Einführung in Linux

Michael Redinger

11. Oktober 2004

Zusammenfassung

Dieses Buch führt in das Betriebssystem Linux ein; Es bezieht sich vor allem auf das System in den Benutzerräumen des ZID der LFU Innsbruck. Die meisten Teile sollten sich aber problemlos auf andere Installationen und auch auf andere Unix Systeme übertragen lassen.

Wenn Sie diesen Text gelesen (besser: durchgearbeitet) haben, sollten Sie in der Lage sein, viele der häufig anfallenden Fragestellungen auf Anhieb zu lösen bzw. eigenständig weiterzuarbeiten und Problemlösungen selbst zu finden.

Wenn Sie Verbesserungsvorschläge, Kritik oder Lob äußern wollen, wenden Sie sich bitte an mich (Michael.Redinger@uibk.ac.at).

Inhaltsverzeichnis

1 Über dieses Dokument

Computer–Benutzer sind Masochisten. Sie trinken den Kakao, durch den sie gezogen werden.

(anonymer Systemadministrator)

Dieser Text bezieht sich auf die Installation von Linux an der LFU Innsbruck. Er soll helfen, erste Einblicke in Linux zu bekommen. Vielleicht gelingt es ja, den einen oder anderen für Linux zu begeistern und ihm dessen Konzepte näherzubringen[1]. Er bietet eine Einführung. Mehr nicht. Im Allgemeinen werde ich die Programme und Eigenschaften nur kurz vorstellen. Es geht darum, eine Einführung in Linux zu geben. Es soll versucht werden, den Benutzern gerade soviel zu zeigen, dass sie sich gut zurechtfinden und eigenständig mehr lernen können. Dazu gibt es unter Linux zahlreiche Möglichkeiten, die ich weiter unten noch nennen werde. Außerdem gibt es natürlich auch zahlreiche weiterführende und spezielle Aspekte behandelnde Bücher.

Ein wesentliches „Problem“ ist sicher die Dynamik von Linux — ständig kommen neue und bessere Programme heraus. Durch diese ständige und schnelle Weiterentwicklung, die in der Computer–Welt wohl einzigartig ist, kann ein Text immer nur ein Momentaneindruck des Systems zu sein. Ich werde mich aber bemühen, den Großteil so zu gestalten, dass er auch noch in fünf Jahren gültig ist.

Dieser Text soll aber keineswegs als alleinige Informationsquelle verstanden werden. Lesen Sie doch ein oder zwei Bücher über Linux. Das hilft ihnen, das System weit besser zu verstehen, als es mit diesem kurzen Text möglich ist. Außerdem schont es ihre Nerven. Es ist viel leichter, produktiv mit einem System zu arbeiten, das man kennt und versteht.

Es wird vor allem die *Benutzung* von Linux behandelt. Themen wie Systemadministration oder Installation werden nicht behandelt. Ich gehe davon aus, dass Sie ein ganz normaler Benutzer sind, der sich mit Aufgaben der Systemadministration nicht befassen muss. Für Personen, die zuhause ihr eigenes Linux System installiert haben und dadurch auch als *root* arbeiten, sind noch viele andere Befehle und Dateien relevant (wenn Sie nicht wissen wovon ich jetzt rede — gut, denn dann sind Sie hier genau richtig…).

Hier wird vor allem die Arbeit mit der Kommandozeile besprochen. Vieles kann zwar auch über grafische Oberflächen erledigt werden — aber dort ist es kaum möglich, die zugrunde liegenden Mechanismen, die Idee hinter Unix/Linux zu verstehen. Wenn Sie einmal mit der Kommandozeile arbeiten gelernt haben, werden Sie mit der Bedienung grafischen Programmoberflächen wohl keine Probleme mehr haben. Viele Aufgaben können auch nur auf der Kommandozeile effizient erledigt und automatisiert werden. Aber auch eine kleine Anzahl grafischer Anwendungen soll besprochen werden, um

[1] Im Folgenden verwende ich sehr oft die männliche Form. Das ist keinesfalls diskriminierend gemeint. Es ist gebräuchlich, vom „Benutzer“ zu sprechen; dieser Audruck beinhaltet auch weibliche Benutzer. Ich werde es in diesem Sinn verwenden.

Ihnen einen kurzen Einblick in die vorhandenen Programme zu ermöglichen. Den Anfang macht eine Einführung in die Bedienung der grafischen Benutzeroberfläche KDE — ich werde mich auf diese beschränken und GNOME nicht besprechen.

Beachten Sie, dass die hier gezeigten Screenshots sich von der Darstellung auf Ihrem System unterscheiden können; das sollte jedoch kein größeres Problem darstellen.

Im Gegensatz zu früheren Versionen dieses Dokuments werde ich (im Bereich der grafischen Oberflächen) die deutschen Bezeichnungen verwenden, da sich diese bei den diversen Distributionen im deutschen Sprachraum durchgesetzt haben. Ich will nicht versuchen, dem weiter (ohne Aussicht auf Erfolg) zu widerstehen.
Speziell im Bereich der Kommandozeile und der allgemeinen Nomenklatur werde ich aber weiter die englischen Begriffe bevorzugen. Dies scheint aufgrund der Zielsetzung dieses Textes sinnvoll: wenn Sie sich näher mit Linux beschäftigen, werden Sie meist auf englische Texte stoßen. Es scheint wenig sinnvoll, hier deutsche Begriffe zu etablieren, wo dies international nicht üblich ist.

Am Ende jedes Abschnittes finden Sie Aufgaben. Diese beziehen sich auf den vorhergehenden Text, ein Teil wurde jedoch noch nicht behandelt (etwa verschiedene Programmoptionen). Diese Aufgabe sollten Sie unter Verwendung der Hilfeseiten lösen. Am Ende des Textes finden Sie die Antworten zu den Aufgabestellungen.

Sollten Sie diesen Text verwenden, um den Umgang mit einem anderen Unix System zu erlernen, sind wahrscheinlich jene Abschnitte, welche die grafische Oberfläche betreffen, nur wenig relevant. Das trifft auch zu, wenn Sie eine andere Oberfläche anstelle von KDE verwenden.
Unterscheiden sich die Syntax oder die Optionen der beschriebenen Programme von denen anderer Unix Derivate, wird dies bei der ersten Verwendung erwähnt.
Einige der besprochenen Programme sind nicht auf allen Unix System vorhanden. Auch darauf wird entsprechend hingewiesen.
Die hier verwendete Shell Syntax gilt für die `bash` und in weiten Teilen auch für die `ksh`. Andere Shells (etwa die Derivate der C Shell) werden nicht behandelt.

1.1 Typographische Konventionen

- Neue Begriffe werden bei ihrer ersten Verwendung *kursiv* gesetzt.
- Für Programmnamen, Verzeichnisse etc. wird eine `Schrift mit fester Buchstabenweite` verwendet, ebenso für die Ausgabe von Programmen.
- Solle ein Befehl eingegeben werden, so wird er **`fett und mit fester Buchstabenweite`** geschrieben. Soweit sinnvoll und den Textfluss nicht störend erfolgt dies in

  `einem eigenen Absatz`

- Menüeinträge, Programmnamen (etwa KDE Kontrollleiste) etc. werden in einer serifenlosen Schrift gesetzt.
- Werden in einem Befehl oder in einer Ausgabe Platzhalter für die tatsächliche Ein– oder Ausgabe verwendet, so erfolgt dies so:

⟨**der zu ersetzende Begriff**⟩

- Optionale Teile werden in

  $\left(\textbf{große runde Klammern eingeschlossen}\right)$

- Wenn eine Taste oder eine Tastenkomination zu drücken ist, so wird sie durch einen Rahmen gekennzeichnet: Taste

2 Einleitung

In the beginning there was data. The data was without form and null, and darkness was upon the face of the console; and the Spirit of IBM was moving over the face of the market. And DEC said, „Let there be registers“; and there were registers. And DEC saw that they carried; and DEC separated the data from the instructions. DEC called the data Stack, and the instructions they called Code. And there was evening and there was morning, one interrupt.

(Rico Tudor, „The Story of Creation or, The Myth of Urk“)

2.1 Betriebssysteme

Computer sind eigentlich nur Maschinen, die besonders gut mit den Zahlen 0 und 1 rechnen können. Alle Befehle werden aus Serien von 0 und 1 gebildet. Das ist zwar sehr effizient. Damit man aber mit dem Computer arbeiten kann, wird ein *Betriebssystem* benötigt. Dieses „vermittelt“ zwischen der Maschinensprache und dem Benutzer und abstrahiert diese Anweisungen.

Das Betriebssystem ist auch für die Ansteuerung der Komponenten und Geräte verantworlich — von der Festplatte über Tastatur und Bildschirm bis hin zu Druckern und Modems.

Eines dieser Betriebssysteme ist Linux. Es ist eine Unix Variante für PCs. Andere Beispiele für Betriebssysteme sind Windows 2000, Windows XP oder Mac OS X.

Neben dem Betriebssystem–Kern, dem *Kernel*, gibt es eine Reihe von Programmen, die zu einem funktionierenden System gehören. Beim Hochfahren von Linux werden viele verschiedene Programme ausgeführt. Diese binden Festplatten ein, zeichnen Systemmeldungen auf, starten Netzwerkdienste, ermöglichen den Benutzerzugriff, initialisieren die Tastaturbelegung etc. Schließlich kann eine grafische Oberfläche gestartet werden, Sie können mit Fenstersystemen arbeiten, auf der Kommandozeile Befehle eingeben oder von einem anderen Computer aus auf Ihr System zugreifen. Die Summe dieser Komponenten wird oft im weiteren Sinne als Betriebssystem bezeichnet.

2.2 Unix

Linux trat nicht in einen leeren Raum, es war keine Neuerfindung. Vielmehr ist es die Fortsetzung einer langen Reihe von Unix–Betriebssystemen.

Im Jahr 1969 arbeiteten mehrere Firmen an einem neuen Betriebssystem namens MULTICS , das auf völlig neuen Ideen und Konzepten beruhte (und bei vielen sogenannten Neuerungen kann man immer noch beobachten, dass die Konzepte dafür bereits in MULTICS existiert haben). Durch den Ausstieg der Bell Laboratories starb das Projekt jedoch. Einer der Programmierer, Ken Thompson, führte es jedoch privat fort und entwarf für das Spiel *Space Travel* eine rudimentäre Programmierumgebung. Ihr gab

er – in Anspielung auf den Namen MULTICS — den Namen Unics, woraus im Laufe der Zeit Unix wurde.

1971 schrieb er das Betriebssystem gemeinsam mit Dennis Ritchie in der von ihm geschaffene Programmiersprache C neu. In diesem Jahr gab es rund 40 Installationen von Unix.

Das Betriebssystem fand immer mehr Echo. Im Jahr 1974 erschien der erste Artikel über Unix. Im Laufe der Zeit teilte sich aber die Entwicklung von Unix, viele Firmen schufen eigene Unix–Derivate, z.B. Sun (Solaris), IBM (AIX), HP (HP/UX) oder SGI (Irix). Alle diese teilen aber gemeinsame Eigenschaften:

- hierarchisches Dateisystem.
- identische Schnittstellen für Daten–, Geräte– und Interprozess–Ein– und Ausgabe.
- Hintergrundprozesse.
- Realisierung synchroner und asynchroner Vorgänge.
- Filtertechnik.
- (Relativ) einheitliche Werkzeuge/Programme.
- Hohes Maß an Portabilität.

Soviele Gemeinsamkeiten es auch gibt: Die Unix Welt ist nachwievor hoffnungslos pluralistisch — auch wenn es in den letzten Jahren immer mehr Bestrebungen gibt, die Systeme wieder enger zusammenzuführen (und auch hier spielt Linux eine maßgebliche Rolle).

Die Aufspaltung begann 1977. Damals gab es das „Ur–Unix“ von AT&T Bell Laboratories. An der University of California in Berkeley begann man jedoch, Unix auf unterschiedliche Weise zu erweitern. Damit begann die Aufspaltung in die zwei Hauptzweige von Unix, die auch heute noch — wenn auch in abgeschwächter Form — erkennbar sind: System V, das sich aus der AT&T Bell Laboratories Version entwickelte, und BSD. Einer der Unterschiede, der für den Benutzer heute am augenfälligsten ist, ist die Verwendung von `ps`: System V verwendet `ps -ef`, BSD `ps aux`.

1988 schien wieder einiges auf eine Annäherung hinzudeuten — das neueste Release von System V brachte in vielen Bereichen eine Annäherung an BSD. Fast gleichzeitig kam es aber zu einer neuerlichen Spaltung: Sun und AT&T beschlossen, System V gemeinsam weiterzuentwickeln. Als Antwort darauf gründeten Firmen wie IBM , DEC und Hewlett–Packard die „Open Software Foundation“. Das Ergebnis war wiederum ein neuer Standard, OSF/1 (und — was heute noch wichtig ist — Motif).

Die Jahre danach waren wiederum vom Bemühen geprägt, einen gemeinsamen Standard zu finden. Das wichtigste Ergebnis daraus ist POSIX, ein Versuch, zumindest für die wichtigsten Systemeigenschaften gemeinsame Richtlinien festzulegen.

Die wichtigsten Unix Derivate sind:

- IRIX
  IRIX von SGI ist ein stark System V orientiertes System. Die früher enthaltenen BSD–Features wurden im Laufe der Zeit aufgegeben. Bekannt ist IRIX vor allem aufgrund seiner grafisch sehr aufwendigen und professionellen Oberfläche und Anwendungen. In letzter Zeit hat dieses System (nicht zuletzt wegen der neuen 3D Fähigkeiten Intel kompatibler PCs und der mangelhaften Weiterentwicklung des Systems) aber an Einfluss verloren.

- Solaris
  Solaris ist das System V Unix Derivat von SUN, das zur Zeit nach Linux am meisten verwendete Unix System. Es enthält zudem einige BSD Features. Solaris kann auch auf IBM kompatiblen (auf Intel kompatiblen Mikroprozessoren basierenden) Rechnern eingesetzt werden.

- HP/UX
  HP/UX von Hewlett–Packard ist grundsätzlich ein System V System. Es wurden jedoch sehr viele Features von BSD und auch von OSF/1 integriert.

- AIX
  AIX von IBM ist ein System V basierendes Betriebssystem. Es unterstützt aber auch zahlreiche Funktionalitäten von BSD und OSF/1. AIX ist dafür bekannt, ein sehr ausgefeiltes Sicherheitssystem zu besitzen und groß Stabilität aufzuweisen und wird daher (neben VMS von Digital) oft in Banken und sicherheitsrelevanten Einrichtungen eingesetzt. Diese Spezialisierung geht aber oft auf Kosten der Bedienbarkeit, die sich teilweise stark von anderen Unix Derivaten unterscheidet.

- Linux
  Linux ist von seinem Ursprung her stark BSD orientiert. Im Laufe der Zeit kamen aber auch zahlreiche System V Features hinzu.
  1991 begann der finnische Student Linus Benedict Torvalds damit, ein einfaches Betriebssystem zu entwickelt, um sein Verständnis für den 80386 Prozessor zu vertiefen. Nach einiger Zeit entwickelte sich daraus Linux. Im Herbst 1991 wurde es erstmals in einer Newsgroup in der Version 0.1 vorgestellt. Es gab zahlreiche Reaktionen darauf, und immer mehr Personen arbeiteten mit Linux. Es entstand eine große Eigendynamik, und bald waren zur Standardsoftware eines Unix–Betriebssystems noch zahlreiche andere Programme verfügbar. Es dauerte aber noch bis 1994, bis die Version 1.0 von Linux vorgestellt wurde. Und nur 2 Jahre später war die Version 2.0 erreicht, mit der Linux endgültig zu einem *peer among peers* der Unix–Betriebssysteme wurde.
  Heute ist Linux das am häufigsten eingesetzte Unix System.

- *BSD
  Neben Linux gibt es eine Reihe weiterer Unix Systeme, die (meist) auf Intel kompatiblen Computern laufen. Neben FreeBSD sind vor allen NetBSD und OpenBSD erwähnenswert, das als die sicherste Unix Varianten gilt. Ihr Nachteil ist, dass bei weitem nicht so viele Hardware–Treiber und Programme wie für Linux existieren.

2.3 Was ist Linux?

Immer wieder gibt es Probleme bei der Definition, was Linux eigentlich ist. Bei kommerziellen Produkten ist es einfach, da weiß man genau, was nun zum entsprechenden Betriebssystem gehört — nicht so bei Linux. Linux ist eigentlich nur der Betriebssystemkern. Das ist sicher der wichtigste Teil eines Betriebssystems. Aber mit dem alleine hat man noch kein funktionierendes System. Dazu gehören noch viele weitere Programme — angefangen von der Shell, dem Compiler, der X Oberfläche bis hin zu den Benutzer–Applikationen.

Linux gibt es in verschiedenen Distributionen, z.B. Red Hat Linux, Debian Linux, Fedora Core, S.u.S.E. Linux usw. Alle diese unterscheiden sich im Umfang der Programme.

Neben dem Kernel bestehen die Distributionen vor allem aus GNU Programmen. Eigentlich ist das, was man als Linux bezeichnet, ein GNU System mit einem Linux Kernel. GNU Programme sind eine große Sammlung freier Software. Sie bildet neben dem Kernel den wesentlichen Teil des Systems. Dazu kommen — abhängig von der Distribution — noch weitere Programme. Manche Distributionen enthalten auch kommerzielle Produkte.

Als Distribution bezeichnet man eine Zusammenstellung von Software nach dem obigen Schema. Mittlerweile gibt es sehr viele Distributionen, die sich alle ein wenig unterscheiden. Schließlich bleibt es jedermanns Geschmack überlassen, welche Distribution bevorzugt wird.

