architekturen lauffähig.

Anfang der 80er-Jahre kam man am MIT ("Massachusettes Institut of Technologies") darauf, dass es ja ein bisschen schade ist, wenn man ein echtes Mehrbenutzer- und Multitasking-System hat, darauf aber nur mit einer Konsole zugreifen kann. Die naheliegende Idee: Mehrere Anwendungen in unterschiedlichen, bewegbaren Fenstern auf dem gleichen Schirm anzeigen. So entstand X als akademisches Projekt; der Quellcode wurde von vielen kommerziellen UNIX-Anbietern aufgegriffen, fortentwickelt und in ihre UNIXe integriert.

X11 selbst entstand im Jahre 1986 als Ergebnis der beiden Vorgängerprojekte V und W. Im Laufe der Entwicklung wurden die grundlegenden Protokolle weiterentwickelt, blieben aber immer zu den älteren Versionen kompatibel. 1992 erschien dann die erste Version von X11 für PCs. Das XFree86-Projekt selbst startete mit einem Referenzserver für die PC-Plattform, der von Thomas Röll (heute Xi Graphics) geschrieben wurde und dem Projekt zur Verfügung gestellt wurde.

Ungeachtet dessen ist es inzwischen ohne weiteres möglich, mit jedem Standard-Linux in den Genuss einer grafischen Benutzeroberfläche zu kommen. Um eine Grafikkarte zu benutzen, benötigt man - wie bei anderen Betriebssystemen auch - einen Treiber. Diese Treiber

[1] http://www.xfree86.org

stehen in Form sogenannter "X-Server" oder Hardwaretreibermodule zur Verfügung. Es werden im folgenden die Installation dieses Servers und einige ausgewählte Anwendungen vorgestellt.

## 9.2 X - Vorteile und Grenzen der Unix-GUI

Entsprechend der UNIX-Philosophie, realisiert X nicht alle Aufgaben integriert in ein einziges Programm ohne durchschaubare innere Struktur, sondern gliedert sich in einzelnen Komponenten:

- Der **X-Server** ist das Programm, das Tastatur- und Mauseingaben entgegennimmt und die Resultate auf dem Bildschirm anzeigt. Er stellt hierfür die passenden Gerätetreiber bereit. Die Events, wie die Benutzereingaben genannt werden, wertet der X-Server nicht selbst aus[2], sondern leitet diese an die betreffenden X-Clients weiter. Diese reichen wiederum ihre Resultate an den X-Server zurück.
- Die **X-Clients** sind praktisch alle Anwendungen (Browser, Textverarbeitung, Editor, PDF-Betrachter, ...), die die grafische Oberfläche benutzen wollen. Vom X-Server erhalten sie die Tastatur- und Mausevents, die sie betreffen, und melden ihm zurück, was auf dem Bildschirm erscheinen soll. Dazu wird das so genannte X-Protokoll verwendet.
- Der **Windowmanager** kümmert sich um die "Verwaltung" der Oberfläche, z.B. das Aussehen und die Funktionalität der Fensterrahmen und -menüs für die Anzeigen der X-Clients, Menüs auf dem Desktop, Minimieren und Maximieren der Fenster.

Das X-Protokoll realisiert durch die Trennung von Server und Client eine Schicht-Architektur, die einen wesentlichen Vorteil gegenüber anderen Systemen beinhaltet: Das X-Protokoll setzt auf dem Internet-Protokoll TCP/IP auf. Das bedeutet, dass Clients und Server auf unterschiedlichen Rechnern laufen können. In einem lokalen Netzwerk, in dem eine rechenintensive mathematische Software nur von einem einzigen Rechner verkraftet werden kann, weil alle anderen zu wenig Hauptspeicher haben, kann man sich einfach von seinem Arbeitsplatz aus dort einloggen, das Programm starten, und es erscheint auf dem eigenen Bildschirm, ohne dass man merkt, dass es auf einem ganz anderen Rechner läuft. Die einzelnen X-Clients müssen "nur" das X-Protokoll beherrschen, die Hardware kann ihnen relativ egal sein; das ist zwar heute allgemein so, aber zur Zeit der Einführung von X war diese Idee sehr fortschrittlich.

- Auf Systemen, die keine grafische Oberfläche brauchen (z.B. Webserver), kann sie einfach weggelassen werden, da sie ja vom Kernel unabhängig ist; das spart Speicher- und Prozessorressourcen.
- Das grafische System kann beliebig herauf- und heruntergefahren werden (z.B. zu Konfigurationszwecken), ohne das das System verändert oder angehalten werden muss.
- Die Textkonsolen arbeiten unabhängig von der grafischen Oberfläche; mit [Strg]-[Alt]-[F1] kann man auf die erste Textkonsole umschalten. Das ist z.B. sinnvoll, falls man schnell an der Systemkonfiguration was ändern will und sich somit auf einer Konsole als Systemadministrator anmeldet. Zum X-Bildschirm geht es standardmäßig mit [Alt]-[F7] zurück.

[2] bis auf bestimmte Ausnahmen, wie das Umschalten der Bildschirmauflösung und das "Abschiessen" (direkte Beenden ohne Umwege) des Servers

- Der Windowmanager ist beliebig wählbar; das macht das Erscheinungsbild individuell konfigurierbar.

Neben diesen schönen Eigenschaften merkt man X jedoch auch sein für Softwareverhältnisse hohes Alter an. Für viele moderne Anwendungen, wie Video- und Bildausgaben, ist X nicht sehr effizient und wird in der Darstellungsleistung von anderen Protokollen (wie z.B. Citrix-Metaframe für Windows) überholt.

- Das X-Protokoll kennt nur einfache Anweisungen wie: "Zeichne Linie von A nach B"; moderne Grafikkarten haben viele Möglichkeiten, um den Bildschirmaufbau durch die Hardware zu beschleunigen, diese können aufgrund dieser "Einfachheit" von X nicht genutzt werden und liegen brach. Solange X nicht durch ein neues Konzept abgelöst wird, wird sich daran auch nichts ändern.

- Die Programmierung von X ist durch seine Architektur recht umständlich. Mittlerweile gibt es allerdings so genannte GUI-Toolkits, die auf X aufsetzen und dem Programmierer das Leben recht einfach machen. Als "Altlast" gibt es aber viele alte Programme für X, von denen jedes eine andere Bedienphilosophie verfolgt und die zusammen einen kunterbunt aussehenden Zoo bilden. Dieses änderte sich zwar durch die Einführung von KDE und GNOME, jedoch verfolgen auch diese Desktop-Projekte durchaus eigene Strategien und Vorstellungen der Benutzerführung.

- X hat absolut nichts mit dem Drucksystem zu tun. Das ist zwar nicht unbedingt eine Designschwäche, hat aber zur Folge, dass es zwei paar Stiefel sind, z.B. eine Schriftart am Bildschirm anzuzeigen und sie später auszudrucken.

- X definiert lediglich ein Verfahren, um mehrere (Text-)fenster anzuzeigen. Von einer grafischen Oberfläche wird aber mehr verlangt. Den Anwender braucht dies aber wenig zu kümmern, denn im Zusammenspiel mit den modernen Desktop Environments von Linux liefert X eine Benutzeroberfläche, die kaum Wünsche offen lässt.

### 9.2.1 Erste Versuche mit X

Linux-Rechnern ist es egal, ob sie ihre grafische Ausgabe mit dem Protokoll X11 auf einem direkt angeschlossenen Bildschirm oder auf den eines anderen PCs schicken und ob sie die Tastatur- und Mauseingaben von direkt angeschlossenen Geräten oder denen an einem anderen PC beziehen. Hier soll kurz demonstriert werden, mit man mit wenigen Befehlen den Bildschirmdialog vom entfernten Linux-PC auf einen Windows- oder anderen Linux-Desktop darstellen kann.

Alle grafischen Applikationen unter Linux, die unter X11 eingesetzt werden, sind automatisch netzwerktransparent. Ohne besondere Massnahmen können grafisch orientierte Programme ihre Ausgaben an Maschinen im Netzwerk senden. Man kann nicht nur die grafische Ausgabe einzelner Applikationen exportieren, sondern ebenso komplette Desktops. Hierzu muss man dann einige Details vorbereiten, wenn noch keine Maschine im Netzwerk für diese Funktion eingerichtet ist.

- Für den Hintergrunddienst Displaymanager ist eines der Programme **xdm**, **gdm**, **kdm** oder **wdm** verantwortlich. Hierzu muss man feststellen, welcher der genannten Displaymanager auf der Maschine läuft, die den Desktop exportieren soll. Am einfachsten ermitteln man dieses durch **top** oder **ps aux**. Hier im Beispiel heißt die Maschine: *xserver.mydomain.local*

- Man suche im Dateisystem die entsprechende Konfigurationsdatei: *xdm-config*, *gdm.conf*, *kdmrc* oder *wdm-config*.
- Sodann editiere man diese Datei und suche nach einem Abschnitt mit der Bezeichnung »xdmcp«. Dann schaltet man »xdmcp« ein. Dieses ist meistens in den kommentierten Zeilen darüber beschrieben. Meistens muss hier nur statt eines »false« ein »true« eintragen werden.
- Anschliessend startet man den Displaymanager neu.

Wenn der Displaymanager KDM und die SuSE-Linux-Distribution am Start sind, sollte man alternativ die Datei */etc/sysconfig/displaymanager* editieren. In dieser Datei setzt man "DISPLAYMANAGER_REMOTE_ACCESS" auf "yes" und startet den Dienst mit `rcxdm restart` neu.


Abbildung 9.1: Grafischer Login einer anderen Maschine im Xnest

Nun kann man von einer anderen Maschine beispielsweise durch die Eingabe von
`X :1 -query xserver.mydomain.local` oder auch
`Xnest :1 -query xserver.mydomain.local`
sich den Desktop im Fenster darstellen lassen. Das erste Kommando öffnet einen komplett neuen grafischen Desktop auf der achten Konsole. Das ist bei den meisten Distributionen die erste freie Konsole nach dem Grafikbildschirm. Das zweite Kommando arbeitet im Fenstermodus. Es öffnet für die Grafikausgabe von *xserver* ein neues Fenster in Ihrem aktuellen grafischen Desktop. Dieses sieht man in Abbildung 9.2.1.

Unter Windows steht die Welt der grafischen X11-Desktops nicht automatisch zur Verfügung. Hierfür muss ein X-Server nachinstalliert werden. Hier stehen freie und kommerzielle Lösungen zur Auswahl.

Zum Fernadministrieren einer Linux-Maschine kann man nicht nur das Text-Terminal der Secure Shell verwenden, sondern auf Secure-Shell-Verbindungen den grafischen Output von Applikationen der Remote-Maschine auf den lokalen Desktop holen:

Der folgenden Dialog zeigt den Aufbau einer Verbindung über die Secure Shell ssh und den Start des SuSE-Konfigurationsprogramms YaST2.

```
dsucho@linux02:~ $ ssh -X -l root s04
Password:
Last login: Sun Jun 13 17:45:02 2004 from linux02.mydomain.local
Have a lot of fun...
s04:~ # yast2 &
s04:~ #
```

Diese kleine Befehlsabfolge kann man dazu nutzen, sich als Systemadministrator mit einer entfernten Linuxmaschine zu verbinden (im Beispiel: *s04*) und auf ihr das Konfigurationswerkzeug YaST2 zu starten. Als Ergebnis zeigt der lokaler Desktop die grafische Oberfläche dieses Tools als normales Fenster an. Man kann nun mit YaST2 von *s04* genauso arbeiten, als würde man direkt vor dieser Maschine sitzen. So kann man diesen Server *s04* administrieren, auch wenn er in einem gesicherten Serverraum steht.

### 9.2.2 Komprimiertes X

Für schmale Bandbreiten, wie sie im WAN-Bereich mit ISDN oder DSL realisiert werden, ist X wegen seines Datenaufkommens eher suboptimal. Hier sollten entweder andere Lösungen wie VNC eingesetzt werden. Möchte man trotzdem X verwenden, hilft vielleicht die folgende Erweiterung des Protokolls weiter.

LBX steht für Low Bandwidth X.11 und führt Caching sowie Kompression in das X-Protokoll ein. Es ist seit Ende 1996 mit Verabschiedung von X11R6.3 eine offizielle Protokollerweiterung. Hierfür läuft auf der Remote-Seite ein Proxyserver. Bevor Applikationen ihre Daten über das Netz geschicken, werden sie vom Proxy komprimiert und gecacht und dann erst an das lokale Display geschickt vor dem der Benutzer sitzt. Um diese Erweiterung nutzen zu können, muss sie sowohl in den X-Server kompiliert sein, als auch das Proxy-Binary in der verwendeten Linux-Distribution vorhanden sein.

### 9.2.3 Spezielle X-Server und Remote-Displays

**Überblick und Betrieb** VNC erlaubt den Zugriff auf einen gemeinsamen Desktop von verschiedenen Rechnern, die sogar unterschiedliche Betriebssysteme verwenden können. VNC löst das Problem, indem es den kompletten Desktop des einen Rechners in einem Fenster auf dem zweiten Rechner darstellt. Ein VNC-Server unter Linux ist im Grund ein doppelter Server. Zum einen stellt er einen vollwertigen X-Server dar - nach dem Start kann man ihn in der Regel über *localhost:1* ansprechen. Angezeigt wird dabei aber nichts: Der Server läuft ohne Ausgabe. Zu sehen bekommt man den, auf dem neuen X-Server laufenden Desktop erst, wenn man einen VNC-Client (vncviewer) startet: Diesem gegenüber tritt der Server dann als VNC-Server auf und überträgt den Desktop-Inhalt als Bildinformation.

Wenn man zunächst die Voreinstellungen des VNC-Servers testen möchte, rufe man einfach das Skript **vncserver** auf. Dieses übernimmt den eigentlichen Start des Servers **Xvnc** mit Standard-Parametern. Dabei beachte man die Ausgabe des Server-Start-Skripts: Dort wird eine Display-Nummer (:1, :2, ...) angegeben, die für den späteren Zugriff auf den Server benötigt wird. Ein VNC-Server unter Linux ist im Grund ein doppelter Server - Er einen stellt er einen vollwertigen X-Server dar und öffnet einen weiteren Port für den VNC-Client-Zugriff. Deshalb belegt er unter Linux zwei TCP-Ports. Angezeigt wird auf dem X11-Port jedoch nichts, die Grafikausgabe erfolgt über den VNC-Port. Hierfür gilt folgende Zuordnung:

| VNC-Display | TCP-Port | X11-Display | TCP-Port |
|---|---|---|---|
| :0 (1. Display, wenn kein X11 läuft) | 5900 | :0 (1. Display, Standard) | 6000 |
| :1 (1. Display, wenn X11 läuft) | 5901 | :1 (2. Display oder **Xnest**) | 6001 |
| :2 (weiterer VNC-Server) | 5902 | :2 (3. Display ...) | 6002 |
| ... | 5903 | ... | 6003 |

Tabelle 9.1: Standard-Ports für VNC- und XFree86-Server

Aus Sicherheitsgründen legt man ein Passwort für den Zugriff fest: Beim Erststart wird die Abfrage von **vncserver** selbst übernommen, später kann dazu das Kommando **vncpasswd** verwendet werden. Das VNC-Passwort ist komplett unabhängig von irgendwelchen System- oder Accountpasswörtern. Es wird verschlüsselt in der Datei */.vnc/passwd* abgelegt.

Für einen ersten Test greife man lokal auf die laufende VNC-Sitzung zu: Dazu setzt man an der Konsole das Kommando `vncviewer localhost:1` ab, wobei ":1" eventuell durch die richtige Display-Nummer zu ersetzen ist. Dann erscheint nach korrekter Eingabe des gesetzten VNC-Passworts der neue Desktop in einem eigenen Fenster. Der VNC-Server führt nach dem Start das Skript */.vnc/xstartup* aus; diese Datei sollte also an die eigenen Bedürfnisse angepasst werden. Wenn **vncviewer** ohne Argument gestartet wird, dann folgt die Frage nach dem Server zusammen mit der Display-Nummer.

Im Gegensatz zu X11 kann VNC mehrere Clients auf einen Server zulassen. Der einfachste Weg, um das zu erreichen, ist, den Server in der Betriebsart "always shared" zu starten. Diese ist beim Kommando-Aufruf mit anzugeben, da sonst automatisch ein anderer Client beim Start des neuen beendet werden würde:

```
vncserver -geometry 1200x1000 -depth 16 -alwaysshared
```

Dieser Aufruf realisiert eine Auflösung des Serverdesktops von *1200x1000* Bildpunkten, eine Farbtiefe von 16 bit und die Möglichkeit, dass mehrere Clients sich gleichzeitig verbinden können. Die gewählte Bildauflösung kann flexibel eingestellt werden. Wenn man auf dem Zielsystem mit dem **vncclient** zwei nebeneinanderliegende Bildschirme von *1280x1024*er Auflösung füllen will, kann der Server auch mit *-geometry 2400x1000* gestartet werden.

Der VNC-Client steht zusätzlich als Java-Applet zur Verfügung, dass im Browser-Fenster ablaufen kann, womit man sehr einfach von fast überall eine Desktop-Sitzung auf einer entfernten Maschine einleiten bzw. wiederaufnehmen kann. Die Netzwerkanforderungen des klassischen VNC sind recht hoch, es gibt jedoch Implementierungen, die mit Kompression arbeiten und damit auch über sehr geringe Bandbreiten (wie ISDN-Verbindungen) noch akzeptabel funktionieren. Sind die Reaktionszeiten nicht akzeptabel, kann durch Verwendung eines sehr kleinen Desktops (etwa 600x400 Punkte) und nur 8 Bit Farbtiefe die notwendige Bandbreite stark reduziert.

**Der ewige Desktop** Im Gegensatz zu einer X11-Sitzung, die mit dem Abschalten des Remote-Displays automatisch geschlossen wird, kann eine VNC-Sitzung ewig "weiterleben". Auch wenn sich alle VNC-Clients abgemeldet haben, bleibt der VNC-Server aktiv, und alle darunter gestarteten Programme laufen weiter. So kommt man bei der erneuten Verbindung zu dem Zustand zurück, indem der Client beendet wurde. Trotzdem sollten aus Sicherheitsgründen alle offenen Dateien gesichert werden, bevor der Client geschlossen wird.

**Anpassungen des Desktopinhaltes** Das Standard-Benutzerinterface, welches VNC anbietet ist nicht besonders spannend. Es startet sehr schnell, da einfach nur ein **xterm** und der alte Windowmanager **twm** geladen werden, die beide im Minimalumfang von XFree86 enthalten sind. Der VNC-Server führt nach dem Start das Skript */.vnc/xstartup* aus, welches nach der Installation:

```
#!/bin/sh

xrdb $HOME/.Xresources
xsetroot -solid grey
xterm -geometry 80x24+10+10 -ls -title "$VNCDESKTOP Desktop" &
twm &
```

die gezeigten Zeilen enthält. Diese Einträge können nun beliebig an die eigenen Bedürfnisse angepasst werden. Hier kann nun z.B. ein Start-Skript für eine der grafischen Desktops, wie GNOME oder KDE, oder einen Windowmanager, wie Windowmaker, eingetragen werden.

**Absicherung durch SSH-Tunnel** VNC kann zwar durch ein Passwort vor unberechtigtem Verbindungsaufbau geschützt werden, jedoch erfolgt die eigentliche Übertragung unverschlüsselt. Dies ist für die meisten Netze inakzeptabel, weshalb ein Tunnel verwendet werden sollte.