Es kommt noch ein weiterer Faktor hinzu: alle diese Distributionen haben spezielle Zielgruppen. S.u.S.E. Linux eignet sich wegen seiner beispielhaften Installation vor allem für Anfänger, die das erste Mal Linux installieren. Seit der Version 5.0 basiert es ebenfalls auf den sogenannten RPM Paketen, wodurch sich Installation, Update und Deinstallation von Software sehr einfach gestalten. Red Hat zielt mit Red Hat Enterprise Linux (und Fedora Core) auf Firmen und etwas fortgeschrittenere Benutzer ab (was nicht heißt, dass ein Anfänger Red Hat Linux nicht installieren könnte). Die Installation ist zwar nicht ganz so benutzerfreundlich, dafür zeichnet sich die Distribution durch ein saubereres Konzept als S.u.S.E. Linux und eine sehr klare Konfiguration aus. Noch mehr gilt das für GNU Debian Linux, das jedoch weniger als Red Hat auf Firmen als Zielkunden setzt. Die Konfigurationsdateien sind vorbildlich, aber für Anfänger ist es, zumindest im Moment, noch nicht ohne die Hilfe eines erfahreneren Linux Benutzers nicht empfehlenswert. Diese Liste ließe sich noch weiter fortsetzen. Die genannten Distributionen sind jedoch die zur Zeit am meisten verwendeten.

3 Einstieg in Linux

A computer lets you make more mistakes faster than any other invention, with the possible exceptions of handguns and Tequilla.

(Mitch Ratcliffe)

3.1 Der Start

Starten Sie den Computer, falls er noch nicht unter Linux läuft, neu. Sie bekommen nun ein Menü, aus dem Sie Linux auswählen können. Nun wird das System gestartet, es erscheinen zahlreiche Meldungen. Schließlich startet die grafische Oberfläche.

Um mit dem Linux System arbeiten zu können, müssen Sie den Dienst „Linux“ beim ZID beantragen. Dazu müssen Sie lediglich das entsprechende Web–Formular auf den Seiten des ZID ausfüllen. Ab dem nächsten Werktag ist der Zugang aktiviert.

Sie können sich nun mit Ihrer Benutzerkennung und Ihrem Passwort beim System anmelden. Beachten Sie, dass das Passwort unabhängig von dem unter Netware/Windows oder dem des Mailsystems verwaltet wird. Zu Beginn sollte Ihr Anfangspasswort gesetzt sein.

3.2 Die grafische Oberfläche

Wenn Sie Linux in den Benutzerräumen des ZID Innsbruck starten, befinden Sie sich auf einer grafischen Oberfläche. Zuerst werden Sie nach Ihrem Benutzernamen und Ihrem Passwort gefragt. Beantworten Sie diese Fragen und schließen Sie jede Eingabe jeweils mit der Eingabetaste ab.

Anschließend wird eine grafische Benutzeroberfläche gestartet. Auf dem Bildschirm befindet sich unten eine Navigationsleiste. Das ist die sogenannte Kontrollleiste (auch als Startleiste bezeichnet) (Abb. 1).

Abbildung 1: Ein Teil der KDE Kontrollleiste.

Ganz links der Kontrollleiste befindet sich das KDE Menü. Wenn Sie auf dieses klicken, öffnet sich ein Menü, über das verschiedene Anwendungen gestartet werden können (Abb. 2).

Neben dem KDE Menü befindet sich der Umschalter (Abb.3). Auf diesem sehen Sie mehrere kleine Rechtecke, die aufsteigend nummeriert sind. Wenn Sie z.B. auf das kleine Rechteck mit der Aufschrift 2 klicken, verschwinden die offenen Fenster. Was ist passiert?

Abbildung 2: Das KDE Menü.

Der Bildschirm besteht aus mehreren virtuellen Arbeitsflächen. Zwischen diesen können Sie über die Rechtecke (die verkleinerte Arbeitsflächen darstellen) hin– und herschalten. Auf jeder Arbeitsfläche können weitere Anwendungen ausgeführt werden, wodurch sich das Arbeiten mit mehreren Anwendungen übersichtlicher gestaltet. Am Umschalter (englisch *Pager*) sehen Sie nicht nur die vorhandenen Arbeitsflächen, sondern auch Miniaturansichten der geöffneten Fenster, sodass man einen groben Überblick hat, welche Fenster auf welcher Arbeitsfläche geöffnet sind.

Beim Klick auf die Arbeitsfläche `1` kommen Sie zurück auf die ursprünglich gestartete Arbeitsfläche und finden dort auch wieder die schon geöffneten Fenster.

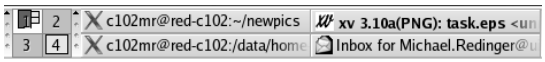

Abbildung 3: Fensterleiste und Umschalter auf der KDE Kontrollleiste.

Auf der Kontrollleiste befindet sich auch die Fensterleiste, in der die laufenden Applikationen angezeigt werden (Abb. 3). Wenn Sie eine Anwendung anklicken, wechseln Sie zu dieser. Wenn Sie die aktuell aktive Anwendung wählen, so wird diese minimiert.
Durch Klicken der rechten Maustaste auf eine Anwendung bekommen Sie ein Menü (Abb. 4).

Hier können Sie Fenster wieder auf andere Arbeitsflächen schicken. Die anderen Menüeinträge werde ich weiter unten besprechen.

Wenden wir uns nun den Fenstern zu. Suchen Sie sich irgendeines aus und sehen Sie es sich an (sollte keines offen sein, starten Sie über das KDE Menü ein beliebiges Programm). Das Fenster besteht aus einem Rahmen (der Begrenzung), einer Titelleiste mit 4 Symbolen sowie dem Fensterinhalt selbst (Abb. 5).

Abbildung 4: Menü der Fensterleiste.

Abbildung 5: Ein Fenster unter KDE.

Wenden wir uns zuerst dem Rahmen zu. Bewegen Sie die Maus auf einen Fensterrahmen. Der Mauszeiger verwandelt sich nun in einen Pfeil. Wenn Sie nun die linke Maustaste drücken und gedrückt halten, können Sie die Fenstergröße vertikal oder horizontal verändern — je nachdem, wo Sie sich befinden. Wenn Sie den Mauszeiger auf eine Ecke des Fensters bewegen, ändert sich dieser auf einen Pfeil, der auf einen rechten Winkel zeigt. Damit können Sie die Größe des Fensters gleichzeitig vertikal und horizontal verändern.

Auf der Titelleiste befinden sich 4 Symbole:

Auf der linken Seite außen befindet sich entweder ein Symbol oder ein Pfeil nach unten. Klicken Sie auf diesen. Hier erscheint nun ein Menü wie in Abbildung 6.

- Verschieben
  Damit können Sie das Fenster am Bildschirm solange verschieben, bis Sie eine Maustaste drücken.

Abbildung 6: Das KDE Fenstermenü.

- Größe ändern
  Das verwandelt den linke und obere Rand des Fensters in eine Linie. Durch Bewegen der Maus können Sie nun die Fenstergröße ändern, bis Sie wieder irgendeine Maustaste drücken.

- Minimieren
  Wenn Sie auf diesen Punkt klicken, verschwindet das Fenster. Die Anwendung bleibt aber weiterhin in der der Fensterleiste sichtbar, ist dort aber nun grau unterlegt. Ein Klick auf die Anwendung in der Fensterleiste macht das Fenster wieder sichtbar.

- Maximieren
  Wenn Sie diesen Punkt anklicken, wird das Fenster auf die Größe des Bildschirms vergrößert (wobei jedoch die Kontrollleiste sichtbar bleibt). Wenn Sie das Menü nun noch einmal öffnen, erscheint daneben ein Häkchen. Wählen Sie den Punkt nun nochmals, so wird die vorige Fenstergröße wiederhergestellt.

- Fensterheben
  Diser Punkt lässt den Fensterinhalt verschwinden, es ist nur noch die Titelleiste mit den Menüs und Symbolen sichtbar.

- Erweitert
  In diesem Untermenü können Sie das Verhalten und Aussehen der Fenster noch genauer einstellen. Über den Punkt Immer im Vordergrund können Sie beispielsweise ein Fenster immer im Vordergrund halten, es wird dann nie von anderen überdeckt, bis Sie diese Funktion wieder abwählen.

- Auf Arbeitsfläche
  Dieser Befehl schickt das Fenster auf eine andere Arbeitsfläche — wenn Sie zum Beispiel auf der aktuellen Arbeitsfläche schon zuviele Fenster offen haben und wieder etwas Überblick gewinnen wollen.
  Hier gibt es den speziellen Eintrag Alle Arbeitsflächen. Wählen Sie diesen als Ziel für das Fenster aus, so wird dieses auf allen Arbeitsflächen dargestellt.

- Schließen
  Dieser Eintrag bewirkt etwas, was wohl niemand vermutet hätte: er beendet die Anwendung.

Neben dem Fenstermenü finden Sie eine Reihe von Symbolen. Diese entsprechen meist Funktionen im Menü, sind aber so schneller zugänglich.

Auf der anderen Seite der Leiste folgen drei Symbole. Das erste sieht aus wie ein Strich, es entspricht dem Menüpunkt Minimize. Darauf folgt das Symbol für Maximieren und schließlich ganz rechts das Pendant zum Eintrag Schließen.

Auf dem Bildschirm befinden sich noch weitere Elemente. Links befinden sich einige Symbole — mit diesen können Sie zum Beispiel schnell auf eine CD-ROM zugreifen, indem durch das Klicken auf das Symbol der Dateimanager gestartet wird.

Wenn Sie mit der mittleren Maustaste (oder, wenn Sie nur 2 Maustasten haben, dann mit beiden gleichzeitig) auf den Hintergrund klicken, bekommen Sie die Fensterliste. Wenn Sie diese anklicken, erscheint eine Liste der zur Zeit laufenden Anwendungen. Außerdem sehen Sie auch, auf welchem Desktop diese laufen (Abb. 7).

Abbildung 7: Die KDE Fensterliste.

Durch Drücken der rechten Maustaste öffnet sich das Arbeitsflächenmenü (Abb. 8).

Abbildung 8: Das Arbeitsflächenmenü am Bildschirmhintergrund.

Hier können Sie über Befehl ausführen... einen Befehl ausführen, die Symblode der Arbeitsfläche bearbeiten oder das Aussehen und Verhalten Ihrer Arbeitsflächen verändern (über Arbeitsfläche einrichen...). Sie finden hier noch eine Reihe anderer Einträge — experimentieren Sie damit. Weitere Hilfestellungen erhalten Sie in der

KDE Hilfe (die Sie ebenfalls über das K Menü starten können). Erwähnt werden sollen vielleicht noch die beiden letzten Punkte. Es ist dies als erstes Sitzung sperren, womit Sie den Arbeitsplatz sichern können — es erscheint ein Bildschirmschoner, der nur mit Ihrem Passwort wieder deaktiviert werden kann. Ganz am Ende des Menüs schließlich befindet sich der Punkt Abmelden. Damit beenden Sie die grafische Oberfläche und melden sich vom System ab.

Damit sollten Sie die wichtigsten Operationen zum Umgang mit der Oberfläche kennen. Weitere Hilfe zur KDE Oberfläche und den KDE Programmen finden Sie in der KDE Hilfe, das Sie über den Eintrag Hilfe im KDE Menü starten.

Ich werde später noch einmal auf die grafische Oberfläche zu sprechen kommen, einige Programme und nützliche Funktionen kennenlernen.

Zum Abschluss dieser kurzen Einführung wollen wir noch die Shell, die Kommandozeile, starten. Wenn nicht anders erwähnt, werden wir im ganzen nachfolgenden Text mit dieser arbeiten.
Öffnen Sie dazu das KDE Menü und wählen Sie den Punkt Befehl ausführen.... Es erscheint ein kleines Fenster. Geben Sie dort

```
xterm -ls
```

ein und drücken Sie die Eingabetaste. Abbildung 9 zeigt das `xterm` (*X Terminal*), die Kommandozeile.

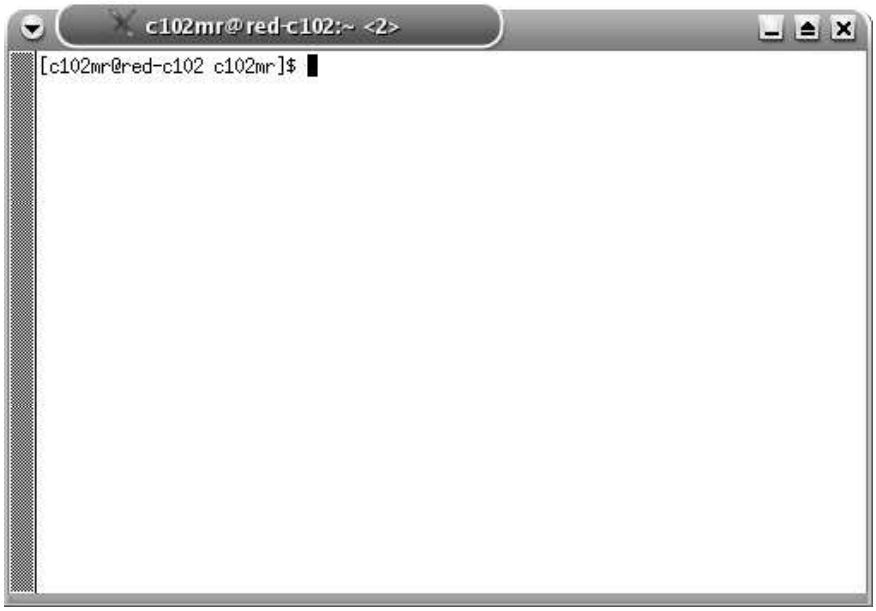

Abbildung 9: Die über `xterm -ls` gestartete Kommandozeile.

Im Folgenden werden wir, sofern nicht anders angegeben, immer auf der Kommandozeile arbeiten.

3.3 Einloggen

Linux ist multiuserfähig. Das heißt es können verschiedene Benutzer gleichzeitig am selben System arbeiten. Daher müssen Sie sich gleich am Anfang einloggen, d.h. anmelden. Dieser Vorgang wurde oben bereits beschrieben (Eingabe des Benutzernamens und dessen Validierung mit dem Passwort).

Beim Starten einer Shell (in unserem Fall heißt diese `bash` und wird in dem Programm `xterm` gestartet) befinden Sie sich in Ihrem HOME Verzeichnis. Das ist sozusagen Ihre Operationsbasis, das Verzeichnis, wo Sie Dateien ablegen können, ohne dass irgendjemand außer Ihnen Zugriff darauf hat.

Zum Ausloggen wählen Sie im KDE Menü den Eintrag Abmelden.

Wie bereits erwähnt ist Linux multiuserfähig. Es gibt viele verschiedene Benutzer. Damit ist es sogar möglich, sich auf einem Rechner mehrmals und mit verschiedenen Namen einzuloggen. Wenn Sie sich später intensiver mit Linux beschäftigen, werden Sie feststellen, dass Sie nicht alles machen dürfen. Sie können z.B. bestimmte Dateien nicht verändern — aber dazu später. Zusätzlich zu den normalen Benutzern gibt es noch einen besonderen Benutzer, den Systemadministrator *root*. Er darf buchstäblich alles machen (zumindest theoretisch). Er überwacht das System, konfiguriert und wartet es. Die Tatsache, dass Sie nicht alle Rechte haben, ist natürlich zu Ihrem eigenen Schutz und zu dem anderer Benutzer.

WICHTIG:

Unter keinen Umständen dürfen Sie während des Betriebs einfach den Rechner abschalten. Sie riskieren sonst den Verlust von Daten!

Zuerst loggen Sie sich aus (drücken Sie auf einer freien Stelle des Bildschirm die rechte Maustaste und wählen Sie Abmelden). Damit gelange Sie wieder auf einen Bildschirm, den Sie schon kennen, den grafischen Anmeldebildschirm. Hier können Sie das System nun über einen eigenen Menüpunkt geregelt herunterfahren.

3.4 Hilfe

Ein neues und noch dazu so mächtiges Betriebssystem wie Linux kennenzulernen, ist für viele faszinierend. Aber schnell kann das auch zum Albtraum werden, wenn man einmal nicht weiß, welche zusätzlichen Optionen ein Programm benötigt usw. Wir wollen daher, bevor wir mit den Linux Befehlen beginnen, die Hilfe kennenlernen.
Zu fast allen Befehlen gibt es eine Hilfe. Diese wird auf der Kommandozeile mit dem Befehl `man` ⟨`Befehl`⟩ (*manual*) aufgerufen. Um also die Hilfe zu `man` selbst zu lesen, geben Sie folgenden Befehl ein:

```
man man
```

Es erscheint nun die Hilfeseite. Hier können Sie mit den Richtungstasten nach oben und nach unten blättern. Mit der Leertaste oder dem Buchstaben f können Sie eine Seite nach vorne blättern, mit der Taste b zurück. An das Ende des Textes gelangen Sie, indem Sie die Taste G eingeben, an den Anfang mit 1 G. Zu einer beliebigen Zeile springen Sie mit *x* G, wobeit *x* die Zeilennummer ist.

Sie können in der Hilfeseite Wörter/Zeichen suchen, indem Sie / eingeben. Dieses Zeichen wird nun ganz unten in der letzten Zeile angezeigt. Geben Sie nun den Suchbegriff ein und drücken Sie die Eingabetaste. Der Begriff wird nun gesucht. Zum nächsten Vorkommen des Suchbegriffes gelangen Sie mit n, zum vorhergehenden mit N.
Die Hilfe beenden Sie mit der Taste q.

`man` verwendet zur Seitendarstellung der Hilfe das Programm `less`. Mit **`man less`** erfahren Sie mehr über dieses Programm.[2]

Die Hilfeseiten sind auf verschiedene Sektionen aufgeteilt:[3]

Sektion	Beschreibung
1	Benutzerkommandos
2	Systemaufrufe der Bibliotheken
3	Bibliotheksfunktionen
4	Beschreibung der Konfigurationsdateien
5	Syntax wichtiger Dateien
6	Spiele
7	Textformatierung und Dateiformate
8	Befehle zur Systemadministration
9	Linux Kernelroutinen
n	nicht zuordenbar

Für Benutzer sind fast ausschließlich die Hilfeseiten in Sektion 1 interessant (wenn Sie nicht programmieren wollen).