Hierzu wird der VNC-Server mit der zusätzlichen Option "-localhost" gestartet, welche verhindert, dass Verbindungen von anderen Rechnern aufgebaut werden dürfen. Damit nun ein Zugriff von einem entfernten Rechner erfolgen kann, wird der VNC-Port über die Secure Shell (**ssh**) getunnelt. Hierzu benötigt man die Port-Nummer des VNC-Servers, die sich leicht aus der Display-Nummer ermitteln läßt, indem man zu dieser 5900 addiert. (Display :1 ist entsprechend über Port 5901 erreichbar!) Wenn der VNC-Server *vncserv01* heisst, lautet das notwendige SSH-Kommando auf dem Client wie folgt:

```
ssh -L 5901:myserver:5901 myserver
vncviewer localhost:1
```

SSH fragt wie üblich nach dem Passwort. Anschliessend wird der **vnc-viewer** gestartet, der sich an das Tunnelende auf dem Client hängt. Der Viewer sieht dieses Tunnelende als lokal laufenden VNC-Server und spricht dazu Port 5901 an. Inzwischen geht das Ganze auch einfacher durch den Aufruf von:

```
vncviewer -via myserver :1
```

**Weitere Lösungen** Einen klassischeren Ansatz bietet das Programm **Xnest**. Es läuft unter X11 als Client in einem normalen Fenster, bietet selbst aber wieder Serverfähigkeiten an. Mit

```
Xnest :1 -query localhost
```

kann eine weitere XDMCP-Session auf der lokalen Maschine gestartet werden. Da bereits ein X-Server aktiv ist - nämlich der Desktop, an dem man bereits sitzt - muss der nächste freie Port gewählt werden. Dieses verläuft analog zu dem, wie es für VNC beschrieben wurde: Port 6000 ist der Standardport für den ersten X-Server, der mit ":0" bezeichnet wird. Die Addition der Display-Nummer ermittelt den jeweils gültigen Port, d.h. im beschriebenen Fall wird der zweite Server auf Port 6001 gestartet.

## 9.3 Desktop Environments

### 9.3.1 Überblick

Zum sinnvollen Betrieb einer grafischen Benutzeroberfläche unter Linux reicht es nicht aus, einen X-Server zu installieren. Neben den eigentlichen Anwendungen wird auch ein geeignetes Desktop Environment benötigt, welches u.a. einen Windowmanager mitbringt und weitere Features realisiert. Das X-Window-System selbst stellt nur ein geöffnetes Fenster für jedes Programm dar - ohne irgendwelche Rahmen, Buttons und Verzierungen.

Was macht ein modernes GUI (Graphical User Interface) aus? Funktionen wie Copy & Paste, Kontextmenüs, einheitliches Look & Feel, Drag & Drop, Desktop Panel - alles Dinge, die X selbst nicht bereitstellt. Dazu wurden die Desktop Environments oder Desktopumgebungen ins Leben gerufen. Desktopumgebungen bringen darüberhinaus eine ganze Ansammlung nützlicher Anwendungssoftware bereits mit (z.B. Web-Browser, Office-Paket, Spiele, Editor...), und bilden somit ein ganzes Software-Bündel. Zudem verwenden sie einheitliche Grafikbibliotheken (GUI-Toolkits), mit denen diese Anwendungen programmiert sind. Diese Toolkits machen das Aussehen für den Anwender gefälliger und nehmen dem Programmierer eine Menge Arbeit ab. Eine Desktopumgebung ist der Schlüssel zum Erfolg, um ein Betriebssystem einer breiten Masse an Anwendern zur Verfügung zu stellen, weil kaum ein Computeranwender gerne auf Komfort bei der Benutzung seines Computers verzichten will. Er möchte sich in erster Linie auf den Kern seiner Bedürfnisse konzentrieren (z.B. Briefe schreiben, im Internet surfen, Mails senden und empfangen). Solche Dinge mit einem Mausklick schnell und bequem zu machen, mit einem Klick Textpassagen aus dem Internet in die eigene Ausarbeitung zu kopieren, Icons auf der Oberfläche, die mit einem Klick eine häufig genutzte Anwendung starten, all das sind die Leistungungen eines Desktopsystems.

Die beiden wohl verbreitesten Desktopsysteme sind:

- KDE (Kool Desktop Environment), basierend auf dem GUI-Toolkit QT
- Gnome (GNU Network Object Model Environment), basierend auf dem GUI-Toolkit GTK

Beide Desktop Environments werden aber mit jeder größeren Distribution mitgeliefert und gleichen sich sehr im Funktionsumfang. Welches ist nun die geeignetere Oberfläche für einen bestimmten Einsatzzweck? Hier gilt der Grundsatz: ausprobieren! Die Geschmäcker sind verschieden und es gibt genug Alternativen, die nahezu die persönlichen Wünsche und Bedürfnisse eines Anwenders abdecken. Neben KDE und Gnome gibt es alternativ auch noch die Möglichkeit, einen “klassischen” Windowmanager wie FVWM oder IceWM zu verwenden - diese bieten zwar geringeren Komfort, schonen aber die Ressourcen, was z. B. auf einem Rechner, der überwiegend als Server eingesetzt wird, sinnvoll sein kann.

Jedes Standard-Linux-System verfügt inzwischen über eine große Anzahl verschiedener Windowmanager. Sie unterscheiden sich im Aussehen, in ihrer Handhabung und im Komfort. Einige dieser Windowmanager können auch mittels sogenannter “Themes” (verschiedene Erscheinungsbilder) während der Laufzeit im Aussehen angepasst werden. Man kann unter Linux deshalb verschiedene Windowmanager installieren und zwischen diesen wechseln. Ein Neustart der laufenden Anwendungen ist hierfür nicht notwendig.

### 9.3.2 Kurzdarstellung weiterer Benutzeroberflächen

Hier folgt ein Überblick über einige weniger verwendete Windowmanager, KDE und GNOME werden in eigenen Abschnitten behandelt.

#### 9.3.2.1 Fvwm(2)

Einer der Klassiker unter den reinen Windowmanagern, aber nicht mehr wirklich aktuell und zeitgemäss ist der **fvwm**. Die aktuelle Version 2 wird auch häufig als "fvwm2" bezeichnet. Ein beliebter Ableger des **fvwm** ist **fvwm95**, welcher einem Redmonder Betriebssystem nachempfunden ist, so dass ein Umstieg leichter fallen kann. Am besten löst man ihn durch den **icewm** ab.

#### 9.3.2.2 Windowmaker

Ein sehr beliebter Windowmanager ist der Windowmaker (Aufruf: **wmaker**), der dem Betriebssystem NeXTStep nachempfunden ist. Eine etwas ältere Adaption dieses Look&Feel bietet auch der " AfterStep".

#### 9.3.2.3 Enlightenment

Enlightenment ist ein weiterer Windowmanager. Ziel der Entwicklung war es, einen möglichst weitgehend konfigurierbaren Windowmanager zu schaffen; dies betrifft sowohl das Aussehen wie auch seine Bedienung.

#### 9.3.2.4 IceWM

Schmaler und schneller, aktuell weiterentwickelter Windowmanager mit einer ganzen Reihe von Themes.

### 9.3.3 GNOME

#### 9.3.3.1 Einführung in GNOME

GNOME ist die Abkürzung für GNU Network Object Model Environment. GNOME ist somit Teil des im Jahr 1984 begonnenen GNU-Projekts, das die Entwicklung eines komplett frei verfügbaren Unix-basierten Betriebssystems zum Ziel hat.

Dieser Abschnitt über GNOME basiert auf dem offiziellen Benutzerhandbuch zu GNOME, das Copyright dafür liegt bei Red Hat Software und David A. Wheeler. Das Benutzerhandbuch zu GNOME unterliegt der GPL. Die deutsche Übersetzung[3] sowie die englischsprachige Originalversion[4] findet man im Internet. Für dieses Skript wurden natürlich eingige Änderungen gegenüber dem ursprünglichen Text zu GNOME vorgenommen.

In diesem Abschnitt geht es um grundlegende Informationen, wie man die vielfältigen Funktionen und Möglichkeiten von GNOME nutzen kann. Auch wenn sich dieser Abschnitt in erster Linie an Benutzer wendet, die bislang noch nicht mit GNOME gearbeitet haben, sind die darin enthaltenen Informationen angesichts der raschen Weiterentwicklung von GNOME sicherlich auch für fortgeschrittene Anwender von Interesse.

**Grundlagen** Bei GNOME handelt es sich um eine komfortable grafische Benutzeroberfläche, mit deren Hilfe man als Nutzer den Computer auf einfache Weise konfigurieren und verwenden kann. GNOME besteht aus einem Panel (für das Starten von Anwendungen und Anzeigen von Statusmeldungen), einem Desktop (auf dem Daten und Anwendungen abgelegt werden können) und einer inzwischen sehr umfangreichen Auswahl (je nach Distribution) von mitgelieferten Hilfs- und Anwendungsprogrammen. Zudem stellt GNOME

[3] http://www.gnome.org/users-guide/de/index.html
[4] http://www.gnome.org/users-guide/index.html

bestimmte Vorgaben (Konventionen) für die Gestaltung weiterer Anwendungsprogramme bereit, um diese nahtlos in die Benutzeroberflche integrieren zu können und deren problemloses Zusammenwirken (z.B. für den Datenaustausch) zu gewährleisten. Sollten Sie schon Erfahrungen mit anderen Betriebssystemen gesammelt haben, werden Sie rasch mit der grafischen Benutzeroberfläche von GNOME zurechtkommen.

Bei GNOME handelt es sich vollständig um Open-Source-Software (d.h. freie Software), deren Quellcode frei verfügbar ist und die von Hunderten von Programmierern auf der ganzen Welt weiterentwickelt wird. Weitere Informationen zum GNOME-Projekt erfährt man wie immer im Internet[5].

GNOME bietet seinen Benutzern eine Reihe von Vorteilen. So macht es einem GNOME einfach, Anwendungen zu konfigurieren und zu benutzen, ohne dabei auf ein Nur-Text-Interface zurückgreifen zu müssen.

Zudem kann man GNOME individuell konfigurieren und somit das Erscheinungsbild und die Funktionen des Desktops nach eigenen Wünschen anpassen. Der in GNOME integrierte Session-Manager speichert die persönlichen Einstellungen der Applikationen und der beendeten Sitzung. Somit steht beim nächsten Starten der Benutzeroberfläche automatisch wieder der individuell gestalteter Desktop zur Verfügung. GNOME unterstützt schon viele Sprachen und ist zudem für die Übersetzung der Benutzeroberfläche in zusätzliche Sprachen vorbereitet. Zudem unterstützt GNOME mehrere Protokolle für Drag and Drop (Ziehen und Ablegen), um dadurch den Datenaustausch mit Anwendungen zu erleichtern, die nicht mit GNOME kompatibel sind.

Darüber hinaus bietet GNOME auch Entwicklern eine Reihe von Vorteilen, die indirekt den Benutzern zugute kommen. So müssen Entwickler keine kostspieligen Softwarelizenzen erwerben, um ihre kommerziellen Anwendungen GNOME-kompatibel zu machen. GNOME wird nicht exklusiv von einem bestimmten Anbieter zur Verfügung gestellt - im Gegenteil, keine der Komponenten von GNOME unterliegt den Rechten eines bestimmten Unternehmens oder Einschränkungen hinsichtlich Änderungen oder Weitergabe. Für das Entwickeln von GNOME-kompatiblen Anwendungen können zudem verschiedene Programmiersprachen verwendet werden. Denn GNOME beruht auf der "Common Object Request Broker Architecture"(CORBA), die das nahtlose Zusammenwirken verschiedener Software-Komponenten ermöglicht, unabhängig davon, welche Programmiersprache für die Implementierung verwendet oder welche Plattform gewählt wurde. Und nicht zuletzt kann GNOME in Verbindung mit einer ganzen Reihe von Unix-basierten Betriebssystemen verwendet werden, zu denen auch Linux zählt.

### 9.3.4 KDE

#### 9.3.4.1 Einführung in KDE

KDE ist die Abkürzung für K Desktop Environement. Das Projekt wurde 1996 von Matthias Ettrich gegründet. Das Ziel des KDE-Projektes ist es die Verbindung des UNIX-Betriebssystems mit dem Komfort einer Benutzeroberfläche. Oder anders gesagt: KDE will UNIX auf den Desktop bringen.

Dieser Abschnitt ist ein Auszug aus den Internetseiten des KDE-Projektes, genauer gesagt aus den FAQ[6], die unter der GNU Free Documentation License dort veröffentlicht sind. Das Ziel dieses Abschnittes ist die Vermittlung von grundsätzlichen Informationen über die Möglichkeiten der Nutzung von KDE.

[5]Die zentrale Website http://www.gnome.org

[6]http://www.kde.org/documentation/faq/index.html

**Grundlagen** Bei KDE handelt es sich um eine Benutzeroberfläche für alle Variationen von UNIX. Da die meisten KDE-Entwickler Linux benutzen, läuft KDE zuverlässig auf einer großen Anzahl von Systemen. Bei KDE handelt es sich nicht nur um einen Window Manager. KDE hat auch einen ausgefeilten Window Manager, den Kwin, ist aber in erster Linie eine ausgewachsene integrierte Desktop Umgebung. Darin enthalten sind ein Web-Browser, ein Dateimanager, ein Fenstermanager, ein Hilfesystem, ein Konfigurationssystem unzählbare Werkzeuge und Hilfsprogramme und eine weiter ansteigende Anzahl von Anwendungen.

KDE ist freie Software und steht unter der GNU General Public License (GPL). Alle KDE Bibliotheken sind unter der **LGPL (Lesser GPL)** verfügbar, die eine kommerzielle Softwareentwicklung für den KDE Desktop ermöglicht. Aber alle KDE Andwendungen sind unter der GPL lizensiert. KDE verwendet das "Qt(TM) C++ crossplatform toolkit" das auch unter der GPL veröffentlicht wurde.

Bei Qt(TM) handelt es sich um eine C basierte Klassenbibliothek zur Erstellung von Benutzereingaben. Es ist ein Produkt der Firma Trolltech, die letzte Version ist auf immer auf deren FTP-Server [7]. Außerdem ist Qt(TM) in den meisten aktuellen Linuxdistributionen enthalten.

KDE bietet dem Benutzer zur Systemkonfiguration das KDE-Kontrollcenter mit dem die Desktopumgebung leicht auf dessen Bedürfnisse angepasst werden kann, auch sprach- und länderspezifische Einstellungen können hier vorgenommen werden. Als Dateiverwaltungswerkzeug ist der Konqueror in KDE enthalten, dieser kann u.a auch als Webbrowser und FTP-Client verwendet werden.

Es gibt verschiedene Möglichkeiten unter KDE Software zu entwickeln. Zum einen ist es möglich, Freeware mit Open Source unter der GPL zu entwickeln. Auch für die Entwicklung von kommerzieller Software sind Möglichkeiten vorhanden. Mithilfe der KDE Bibliotheken können kommerzielle Anwendungen unter Offenlegung der Quellen (commercial and open source) erstellt werden, dafür und für die Entwicklung von freier Software ist die von der Firma Trolltech herausgegebene Version Qt(TM) free edition vorgesehen. Für kommerzielle Anwendungen ohne Offenlegung der Quellen kann die Qt(TM) Professionell Edition verwendet werden.

## 9.4 Aufgaben

### 9.4.1 XFree86 - Der Grafikserver

1. Wie schaltet man von der Textkonsole zur Grafik und wie kommt man da wieder weg?

2. Wie kann man die Auflösung des X-Servers umschalten?

3. Wie kann man den X-Server beenden und wo kann man einstellen, wenn dieses verhindert werden soll?

4. Auf welcher Konsole(nnummer) landet üblicherweise die grafische Ausgabe? Wie kann man das ändern (in welcher Datei)? Wo landen dann weitere geöffnete Grafikkonsolen? Wie kann man zwischen den einzelnen Sessions umschalten?

[7] ftpl.trolltech.com

5. Mit dem Programm **xhost** kann man Freigaben auf seinen X-Server erstellen. Man gebe seinen Server grosszügigst frei und lasse einen Nachbarn durch Ändern seiner Displayvariablen (wo steht die drin, wie zeigt man sie an, wie ändert man sie?) ein Programm auf seine Ausgabe umlenken.

6. Man experimentiere mit dem Programm **Xnest** herum, um unterschiedliche Anfragen (chooser, direkt, broadcast) zu realisieren.

7. Man probiere mal das Tool **xvidtune** aus und lasse sich die entsprechenden Modlines (Steuerzeilen für den Monitor) ausgeben!

8. Welche Datei muss man editieren, um bestimmten Hosts den Zugang zum Xserver zu gewähren, oder zu verwehren?

9. Was macht der Befehl `X:1 -bpp 16-query localhost`?

10. Was ist **Xnest**? Wie lautet die Syntax zum Aufbau einer Verbindung zu einem anderen System? Welche Ports werden hierfür benutzt?

11. Wofür ist die Datei *Xserver* zuständig? Wo befindet sich diese üblicherweise? Wie kann ich sie finden?

12. Welche Funktion hat der **xdm**? Welche Alternativen gibt es? Nenne zwei! Wo befinden sich die jeweiligen Konfigurationsdateien?

13. Weshalb macht **xvidtune** auf einem digitalen TFT-Display, bzw. für die Ausgabe auf einem Fernseher mit TV-Out wenig Sinn? (Weshalb wird man bei TFT-Displays üblicherweise nur eine Auflösung verwenden? Welche?)

14. Was leistet VNC? Worin liegt der Unterschied zum X11-Konzept?

### 9.4.2 Benutzeroberflächen

1. Welche grafischen Benutzeroberflächen stehen zur Verfügung? Worin unterscheiden sie sich?

2. Wo werden KDE bzw. Gnome üblicherweise installiert? Wo findet man die dazugehörigen Include-Dateien? Wo sind üblicherweise die Binärdateien des KDE/Gnome, bzw. die entsprechenden Bibliotheken abgelegt? (Wo muss man evtl. dafür sorgen, dass die Bibliotheken/Binaries gefunden werden?)

3. Welcher Windowmanager wird zusammen mit Gnome benutzt oder besser: installiert? Wie bekommt man das heraus?

4. Weshalb kann man nicht als normaler Benutzer den KDM (K-Displaymanager) konfigurieren? Wo liegt dessen Konfigurationsdatei?

5. Welches Programm zaubert die vielen (oder auch nicht so vielen) Icons auf den Desktop von KDE bzw. GNOME?

6. Welche Prozesse sind überlicherweise bei einer Gnome bzw. KDE-Session am Start? Welche Prozesse unter IceWM?

7. Wo müßte man die Language-Variable setzen, damit sie unter den verschiedenen grafischen Oberflächen und im Textmodus ausgewertet wird? Welche anderen Möglichkeiten bestehen, wenn man sich die **Xsession** bzw. den **gdm** ansieht?

# Kapitel 10

# Nachschlageteil

In diesem Teil des Skriptes werden alle (nach Meinung des Autors) mehr oder weniger wichtigen bzw. häufig benutzten Kommandos vorgestellt. Benutzer dieses Skriptes hatten vielfach nach einer solchen Liste gefragt, so dass im Folgenden versucht wird, auf diesen Wunsch einzugehen. Die Beschreibung des jeweiligen Befehls ist beileibe nicht vollständig, sondern soll nur einen ersten Anhaltspunkt liefern, wenn die Lösung zu einen Problem gesucht wird, oder unklar ist, wozu ein gegebenes Kommando gut sein soll.