Manchmal kommt es vor, dass es in verschiedenen Sektionen Hilfeseiten gleichen Namens gibt. In diesem Fall können Sie die Sektion der Hilfeseite voranstellen, um auf jeden Fall die gewünschte Hilfeseite zu erhalten. Die Hilfeseite von `man` hätten wir auch so aufrufen können:

```
man 1 man
```

Die Hilfeseiten enthalten auch Stichworte. Wenn Sie eine Hilfe zum Thema Hilfeseiten (*manual*) erhalten wollen, geben Sie Folgendes ein:

```
man -k manual
```

Die Ausgabe hilft Ihnen im Moment noch nicht viel weiter — behalten Sie diese Option aber in Erinnerung.

Sie können die Hilfe auch in einem grafisch orientierten Programm nutzen (Abb. 10). Öffnen Sie dazu wieder das KDE-Hilfezentrum , indem Sie im KDE Menü

[2] Das Programm zur Darstellung der Hilfeseiten kann sich auf anderen Unix Systemen unterscheiden. Oft wird **`more`** verwendet, das nur ein Subset der Funktionalität von **`less`** bietet.

[3] Die Einordnung in den einzelnen Gruppen ist auf vielen Unix Systemen unterschiedlich. Mehr Information finden Sie in den Hilfeseiten von `man`. Benutzerbefehle befinden sich aber immer in Sektion 1.

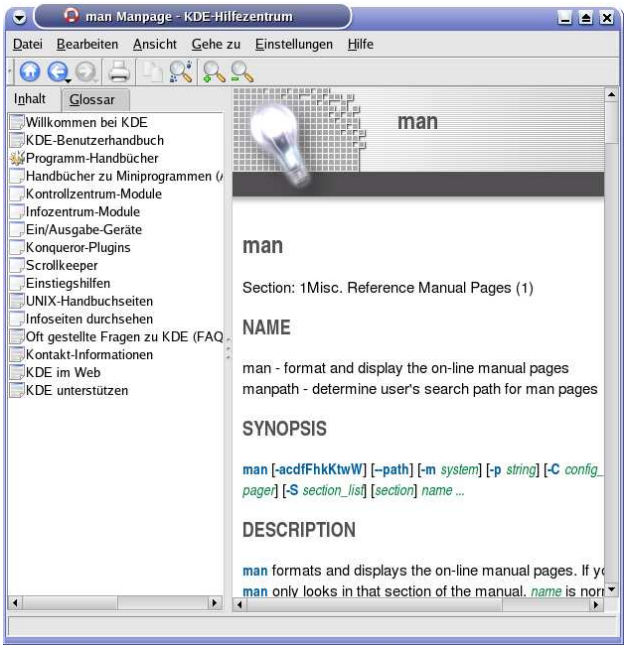

Abbildung 10: Das KDE-Hilfezentrum bei der Anzeige der `man` Hilfeseite.

den Punkt Hilfe wählen. Hier können Sie in der Übersicht links den Punkt Unix-Handbuchseiten anwählen. Nun erscheint eine Liste der Hilfesektionen. Wählen Sie hier die erste Sektion und dort die gewünschte Hilfe (z.B. `man`).

Im Folgenden werden Sie viele Befehle kennenlernen. Die meisten haben einen weit größeren Funktionsumfang als ich hier besprechen kann. Sie sollten daher immer wieder einen Blick auf die Hilfeseiten der Programme werfen.

3.5 Aufgaben

1. Machen Sie sich mit der grafischen Oberfläche vertraut. Arbeiten Sie mit Fenstern und Desktops, versuchen Sie, deren Aussehen zu verändern. Rufen Sie die Hilfe zu KDE allgemein sowie zu KDE Programmen auf und versuchen Sie, über die grafische Oberfläche `man` Hilfeseiten anzuzeigen.

Die Lösungen finden Sie auf Seite 122.

4 Die Shell

Some people claim that the UNIX learning curve is steep, but at least you only have to climb it once.

4.1 Dateien und Verzeichnisse

Geben Sie Folgendes ein:

`pwd`

Als Ausgabe am Bildschirm erhalten Sie beispielsweise:

`/home/c102/c102mr`[4]

Das ist Ihr HOME Verzeichnis. Daran sieht man, dass das Verzeichnistrennzeichen eines Pfades unter Linux/Unix der Schrägstrich (englisch *Slash*) / (und nicht wie unter Microsoft Windows der Backslash \) ist. Dadurch können wir auch definieren: ein Pfad ist eine Zeichenkette, bei der eine beliebige Anzahl von Verzeichnissen jeweils durch einen Schrägstrich getrennt sind.

Eine weitere Unterschied gegenüber Microsoft Windows ist, dass es keine Laufwerke (*C:*, *D:* usw.) gibt. Unter Unix bzw. Linux gibt es nur ein sogenanntes *root* Verzeichnis — im Deutschen auch mit *Wurzelverzeichnis* übersetzt. Die Bezeichnung Laufwerk wird unter Unix nicht verwendet. Man spricht immer von Partitionen, Medien und Mountpunkten. Weiter unten werden wir sehen, wie verschiedene Partitionen in dieses *root* Verzeichnis integriert werden.

Unter Unix wird, im Gegensatz zu Microsoft Windows, auch zwischen Groß– und Kleinschreibung unterschieden.

`/HOME/C102/C102MR`

und

`/home/c102/c102mr`

bezeichnen zwei völlig verschiedene Verzeichnisse.

Eine Trennung nach Dateierweiterungen wie unter Microsoft Windows (z.B. `*.exe` oder `*.doc` ist unter Unix in manchen Gebieten zwar üblich (z.B. `*.tex` für LaTeX Dokumente) aber keineswegs zwingend — so haben ausführbare Dateien meist gar keine Endung (z.B. lautet der Dateiname der meistverwendeten Shell `bash`).

Unter Unix gibt es die verschiedensten Shells. Das sind Befehlsinterpreter. Mit einer Mini–Version einer solchen Shell wäre etwa `command.com` unter Microsoft Windows vergleichbar. Unter Linux in den Benutzerräumen des ZID der LFU Innsbruck ist — wie bei den meisten Linux Versionen — die `bash` als Standard–Shell verfügbar. Sie können ihre Shell auch ändern. Andere beliebte Shells sind die Korn Shell oder die

[4] Auf den meisten „kleineren“ Linux – und Unix Systemen befindet sich das HOME Verzeichnis des Benutzers unter `/home/⟨Benutzerkennung⟩`.

C Shell. Wir werden uns vorerst auf die Arbeit mit der `bash` (*Bourne–Again Shell*) konzentrieren.

Bei der `bash` und einigen anderen Shells ist eine Funktion namens *name completion* vorhanden. Wenn Sie einen Befehl oder einen Dateinamen unvollständig angeben und dann die *TAB* Taste (das sollte auf Ihrer Tastatur die vierte von unten ganz links sein) drücken, wird der Name vervollständigt, falls er eindeutig ist, d.h. wenn es nur eine Möglichkeit zur Vervollständigung gibt. Gibt es mehrere Möglichkeiten, so können Sie zweimal die *TAB* Taste drücken. Nun listet Ihnen die `bash` sämtliche Möglichkeiten auf (ab einem voreingestellten Wert wird abgefragt, ob Sie wirklich alle Möglichkeiten sehen wollen). Dieses Feature mag speziell für Anfänger sehr hilfreich sein – man sollte sich jedoch hüten, dies zu extensiv zu benutzen. Auf anderen Unix Systemen — und wer weiß, vielleicht arbeitet man später einmal mit solchen — ist dies meist nicht vorhanden. Und wenn man sich erst einmal daran gewöhnt hat, ist das Umlernen schmerzhaft.

4.2 Arbeiten mit Verzeichnissen

Nach dem Einloggen — sei das nun bei einer *telnet* Sitzung oder im *xterm* — befinden Sie sich in Ihrem HOME Verzeichnis. Wir wollen nun einmal sehen, wo wir sind. Geben Sie Folgendes ein:

`pwd`

Der Befehl `pwd` (*print working directory*) zeigt Ihnen an, in welchem Verzeichnis Sie sich gerade befinden. Sie bekommen nun eine Antwort, die Ausgabe des Befehles, z.B.:

```
/home/c102/c102mr
```

Wie auch unter Microsoft Windows dient zum Wechseln des Verzeichnisses der Befehl `cd`. Geben Sie nun ein:

`cd bin`[5]

Nun geben Sie wieder `pwd` ein. Die Ausgabe sollte wie folgt aussehen:

```
/home/c102/c102mr/bin
```

Nun wollen wir noch einmal ein `cd` ausführen. Geben Sie ein:

`cd /bin`

Wenn Sie nun `pwd` ausführen, so lautet das Ergebnis:

```
/bin
```

Achten Sie bitte auf den feinen Unterschied zwischen den beiden Eingaben von `cd`. Beim zweiten Mal steht vor dem `bin` ein `/`. Dieses Zeichen, den *Slash*, haben wir

[5] Dies setzt voraus, dass dieses Verzeichnis vorhanden ist. Sollten Sie hier eine Fehlermeldung erhalten, geben Sie
`mkdir bin`
ein und wiederholen dann den `cd` Befehl. Später werden wir sehen, was Sie da gerade gemacht haben.

bereits als Trennzeichen für Verzeichnisse kennengelernt — und genau das ist er auch hier.

Was wir hier lernen ist der vielleicht bereits von Microsoft Windows her bekannte Unterschied zwischen absoluten und relativen Dateiangaben.

Der erste Befehl wechselt in das Verzeichnis `bin` — und zwar abhängig vom aktuellen Verzeichnis. Da wir uns im HOME Verzeichnis befinden, wechselt er dort in das Verzeichnis `bin`, falls es vorhanden ist. Wären wir z.B. in `/usr`, würde `cd` dort in das `bin` Verzeichnis wechseln (also nach `/usr/bin`). Das nennt man *relative Dateiangaben*.

Der zweite Befehl wechselt in das Verzeichnis `/bin`. Das ist ein *absoluter Verzeichnisname*. Unabhängig davon, wo wir uns gerade befinden, wechselt dieser Befehl nach `/bin`. Absolute Dateiangaben listen immer sämtliche Verzeichnisse vom *root* Verzeichnis ausgehend bis zum Zielverzeichnis oder bis zur Zieldatei auf. Wollten wir in das `bin` Verzeichnis in unserem HOME Verzeichnis wechseln, so müsste ich (für Sie gehörte das entsprechend angepasst) eingeben:

`cd /home/c102/c102mr/bin`

Wechseln Sie einmal in das *root* Verzeichnis. Geben Sie also ein:

`cd /`

Wir wollen versuchen, wieder in unser HOME Verzeichnis zu kommen. Ich könnte jetzt eingeben `cd /home/c102/c102mr`. Aber das ist dann doch etwas zu lang. Hier gibt es einige ganz einfache Lösung. Sie verwenden dazu das *Tilde* Zeichen `~`. Dieses bezeichnet immer Ihr HOME Verzeichnis. Sie geben also ein:

`cd ~`

Oben sind wir in das `bin` Verzeichnis in unserem HOME Verzeichnis gewechselt — das war nun doch etwas kompliziert, wie wir jetzt wissen. Es wäre wesentlich einfacher gegangen:

`cd ~/bin`

Eine Parallele zu Microsoft Windows stellt die Bezeichnung des übergeordneten Verzeichnisses mit **`..`** dar. Sie befinden sich, wenn Sie die obigen Schritte genau nachvollzogen haben, jetzt im Verzeichnis `bin` in Ihrem HOME Verzeichnis. Existiert ein Verzeichnis namens `Desktop`, so können Sie mit der Eingabe von

`cd ../Desktop`

in dieses wechseln.[6]

Im Gegensatz dazu wird das aktuelle Verzeichnis mit „**`.`**“ bezeichnet. Würde sich in Ihrem Verzeichnis ein Programm namens `test` befinden, so könnten Sie dieses durch folgende Zeile ausführen: **`./test`** Wir wissen nun also schon, wie wir von Verzeichnis zu Verzeichnis wechseln.

[6] Ansonsten können Sie davor
`mkdir ../Desktop`
eingeben.

Der Befehl zum Erstellen von Verzeichnissen ist `mkdir` (*make directory*). Wenn Sie also in Ihrem HOME Verzeichnis ein Verzeichnis namens `junk` erstellen wollen, geben Sie ein:

`mkdir ~/junk`[7]

Zum Entfernen von Verzeichnissen wird `rmdir` verwendet. Dazu muss aber das Verzeichnis leer sein![8] Dazu erstellen wir vorher noch ein neues Verzeichnis:

`mkdir ~/test`

Dieses Verzeichnis können wir nun nicht gebrauchen. Wir entfernen es daher wieder mit dem Befehl `rmdir` (*remove directory*):

`rmdir ~/test`

Das `junk` Verzeichnis können Sie, wenn Sie wollen, bestehen lassen. Es ist wohl eines der Verzeichnisse, das in den meisten HOME Verzeichnissen vorhanden ist (`junk` heißt auf englisch soviel wie „Trödel“ oder „Plunder“...) — es wird meist zum Testen von Software usw. verwendet.

Zum Verschieben von Verzeichnissen können Sie den Befehl `mv` (*move*) verwenden. Erstellen Sie also noch einmal das `test` Verzeichnis. Dann verschieben Sie es nach `Bloedsinn`:

`mv ~/test ~/Bloedsinn`

Wie wir später sehen werden, wird der `mv` Befehl auf dieselbe Weise auch bei normalen Dateien verwendet.

4.3 Arbeiten mit Dateien

Bei der Arbeit am Computer sind Dateien von ganz entscheidender Bedeutung. Mehr noch als für alle anderen Betriebssysteme gilt das für Unix. Hier werden z.B. auch die Festplatte oder das Diskettenlaufwerk über eine Dateien angesprochen (ich wiederhole mich — aber was das genau heißt, werden wir wieder später sehen). Dementsprechend zahlreich und wichtig sind die Programme unter Linux, mit denen man Dateien manipulieren kann. Wir werden im Folgenden nur die wichtigsten besprechen, andere folgen weiter unten, wieder andere habe ich aufgrund der großen Menge weglassen müssen.

4.3.1 Informationen über Dateien

Viele werden mich im letzten Abschnitt wohl schon heimlich verflucht haben — da zeige ich nun, wie man in Verzeichnisse wechselt, aber Sie haben noch nicht einmal eine Ahnung, was sich dort eigentlich befindet. Aber alles zu seiner Zeit...

[7] Beachten Sie hier, dass `~` in allen Befehlen, nicht nur bei `cd` verwendet werden kann.

[8] Das Entfernen nicht–leerer Verzeichnisse wir im Abschnitt 4.3.5 behandelt.

Einer der grundlegendsten und meistverwendetsten Befehle ist `ls`, das Pendant zum Microsoft Windows Befehl `dir`. Mit diesem ist es nun möglich, die Dateien und Verzeichnisse aufzulisten — daher auch der Name (`list`).
Gibt man einfach nur

```
ls
```

ein, so wird der Inhalt des aktuellen Verzeichnisses aufgelistet (also dessen Dateien und Verzeichnisse).

Aber Unix wäre nicht Unix, gäbe es hier nicht zahlreiche Möglichkeiten.

Nehmen wir einmal an, Sie befinden sich in Ihrem HOME Verzeichnis. Sie wollen nun wissen, welche Dateien sich unter `/etc` befinden (wir werden später noch besprechen, in welchen Verzeichnissen sich was befindet). Dazu geben Sie ein:

```
ls /etc
```

Nun möchten Sie — unter Linux sind ausführbare Dateien ja nicht an der Endung zu erkennen, und auch der Unterschied zwischen Verzeichnissen und Dateien ist mit dem normalen `ls` Befehl nicht erkennbar[9] —, vielleicht auch angezeigt bekommen, was für eine Art von Datei Sie vor sich haben. Dazu verwenden Sie die Option `-F`. Geben Sie also ein:

```
ls -F ~
```

Nun sehen Sie hinter den Verzeichnissen das Zeichen `/`, das wir ja schon als Zeichen für Verzeichnisse kennengelernt haben. Es handelt sich also bei den so dargestellten Namen um Verzeichnisse. Ähnliches gilt auch für ausführbare Dateien (also einem „Programm“ im weitesten Sinn des Wortes) — auch sie werden mit einem speziellen Zeichen am Ende, dem `*`, angezeigt. Versuchen Sie's:

```
ls -F /bin
```

Weiter unten werden wir noch weitere Möglichkeiten kennenlernen, die Art und die Eigenschaften einer Datei kennenzulernen.

Um noch mehr über eine Datei zu erfahren, können Sie den Befehl `file` verwenden. Dieser sagt ihnen z.B. bei ausführbaren Dateien, ob es ein Programm oder ein Shell Script (was das ist, werden wir wieder einmal später besprechen — jaja, ich weiß, das avanciert langsam zu meiner Lieblingsphrase...) ist. Bei Textdateien erfahren Sie, ob der Text englischsprachig ist oder nicht. Mit `file` erfahren Sie z.B. Art einer Bilddatei (z.B. die JPEG oder die Ghostscript Version) und vieles mehr. Probieren Sie's:

```
file ~/liesmich
```[10]

[9] Neuere Linux Systeme zeigen aber, wenn möglich, ausführbare Dateien und die Verzeichnisse in einer anderen Farbe an

[10] Ist diese Datei nicht vorhanden, dann sollten Sie diese anlegen. Wir werden dies später besprechen. Geben Sie vorerst einfach die beiden folgenden Befehle ein:

```
cp /etc/services ~/liesmich
echo Unix und Linux >> ~/liesmich
```

4.3.2 Exkurs: Wildcards

Bevor wir nun zu weiteren Arbeiten mit Dateien kommen, wollen wir uns mit den sogenannten Wildcards[11] beschäftigen. Auch unter Microsoft Windows sind sie rudimentär vorhanden (v.a. `*`, der *Asterisk*), aber erst unter Unix kommen sie voll zum Einsatz (und werden von großer Bedeutung).

Wissen Sie etwa, dass sich im Verzeichnis `/etc` Dateien befinden, die ein „e“ 2 Positionen vor einem Punkt haben, nach dem noch 4 weitere Zeichen folgen, so könnten Sie alle entsprechenden Dateien mit folgendem Befehl auflisten:

```
ls -d *e??.????
```

Das `*` steht also für beliebig viele beliebige Zeichen. Das `?` hingegen steht für ein einzelnes beliebiges Zeichen.

Dehnen wir das Beispiel noch etwas aus. Nehmen wir an, dass das Zeichen kein „e“ ist, sondern irgendein Zeichen von `b` bis `h` (also `b`, `c`, `d`, `e`, `f`, `g` oder `h`) ist. Dann würde der Befehl wie folgt aussehen:

```
ls -d /etc/*[b-h]??.????
```

Das sollte fast selbsterklärend sein: `*` und `?` werden wie oben verwendet. In eckigen Klammern, getrennt durch einen Bindestrich, steht der Bereich, der für die Suche verwendet werden soll.

Auch das kann man noch steigern. Nehmen wir an, Sie suchen wie oben, aber statt dem Bereich nach den Zeichen `c`, `e`, `g`, `J`, `K`, `L`, `X`, `1`, `4`, `5`, `6` und `9`. Sie würden dann eingeben

```
ls -d /etc/*[cegJ-LX14-69]??.????
```

Aus obigem Beispiel sehen wir auch schon recht gut, dass man die verschiedenen Wildcards kombinieren kann.

Neben eckigen Klammern kann man auch geschwungene verwenden. Deren Syntax ist etwas unterschiedlich:

```
ls l{ie,a,e,alala}smich
```

Die einzelnen Möglichkeiten sind hier nun durch Beistriche getrennt. Dieses Beispiel wird von der Shell nun expandiert, es ergeben sich folgende Auflösungen:

```
ls liesmich
ls lasmich
ls lesmich
ls lalalasmich
```

Vielleicht fragen Sie sich, wozu man das brauchen kann. Hier ein Beispiel. Nehmen Sie an, Sie haben in einem Verzeichnis die Dateien `tests`, `Teste` und `tests1` abgelegt. Um diese nun mit einem Befehl aufzulisten, verwenden Sie den folgenden Befehl:

[11] Genauer: Mit Wildcards, *bracket* – und *brace expansion* und ähnlichen Konstrukten. Die ganze erschreckende Wahrheit dazu finden Sie in der `bash` Hilfeseite.

`ls [Tt]est[se]*`

Relativ häufig werden auch Befehle wie folgender verwendet:

`mkdir /tmp/{1,2,drei}`

Durch die Expandierung werden hier folgende Verzeichnisse erstellt:

```
/tmp/1
/tmp/2
/tmp/drei
```

4.3.3 Inhalt von Dateien

Wir haben nun schon gesehen, wie wir Informationen über Dateien anzeigen können, nun wollen wir zum Anzeigen der Dateien selbst kommen.

Die einfachste Methode, eine Datei anzuzeigen, ist mit dem Programm `cat` (*catenate*). Dieses Programm gibt einfach den Inhalt einer Datei auf dem Bildschirm aus. Probieren Sie's:

`cat ~/liesmich`

Sie werden bemerken, dass die Zeilen einfach auf dem Bildschirm ausgegeben werden. Wenn die Datei zu lange ist — Pech gehabt. Dann sehen Sie nur die Zeilen, die als letzte ausgegeben werden. In solchen Fällen gibt es ein weiteres Programm, `less`.[12] Dieses zeigt die Dateien Seite für Seite an. Sie kennen das Programm schon: Das Hilfeprogramm `man` verwendet es auch, um die Hilfe Seite für Seite auszugeben. `less` zählt zu den Programmen, die man bei der Arbeit in der Shell ständig brauchen wird, es wird uns noch mehrmals begegnen. Ein einfaches Beispiel:

`less ~/liesmich`

Damit haben wir nun die Datei angezeigt. Aber wie editieren wir sie? Vorweg: der von Administratoren und vielen Benutzern bevorzugte Editor ist vi. Er ist jedoch für den Anfang etwas zu komplex, wird daher später (Abschnitt 7) besprochen.

Ein einfach Editor ist nano[13] (Abb. 11).[14] Er ist auch für den Anfänger leicht zu bedienen.

Starten Sie nano mit dem Befehl:

`nano`

Unten finden Sie einige Tastenkürzel, wobei ^ für die Strg Taste steht. Mit ^ – G bekommen Sie Hilfe.

Eine grafischer, sehr einfach zu bedienender, aber trotzdem funktionell gut ausgestatteter Editor ist `kate` (Abb. 12).[15]

[12] Auf manchem Unix Systemen ist dieses Programm nicht vorhanden. In diesem Fall können Sie more verwenden; es steht Ihnen dann aber nur ein Subset der Funktionalität von `less` zur Verfügung.

[13] Auf vielen Systemen ist stattdessen pico vorhanden, das praktisch identische Funktionalität aufweist.

[14] Auf vielen Unix Systemen (und bei einigen Linux Installationen) ist dieses Programm nicht verfügbar. In diesem Fall können Sie entweder einen grafische Applikation oder den Editor vi (siehe

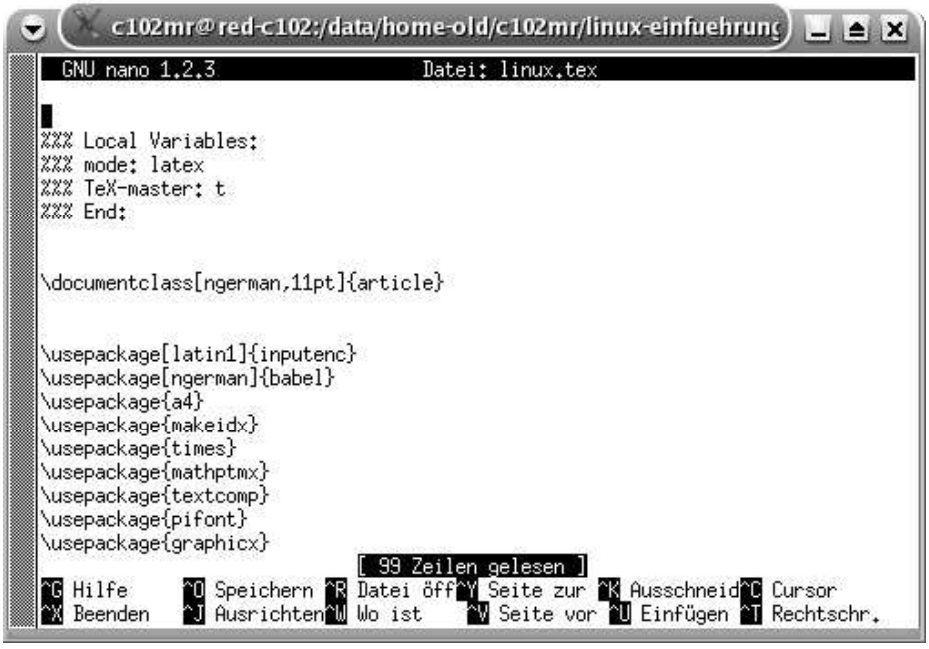

Abbildung 11: Ein einfacher textbasierender Editor, nano.

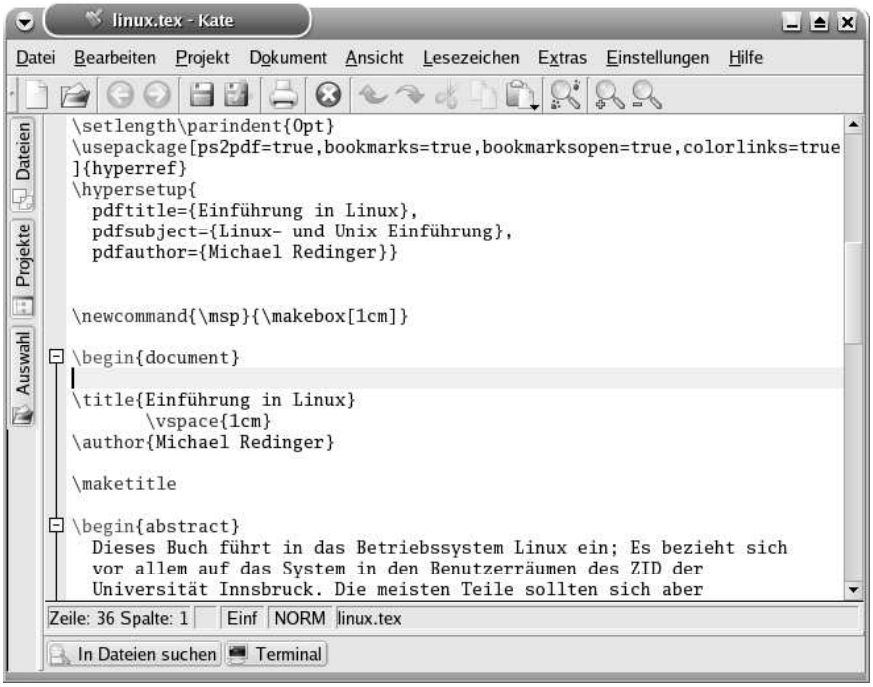

Abbildung 12: Der Erweiterte KDE Editor, kate.

Der Aufruf erfolgt über das KDE Menü im Untermenü Programmieren. Alternativ kann man ihn über die Shell mit `kate` direkt aufrufen. Dieser Editor ist am besten in die KDE Oberfläche integriert und wird so für viele Anfänger die erste Wahl sein.

`kate` ist nur einer von vielen Editoren unter Linux. Andere finden Sie im KDE Menü — z.B. NEdit oder Emacs.

4.3.4 Exkurs: Ein– und Ausgabeumleitung

Ein zentrales Konzept von Unix ist die Abstraktion der Ein– und Ausgabe. Bei nicht–interaktiven (und vielen anderen) Programmen gibt es zwar einen Standardwert für die Ein– und Ausgabe, diese sind aber nicht statisch, sondern können geändert werden. Man unterscheidet folgende Schnittstellen:

Name	Kürzel	Bedeutung
Standardeingabe	STDIN	Die Eingabe eines Programmes.
Standardausgabe	STDOUT	Hier erfolgt die Ausgabe eines Programmes.
Standardfehler	STDERR	Hier werden die Fehlermeldungen ausgegeben.

Bei den meisten Programmen ist die Standardeingabe die Kommandozeile, z.B.:

```
ls ~/liesmich
```

Es wird über die Kommandozeile dem Befehl `ls` die Anweisung gegeben, die Datei `liesmich` aufzulisten.
Die Standardausgabe erfolgt, ebenso wie der Standardfehler, auf dem Bildschirm — so auch beim vorigen Beispiel.

Mit Hilfe der Ein– und Ausgabe–Umleitung können diese jedoch umdefiniert werden. So kann die Ausgabe von `ls` in eine Datei geschrieben werden. Am Bildschirm erfolgt nun keine Ausgabe mehr, diese wird in die Datei geschrieben:

```
ls > ~/ls.out
```

„>“ bewirkt, dass die Ausgabedatei neu erstellt wird. War sie bereits vorhanden, wird der Inhalt überschrieben. Soll die Ausgabe an eine Datei angehängt werden, so verwendet man „>>“:

```
ls etc >> ~/ls.out
```

Wenn Sie aber nun eingeben:

```
ls ~/gibts.nicht >> ~/ls.out
```

(und die Datei `~/gibts.nicht` — was zu erwarten ist — nicht existiert), so wird von `ls` nichts an die Datei `~/ls.out` angehängt (da nichts auf der Standardausgabe ausgegeben wird). Auf der Kommandozeile erhalten Sie aber folgende Ausgabe:

Abschnitt 7) verwenden.

[15] Auf anderen Unix Systemen ist dieser Editor meist nicht vorhanden. Versuchen Sie, durch Eingabe von `nedit`, ob dieser Editor vorhanden ist, oder lesen Sie die lokale Dokumentation, um die vorhandenen grafischen Editoren kennenzulernen.

```
ls: /home/c102mr/gibts.nicht: No such file or directory
```

Dies ist eine Fehlermeldung, die besagt, dass die Datei nicht existiert. Die Fehlermeldungen werden als Standardfehler ausgegeben. Auch diesen können Sie in eine Datei umleiten:

```
ls ~/gibts.nicht 2> ~/ls.err
```

„`2>`“ gibt an, wohin der Standardfehler umgeleitet wird.

Sie können die Standardausgabe und den Standardfehler in zwei verschiedene Dateien umleiten:

```
ls ~/liesmich ~/gibts.nicht > ~/ls.out 2> ~/ls.err
```

`ls` versucht, sowohl `~/liesmich` als auch `~/gibts.nicht` aufzulisten. Die Datei `~/liesmich` existiert, daher wird die Ausgabe nach `~/ls.out` geschrieben. Bei `~/gibts.nicht` tritt ein Fehler auf, da diese Datei nicht exisitert. Dieser wird nun nach `~/ls.err` geschrieben.

Sie können auch beide Ausgaben in eine Datei schreiben:

```
ls ~/liesmich ~/gibts.nicht > ~/ls.out 2>&1
```

„`2>&1`“ gibt an, dass die Fehler dorthin geschrieben werden sollen, wohin auch die Standardausgabe erfolgt.

Die Ausgabe kann auch an ein anderes Programm übergeben werden:

```
ls -lR /etc | less
```

Dies bewirkt, dass die Standardausgabe von `ls` umgeleitet wird und als Standardeingabe des Befehls `less` verwendet wird. `ls` wird ausgeführt, die Ausgabe durch das Zeichen „`|`“ (das als *Pipe* — dt. Rohr — bezeichnet wird. Auf der Tastatur bekommen Sie dieses Zeichen, wenn Sie AltGr – < drücken) an das Programm `less` übergeben. Dieses nimmt es nun als Standardeingabe; dadurch wird die Ausgabe von `ls` durch `less` seitenweise dargestellt.
Wollte man hier auch die Fehler — falls vorhanden — von `less` darstellen lassen, würde man folgenden Befehl verwenden:

```
ls -lR /etc 2>&1 | less
```

Hier wird die Standardausgabe über die Pipe wiederum an `less` übergeben. Mit „`2>&1`“ teilt man der Shell mit, dass mit dem Standardfehler genauso zu verfahren ist.

Wir haben mit der Pipe bereits gesehen, dass man die Standardeingabe umleiten kann. Man kann diese auch so ändern, dass sie aus einer Datei erfolgt.
Das Programm `tr` (*translate*) ersetzt Zeichen durch andere (mehr interessiert uns vorerst nicht — wir werden das Programm später noch einmal behandeln). Das folgende Beispiel ersetzt alle Groß– durch Kleinbuchstaben, wobei es die Eingabe aus einer Datei bezieht; dafür wird das Zeichen „<“ verwendet. Die Ausgabe wird in eine andere Datei geschrieben:

```
tr '[A-Z]' '[a-z]' < ~/liesmich > ~/liesmich.klein
```

4.3.5 Kopieren, Umbenennen und Löschen von Dateien

Wir haben nun schon einiges über Dateioperationen gelernt. Wir können die Dateien auflisten und am Bildschirm anzeigen. Nun folgt ein weiterer Teil: das Kopieren und Verschieben.

Zum Kopieren verwendet man den Befehl `cp` (*copy*). Wollen Sie z.B. die Datei `/etc/fstab` in Ihr HOME Verzeichnis kopieren, so erfolgt das mit

```
cp /etc/fstab ~/
```
[16]

`cp` nimmt den ersten Parameter, `/etc/fstab`, als Quelldatei und kopiert sie in das HOME Verzeichnis. Wollten Sie die Datei im HOME Verzeichnis z.B. unter dem Namen `fstab.copy` speichern, so würden Sie eingeben:

```
cp /etc/fstab ~/fstab.copy
```

Wenn Sie mehrere Dateien angeben und der letzte Parameter ein Verzeichnis ist, werden alle Dateien in dieses Verzeichnis kopiert. Das folgende Beispiel kopiert mehrere Dateien aus `/etc` in das Verzeichnis `testdir`.

```
mkdir  /testdir
cp /etc/fstab /etc/profile  /testdir
```

Ehrlich gesagt: die Dateien `fstab` und `fstab.copy` haben nun wirklich nichts in Ihrem HOME Verzeichnis verloren. Mit dem Befehl `rm` (*remove*) können sie wieder gelöscht werden:

```
rm ~/fstab ~/fstab.copy
```

Wie wir oben gesehen haben, wäre das auch mit einem Befehl gegangen:

```
rm ~/fstab{.copy,}
```

Für `rm` gibt es auch eine Option, die bewirkt, dass ein Verzeichnis rekursiv gelöscht wird, d.h. es kann auch gelöscht werden, wenn sich darin noch weitere Dateien oder Verzeichnisse befinden. Die Eingabe von

```
rm -r ~/testdir
```

würde also dieses Verzeichnis mit allen darin enthaltenen Verzeichnissen und Dateien löschen.
Verwenden Sie diesen Befehl mit größter Vorsicht! Es gibt (fast) keine Möglichkeit, einmal gelöschte Dateien wiederherzustellen!

Eine Datei entsteht, wenn ein Programm oder der Benutzer diese erstellt. Das kann entweder explizit durch einen Befehl erfolgen oder aber, wenn eine Datei geschrieben werden soll, die noch nicht besteht. Oben haben wir — ohne das zu erwähnen — bereits Dateien erstellt, z.B. mit dem Befehl

```
ls > ~/ls.out
```

Da diese Datei noch nicht vorhanden war, wurde sie erstellt.

[16] Ist die Datei `/etc/fstab` nicht vorhanden, verwenden Sie stattdessen `/etc/vfstab` oder `/etc/profile`.

Um unter Unix explizit eine neue, leere Datei zu erstellen, verwendet man den Befehl `touch`, z.B.:

```
touch ~/neue.Datei
```

Ein Befehl fehlt uns noch, dann haben wir die wesentlichsten Dateioperationen kennengelernt: `mv` (*move*). Dieser Befehl entspricht in etwa dem Microsoft Windows Pendant `move`. Mit diesem werden Dateien und Verzeichnisse verschoben bzw. umbenannt. Wir haben gerade erst die Datei `neue.Datei` erstellt. Nun, der Name ist nicht eben besonders bedeutungsvoll. Verschieben wir die Datei doch:

```
mv ~/neue.Datei ~/meine.erste.selbsterstellte.Datei
```

Der Befehl funktioniert auch für Verzeichnisse:

```
mv ~/ein.Verzeichnis ~/anderes.Verzeichnis
```
[17]

Dadurch wird `ein.Verzeichnis`, falls es vorhanden ist, nach `anderes.Verzeichnis` in Ihrem HOME Verzeichnis verschoben. Es wird auch der Inhalt dieses Verzeichnisses mitverschoben.