## 10.1 Wichtige Programme in der Shell

### 10.1.1 Umsehen auf dem System

| Kommando | Aufgabe |
|---|---|
| cal | zeigt Kalender für den laufenden Monat, das ganze Jahr kann durch `cal 2002` dargestellt werden |
| date | meldet aktuelle Systemzeit und Systemdatum, der Sysadmin kann mit diesem Kommando die Systemzeit neu setzen |
| finger | aktuell angemeldete Benutzer mit Herkunftsshell und gerade getätigtem Kommando, netzwerkfähig aber kaum noch anwendbar |
| free | zeigt Belegung des Arbeitsspeichers incl. Cache an |
| id | zeigt die eigene UserID und Gruppenzugehörigkeit an |
| last | zeigt alle zuletzt an der Maschine angemeldeten Benutzer an |
| uname | meldet die Art des Unix-Systems (Linux), mit `uname -a` sieht man mehr, z.B. auch die Kernelversion |
| uptime | meldet die Systemauslastung (drei Werte: Durchschnitt einer, fünf und fünfzehn Minuten) mit vergangener Zeit seit dem Neustart der Maschine |
| w | analog wie lokales **finger** mit Informationen zur Uptime |
| who | ähnlich wie lokales **finger** |
| whoami | meldet den eigenen Accountnamen |

Hat man sich auf einem Linux-System über lokale Konsole oder über das Netzwerk mittels **ssh** angemeldet, kann man sich einige Standardinformationen über das System anzeigen lassen. Die Zahlendarstellung von **free** kann mit den Optionen "-b" auf Byte, "-k" auf KiloByte (das ist der Default) und "-m" auf MegaByte eingestellt werden.

```
dirk@hermes:~> free -m
             total       used       free     shared    buffers     cached
```

```
Mem:           629      535       93        0      174       97
-/+ buffers/cache:      264      365
Swap:          516       24      492
```

Die Kommandos **finger**, **w** und **who** liefern einen recht ähnlichen Output (in unterschiedlichem Format) über gerade am System angemeldete Benutzer.

```
dirk@dozent:~/text/lak> w
 12:50pm  up 34 days, 14:57,  1 user,  load average: 0.02, 0.04, 0.00
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
root     tty1     -                17Feb03 34days  0.43s  0.32s  -bash
dirk@randy2:~/text/lak> who
root     tty1     Feb 17 21:57
dirk@randy2:~/text/lak> finger
Login      Name                 Tty      Idle  Login Time   Where
root       root                 1          4d  Sep 17 21:57
```

Als die Welt des Internets noch in Ordnung war :-), konnte man sich mittels `finger@rechner name.domain` alle auf entfernten Rechnern eingeloggten Benutzer anzeigen lassen. Der Finger-Daemon war nicht, wie heute üblich, standardmäßig abgeschaltet. Das Kommando `last | less` wirft einen Blick in die Geschichte der angemeldeten Benutzer. Da diese Liste durchaus länger als ein Bildschirm werden kann, schickt man die Ausgabe von **last** gleich an einen Pager, in diesem Fall an **less**.

```
dirk@linux2:~/text/lak> last|less
root    pts/7      pd9e29571.dip.t- Sat Mar 22 16:45 - 22:29  (05:44)
dirk    pts/9      pd9e29571.dip.t- Sat Mar 22 16:27 - 22:29  (06:02)
dirk    pts/7      pd9e295cb.dip.t- Sat Mar 22 12:18 - 16:40  (04:22)
root    pts/7      java1.ruf.uni-fr Fri Mar 21 16:13 - 16:17  (00:04)
root    pts/7      java1.ruf.uni-fr Fri Mar 21 15:58 - 15:59  (00:00)
dirk    pts/7      dozent.lp2.ruf.u Thu Mar 20 10:49 - 13:00  (02:11)
dirk    :0                          Wed Mar 19 11:21   gone no logout
dirk    pts/0      suenne.stud.uni- Fri Mar 14 15:12 - 16:57  (01:44)
dirk    tty2                        Mon Mar 10 19:42 - 19:42  (00:00)
dirk    :0                          Mon Mar 10 14:51 - 13:32 (2+22:40)
dirk    pts/11     np9.ruf.uni-frei Wed Mar  5 19:30 - 19:30  (00:00)
dirk    pts/8      p5083ffa6.dip.t- Sun Mar  2 16:42 - 22:54  (06:11)
dirk    pts/0      acbd9832.ipt.aol Sat Feb 22 19:10 - 21:23  (02:13)
dirk    pts/0      acbd7ced.ipt.aol Thu Feb 20 00:04 - 02:16  (02:12)
root    pts/2      randy3.ruf.uni-f Wed Feb 19 10:51 - 11:08  (00:16)
dsuchod pts/2      randy2.rz.uni-fr Tue Feb 18 13:29 - 13:30  (00:00)
dirk    :0                          Mon Feb 17 21:59 - 14:51(20+16:52)
root    tty1                        Mon Feb 17 21:57   still logged in
reboot  system boot 2.4.19-4GB      Mon Feb 17 21:55        (34+14:59)
dirk    pts/14                      Mon Feb 17 20:51 - down   (01:02)
dirk    pts/13                      Mon Feb 17 19:36 - down   (02:18)
dsuchod pts/13     localhost        Mon Feb 17 19:31 - 19:33  (00:01)
dirk    pts/12                      Mon Feb 17 19:31 - down   (02:23)
[ ... ]
dirk    :0         console          Mon Jan  6 19:38 - 11:21 (7+15:43)
reboot   system boot 2.4.19-4GB     Mon Jan  6 19:37        (20+21:34)

wtmp begins Mon Jan  6 18:01:01 2003
```

Auch hier finden sich viele der Informationen wieder, welche die oben gezeigten Kommandos generieren. Die Daten werden in zwei Dateien hinterlegt: */var/run/utmp* liegen die kurzfristigen Informationen zu gerade angemeldeten Benutzern und in */var/log/wtmp* liegen die Daten, die z.B. **last** auswertet.

### 10.1.2 Shelleigene Standardkommandos

| Kommando | Aufgabe |
|---|---|
| echo | Ausgabe von Variableninhalten, z.B. `echo $PATH` meldet die aktuelle Liste des Suchpfades für ausführbare Programme |
| export | zeigt alle definierten Variablen an, die an Subshells weitergereicht werden, Untermenge von **set** |
| help | liefert eine Liste der in der aktuellen Shell zur Verfügung stehenden, eingebauten Kommandos |
| history | gibt eine Liste der am Prompt abgesetzten Kommandos an |
| pwd | meldet das aktuelle Arbeitsverzeichnis |
| set | zeigt alle in der aktuellen Shell definierten Variablen an |
| which | meldet die Lage von ausführbaren Programmen. Dazu wird im Bereich der in "PATH" angebenen Verzeichnisse gesucht |

Die shelleigenen Befehle machen die Kommandozeile erst zu einem mächtigen Werkzeug. Viele Skripten sind sogenannte Shellskripten. Diese sind "Stapel"[1] von nacheinander ausgeführten System- oder Shellkommandos. Hier liefern die shelleigenen Befehle den "Kitt", der häufig ein Systemkommando erst sinnvoll in ein Skript einbinden hilft.

### 10.1.3 Shelleigene Strukturen und Schleifen

Die Unix-Shell kann durchaus mit einer Interpreter-Programmiersprache, wie Perl oder Python verglichen werden. Sie kennt Variablen, Kontrollstrukturen und Schleifen, die bereits fest eingebaut sind.

| Kommando | Aufgabe |
|---|---|
| break | bricht eine Schleife ab |
| case | Switchanweisung in der Shell |
| continue | bricht nur den aktuellen Schleifendurchlauf ab |
| declare | erlaubt Nicht-String-Variable in der Shell zu definieren, z.B. `declare -i n` erzeugt die Ganzzahlvariable *n* |
| echo | Ausgabe von Textzeilen, z.B. `echo -n ‘‘Das ist ein Test!’’` Das "-n" schaltet das sonst obligatorische Newline ab |
| for | einfache Schleife zur Abarbeitung von Listen, z.B. `for i in * ; do echo $i; done` ist ein einfacher Ersatz für **ls** |
| else | Alternative beim Nichtzutreffen von **if** |
| exit | Rückgabewert einer Funktion oder des Shellskriptes |
| if | Bedingung, z.B. `if [ $i -gt 10 ]; then echo ‘‘i ist größer als 10’’; else ‘‘i ist kleiner oder gleich 10’’` |
| return | beendet eine Subroutine und kann einen Wert zwischen *0* und *255* zurückliefern |
| while | einfache Schleife, die auf eine Abbruchbedingung prüft. `while [ $i -lt 100 ]; do echo $i; i=$i+1; done` läuft solange die Variable *i* kleiner als *100* ist |

Mit eine WHILE-Schleife lässt sich beispielsweise regelmäßig überprüfen, ob beispielsweise die Netzwerkverbindung noch besteht. Falls es nicht der Fall ist, wird ein Kommando angestossen, welches sie wieder herstellt:

[1] Im Zusammenhang auch mit Windows oder DOS findet man die englische Bezeichnung "Batch". Ein Batch war mal ein Stapel von Lochkarten, die der Ausführung von einem oder mehreren Programmen nacheinander entsprachen.

```
while [ TRUE ] ; do ping -c 1 -W 1 10.8.4.254 || { echo "Mist Netzwerk weg!!" ;
rcnetwork restart; } ; sleep 20; done
```

### 10.1.4 Operationen auf Dateien

| Kommando | Aufgabe |
|---|---|
| cp | Kopieren von Dateien und Verzeichnissen (Quelle bleibt erhalten) |
| chmod | ändert Zugriffsrechte: Schreiben, Lesen, Ausführen bzw. Directorylisting auf Dateien oder Verzeichnissen |
| chown | ändert die Besitzer einer Datei (Benutzer und Gruppe) |
| file | ermittelt die Art einer Datei (Text, Bibliothek, ausführbares Programm, PNG-Bild, TeX, ...) |
| find | Suchen nach Dateien, z.B. alle Dateien unterhalb von */bin* anzeigen, die größer als 100 kByte sind: `find ‘‘/bin’’ -size +100k` |
| ln | Anlegen von Links (eine Art Verweis) auf Dateien, z.B. mit `ln -s quelle ziel` wird ein Verweis von “ziel” auf “quelle” angelegt |
| locate | Arbeitet erst nachdem mittels **updatedb** eine Hashtable mit allen Dateien erzeugt wurde. **updatedb** bedient sich dabei wiederum der Hilfe von **find** ... |
| ls | Anzeigen von Verzeichnisinhalten |
| mkdir | legt Verzeichnisse an |
| mv | verschiebt/umbenennt Dateien, wobei die Quelle anschließend nicht mehr verfügbar ist |
| rm | löscht Dateien oder Verzeichnisse |
| rmdir | löscht leere Verzeichnisse. Nichtleere Verzeichnisse lassen sich mittels `rm -r` entsorgen, wobei gerade als Sysadmin etwas Vorsicht geboten ist :-)) |
| touch | legt eine leere Datei an oder trägt die aktuelle Zeit auf eine Datei ein `touch datei` |

Das Kommando **ls** kennt die Option ”-h”, um eine besser lesbare Darstellung in Mega-, GigaByte-Angaben zu erhalten. Mit der Option ”-i” gibt **ls** die Inode-Nummer aus. So kann man beispielsweise feststellen, ob eine Datei ein Hardlink ist, wenn sie an zwei verschiedenen Stellen im Dateisystem[2] steht.

Mit dem Befehl **find** lassen sich direkt Aktionen verknüpfen. Ein Aufruf `find Einleitung_Netzwerk -name "*.texexec echo " {"{}"}"\;` sucht alle Dateien mit der Endung *tex* im Unterverzeichnis *Einleitung_Netzwerk* und gibt diese durch **echo** beispielsweise so aus:

```
\input{Einleitung_Netzwerk/0000-chapter.tex}
\input{Einleitung_Netzwerk/0100-diese-unterlagen.tex}
\input{Einleitung_Netzwerk/0200-layout.tex}
\input{Einleitung_Netzwerk/0300-begriffserklaerung.tex}
```

Wichtig ist dabei das Symikolon zu ”escapen”, da es sonst von der Shell verfrühstückt wird und **find** mit der recht unverständlichen Fehlermeldung nervt, dass es kein Argument zu ”-exec” finden würde. Ebenfalls sind die Anführungszeichen um den Suchausdruck hilfreich.

[2] nicht über Dateisystemgrenzen hinweg, d.h. nicht auf verschiedenen Partitionen

### 10.1.5 Verzeichnisstruktur und Filesysteme

| Kommando | Aufgabe |
|---|---|
| df | Diskfree zeigt freien Speicherplatz auf verschiedenen Datenträgern nach Mountpoints sortiert |
| du | Diskusage 'Speicherbelegung in einem (Unter-)verzeichnis |
| eject | Auswerfen (und Einziehen) von "Removable Devices", z.B. `eject /dev/hdd` wirft das Medium aus, welches am Secondary Slave IDE hängt |
| mount | Einhängen von neuen Partitionen, Netzwerkshares etc. in das Dateisystem der Maschine |
| mkfs | Formatieren einer Festplattenpartition |
| tune2fs | Einstellungen für EXT2/3-basierte Dateisysteme, z.B. Häufigkeit von erzwungenen Dateisystemüberprüfung beim Mounten |
| umount | Aushängen der mit **mount** eingehangene Dateisystemteile |

Die Kommandos **df** und **du** kennen analog zu **ls** die Option "-h" für die besser lesbare Zahlendarstellung. Das Formatkommando wird üblicherweise mit einer Extension aufgerufen, die bestimmt, mit welchem Dateisystem eine Partition eingerichtet werden soll, z.B.

```
linux02:~ # mkfs [Tab],[Tab]
mkfs           mkfs.ext2      mkfs.minix     mkfs.vfat
mkfs.bfs       mkfs.ext3      mkfs.msdos     mkfs.xfs
mkfs.cramfs    mkfs.jfs       mkfs.reiserfs
```

Die gezeigte Ausgabe erhält man durch die Complete-Funktion der Bash: Tippt man nach "mkfs" zwei Mal auf die Tabulator-Taste erscheint die Ausgabe des Beispiels.

### 10.1.6 Texteditoren

Texteditoren gibt es für Linux richtig viele. Einige davon sind etliche Jahre alt, andere sind erst in jüngster Zeit programmiert worden. Wenn es um die Administration von Systemen geht, benötigt man, gerade wenn man sich entfernt über das Netzwerk eingeloggt hat, Editoren, die in der Kommandozeile arbeiten. **joe** ist ein einfacher word-star-kompatibler Editor. Die Hilfe innerhalb des Editors kann mit [STRG]-KH ein und ausgeschaltet werden. Der Editor **pico** ist noch etwas rudimentärer; er wird mit dem textbasierten Mailprogramm **pine** mitgeliefert wird. Er eignet sich für Benutzer, die mit dem besagten Mailprogramm arbeiten. Seine Beliebtheit hat inzwischen für eine Weiterentwicklung gesorgt; unter dem Namen **nano** kann er aufgerufen werden (wenn die Weiterentwicklung auf dem System installiert ist. **vi** kann man als den Standard-Unix-Editor bezeichnen. Ihn findet man auf allen klassischen Unixsystemen vor. Nichtbenutzer sollten zumindest wissen, wie man ihn wieder verläßt :-)) (mit [Esc],dann [:] und dann [q].

Unter den diversen grafischen Benutzeroberflächen existieren jeweils etliche Editoren mit mehr oder weniger gelungener Benutzerschnittstelle. Mit **xemacs** und **gvim** stehen jeweils grafisch orientierte Vertreter ihren textorientierten Vorlagen gegenüber. KDE kennt daneben weitere, wie **kedit**, **kate**, **kwrite**. Ähnlich sieht es unter Gnome aus. Andere Editoren verwenden wiederum andere Grafikbibliotheken, so dass sich jeder Anwender den passenden Editor heraussuchen können sollte.

Die Bedienung der einzelnen Editoren weicht dabei stark voneinander ab, wobei die Ähnlichkeit jeweils zwischen der grafischen und der Textausgabe eines Editors, wie **vi** am größten sind. Etliche Editoren eignen sich gut für die Programmierung in den diversen Sprachen, der Shell oder LaTeX, da sie über ein gutes Syntax-Highlighting verfügen.

### 10.1.7 Operation auf Textdateien

In vielen Fällen will man für regelmäßig auftretende Aufgaben oder für die gleichzeitige Anwendung auf viele Dateien, nicht jedesmal einen interaktiven Editor öffnen müssen. Für diese Tasks bietet sich die Shellprogrammierung gemeinsam mit den nachstehend genannten Standard-Tools für die Bearbeitung von Textdateien an.

Viele Kommandos lassen sich dabei mittels Pipes ("|") hintereinanderschalten. Gute Beispiele für die verschiedenen Situationen und deren Beherrschung bieten z.B. die Runlevel-Skripte unterhalb von */etc/init.d* oder die Programme zur Steuerung der grafischen Benutzeroberfläche unterhalb von */etc/X11/xdm*.

| **Kommando** | **Aufgabe** |
|---|---|
| awk | Eigene Programmiersprache für zeilenorientierte Textbearbeitung, z.B. zum Auftrennen von Zeilen: `cat /etc/passwd\|awk -F : '{print $1 `` '' $2 `` '' $3 `` '' $4}'` |
| cat | Einfache Ausgabe von Textdateien ohne interaktive Steuerungsmöglichkeit, deshalb besser zur Shellprogrammierung geeignet |
| grep | Durchsuchen von Zeichenketten nach Mustern auch Regular Expressions |
| iconv | Einfacher Umkodierer. Das Umwandeln einer UTF-8-Datei (als Beispiel *utf8.txt*) in eine Datei (*iso-alt.txt*) nach dem alten ISO8859-1-Standard geschieht so: `iconv -f UTF-8 -t ISO-8859-1 -o iso-alt.txt utf8.txt`. Das Ergebnis kann man mit `file -i iso-alt.txt` einfach überprüfen. |
| less | Komfortabler Pager zum interaktiven Blättern innerhalb von Textdateien. Wird auch in Verbindung mit **man** eingesetzt. Verlassen deshalb in jedem Fall mit der Taste "q" (statt [CTRL]-C) |
| more | Einfacher Bruder von **less** mit weniger Möglichkeiten |
| recode | Umkodieren von Textdateien von einem Zeichensatz in einen anderen. Mit dem folgenden Beispiel wird die Datei *test.txt* vom alten ISO8859-1 Zeichensatz nach UTF8 umgewandelt: `recode latin1..utf-8 test.txt` |
| sed | Streameditor zum zeilenweisen Bearbeiten von Strings, z.B. zum Austauschen des Trennzeichens ":" gegen ein Leerzeichen: `cat /etc/passwd\|sed -e ``s,:, ,g''` |
| sort | Sortieren von Ausgaben eines Kommandos oder einer Datei |
| wc | WordCount zum Zählen von Zeilen und Wörtern |

### 10.1.8 Textsatzsystem und Darstellung

Dieses Skript[3] wurde mit Hilfe von LaTeX erstellt, welches einen ziemlich anderen Ansatz als bekannte Textverarbeitungsprogramme wählt. Der zu formatierende Text wird in einem einfachen Texteditor nach Wahl erstellt und mit Formatanweisungen versehen. Dieses Konzept kann vielleicht mit HTML verglichen werden. Anschliessend wird dieser "Rohtext" mit dem Kommando **latex** "kompiliert" und in ein Device-unabhängiges[4] Format umgewandelt. Die ".dvi"-Dateien kann man sich mit DVI-Viewern (z.B. bf xdvi, **kdvi**) ansehen. Die im Folgenden genannten Befelhe gelten natürlich für jede Datei unabhängig vom Textsatzsystem LaTeX.

[3] Es ist ein gemeinsames Projekt des Lehrstuhls für Kommunikationssysteme mit dem Mathematischen Institut in Göttingen, siehe hierzu auch: http://www.ks.uni-freiburg.de/projekte/las Das Skript ist frei verfügbar und kann von Interessenten mit **cvs** vom Server in Göttingen bezogen werden.