Wenn Sie als letzten Parameter ein Verzeichnis angeben, das bereits existiert, wird die zu verschiebende Datei oder das zu verschiebende Verzeichnis in dieses Verzeichnis verschoben.

4.3.6 Suchen und Durchsuchen von Dateien

Wir haben bisher also Verzeichnisse und Dateien aufgelistet, auf dem Bildschirm dargestellt, kopiert, verschoben usw. Das bezog sich aber immer nur auf einzelne Verzeichnisse oder Dateien. Will man aber wissen, welche Dateien sich in unserem HOME Verzeichnis befinden, so kann natürlich `ls` verwendet werden. Wenn wir aber bereits wissen, welche Datei wir suchen, aber nicht mehr wissen, wo sie sich befindet und eine Verzeichnisstruktur sehr umfangreich ist, wird das schnell umständlich. Das führt uns zu einem sehr praktischen Befehl: `find`. Damit durchsucht man ein Verzeichnis mit allen Unterverzeichnissen nach Dateien mit einem bestimmten Namen. In Ihrem HOME Verzeichnis befindet sich (hoffentlich) eine Datei namens `liesmich`. Um nach dieser zu suchen verwenden Sie:

```
find ~/ -name 'liesmich'
```
[18]

Nun wird nach der Datei `liesmich` gesucht, und alle Dateien mit diesem Namen werden aufgelistet. Nach `find` steht zuerst einmal das Verzeichnis, in dem gesucht werden soll — das ist hier Ihr HOME Verzeichnis. Dann folgt `-name`. Das zeigt an, dass nach einem Namen gesucht werden soll. Das ist von Bedeutung, da mit `find` auch nach Dateien mit einer bestimmten Größe, mit einem bestimmten Erstellungsdatum usw. gesucht werden kann. Näheres erfahren Sie wie immer mit `man find`. Als letztes folgt nun der Name der Datei, nach der gesucht wird. Dieser steht in einfachen Anführungszeichen. Das wäre zwar hier nicht notwendig, aber sobald Sie Wildcards

[17] Das zu verschiebende Verzeichnis muss zuerst mit `mkdir` erstellt werden.

[18] Bei manchen `find` Implementationen anderer Unix Systeme müssen Sie am Ende `-print` angeben, damit die gefundenen Dateien aufgelistet werden.

verwenden, müssen Sie diese verwenden — gewöhnen Sie sich also lieber gleich daran.

Nehmen wir an, Sie wissen nicht den vollständigen Namen, sondern nur, dass die Datei mit `lies` beginnt. Außerdem wissen Sie nicht mehr, ob die Datei mit einem Groß– oder einem Kleinbuchstaben beginnt. Klarer Fall — hier kommen natürlich wieder die schon besprochenen Wildcards zum Einsatz:

```
find ~/ -name '[lL]ies*'
```

Wenn Sie nicht mehr wissen, was das heißt, dann lesen Sie bitte auf Seite 28 noch einmal nach.

Nehmen wir nun an, Sie wollen nicht nach Dateien, sondern nach Wörtern in diesen suchen. Dazu wird der Befehl `grep` verwendet. Wenn Sie z.B. wissen wollen, ob in `~/liesmich` (falls diese Datei besteht) das Wort `Linux` vorkommt, so verwenden Sie dazu:

```
grep "linux" ~/liesmich
```

Falls das Wort vorkommt, wird die entsprechende Zeile angezeigt.

Oft wissen Sie aber nicht, ob das gesuchte Wort groß oder klein geschrieben wird. In diesem Fall hilft die Option `-i` von `grep` weiter. `-i` steht für *case insensitive*, durchsucht also die Datei nach der gegebenen Zeichenkette, unabhängig von Groß– oder Kleinschreibung:

```
grep -i "linux" ~/liesmich
```

`grep` durchsucht normalerweise nur die angegebenen Dateien, nicht aber ganze Verzeichnisse. Wollen Sie einen Verzeichnisbaum nach einer Zeichenkette durchsuchen, verwenden Sie dazu die Option `-r` (*rekursiv*).[19] Das folgende Beispiel durchsucht alle Dateien in Ihrem HOME Verzeichnis nach der Zeichenkette `Linux` (und zwar unabhängig von Groß– oder Kleinschreibung):

```
grep -ri "linux" ~
```

4.3.7 Links

Durch die Verwendung von symbolischen Links kann einer Datei ein zusätzlicher Name gegeben werden. Dieser Name „zeigt“ dann auf diese Datei. Es kann z.B. vorkommen, dass verschiedene Versionen eines Programmes aufgehoben werden sollen, aber die jeweils benutzte Version immer unter dem gleichen Namen verfügbar sein soll. Die Lösung liegt im Erstellen eines symbolischen Links, der auf die jeweils benutzte Version zeigt. Symbolische Links verhalten sich wie die Dateien, auf die sie zeigen. Die folgende Eingabe

```
ln -s ~/liesmich ~/liesmich.1
```

erzeugt einen symbolischen Link `~/liesmich.1`, der auf die Datei `~/liesmich` zeigt. Würden Sie nun den Link editieren, so würde sich auch die ursprüngliche Datei ändern und umgekehrt. Das hat den Vorteil, dass Sie immer nur eine Datei ändern

[19] Diese Option ist auf den meisten anderen Unix Systemen nicht vorhanden.

müssen. Außerdem wird aber der Platz für die Datei nur einmal belegt, Sie brauchen nicht soviel Speicherplatz. Entfernen Sie nun den Link, so verschwindet dieser. Die ursprüngliche Datei bleibt aber erhalten. Wenn Sie hingegen die ursprüngliche Datei entfernen, so wird der Link weiterhin aufgeführt — er zeigt aber ins Leere, die ursprüngliche Datei ist ja nicht mehr vorhanden.

Das gilt für Softlinks. Diese werden mit der oben verwendeten Option `-s` erzeugt. Im Gegensatz dazu stehen die Hardlinks. Sie werden erstellt mit:

```
ln ~/liesmich ~/readme
```

Hier fehlt die Option `-s`. Auch hier gilt: wenn Sie eine Datei ändern, ändert sich auch der Link und umgekehrt. Wenn Sie aber die ursprüngliche Datei löschen, bleibt der Link bestehen.

Bei Softlinks wird auf der Festplatte an einer freien Stelle hingeschrieben: „siehe Datei …“, und es wird dann beim Zugriff auf die Datei überprüft, ob der Punkt, auf den der Verweis zeigt, vorhanden ist.

Hardlinks funktionieren anders: bei ihnen wird einfach in die Liste, die dem Betriebssystem verrät, wo welche Datei ist, für dieselbe Stelle ein zweiter Name eingetragen.

Dieser für den Anfänger möglicherweise klein erscheinende Unterschied hat große Folgen: löscht man nämlich eine Datei, auf die ein Softlink zeigt, zeigt dieser ins Leere und ergibt, wenn darauf zugegriffen werden soll, einen Fehler. Wird hingegen die Datei, auf die ein Hardlink erstellt wurde, gelöscht, bleibt der Hardlink bestehen — aus der Liste der Dateinamen wurde ja nur der Originalname gelöscht, der zweite und damit auch der Dateiinhalt ist ja noch vorhanden.

Langer Rede kurzer Sinn: es gibt 2 Typen von Links. Dem Anfänger sei empfohlen, der Übersichtlichkeit halber vorerst nur Softlinks zu verwenden.

4.3.8 Dateien archivieren und sichern

Jetzt wissen wir, wie man Dateien kopiert und verschiebt. Aber versuchen Sie einmal, 20 oder mehr Dateien zu kopieren. Das artet in ein mechanisches, zeitaufwändiges Wiederholen von Befehlen aus. Als Ausweg packt man die Dateien in ein Archiv, verschiebt sie an den neuen Platz und extrahiert sie dort wieder. Als Archiv bezeichnet man eine Datei, die mehrere Dateien enthält. Unter Microsoft Windows sind vor allem die kombinierten Archivier– und Komprimierungsprogramme `zip` und `arj` bekannt. Unter Unix wird meist `tar` (`tape archive`) verwendet. Üblicherweise haben diese Archive die Endung `.tar`. Sie können auch noch mit dem Komprimierprogramm `gzip`[20] komprimiert sein. Dann haben sie die Endung `.tar.gz` oder manchmal auch `.tgz`. Man kann hier zwischen 3 verschiedenen Modi unterscheiden: dem Erzeugen, dem Auspacken und dem Auflisten eines Archivs.

Zum Erzeugen eines Archivs wird `tar` mit den Optionen `cf` verwendet. Wenn Sie es dann auch gleich noch komprimieren wollen, fügen Sie die Option `z`[21] hinzu. Die

[20] `gzip` ist auf anderen Unix Systemen optional, aber meist vorhanden.

[21] Diese Option ist bei den meisten Unix Derivaten nicht vorhanden, hier müssen Sie das Archiv nach der Erstellung beziehungsweise vor dem Entpacken manuell mit `gzip` komprimieren oder mit `gunzip` dekomprimieren. Mehr dazu finden Sie auf Seite 91.

Option `v` (*verbose*) bewirkt, dass während des Archivierens angezeigt wird, was das Programm gerade macht. Um zum Beispiel das Verzeichnis `Desktop` und die Datei `liesmich` zu archivieren, würden Sie folgenden Befehl verwenden:

```
tar cvf mein.archiv.tar  /liesmich  /Desktop
```

Wenn Sie nun ein Archiv erhalten und erst einmal wissen wollen, welche Dateien es enthält, bevor Sie es entpacken, verwenden Sie die Optionen `tf`. Bei komprimierten Archiven ist wiederum die Option `z` zu verwenden.

```
tar tf mein.archiv.tar
```

Wenn Sie sich schließlich versichert haben, dass das Archiv die gewünschten Dateien enthält, können Sie das Archiv nun auspacken. Dazu verwenden Sie die Optionen `xvf` und gegebenenfalls `z`.

```
tar xvf mein.archiv.tar
```

Wollen Sie nur eine einzelne Datei aus dem Archiv auspacken, so geben Sie diese nach dem Archiv an:

```
tar xvf mein.archiv.tar liesmich
```

4.4 Zusammenfassung

In diesem Kapitel wurden sehr viele Befehle vorgestellt. Zum Schluss daher eine kurze tabellarische Zusammenfassung.

Befehl	Bedeutung
cat	Dateiinhalt ausgeben.
cd	Wechseln des Verzeichnisses.
cp	Datei kopieren.
echo	Zeichenkette ausgeben.
file	Dateityp anzeigen.
find	Datei suchen.
grep	Datei durchsuchen.
less	Datei seitenweise darstellen.
ln	Link erstellen.
ls	Verzeichnisinhalt auflisten.
mkdir	Verzeichnis erstellen.
mv	Datei oder Verzeichnis verschieben.
pico	Textbasierender Editor.
pwd	Name des aktuellen Verzeichnisses ausgeben.
rm	Datei löschen.
rmdir	(Leeres) Verzeichnis löschen.
tar	Archiv erstellen.
touch	Leere Datei erstellen.

4.5 Aufgaben

1. Wechseln Sie mit `cd /usr` in das Verzeichnis `/usr`. Was ist der Unterschied zwischen `cd /bin` und `cd bin`? Wohin wechseln Sie mit den Befehlen `cd .`, `cd ..` beziehungsweise `cd ~`? Was bedeuten diese beiden speziellen Verzeichnisse?

2. Was passiert, wenn Sie in ein nicht vorhandenes Verzeichnis wechseln?

3. Erstellen Sie in Ihrem HOME Verzeichnis das Verzeichnis `junk` und kopieren Sie eine beliebige Datei dorthin. Warum können Sie das Verzeichnis nicht mit `rmdir` löschen? Mit welchem Befehl gelingt dies?

4. Erstellen Sie noch einmal ein Verzeichnis `junk` sowie eine Kopie einer beliebigen Datei.
   Benennen Sie diese nun um und verschieben Sie sie in das Verzeichnis `junk`. Wie erreichen Sie das mit 2 Befehlen, wie mit einem?
   Erstellen Sie noch drei unterschiedlich benannte Kopien einer Datei (und verwenden Sie andere Namen als zuvor). Was geschieht, wenn Sie nach dem Befehl `mv` nur diese Dateien angeben? Wie können Sie diese Dateien mit einem Befehl nach `junk` verschieben?

5. Lesen Sie die Hilfeseite zu `ls` und experimentieren Sie mit den möglichen Optionen. Was bedeuten „`-A`“, „`-h`“[22] und „`-r`“?[23]
   Wenn Sie `ls /tmp` angeben, wird der Inhalt des Verzeichnisses `/tmp` angezeigt. Können Sie auch nur das Verzeichnis `/tmp`, nicht dessen Inhalt anzeigen?

6. Was bewirkt der Befehl `cat /etc/profile /etc/services > ~/sprof`?

7. Welche Ausgabe liefert der Befehl `ls /etc/m[ao]*.conf{ig,}`? Erklären Sie diesen![24]

8. Was macht der Befehl `grep -ri "xxx"/etc/s* 2> /dev/null | less`? Was ist der Unterschied zu `grep -ri "xxx"/etc/s* 2>&1 | less`?[25]

9. Ändern Sie das Änderungsdatum einer Datei mit `touch` auf den 31. März 2004 12:35.

10. Suchen Sie in Ihrem HOME Verzeichnis alle Verzeichnisse, nicht aber Dateien, und lassen Sie diese anzeigen.

11. Suchen Sie in Ihrem HOME Verzeichnis alle Dateien, die neuer sind als das Verzeichnis `Desktop`.

[22] Diese Option ist auf den meisten Unix Systemen nicht vorhanden.
[23] Tipp: Verwenden Sie dabei immer auch die Option `-l`
[24] Das Resultat hängt stark von der verwendeten Unix Variante ab.
[25] Tipp: Den größten Unterschied sehen Sie bei `grep -ri "xxx"/etc/s* >/dev/null`.

12. Erstellen Sie ein TAR Archiv. Fügen Sie dann mit einem `tar` Befehl dem Archiv eine Datei hinzu. Lassen Sie sich dann den Inhalt des Archives auflisten.

Die Lösungen finden Sie auf Seite 122.

5 Dateizugriff

There are two major products that come out of Berkeley: LSD and UNIX. We don't believe this to be a coincidence.

(Jeremy S. Anderson)

5.1 Zugriffsrechte

Nur der Superuser `root` — das ist der Systemadministrator — hat uneingeschränkten Zugriff auf alle Dateien und Verzeichnisse. Alle anderen Benutzer haben lediglich beschränkte Zugriffsmöglichkeiten. Volle Kontrolle haben Sie über die Dateien in Ihrem HOME Verzeichnis und an der LFU Innsbruck auch über die Dateien und Verzeichnisse in `/tmp` und `/scratch`[26], die Sie angelegt haben. Aus Sicherheitsgründen haben Sie sonst nirgends im Dateisystem die Möglichkeit, Dateien zu erstellen oder vorhandene Dateien zu ändern. Das ist übrigens auch zu Ihrem eigenen Schutz — sonst könnte ja jeder (absichtlich oder unabsichtlich) das System unbrauchbar machen.

Dieses System wird verwirklicht, indem jede Datei und jedes Verzeichnis bestimmte Rechte hat, die anzeigen, wer was damit machen darf. Diese Rechte kann man mit `ls -l` ansehen.

`ls -l liesmich`

Das Ergebnis sieht wie folgt aus:

```
- rw- r-- r--  1  c102mr  c102  4387  Dec 12 13:38  liesmich
- --- --- ---  |  ------  ----  ----  ------------  --------
|   |   |   |  |     |     |     |         |            |
|   |   |   |  |     |     |     |         |        Dateiname
|   |   |   |  |     |     |     |         |
|   |   |   |  |     |     |     |        Änderungsdatum
|   |   |   |  |     |     |     |
|   |   |   |  |     |     |    Dateigröße
|   |   |   |  |     |     |
|   |   |   |  |     |    Gruppe
|   |   |   |  |     |
|   |   |   |  |    Eigentümer
|   |   |   |  |
|   |   |   |  Hardlinks
|   |   |   |
|   |   |  Rechte aller anderen
|   |   |
|   |  Rechte der Gruppe
```

[26] Dieses Verzeichnis ist spezifisch für die Installation an der LFU Innsbruck.

```
|   |
|   Rechte des Eigentümers
|
Dateiart
```

- Dateiart
  Hier steht, welche Art von Datei das ist. Bei - handelt es sich um eine normale Datei. Wäre es ein Verzeichnis, so stünde hier `d`. Bei Softlinks steht ein `l`, bei Gerätedateien `c`.

- Rechte des Eigentümers
  Dieser Block gibt an, welche Rechte der Eigentümer der Datei (das ist normalerweise die Person, die die Datei angelegt hat. In Ihrem HOME Verzeichnis sind Sie normalerweise bei allen Dateien der Eigentümer) hat. Der erste Teil dieses Blocks enthält entweder den Buchstaben `r` (das heißt Lesezugriff; der Eigentümer darf die Datei lesen) oder -, im zweiten `w` (d.h. Schreibzugriff; der Eigentümer darf die Datei verändern) oder -, und im dritten Teil steht `x` (d.h. Ausführrechte; der Benutzer kann die Datei ausführen: es ist also ein "Programm") oder -.
  Die Rechte hier dienen eigentlich nur dazu, dass der Eigentümer eine Datei vor irrtümlicher Löschung durch sich selbst schützt oder die Datei als ausführbar markiert. Wie wir später sehen werden, kann der Eigentümer die Rechte für seine Dateien ändern.
  Bei Verzeichnissen steht ein `x`. Das bedeutet, dass man das Verzeichnis „ausführen" kann — man wechselt also in dieses.

- Rechte der Gruppe
  Dieser Block zeigt die Rechte der Gruppe an — die möglichen Buchstabenkombinationen entsprechen wieder den zuvor genannten. Jeder Benutzer gehört zu einer Gruppe. Diese Rechte treffen nun für alle Benutzer dieser Gruppe zu. Mehr zu Benutzern und Benutzergruppen finden Sie auf Seite 43.