[4] device independent

| Kommando | Aufgabe |
|---|---|
| dvips | Umwandlung von DVI-Dateien nach Postskript |
| latex | Erweiterter Textsatzkompiler, der auf Donald Knuth' TeX basiert und DVI-Dateien generiert |
| makeindex | Generieren einer Indexdatei, welche nach erneutem Lauf in die von **latex** generierte DVI-Datei eingebunden wird |
| pdf2ps | Umwandlung von PDF zu Postskript, wie es zum Drucken notwendig werden kann |
| ps2pdf | Umwandlung von Postskript in PDF, nutzt jedoch bei weitem die Fähigkeiten von PDF nicht aus |
| xdvi | zum Betrachten einer *.dvi* Datei |
| gv | zum Betrachten einer *.ps* oder *.pdf* Datei |

Standardmäßig nehmen alle Programme immer die Inputdatei als Argument. Wenn nichts anderes spezifiziert wird, behält die Outputdatei den Originalnamen bei, nur die Endung wird entsprechend neu gesetzt, z.B.

```
dirk@hermes:~tex/lak> latex lak.tex
dirk@hermes:~tex/lak> makeindex lak.idx
dirk@hermes:~tex/lak> latex lak.tex
dirk@hermes:~tex/lak> dvips lak.dvi
dirk@hermes:~tex/lak> ps2pdf lak.ps
```

## 10.2 Systemprogramme

### 10.2.1 Prozess-Steuerung, Runlevel

| Kommando | Aufgabe |
|---|---|
| insserv | SuSE-spezifisch zum Ein- und Austragen von Runlevelskripten |
| free | zeigt freien Systemspeicher an. Da Caches für Plattenblöcke und der Verzeichnisstruktur angelegt werden, scheint dieser meistens lächerlich niedrig, was aber selten ein ernstes Problem darstellt |
| kill | Abschiessen von Prozessen über ihre PID |
| killall | Abschiessen von Prozessen nach ihrem Namen, z.B. `killall mozilla` |
| lsof | Anzeigen geöffneter Filehandles |
| netstat | Anzeigen offener Netzwerk- und Dateisystemsockets |
| nice | lässt ein Programm mit veränderter Priorität laufen |
| ps | Anzeigen der Liste aktuell laufender Prozesse |
| pstree | Anzeigen der Liste der von einem Prozess ausgehenden Kindprozesse |
| top | Interaktives Tool für das Prozessmanagement |

### 10.2.2 Dämonen

Dämonen sind im Hintergrund laufende Prozesse, die bestimmte Standardaufgaben des Systems übernehmen und überlicherweise nicht vom Administrator aus einer Shell heraus, sondern durch meistens gleichbenannte Runlevel-Skripte gestartet werden. Ihre dämonische Eigenschaft wird meistens durch das an den Protokollnamen angehängte *d* kenntlich gemacht. So wird aus dem Server für den Dienst "DHCP" der **dhcpd**. Das muss aber nicht immer so sein, wie das Beispiel "DNS" zeigt, wo der Dämon **named** heißt.

| Kommando | Aufgabe / Beschreibung | Konfiguration / Datenbereich |
|---|---|---|
| atftpd | Erweiterter Trivial-FTP-Server, der auch Anfragen von PXE-Clients bedienen kann | keine direkte, mittels */etc/sysconfig/atftpd* |
| dhcpd | Der Server für dasDynamic Host Configuration Protocol | */etc/dhcpd.conf*, */var/lib/dhcp/...* |
| httpd | Der Webserver. Meistens kommt hierbei der "Apache" zum Einsatz | */etc/httpd/...* , aber stark distributionsabhängig |
| inetd | Internet-Super-Daemon - stößt andere auf der jeweiligen Maschine eher seltener benutzte Dienste, wie **telnetd**, **in.ftpd**, **in.tftpd**, ... an und öffnet hierfür die entsprechenden Netzwerkports | */etc/inetd*, evtl. haben einzelne Dienste Datenverzeichnisse |
| mysqld | Der Server für die MySQL-Datenbank | */var/lib/mysql/...* |

Dämonen können für eine sehr weite Spanne von Diensten im Einsatz sein. Letzendlich gibts für alle Client-Server-Protokolle immer einen Client- und einen Serverprozess, so dieser auf einer bestimmten Plattform implementiert ist.

| Kommando | Aufgabe / Beschreibung | Konfiguration / Datenbereich |
|---|---|---|
| named | Serverprozess des Domain Name Systems | */etc/named.conf*, */var/(lib/)named/...* |
| nmbd | Server für das Windows-Name-System (WINS); kann mit dem DNS zusammenarbeiten und gehört zur Samba-Suite gemeinsam mit dem **smbd** | */etc/samba/...*, */var/lib/samba/...* |
| proftpd | Einer der möglichen FTP-Server | */etc/proftpd.conf*, stark distributionsabhängig |
| rpc.mountd | RPC-Dienst für das Network File System, arbeitet mit dem **rpc.nfsd** und evtl. einem geeigneten **lockd** zusammen | */etc/exports* |
| rpc.nfsd | RPC-Dienst für das Network File System | */etc/exports* |
| slapd | LDAP-Server | */etc/openldap/slapd.conf*, */var/lib/(open)ldap* |
| smbd | Samba-Server für SMB-Dienste in Windowsnetzwerken | */etc/samba/...*, */var/lib/samba/...* |
| wuftpd | Ein anderer FTP-Server | */etc/wuftpd.conf*, stark distributionsabhängig |
| xinetd | Die erweiterte Ausgabe des Internet-Super-Daemon mit besserer (Netzwerk-) Zugriffskontrolle | */etc/xinetd.conf*, analog zum **inetd** |

Die Tabelle kann nur eine unvollständige Auflistung einiger häufig zu findender Serverprogramme bieten. Die oben genannten findet man relativ häufig in der Prozessliste von Servermaschinen.

## 10.3 Nützliche Tools

### 10.3.1 Packprogramme

Möchte man viele zusammenhängende Dateien kopieren oder weitergeben, sollte man sie als eine einzige Datei behandeln können. Mit dem Kommando **tar** (Tape Archiver) können Dateien oder ganze Verzeichnisse zu einem großen Block zusammengefasst werden. Am Zielort können die in den Block eingepackten Dateien dann wieder ausgepackt werden. Auch die Verzeichnisstruktur und die Zugriffsrechte werden gespeichert, d.h. die Dateien liegen nach dem Auspacken in denselben Verzeichnissen mit ihren ursprünglichen Rechten. Mit **tar** wird noch kein Platz gespart. Die TAR-Datei lässt sich jedoch mit einem Komprimierungsprogramm in ihrem Umfang reduzieren.

| Kommando | Aufgabe / Beschreibung |
|---|---|
| compress | Altes Unix-Packprogramm mit historischen Kompressionsraten |
| bzip2 | Pack- und Entpackprogramm mit den derzeit besten Kompressionsraten für Standarddateien, kommandozeilenkompatibel zu **gzip** |
| gzip | Das Standardpack- und Entpackprogramm, welches auch kompatibel ist zu WinZIP und Konsorten |
| tar | Tape Archiver, Komprimiert nicht, aber darf in diesem Zusammenhang nicht fehlen, da die anderen Packprogramme zum Teil nicht auf Listen von Dateien und Verzeichnissen losgelassen werden können |
| unarj | Auspacken von ARJ-gepackten Dateien (ARJ war längere Zeit ein Standard unter DOS) |
| unzip | Auspacken von PKZIP-Archiven |
| unrar | Auspacken von RAR-gepackten Archiven (ein Standardprogramm unter Windows) |

Das Programm **gzip** sollte auf jedem UNIX-System vorhanden sein. Eine komprimierte Datei erhält die Endung ".gz"; ein komprimiertes Tarfile heißt also z.B. *tarfile.tar.gz*. Leistungsfähiger ist das ebenfalls weit verbreitete Programm **bzip2**. Mit **bzip2** komprimierte Dateien erhalten die Endung ".bz2". Die Dekomprimierung erfolgt entsprechend des Packers mit `gzip -d` oder **gunzip** bzw. mit `bzip2 -d`.

Die Optionen von **tar** müssen nicht wie üblich von einem "-" eingeleitet werden. **tar** kennt drei Modi: Einpacken, Auspacken und Inhalt anzeigen. Den Modus wählt man mit einer der folgenden Optionen aus: "c" (create - Erstellen/Einpacken eines tarfiles), "x" (extract - Auspacken eines tarfiles, wobei vorhandene gleichnamige Dateien überschrieben werden) und "t" (type - zeigt den Inhalt eines tarfiles an, ohne dieses auszupacken). **tar** wurde früher vor allem zur Datensicherung (Backup) auf Magnetbändern eingesetzt, woher der Name rührt. Will man das Archiv in einer Datei speichern oder eine Archivdatei auspacken, muss man die Option "f" verwenden und den Dateinamen dahinter angeben. Zusatzinformationen über die Aktivitäten des Programms liefert die Option "v". Der Tape Archiver kennt die Möglichkeit, das Packprogramm aus sich heraus zu invozieren, dieses geschieht mit den Schaltern "z" oder "i" für **gzip** und "j" für **bzip2**.