- Rechte aller anderen
  Dieser Block gibt an, was alle anderen mit dieser Datei machen dürfen. Wieder entsprechen die möglichen Buchstaben den zuvor angegebenen.

- Hardlinks
  Hier steht, wieviele Hardlinks auf diese Datei existieren. `1` bedeutet, dass kein Hardlink besteht, 2 würde bedeuten, dass es einen Hardlink gibt usw. Eigentlich wird hier angegeben, wieviele Einträge es in der Dateizuordnungstabelle des Betriebssystems für diese Datei gibt. Wir haben ja oben erfahren, dass für jede Datei ein solcher Eintrag existiert und dass für Hardlinks einfach ein weiterer Eintrag angefertigt wird.
  Etwas anders ist die Bedeutung bei Verzeichnissen. Hier steht dann, wieviele direkte Unterverzeichnisse existieren. Nehmen wir an, Sie haben ein Verzeichnis mit zwei Unterverzeichnissen. In der Dateizuordnungstabelle steht nun dieser Verzeichnisname. Gleichzeitig gibt es in diesem Verzeichnis das spezielle Verzeichnis „.", das (wie wir schon gehört haben) auf den Verzeichnisnamen zeigt

und soviel bedeutet wie „das aktuelle Verzeichnis“. Damit wären wir für diese Spalte bei der Zahl 2. Nun existiert aber noch in jedem der 2 Unterverzeichnisse ein spezielles Verzeichnis „`..`“, das für das übergeordnete Verzeichnis steht, also ebenfalls wieder auf unseren Verzeichnisnamen zeigt (Sie verwenden das beispielsweise, wenn Sie `cd ..` eingeben). Somit ergibt sich bei diesem Beispiel für diese Spalte schließlich die Zahl 4.

- Eigentümer
  Dieser Block zeigt nun endlich, wer eigentlich der Eigentümer der Datei ist.
- Gruppe
  Dieser Block zeigt die Gruppe des Eigentümers und damit die Gruppe, welche die Rechte in Block 3 besitzt, an.
- Dateigröße
  Hier steht, wie groß die Datei ist. Die Angabe erfolgt in Bytes. Um daraus die Größe in Kilobytes zu erhalten, dividieren Sie dies durch 1024, für Megabyte dann noch einmal durch 1024.
- Änderungsdatum
  Hier wird angezeigt, wann die Datei das letzte Mal geändert wurde.
- Dateiname
  Hier steht schließlich der Dateiname.

Ist die Datei oder das Verzeichnis ein Softlink, so folgt ein 11. Block, der wie folgt aussieht:

```
-> wohin.der.link.zeigt
```

Wie bereits oben erwähnt kann der Eigentümer einer Datei deren Zugriffsrechte ändern. Das geschieht mit dem Befehl `chmod` (*change mode*). Dieses Programm wird mit zwei Argumenten aufgerufen: den zu ändernden Rechten und der Datei, auf welche die Änderung anzuwenden ist. Bei den Rechten unterscheidet man drei Kategorien. Die des Eigentümers werden mit `u` angegeben, die der Gruppe mit `g` und die aller anderen mit `o`. Dem entsprechenden Buchstaben folgt ein = (dieses zeigt an, dass die Rechte auf den folgenden Wert gesetzt werden sollen), ein + (das besagt, dass die nachfolgenden Rechte hinzugefügt werden sollen) oder ein − (das angibt, dass die nachfolgenden Rechte entfernt werden sollen). Darauf folgen die Rechte (`r`, `w` und/oder `x`). Das hört sich ziemlich kompliziert aus. An einem Beispiel kann gezeigt werden, dass dem bei näherer Betrachtung nicht so ist:

```
chmod u=rw liesmich
```

Hier werden der Datei `liesmich` für den Benutzer die Rechte Lesen (`r`) und Schreiben (`w`) zugewiesen. Die Rechte der Gruppe und anderer bleiben unverändert.

Ein weiteres Beispiel:

```
chmod u=rwx,g+x,o-x junk
```

Hier werden der Datei `junk` die Rechte Lesen, Schreiben und Ausführen für den Eigentümer zugewiesen. Die Gruppe bekommt zusätzlich zu den bisherigen Rechten das Recht Ausführen, und für alle anderen wird das Recht Ausführen entfernt.

Das ist aber etwas unzuverlässlich. Man weiß ja die Rechte nicht, die vorher schon gesetzt sind (obwohl man das natürlich mit `ls -l` sehen kann). Man könnte die Rechte daher auch so setzen:

```
chmod u=rwx,g=rx,o=r junk
```

Hier sollte eigentlich schon klar sein, was das bedeutet. Dasselbe Ergebnis würde man mit folgendem Befehl erreichen:

```
chmod 754 junk
```

Die Erklärung: Jedem Recht wird eine Zahl zugeordnet:

Recht	Zahl
r	4
w	2
x	1

Diese Zahlen können dann jeweils addiert werden:

Benutzer	Gruppe	Alle
$r+w+x=4+2+1=7$	$r+x=4+1=5$	$r=4$

Das ergibt aneinandergereiht 754. Mit dieser Berechnungsmethode kann man die Rechte sehr viel schneller setzen.

5.2 Benutzer und Benutzergruppen

Wir haben oben schon gesehen, dass es unter Linux verschiedene Benutzer und verschiedene Gruppen gibt. Jeder Benutzer hat eine eigene Benutzerkennung. Am ZID der LFU Innsbruck bekommt man diesen durch Ausfüllen des entsprechenden Web–Formulars. Dadurch erhält man einen Benutzernamen und kann sich mit diesem und seinem Passwort einloggen. Gleichzeitig wird unter Unix auch eine Gruppe zugewiesen, in die man gehört — also z.B. `csaa`, `csab` oder `c102`.

Wie wir ebenfalls schon gesehen haben, erhält jeder Benutzer ein HOME Verzeichnis. In dieses werden beim Anlegen Dateien abgelegt. Diese und vom Benutzer abgelegte Dateien gehören dem Benutzer. Neben dem HOME Verzeichnis stehen unter Linux in den Benutzerräumen des ZID zur kurzfristigen Dateiablage (und nur dazu) auch die Verzeichnisse `/scratch` und `/tmp` zur Verfügung.

Der Benutzer kann viele Einstellungen, die ihn betreffen, in vorhandenen oder zu erstellenden Dateien vornehmen. Näheres dazu entnehmen Sie der Dokumentation der einzelnen Programme.

5.3 Aufgaben

1. Listen Sie die Berechtigungen für Ihr HOME Verzeichnis auf. Was bedeuten diese? Was bedeuten sie für die darunterliegenden Dateien?

2. Erstellen Sie in `/tmp` ein Verzeichnis, in dem nur Sie Dateien anlegen können. Andere Benutzer sollen in dieses Verzeichnis zwar hineinwechseln können, aber sonst keine Rechte haben.

Die Lösungen finden Sie auf Seite 125.

6 Dateisysteme

Feel free to contact me (flames about my english and the useless of this driver will be redirected to /dev/null — oh no, it's full…).

(Michael Beck)

6.1 Gerätedateien

Ich habe schon erwähnt, dass unter Unix Dateien eine zentrale Rolle spielen. Am besten ist das dadurch erkennbar, dass sogar für den Zugriff auf die Hardware spezielle Dateien, die Gerätedateien, verwendet werden. Diese befinden sich im Verzeichnis `/dev` (*device*). Hier einige der wichtigsten Gerätedateien:[27]

- `/dev/null`
  „Müllschlucker“
- `/dev/mouse`
  Maus
- `/dev/printer`
  Drucker
- `/dev/cdrom`
  CD-ROM Laufwerk
- `/dev/fd0`
  1. Diskettenlaufwerk („A:“)

6.2 Einbinden von Dateisystemen

Wie schon oben erwähnt gibt es unter Unix keine Laufwerksbuchstaben wie unter Microsoft Windows. Vielmehr werden alle Speichermedien — Festplatten, Diskettenlaufwerke, ZIP Laufwerke, CD ROMs usw. — in einem Verzeichnisbaum integriert.

Diesen Einbinden erfolgt mittels des Befehles `mount`. Ein normaler Benutzer kann `mount` nur für das Diskettenlaufwerk, die CD–ROM und das ZIP–Laufwerk ausführen.[28] Sie können außerdem die sogenannten `mtools` verwenden, um Daten auf eine Diskette zu übertragen.[29]

[27] Ein Großteil dieses Abschnittes ist Linux–spezifisch. Viele Befehle und Dateien sind auf anderen Unix Systemen nicht verfügbar.

[28] Bei vielen Linux Standard–Installationen gilt das nur für interne, nicht aber für externe ZIP Laufwerke.
Bei anderen Unix Systemen kann ein Benutzer oft keines dieser Medien mounten.
Auf einigen Systemen wird eine eingelegte CD–ROM automatisch gemountet.

[29] Ich verwende hier weiterhin die etwas „veralteten“ Disketten als Beispiel, da der Umgang mit USB Geräten bei den verschiedenen Linux Distributionen noch zu unterschiedlich ist.

Mit

`mformat A:`

erstellen Sie dann auf der Diskette ein Microsoft Windows Dateisystem.

Eine Diskette können Sie mit

`mount /mnt/floppy`[30]

einbinden. Bei CD–ROM– und ZIP–Laufwerken lauten die Befehle:

`mount /mnt/cdrom`
`mount /mnt/zip`

Das Aushängen einer Diskette (und damit die Entfernung aus dem Verzeichnisbaum) erreichen Sie mit

`umount /mnt/floppy`

(analog dazu für ZIP und CD–ROM).
Beachten Sie, dass Sie die Diskette nicht entfernen dürfen, bevor Sie diese nicht aus dem Dateisystem entfernt haben! Ansonsten kann es zu Datenverlust kommen! CD–ROMs und ZIP Medien können Sie aufgrund der elektronischen Verriegelung ohnehin erst entfernen, wenn Sie sie ausgehängt haben.

Unter KDE können Sie CD–ROM, Diskettenlaufwerk und ZIP–Laufwerk sehr einfach mounten. Starten Sie dazu den Dateimanager über den Punkt Persönlicher Ordner im KDE Menü. Hier wählen Sie links den Punkt Geräte (Abb. 13).

Wenn Sie nun mit der rechten Maustaste auf Diskette klicken, können Sie über Laufwerk einbinden die Diskette einbinden (Abb. 14).

Wie erwähnt dürfen Sie Disketten nie einfach entfernen. Sie müssen dem System dies zuerst mitteilen. Damit stellen Sie sicher, dass auch wirklich alle Daten auf die Diskette geschrieben werden. Gehen Sie dazu wie beim Einbinden der Diskette vor. Ist die Diskette eingebunden, so erscheint im Dialog statt Laufwerk einbinden nun der Eintrag Laufwerkeinbindung lösen. Wählen Sie diesen, um die Diskette geregelt zu entfernen.

6.2.1 Die mtools

Zum Arbeiten mit Microsoft Windows (beziehungsweise eigentlich MS–DOS) Disketten gibt es die `mtools`. Zwar kann man diese auch mit `mount` einhängen, mit den `mtools` ist es möglich, auf Disketten zugreifen, ohne sie extra mounten zu müssen. Die `mtools` sind eine Reihe von Programmen, die in Funktionalität und Aufruf ihren Microsoft Windows Pendants entsprechen. Dem Namen des Befehls unter Microsoft Windows ist jeweils ein `m` vorangestellt.

Die Dateien werden mit den üblichen 8+3 MS–DOS Namen gespeichert. Die `mtools` unterstützen jedoch VFAT. Sie legen für Dateien, die „verstümmelt“ werden müssen,

[30] Die Stellen, wo das Diskettenlaufwerk, CD–ROM und ZIP–Laufwerk in das Dateisystem eingebunden werden — die sogenannten Mountpunkte — können sich bei den verschiedenen Distributionen unterscheiden.

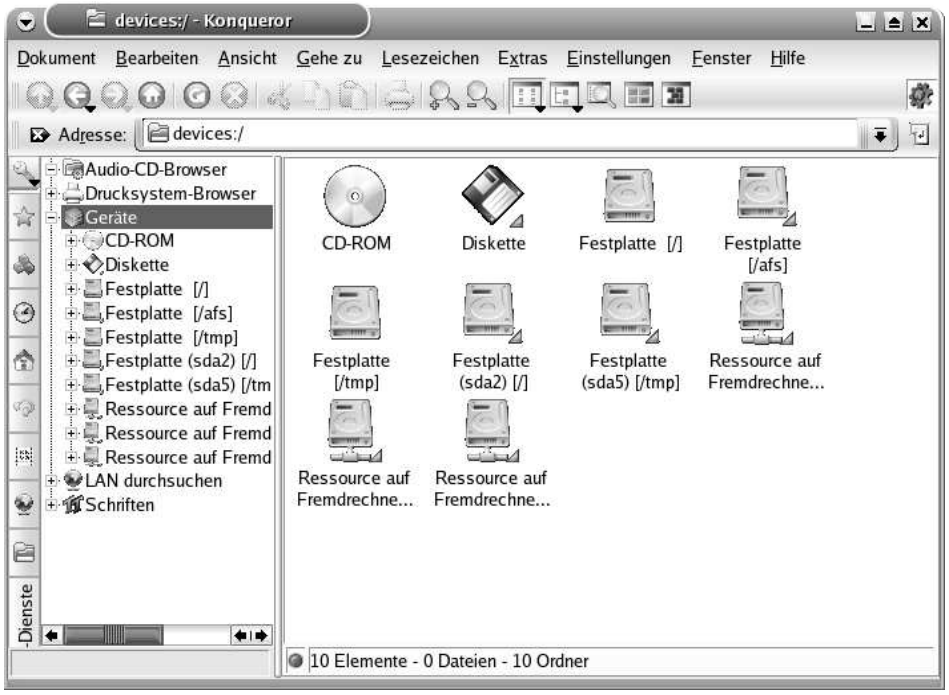

Abbildung 13: Geräte im KDE Dateimanager.

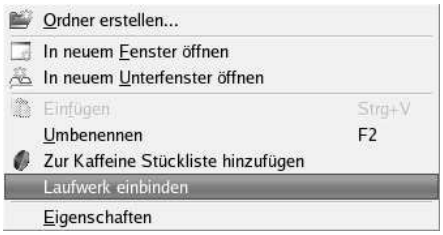

Abbildung 14: Diskette im KDE Dateimanager einbinden.

damit sie ins 8+3 Schema passen, einen primären Eintrag in der VFAT mit dem Originalnamen an und einen sekundären, der an das 8+3 Muster angepasst wurde. So sehen Sie, etwa bei Eingabe von `mdir A:` sowohl den langen als auch den kurzen Dateinamen. Dadurch ist es möglich, die Dateien auch unter Microsoft Windows zu verwenden und gleichzeitig zu gewährleisten, dass die langen Dateinamen beim Zurückkopieren auf eine Linux Partition wiederhergestellt werden. Auch Windows erkennt die von den `mtools` erstellten langen Dateinamen problemlos.

Die `mtools` beinhalten eine ganze Reihe von weiteren Programmen. Die meisten Dateioperationen, die Sie von Microsoft Windows her kennen, sind auch bei den `mtools` vorhanden. Lesen Sie bitte die Hilfeseite der `mtools` sowie der einzelnen Programme für nähere Informationen.

Die wichtigsten Befehle sind `mdir` zum Anzeigen von Verzeichnissen und Dateien, `mformat` zum Anlegen eines Microsoft Windows Dateisystems auf einer Diskette, `mmd` zum Erstellen von Verzeichnissen und natürlich `mcopy` zum Kopieren von Da-

teien auf oder von einer MS–DOS Partition.

Im Folgenden einige Beispiele für die Verwendung von Disketten. Als erstes muss das MS–DOS Dateisystem angelegt werden, falls die Diskette nicht formatiert ist:

`mformat A:`

Erstellen wir einmal ein Verzeichnis auf der Diskette:

`mmd A:/langer.Verzeichnisname`

Nun kopieren wir eine Datei auf die Diskette

`mcopy ~/liesmich A:/langer.Dateiname`

und schauen uns das Ergebnis an:

`mdir A:`

```
 Volume in drive A has no label
 Volume Serial Number is 06CC-592D
Directory for A:/

LANGER~1 VER <DIR>    05-06-2001 18:17 langer.Verzeichnisname
LANGER~1 DAT      936 05-06-2001 18:17 langer.Dateiname
        2 files               3 936 bytes
                          1 453 056 bytes free
```

Sie sehen links die „verstümmelten“ MS–DOS 8+3 und rechts die langen Dateinamen. Wenn Sie die Datei zurückkopieren, werden die langen Dateinamen wiederhergestellt. Erstellen Sie dazu ein Verzeichnis `~/dos-test`.

`mcopy A:/langer.Dateiname ~/dos-test/`

6.3 Der Verzeichnisbaum von Linux

Die Methode des Mountens wird für alle Teile des Verzeichnisbaumes verwendet. Sehen Sie sich doch die Ausgabe von `mount` an. Dieser Befehl zeigt an, welches Dateisystem wo gemountet ist. Unter anderem sind die Verzeichnisse `/home`, das `/usr`, das `/scratch` und `/tmp` Mountpunkte für verschiedene Dateisysteme.[31]

Hier ein (vereinfachter) Verzeichnisbaum, wie Sie ihn auch auf Ihrem Computer finden, mit einer kurzen Funktionsbeschreibung der Verzeichnisse.[32]

```
/              Wurzel des Verzeichnisbaums
|              (engl. root directory)
|
|- bin         für den Systemstart benötigte Programme
|
```

[31] Die Partitionierung kann je nach Installation sehr unterschiedlich sein.

[32] Der Verzeichnisbaum unterscheidet sich auf anderen Unix Systemen mehr oder weniger stark.

```
+- dev          Gerätedateien (devices)
|
+- etc          Konfigurationsdateien
|
+- home         Verzeichnisse der Benutzer
|
+- lib          gemeinsame Bibliotheken verschiedener Programme
|
+- proc         "virtuelles Dateisysten", z.B. Systemstatistiken
|
+- root         Verzeichnis des Systemadministrators
|
+- sbin         Programme zur Systemadministration
|
+- scratch      Verzeichnis für temporäre Dateien
|               (ZID LFU Innsbruck spezifisch)
|
+- tmp          Verzeichnis für temporäre Dateien
|
+- usr          nicht für den Systemstart benötigte
|  |            Programme, die sich nicht ändern
|  |
|  +- X11R6     X Window System, Version 11
|  |
|  +- bin       Programme
|  |
|  +- lib       gemeinsame Bibliotheken verschiedener Programme
|  |
|  +- include   Deklarationen für C-Bibliotheken
|  |
|  +- local     manuell installierte Software
|  |
|  +- sbin      Programme zur Systemadministration
|  |
|  |- share     plattformunabhängige Dateien
|  |  |
|  |  |-doc     Dokumentation
|  |  |
|  |  |-man     Hilfeseiten
|  |
|  |- site      Zusatzsoftware (z.B. Matlab)
|  |            (ZID LFU Innsbruck spezifisch)
|  |
|  +- src       Quelltexte (sources)
|
+- var          veränderliche/variable Dateien
```

6.4 Aufgaben

1. Was passiert, wenn Sie `/dev/null` in Ihr HOME Verzeichnis kopieren? Was, wenn Sie eine Datei nach `/dev/null` kopieren?

2. Leiten Sie die Ausgabe eines Befehls nach `/dev/null` um. Was passiert?

3. Erstellen Sie mit den `mtools` auf einer Diskette ein Verzeichnis mit einem Unterverzeichnis und kopieren Sie dorthin eine beliebige Datei. Löschen Sie diese Verzeichnishierarchie rekursiv.

4. Benennen Sie eine Datei oder ein Verzeichnis auf der Diskette mit Hilfe der `mtools` um.

5. Die `bash` beinhaltet eine Reihe von Dokumentation, etwa Beispiel–Scripts. Wo finden Sie diese?
   In welchem Verzeichnis befindet sich die Hilfeseite zur `bash`?

Die Lösungen finden Sie auf Seite 125.

7 Der Editor vi

The last time somebody said, „I find I can write much better with a word processor.“, I replied, „They used to say the same thing about drugs.“

(Roy Blount, Jr.)

Beim Editor vi scheiden sich die Geister. Für viele ist er ein urtümliches Monster, das man möglichst schnell vergessen sollte. Vor allem für Systemadministratoren ist er jedoch der Editor, mit dem schnell eine Datei editiert und verändert kann — auch, wenn man nur eine *telnet* Verbindung und keine grafische Oberfläche zur Verfügung hat. Mit kaum einem anderen Editor ist es möglich, so schnell und effizient zu arbeiten — Beherrschung der Kommandos verausgesetzt.

Der Aufruf ist auf drei verschiedene Arten möglich:

- **`vi`**
  Dies startet den „klassischen“ vi, wie er auf allen Unix Plattformen vorhanden ist.[33]
- **`vim`**
  Die `vi` Variante unter Linux nennt sich `vim`. *vi improved*. Sie zeichnet sich durch erweiterte Funktionalität aus (zum Beispiel Syntax–Highlighting).[34]
- **`gvim`**
  Dies ist die „grafische“ Variante des `vi`. Sie funktioniert genauso wie die Textvariante, bietet jedoch einige Menüpunkte, sodass man sich hier nicht immer an alle Befehle erinnern muss.

Die Bedienung des `vi` ist etwas gewöhnungsbedürftig. Er wird an dieser Stelle anderen Editoren vorgezogen, weil er auf jedem Unix–ähnlichen Betriebssystem zum standardmäßigen Installationsumfang gehört; weiters, weil seine Bedienung eindeutig ist und dadurch i.a. keine Missverständnisse auftreten. Diese Kurzanleitung sollte Sie in die Lage versetzen, mit Hilfe des `vi` z.B. diverse Konfigurationsdateien zu editieren — mehr nicht. Wer mehr will, kann die Dokumentation von `vi` lesen. Außerdem wurden über diesen Editor ganze Bücher geschrieben.

Der `vi` kennt 3 Betriebsarten (Modi)

- command mode
  jeder Tastendruck wird als Teil eines Befehls interpretiert.
- input mode
  Tastendrücke werden als Texteingaben interpretiert.

[33] Bei Red Hat Linux und Fedora Core ist `vi` ein Alias auf `vim`.
[34] `vim` und `gvim` sind auf den meisten anderen Unix Derivaten nicht vorhanden.

- last line mode
  für komplexere Kommandos, die in der letzten Zeile editiert werden.

Am Anfang befindet man sich immer im command mode. Mit [:] ⟨`Befehl`⟩ wechselt man in den last line mode und führt dort einen Befehl aus. Vom input und last line mode kommt man mit [ESC] wieder in den command mode.

Die Navigation erfolgt im command mode. Zwar kann man dafür im `vim` unter Linux auch die Richtungstasten verwenden, aber auf anderen Unix Systemen oder beim Zugriff über `telnet` funktioniert das oft nicht.

Verwenden Sie stattdessen die normale Tastatur im command mode. Das hat den Vorteil, dass Sie, wenn Sie sich einmal daran gewöhnt haben, schneller arbeiten können. Außerdem kommen Sie damit nie in Gefahr, eines Tages ratlos vor einem Computer zu stehen, der die Richtungstasten unter `vi` nicht unterstützt.

- [h] bedeutet 1 Buchstabe nach links.
- [j] steht für eine Zeile nach unten.
- [k] bringt Sie eine Zeile nach oben.
- [l] bedeutet 1 Buchstabe nach rechts.
- [b] bringt sie ein Wort nach links.
- [w] steht für ein Wort nach rechts.
- [^] bringt Sie an den Zeilenanfang.
- [$] wechselt zum Zeilenende.

Die wichtigsten Befehle des command mode sind:

- [i] wechselt in den input mode (Zeichen werden an der aktuellen Cursorposition eingegeben).
- [a] wechselt in den input mode (Zeichen werden nach der aktuellen Cursorposition eingegeben).
- [A] wechselt in den input mode (Zeichen werden am Ende der Zeile angehängt).
- [R] wechselt in den input mode (überschreibt den alten Text).
- [r] wechselt zum Überschreiben eines Zeichens in den input mode.
- [s] wechselt in den input mode (das Zeichen, auf dem der Cursor steht, wird durch die Eingabe überschrieben und durch beliebig viele neue ersetzt).

- C wechselt in den input mode (der Rest der Zeile wird durch den neuen Text ersetzt).
- o wechselt in den input mode (nach der aktuellen Zeile wird eine neue Zeile eingefügt).
- O wechselt in den input mode (vor der aktuellen Zeile wird eine neue Zeile eingefügt).
- x löscht das aktuelle Zeichen.
- d d löscht die aktuelle Zeile.
- d w löscht bis zum Ende des aktuellen Worts.
- c w wechselt in den input mode (der Rest des aktuellen Worts wird durch die Eingabe überschrieben).
- u nimmt das letzte Kommando zurück.
- y kopiert die angegebenen Zeichenketten.
- J hängt die folgende Zeile an die aktuelle an.
- . wiederholt das letzte Kommando.
- : wechselt in den last line mode.
- / sucht nach der angegebenen Zeichenkette.
- n wiederholt die letzte Suche.
- x löscht das aktuelle Zeichen.

Allen Kommandos (auch denen zum Bewegen des Cursors!) kann eine Zahl vorangestellt werden, die angibt, auf wieviele Objekte sich der folgende Befehl beziehen soll. So können durch Eingabe von 3 d w (oder d 3 w) drei Wörter auf einmal gelöscht werden. Durch Eingabe von 1 0 x erreicht man das Löschen von zehn Zeichen ab der Cursorposition, 2 0 d d (oder d 2 0 d) löscht 20 Zeilen.

Die wichtigsten Befehle des last line mode:

- : q beendet `vi`, falls alle Änderungen gespeichert wurden.
- : q ! verlässt den `vi`, ohne Änderungen zu speichern.
- : w ⟨`Datei`⟩ speichert unter ⟨`Datei`⟩; wurde `vi` auf der Kommandozeile mit einer bestimmten Datei aufgerufen, kann man ⟨`Datei`⟩ weglassen.
- : x speichert die geänderte Datei und verlässt den Editor.
- : e ⟨`Datei`⟩ editiert die Datei ⟨`Datei`⟩.

- : e ! verwirft alle nicht gespeicherten Änderungen.

`vi` (bzw. das unter Linux verwendete verbesserte `vim`) hat gute Hilfetexte, die sich unter `/usr/share/doc/vim*` befinden.[35] Hier noch ein paar Möglichkeiten, wie sie `vi` einsetzen können:

Um Wörter zu suchen, drücken Sie im command mode /. Dann geben Sie die die zu suchenden Zeichen ein. Achtung! `vi` verwendet zum Suchen *regular expressions*. Diese werden im Abschitt 13 ausführlicher behandelt. Vorerst sollten Sie wissen, dass Zeichen wie „`;`“, „`.`“ oder „/“ eine spezielle Bedeutung haben. Sie müssen mit \ von dieser speziellen Bedeutung „befreit“ werden. Um z.B. `/root` zu suchen, geben Sie im command mode ein:

```
/\/root
```

/ beginnt die Suche. Nun wollen Sie `/root` suchen. Da aber / eine spezielle Bedeutung hat, müssen Sie `vi` mitteilen, dass es dieses Zeichen hier als normales Zeichen interpretieren soll. Dazu verwenden Sie das vorangestellte \.

Wollen Sie nun z.B. ein Wort suchen, das mit `Unix-` beginnt, dann ein paar andere Zeichen hat und schließlich von `Linux` gefolgt wird, würden Sie tippen:

```
/Unix-.*Linux
```

Im Gegensatz zur Shell ist * hier nicht die bekannte Wildcard . * gibt an, dass der vorangegangene Buchstabe beliebig oft vorkommen kann. Da davor eine Wildcard steht („`.`“ steht für ein beliebiges Zeichen, wie „?“ als `bash` Wildcard), kann jedes beliebige Zeichen beliebig oft vorkommen.

Wollen Sie z.B. nach einer Zeile suchen, die mit `Unix` beginnt und mit `Linux` endet, so geben Sie ein:

```
/^Linux.*Unix$
```

Das ^ Zeichen steht für den Zeilenanfang, das $ Zeichen für das Zeilenende.

Wahrscheinlich wollen Sie Zeichen nicht nur suchen, sondern auch ersetzen. Dazu wechseln Sie mit : in den last line mode. Wollen Sie z.B. `/home/c102/c102mr` durch ~ / ersetzen, geben Sie ein:

```
:%s/\/home\/c102\/c102mr/\~/gc
```

Die allgemeine Syntax lautet:

```
:%s/⟨zu ersetzende Zeichen⟩/⟨Zeichen⟩/⟨Optionen⟩
```

Nun zum obigen Beispiel: es beginnt mit `:%s/` — wie in der allgemeinen Syntax angegeben. Dann folgt `/home/c102/c102mr` als zu ersetzende Zeichenkette und ~ als ersetzende Zeichenkette. Die Zeichen / und ~ wurden mit \ von ihrer speziellen Bedeutung befreit. „g“ und „c“ sind schließlich 2 Optionen. „g“ bewirkt, dass nicht nur das erste Vorkommen in einer Zeile gesucht und ersetzt wird. Durch „c“ werden Sie bei jedem Ersetzen um Bestätigung gefragt.

[35] Auf anderen Linux Distributionen und Unix Derivaten ist dies wahrscheinlich ein anderes Verzeichnis.

Nun zu einem etwas weniger schwierigen Beispiel: Nehmen wir an, Sie wollen einen Text kopieren und später wieder wo einfügen. Eine Katastrophe, oder? Wie soll das mit `vi` gehen? Nun — für alle, die es gewohnt sind, einfach eine Stelle zu markieren und dann Strg – C zu drücken, mag es schwierig sein, aber im Endeffekt ist man mit `vi` weit flexibler.
Es ist gar nicht so schwer, man muss nur die Bedeutung von Bereichen für `vi` verstanden haben.

Wechseln Sie in den command mode und gehen Sie zu der Stelle, wo Sie kopieren möchten. Drücken Sie nun y und wählen Sie mit dem entsprechenden Tastaturkürzel den Bereich, den Sie kopieren möchten — hier einige Beispiele:

- y 1 w kopiert 1 Wort (Achtung: „/“ „,“ oder „.“ werden als Worttrennzeichen gezählt!).
- y 2 w kopiert 2 Wörter.
- y 1 W kopiert 1 Wort (bis zum nächsten Leerzeichen, ein „.“ wird also beispielsweise mitkopiert).
- y 3 l kopiert 3 Zeichen (nach rechts).
- y $ kopiert bis zum Zeilenende.
- y ^ kopiert zurück bis zum Zeilenanfang.
- y 3 $ kopiert die aktuelle Zeile ab der aktuellen Position und die 2 folgenden.
- y 2 j kopiert die aktuelle Zeile und die beiden folgenden.
- y 2 h kopiert die aktuelle Zeile und die beiden vorhergehenden.
- y t a kopiert ab der aktuellen Position bis zum nächsten auftreten des Zeichens `a` in der aktuellen Zeile.

Mit dieser Methode können Sie, wenn Sie sie einmal durchschaut haben, sehr viel schneller und bequemer arbeiten als mit jedem anderen Editor.

Zum Abschluss noch ein kleiner Trost: am Anfang ist das Erlernen von `vi` wirklich etwas schwierig. Aber wenn man sich einmal eingearbeitet hat, ist man schneller als mit jedem anderen Editor und würde ihn auch gegen keinen anderen eintauschen. So wurde etwa dieser Text mit `vi` erstellt.

7.1 Aufgaben

1. Wie rufen Sie im `vi` die zugehörige Hilfe auf?

2. Experimentieren Sie mit dem `vi`. Öffnen Sie Dateien, ändern Sie diese, probieren Sie möglichst viele Befehle aus.

3. Beantworten Sie die Fragen der nachfolgenden Abschnitte mit dem `vi` als Editor.

Die Lösungen finden Sie auf Seite 126.

8 Prozessmanagement

As in certain cults it is possible to kill a process if you know its true name.

(Ken Thompson and Dennis M. Ritchie)

Wir haben bisher immer von Programmen gesprochen, ohne auf Details einzugehen, wie diese verwaltet werden.
Jedes „Programm“ besteht aus einem oder mehreren sogenannten Prozessen. Jedem Prozess wird beim Start eine eindeutige Nummer, die *PID* (*Process Identification*) zugeordnet.

8.1 Prozesse

Eine Liste aller laufenden Befehle erhalten Sie mit dem Befehl

`ps aux | less`[36]

Hier eine verkürzte Ausgabe:

```
USER  PID %CPU %MEM  VSZ  RSS TTY   STAT START TIME COMMAND
root    1  0.1  0.1 1368   72 ?     S    22:26 0:04 init [5]
...
c102 1039  0.0  4.1 5544 2600 ?     S    22:44 0:00 xterm -ls
c102 1041  0.0  2.2 2420 1388 pts/3 S    22:44 0:00 -bash
c102 2403  0.0  1.2 2632  788 pts/1 R    23:04 0:00 ps aux
```

In der ersten Spalte sehen Sie den Benutzer, der den Befehl gestartet hat, gefolgt von der PID. Die nächsten beiden Spalten zeigen an, wieviel Prozent Prozessorzeit beziehungsweise Arbeitsspeicher der Prozess im Moment verwendet. Die restlichen Zeilen wollen wir vorerst einmal vergessen — wie immer können Sie diese mit `man ps` nachschlagen, ebenso wie die möglichen Optionen zu `ps` (*process status*).

Ein verwandter Befehl ist **`pstree`**[37] — er zeigt, welches Programm von welchem Programm aufgerufen wurde.

```
init-+-apmd
     |-...
     |-xterm---bash---bash
     |              |-pstree
     ...
```

An der Wurzel befindet sich das Programm `init`, das beim Systemstart des Rechners als erstes ausgeführt wird und für das Starten der restlichen verantwortlich ist. Weit unten sehen Sie, dass ein `xterm` gestartet wurde, das seinerseits als Shell die `bash`

[36] Auf vielen Unix Systemen wird `-ef` statt `aux` verwendet
[37] Dieses Programm ist nicht auf allen Linux– und Unix–Systemen vorhanden.

startete. In dieser wurde nun schließlich `pstree` ausgeführt. Jeder Prozess erbt die Umgebung des aufrufenden Prozesses; dazu später mehr.

`top` (Abb. 15) funktioniert wie `ps`, ist allerdings interaktiv.

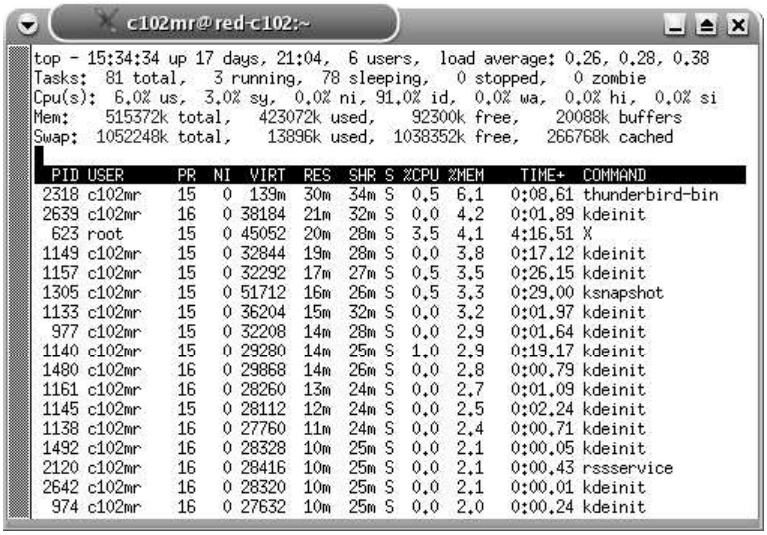

Abbildung 15: Die Anzeige von `top`.