### 10.3.2 Zugriff auf (DOS)-Disketten

Will man Dateien oder Programme auf Disketten abspeichern und transportieren, kann man die "mtools" verwenden. Diese stellen einen eingeschränkten Befehlssatz zum Arbeiten mit MS-DOS-formatierten Disketten zur Verfügung, daher das vorangestellte "m" vor den Kommandonamen. Ein wichtiger Unterschied zu DOS besteht im Verzeichnisdelimiter: Da

der Backslash ("\") in der Unix-Shell eine besondere Bedeutung hat, wird er durch den "/" ersetzt. Ähnliches kennt man vielleicht auch von der Benutzung der Samba-Tools.

| MTool | Funktion | Beispiel |
|---|---|---|
| mcopy | Kopieren vom Laufwerk in ein UNIX-Verzeichnis oder umgekehrt | `mcopy lak.ps a:` kopiert die Datei lak.ps auf Diskette. |
| mdel | Löschen einer DOS-Datei | `mdel a:lak.pdf` löscht die Datei *lak.pdf* von der Diskette. |
| mdir | Anzeigen eines DOS-Verzeichnes | `mdir a:/psfiles` zeigt den Inhalt von *a:psfiles* an. |
| mmd | Anlegen eines DOS-Verzeichnisses auf der Diskette | `mmd a:/testdir` |
| mrd | Löschen eines (leeren) DOS-Verzeichnisses | `mrd a:/testdir` |
| mren | Umbenennen eines (existierenden) DOS-Verzeichnisses. | `mrd a:/testdir a:/dirtest` |
| mtype | Zeigt den Inhalt einer DOS-(Text-)Datei an. | `mtype a:/test.txt` |

### 10.3.3 Netzwerk

Da der Zugriff auf das Netzwerk durchaus als sensibel eingestuft wird, lassen sich die meisten der nachfolgenden Kommandos nur sinnvoll mit Root-Rechten ausführen.

| Kommando | Aufgabe / Beschreibung |
|---|---|
| arp | Programm zur Anzeige und Manipulation der Kernel-Arp-Tabelle |
| dig | Abfrage von Nameserverinformationen, Nachfolger von bf nslookup |
| ifconfig | Anzeigen der Konfiguration von Netzwerkinterfaces, genügt für die meisten anfallenden Standardaufgaben |
| ifuser | Über welches Interface verläßt ein Datenpaket an eine bestimmte IP-Adresse den Rechner |
| ip | Nachfolger von **ifconfig** mit erweiterten Fähigkeiten aber deutlich abweichender Bedienung. Übernimmt auch die Aufgaben von **route** |
| nslookup | Traditionelles Programm zum Test von Nameservern |
| ping | Einfaches Programm zum Test auf Erreichbarkeit von Rechnern |
| route | Manipulieren der nicht automatisch mit **ifconfig** angelegten Netzwerkrouten |
| tc | Traffic Control. Mit diesem Programm lassen sich Queues für das Management und die Priorisierung von Netzwerkdatenströmen einstellen |
| whois | Programm zur Anzeige des Personen- oder Firmennamens der zu einer IP oder Domäne gehört |

### 10.3.4 Netzwerküberwachung

| Kommando | Aufgabe / Beschreibung |
|---|---|
| arpwatch | Meldet neu im Netzwerk auftauchende MAC-Adressen. Eignet sich für die einfache Überwachung von Ethernets auf ungeschickt ins lokale Subnetz gehängte Maschinen |
| ethereal | Sehr komfortables Analysewerkzeug für Netzwerkdatenströme mit komfortabler grafischer Benutzeroberfläche |
| fping | Testen auf das Antwortverhalten von Rechnergruppen. Besonders für die Verwendung in Skripten geeignet |
| nmap | ist ein mächtiger Portscanner zum Aufspüren von offenen Netzwerkports auf (entfernten) Maschinen |
| netstat | Zeigt offene Netzwerkverbindungen und Ports an |
| ntop | Komfortables textorientiertes Traffic-Analyse-Tool mit weitgehenden Auswertungsmöglichkeiten |
| tcpdump | Einfacher Packet-Sniffer für die Kommandozeile, der die Paket-Header anzeigt |
| telnet | Einfache Überprüfung auf Erreichbarkeit von TCP-basierten Diensten, z.B. `telnet localhost 80` testet, ob der Webserver antwortet |

Beispiel: `tcpdump -i eth1 host 1.2.3.4 and not port ssh`
Mit einer ganzen Reihe von TCP-basierten Netzwerkdiensten, kann man sich mittels **telnet** interaktiv “unterhalten” und diese so auf Erreichbarkeit überprüfen. Im folgenden Beispiel wird die “Unterhaltung” mit einem Pop3-Server dargestellt.

```
dirk@hermes:/home/dirk/SharedFiles/tex/lak> telnet pop3.test.local 110
Trying 10.16.20.86...
Connected to pop3.test.local.
Escape character is '^]'.
+OK ready  <7889.1060520932@pop3.test.local>
user dirk
+OK Password required for dirk.
pass geheim
+OK dirk has 117 visible messages (1 hidden) in 2844396 octets.
quit
+OK Pop server at pop3.test.local signing off.
Connection closed by foreign host.
```

Dieses Beispiel zeigt die ”Unterhaltung” mit einem Pop3-Server.

## 10.4 Grafische Oberflächen

### 10.4.1 X-Programme

Es wird kaum möglich sein, alle Programme, die unter der grafischen Oberfläche des X11 laufen aufzuzählen. Deshalb folgen einige wichtige Standardprogramme, die nicht direkt nur einer der beiden wichtigen grafischen Benutzeroberflächen, wie KDE oder GNOME zuzuordnen sind.

| Kommando | Aufgabe / Beschreibung |
|---|---|
| Xnest | Eingebetteter X-Server, der als normales Fenster unter X11 erscheint |
| Xvnc | Spezieller X-Server für VNC |
| acroread | Der offizielle Adobe PDF-Viewer, nicht besonders schön, da Motif-GUI, aber als Plugin für die diversen Browser ebenfalls verfügbar |
| mozilla | Der Netscape-Nachfolger und Open-Source-Browser, der fast alles kann, was ein Standardbrowser hinbekommen sollte |
| gv | Ghostview - Standardbetrachter für Postskript-Dateien unter Linux |
| netscape(6) | Der alte bzw. neue "kommerzielle" Browser von Netscape. Netscape 6 setzt stark auf Mozilla auf und bastelt Zusatztools hinzu, auf die man zum Teil gut verzichten kann |
| openoffice | Die Office-Suite von Sun Microsystems, die sich durchaus mit der Redmonter Konkurrenz in etlichen Punkten messen kann. Das Programm sieht vom Look&Feel noch sehr windows-like aus, es verwendet Fenster in Fenstern; in der Form X11-unüblich |
| xdvi | Ein alter direkt X11-basierter DVI-File Viewer |

### 10.4.2 GNOME

| Kommando | Aufgabe / Beschreibung |
|---|---|
| abiword | Textverarbeitung des Gnome-Projektes, welche Dateien auch im TEX-Format speichern kann |
| dia | Einfaches Malprogramm für Ablaufdiagramme mit etlichen Postscript-Bildern für Netzwerkgrafiken etc. |
| evolution | Ein Mail- und Kalenderwerkzeug; ähnlich wie M$-Outlook |
| galeon | Auf der Gecko-Engine basierender, schlanker Webbrowser |
| gimp | Mächtiges Grafikwerkzeug mit Funktionalität ähnlich Photoshop |
| ggv | Der Gnome-Betrachter für Postskript-Dateien |
| gnumeric | Tabellenkalkulation des Gnome-Projektes |
| nautilus | Ein Filemanager |

### 10.4.3 KDE

Das KDE-Projekt (inzwischen in der dritten Version) widmet sich seit Ende der 90er Jahre der Entwicklung einer einheitlichen ansprechenden Benutzeroberfläche. Im folgenden werden einige wichtige Programme aufgelistet, wobei sie jedoch nur einen winzigen Ausschnitt der verfügbaren Werkzeuge repräsentieren.

| Kommando | Aufgabe / Beschreibung |
|---|---|
| k3b | CD-Erstell-, Kopier- und Brennprogramm |
| kaddressbook | Wie der Name schon sagt ... |
| kbear | Downloadmanager |
| kdvi | Betrachter für DVI-Files, die z.B. durch TEX erzeugt werden |
| kghostview | Ghostscript-Frontend zum Ansehen von Postskript und PDF-Dokumenten |
| kmail | Standard-Email-Programm unter KDE, welches alle wichtigen Funktionen einer Mailapplikation beherrscht |
| koncd | CD-Erstell-, Kopier- und Brennprogramm |
| konqueror | mächtiger Datei- und Webbrowser mit vielen eingebauten Zusatzfunktionen |
| kspread | Tabellenkalkulation der K-Office-Suite |
| kword | Textverarbeitungsprogramm aus der K-Office-Suite |
| kwrite | KDE-Texteditor |
| pixie | Bildbetrachter mit Thumbnail-Funktion etc. |

## 10.5 Software

### 10.5.1 Installation und Management

Das Standardporgramm zum Paketmanagement ist üblicherweise der RedHat-Packet-Manager **rpm**. Es gibt andere Ansätze, die bei Debian mit **dpkg** oder **aptget** verfolgt werden. Gentoo hat ebenfalls eigene Tools für diese Tasks am Start.

### 10.5.2 Entwicklung

Zur Programmentwicklung unter Linux gehören eine ganze Reihe von Werkzeugen. Die Editoren wurden gesondert abgehandelt. Es gibt integrierte Entwicklungsumgebungen, z.B. **kdevelop**, die viele der genannten Programme unter einer einheitlichen Oberfläche zusammenfassen.

| Kommando | Aufgabe / Beschreibung |
|---|---|
| autoconf | Zusammen mit **automake**, um Software zur Kompilation auf einer Zielplattform geeignet einzurichten |
| automake | Generieren von Makefiles für die Zielplattform. Wenn man `./configure --prefix=/usr --nextoption ...` aufruft, steckt genau dieses dahinter |
| cvs | Concurrent Versions System (CVS) ist ein Tool zur Versionskontrolle |
| diff | Erstellt eine Differenzdatei von zwei unterschiedlichen Dateien oder Dateibäumen |
| gcc | Der GNU C-Compiler. Eines der wichtigsten Programme überhaupt |
| ld | Der Linker (dynamisches Binden von Programmen und Bibliotheken) |
| ldconfig | Einrichten des schnellen Hashes */etc/ld.so.cache* für das schnelle Auffinden dynamisch gelinkter Bibliotheken. Es wird normalerweise nach dem Systemstart automatisch aufgerufen, sollte aber auch nach der Installation neuer Bibliotheken angestoßen werden |
| ltrace | Verfolgen von Bibliotheksaufrufen zum Debugging von Programmen |
| make | Aufwändige spezielle Skriptsprache zur Analyse der Makefiles um bequem ganze Softwarepakete zu bauen |
| nm | zeigt in einem Programm oder einer Bibliothek enthaltene Funktionen an |
| patch | fügt durch **diff** erstellte Differenzdateien in eine Datei oder ein Verzeichnis ein |
| strace | Verfolgen von Systemaufrufen zum Debugging von Programmen, analog zu **ltrace** |
| strings | zeigt in einem Programm oder einer Bibliothek enthaltene Textstrings an |

# Index