In konfigurierbaren Abständen wird die Prozessliste auf den aktuellen Stand gebracht. Sie können die Liste nach verschiedenen Kriterien sortieren, z.B. nach Arbeitsspeicherbedarf (mit M) oder nach CPU–Verwendung (mit P). Mit `top` könen Sie auch Prozesse beenden — siehe dazu unten.[38]

Ähnlich funktioniert die KDE-Systemüberwachung (kpm), die in Abbildung 16 gezeigt wird. Sie hat zusätzlich eine grafische Oberfläche und ist einfacher zu bedienen. So kann man einen Prozess durch Anklicken mit der rechten Maustaste manipulieren.

Abbildung 16: Die KDE-Systemüberwachung (`kpm`).

Oft wollen Sie nur nach einem bestimmten Prozess suchen. Dies ist einfach durch die Verknüpfung bereits bekannter Befehle möglich, z.B.:

[38] Die Bedienung von `top` über die Tastatur ist bei den einzelnen Unix Systemen unterschiedlich.

ps auxw | grep xterm

Ähnlich funktioniert es, wenn Sie nur Ihre Befehle sehen wollen, zum Beispiel:

ps auxw | grep c102mr

Sie können laufende Prozesse auch beenden. Bei Kommandozeilenprogrammen ist das durchaus üblich, bei grafischen Applikationen sollte man das mit Vorsicht verwenden. Um beispielsweise ein `xterm` zu beenden, würde ich es, wie oben beschrieben, zuerst mit

ps auxw | grep xterm

suchen. Ich erhalte eine Zeile wie folgt zurück:

```
c102mr 940 0.0 2.7 59001684 ?   S  Apr13 0:01 xterm -ls
```

Nun kann ich das `xterm` mit folgendem Befehl beenden:

kill 940

`940` ist die Prozessnummer, die PID des Prozesses.

Manchmal kann es vorkommen, dass ein Prozess hängen bleibt, nicht mehr reagiert und mit einem normalen `kill` nicht mehr beendet werden kann. In diesem Fall kann man den Prozess immer noch mit

kill -9 [PID]

beenden. Das hat einen entscheidenden Nachteil — der Prozess beziehungsweise das betroffene Programm wird unmittelbar beendet, hat keine Zeit mehr, beispielsweise Daten oder die Einstellungen zu speichern.

8.2 Jobs

Nehmen wir an, Sie haben ein `xterm` gestartet. In diesem starten Sie nun mit `xterm -ls` ein weiteres X Terminal. Nun können Sie zwar im zweiten schreiben, das erste ist aber blockiert. Sie können das zweite `xterm` unterbrechen, indem Sie im ersten `xterm` Strg – Z drücken. Wenn Sie nun im zweiten `xterm` versuchen, etwas einzugeben, funktioniert das nicht. Sie haben das Programm ja suspendiert (mit Strg – C hätten Sie es übrigens abgebrochen. Das entspricht in etwa einem `kill`). Mit `bg` (*background*) können Sie den `xterm` Prozess *im Hintergrund* weiterlaufen lassen. Mit `fg` (*foreground*) können Sie den Prozess wieder in den Vordergrund holen.

Sie können ein Programm auch gleich in den Hintergrund stellen, indem Sie danach ein „`&`“ eingeben, zum Beispiel:

xterm -ls &

Wiederum können Sie es mit `fg` in den Vordergrund holen.

Programme, ob im Vorder– oder Hintergrund, bezeichnet man aus Sicht der Shell als *Jobs*. Wenn Sie mehr als ein Programm im Hintergrund gestartet haben, können Sie diese mit dem Befehl

jobs

auflisten. Die Ausgabe sieht wie folgt aus:

```
[1]   Running                 emacs linux.tex &
[2]+  Stopped                 less /etc/fstab
[3]-  Running                 xterm -ls &
```

In eckigen Klammern sehen Sie die Jobnummer. Danach folgt bei einem Prozess ein „+“ — das ist der aktuelle Job; der Jobnummer des vorigen Prozesses folgt ein „–“. Danach wird der Status des Jobs angezeigt. Obiges Beispiel zeigt 2 laufende und 1 gestoppten Prozess. Schließlich folgt die Kommandozeile, mit welcher der Job gestartet wurde.
Wenn Sie nun `fg` eingeben, wird der aktuelle Job in den Vordergrund geholt. Jeden Job können Sie über die Prozessnummer ansprechen:

fg %2

würde beispielsweise den zweiten Prozess in den Vordergrund holen.
Sie können `kill` auch auf die Jobnummer anwenden. So würde

kill %1

den ersten Job beenden.

Ein Job wird meist beendet, wenn Sie den Mutterprozess beenden. Geben Sie beispielsweise ein:

less ~/liesmich

Mit Strg – Z können Sie diesen Prozess nun unterbrechen. Wenn Sie nun das `xterm` mit `exit` (oder Strg – D) beenden wollen, bekommen Sie die Meldung:

```
There are stopped jobs.
```

Mit einem weiteren `exit` können Sie zwar trotzdem das `xterm` beenden, `less` wird damit aber auch beendet.

Wenn Sie ein Programm starten wollen, das auch weiterlaufen soll, wenn die Shell beendet wurde (beispielsweise bei länger laufenden Berechnungen), verwenden Sie den Befehl `nohup` (*no hangup*):

nohup ⟨**Befehl**⟩ **&**

Die Ausgabe des Befehls `nohup` wird in die Datei `nohup.out` geschrieben. Sie können den Mutterprozess (z.B. Ihr `xterm`) beenden, Ihr Prozess läuft dennoch weiter.

8.3 Nettigkeiten

Oft werden Prozesse gestartet, die lange laufen und viel CPU brauchen. Das stört, speziell bei langen Laufzeiten, die anderen Programme (und Benutzer). Sie können

festlegen, mit welcher Priorität Ihre Programm ausgeführt wird, wieviele der CPU Zyklen es bekommt. Die möglichen Werte reichen von -19 bis $+19$. Je kleiner die Zahl, desto vordringlicher wird der Prozess behandelt. Prioritäten kleiner als 0 können von normalen Benutzern nicht vergeben werden, nur von `root`. Die Standard–Priorität ist 0.

Um einen Prozess mit Priorität 5 zu starten, verwenden Sie

`nice -n 5` ⟨`Befehl`⟩

Die Priorität eines Prozesses können Sie auch im Nachhinein ändern (allerdings nur nach oben, d.h. auf weniger „wichtig“, wenn Sie nicht `root` sind):

`renice 5` ⟨`PID`⟩

Dies erhöht den *nice* Wert (beziehungsweise senkt die Priorität) um 5.

8.4 Kommandoverkettung

Bisher haben wir immer nur ein Progamm pro Eingabezeile gestartet. Es geht aber auch anders. Sie können mehrere Programme hintereinander starten, indem Sie diese durch einen Strichpunkt trennen:

```
ls ; echo "ls fertig"
```

führt zuerst den Befehl `ls` aus. Wenn dieser beendet ist, wird der zweite ausgeführt — egal, ob der erste erfolgreich war oder nicht.

Ebenso können Sie das schon besprochene „`&`“ verwenden:

```
ls & echo "ls läuft"
```

Damit wird `ls` gestartet und in den Hintergrund gestellt, dann wird sofort `echo` gestartet (noch bevor `ls` fertig ist).

Jeder Befehl hat einen *exit status*. Wurde ein Befehl erfolgreich ausgeführt, so ist der *exit status* 0, andernfalls ist er ungleich 0 (je nach Fehler). Den *exit status* des jeweils zuvor ausgeführten Programmes können Sie übrigens mit folgendem Befehl abfragen:

```
echo $?
```

Um nun einen Befehl nur dann auszuführen, wenn der vorige erfolgreich war, verwendet man „`&&`“:

```
ls lies* && echo "ls war erfolgreich"
```

Das Gegenteil — die Ausführung, wenn ein Befehl nicht erfolgreich war — erreicht man mit „`||`“:

```
ls lies* || echo "ls war nicht erfolgreich"
```

Man kann das (natürlich) auch verknüpfen:

```
ls lies* && echo "erfolgreich" || echo "nicht erfolgreich"
```

Ist der erste Befehl erfolgreich, so wird der zweite ausgeführt, andernfalls der dritte.

8.5 Etwas Geschichte

Die *history*, die Liste der zuletzt ausgeführte Befehle wird in der `bash` in der Datei `.bash_history` in Ihrem HOME Verzeichnnis gespeichert. Mit dem Befehl

`history`[39]

bekommt man diese aufgelistet. Links steht die History–Nummer des Befehls, rechts die Aufrufzeile. Mit

`!`⟨**`history Nummer`**⟩

kann man einen Befehl aus der History wiederholen. Den zuvor ausgeführten Befehl kann man mit

`!!`

wiederholen, den vorletzten mit

`!-2`

Will man den letzten Befehl wiederholen, der mit „`xt`“ begonnen hat, so kann man folgende Zeile verwenden:

`!xt`

Man kann auch den letzten Befehl mit Änderungen wiederholen. Hat man den Befehl

`ls liesmich`

ausgeführt, so kann man durch die Eingabe von

`^liesmich^README^`

den folgenden Befehl ausführen:

`ls README`

8.6 Aufgaben

1. Zeigen Sie mit `top` nur jene Prozesse an, die unter Ihrer Benutzerkennung laufen.
2. Finden Sie mit `top` die Prozessnummer von `top` selbst heraus und beenden Sie das Programm mit einem `kill` Signal.
3. Suchen Sie mit `ps` den `xterm` Prozess heraus und beenden Sie ihn mit `kill`.
4. Welche `kill` Signale gibt es (`man 7 signal`)?[40]
5. Welchen Signalnamen können Sie statt 9 (z.B. in `kill -9`) schicken?

[39] Die Verwendung des `history` Mechanismus unterscheidet sich je nach verwendeter Shell.

[40] Bei anderen Unix Implementationen befindet sich diese Beschreibung oft in einer anderen Hilfeseite. Suchen Sie gegebenenfalls nach dieser mit **`man -k signal`**.

6. Was bewirkt das Signal `SIGTSTP`? Öffnen Sie mit `less` eine Datei und schicken Sie mit `kill` dieses Signal an den `less` Prozess. Welche Methode haben wir bereits kennengelernt, die dieselbe Wirkung hat?

7. Welchen *exit status* liefert `ls ~/notthere || true`? Warum?

8. Durchsuchen Sie die `history` nach allen `man` Befehlen.

Die Lösungen finden Sie auf Seite 127.

9 Informationen über den Systemzustand

I didn't do it, I wasn't there, and the sheep are lying!

Nun sollten Sie schon einige Aspekte von Linux kennengelernt haben. Vielleicht wollen Sie es auch einmal wagen, einen Blick auf Linux bzw. ihren Computer zu werfen. Hier helfen Befehle wie `df`, `free`, `vmstat` und die Dateien im Verzeichnis `/proc`. Aber der Reihe nach:

Mit **`df`** (*disk free*) können Sie feststellen, wieviel Platz noch verfügbar ist. Die Ausgabe sieht ungefähr wie folgt aus:[41]

```
Filesystem     1k-blocks     Used Available Use% Mounted on
/dev/hda5        2213668   205524    459680   9% /
/dev/hda2          31111     3886     25619  13% /boot
/dev/hda6        1748494  1271898    386232  77% /home
```

Die erste Spalte bezeichnet das Dateisystem. Es ist in etwa mit einer Partition unter Windows vergleichbar. Die zweite, dritte und vierte geben an, wieviele Kilobyte (1024 Kilobyte = 1 Megabyte) insgesamt vorhanden, schon gebraucht und noch frei sind. Als `Use%` wird jene Prozentzahl bezeichnet, die schon belegt ist. Zum Schluss wird noch angegeben, wo das Dateisystem eingebunden ist. Im Gegensatz zu Windows gibt es ja keine Laufwerke. Vielmehr werden die Dateisysteme in den Verzeichnisbaum eingebunden. Im obigen Beispiel ist die Partition `/dev/hda6` unter `/home` eingebunden.

`free`[42] zeigt die Auslastung des Arbeitsspeichers und der Swap Partition an.

```
             total       used       free     shared buffers     cached
Mem:        257480     229772      27708      33488      17948     161740
-/+ buffers/cache:      50084     207396
Swap:       979956      14468     965488
```

In diesem Beispiel hätte das System 256 MB Arbeitsspeicher (unter `total` wird in Kilobyte etwas weniger angezeigt, da der Bereich, den der Kernel selbst belegt, schon abgezogen ist). Außerdem wären fast 1 GB *Swap* vorhanden. Als *Swap* bezeichnet man jenen Bereich der Festplatte, der für das Auslagern von Teilen, die im Moment nicht verwendet werden, reserviert ist; das ist zwar wesentlich langsamer als der Arbeitsspeicher, zum Auslagern gerade nicht benötigter Teile reicht es aber.

Mit `vmstat` (*virtual memory status*)[43] kann man sich die Systemstatistiken ansehen, z.B. wieviel CPU Zeit im Moment von allen Prozessen verbraucht wird, ob und in

[41] Bei vielen Unix Systemen müssen Sie die Option `-k` verwenden, um die Ausgabe in Kilobytes zu erhalten.

[42] Dieser Befehl ist Linux–spezifisch.

[43] Auf vielen Unix Systemen ist dieser Befehl nicht vorhanden.

welchem Umfang gerade auf die Festplatte geschrieben wird usw. Am Besten rufen Sie es mit

`vmstat 1`

auf. Nun läuft `vmstat` ständig, wobei jede Sekunde eine neue Ausgabe erfolgt. So sieht man z.B. auch, zu welchem Zeitpunkt ein Programm in welchem Umfang Arbeitsspeicher beansprucht. Sie beenden die Ausgabe, indem Sie Strg – C drücken.

Das `/proc` Dateisystem ist ein virtuelles Dateisystem.[44] Es stellt eine Abbildung des Kernel Zustandes dar. Dieses Dateisystem belegt keinen Festplattenplatz. Hier befinden sich verschiedene Dateien, die nützliche Informationen über das System enthalten. Einige Beispiele:

Mit **`cat cpuinfo`** werden Informationen über die CPU des Computers angezeigt.

In `meminfo` stehen Statistiken zum Arbeitsspeicher. Diese Information wird — in überarbeiteter Form — auch von `free` verwendet.

In `pci` finden Sie Details über die Hardware des Systems, z.B. über die verwendete Grafikkarte.

9.1 Aufgaben

1. Versuchen Sie, mit Hilfe der `man page` von `vmstat` dessen Ausgabe zu interpretieren.
2. Schauen Sie sich die Datei `/proc/pci` an. Was erkennen Sie?

Die Lösungen finden Sie auf Seite 128.

[44] `/proc` ist nur auf wenigen Unix Systemen vorhanden. Selbst wenn es existiert, kann sich der Inhalt stark von dem unter Linux unterscheiden.

10 Shell Variablen

The most likely way for the world to be destroyed, most experts agree, is by accident. That's where we come in; we're computer professionals. We cause accidents.

(Nathaniel Borenstein)

Die Shells unterstützen, wie alle „modernen" Programmiersprachen, die Verwendung von Variablen. Diesen kann ein beliebiger Wert zugewiesen und später wieder abgefragt werden. Im Gegensatz zu Programmiersprachen wie C gibt es keine Typisierung, d.h. einer Variable kann ein beliebiger Datentyp — ob Zahl oder Zeichenkette — jederzeit zugewiesen werden, beziehungsweise wird diese standardmäßig nicht verwendet. Weiters ist nicht zwischen Deklaration und Definition zu unterscheiden (eine Variable ist verfügbar, sobald sie verwendet wird). Ein einfaches Beispiel:

```
HALLO="Hallo Welt"
export HALLO
echo $HALLO
```

Die erste Zeile ordnet der Variablen `HALLO` den Wert `Hallo Welt` zu. Die Variable und der zuzuweisende Wert sind durch das Gleichzeichen verknüpft; dazwischen darf kein Leerzeichen stehen. Da die Zeichenkette ein Leerzeichen enthält, muss sie unter Anführungszeichen gestellt werden. Damit ist die Variable innerhalb der Shell verfügbar. Soll die Variable auch in aus der Shell aufgerufenen Programmen verfügbar sein, so muss sie exportiert werden (man bezeichnet das als „Exportieren in die Prozessumgebung"). Dies geschieht durch die zweite Zeile.
Die dritte Zeile schließlich greift auf den Wert der Variable zu. Dazu wird dieser ein „`$`" vorangestellt. Diesen Vorgang, die Substitution der Variable durch ihren Wert, bezeichnet man als *Variablen–Expandierung*.

Variablen können auch den Ausgabewert von Programmen beinhalten. Dafür gibt es zwei Varianten:

```
LS=`ls`
LS=$(ls)
```

Die erste Variante, die nach links geklappten einfachen Anführungszeichen, sind in den meisten Shells verfügbar. Erst seit Version 2 der `bash` (und schon länger in der der `ksh`) ist die zweite Variante verfügbar. Sie ist in der Verwendung unproblematischer und sollte deshalb bevorzugt werden. Beide Varianten machen dasselbe. Sie rufen `ls` auf und speichern dessen Ausgabe in der Variablen `LS`.

Variablen müssen mit einem Buchstaben beginnen. Danach können als Zeichen Buchstaben, Zahlen und der Unterstrich verwendet werden. Beachten Sie, dass auch bei den Variablen zwischen Groß– und Kleinschreibung unterschieden wird.

Es gibt eine Reihe besonderer Shell Variablen, über die die Arbeitsumgebung definiert ist beziehungsweise eingestellt werden kann, zum Beispiel:

- $DISPLAY
  Setzt fest, wo grafische Applikationen angezeigt werden sollen. Da die grafische

Oberfläche netzwerktauglich ist, kann das auch ein anderer Rechner sein. Wenn man vor dem Rechner sitzt (im Gegensatz zum Arbeiten über eine Netzwerkverbindung), so ist diese Variable auf `:0` gesetzt.

- $HOME
  Bezeichnet das HOME Verzeichnis eines Benutzers.

- $PATH
  Dies listet, durch Doppelpunkte getrennt, den Suchpfad für Programme auf. Wenn Sie einen Befehl eingeben, wird er in der Reihenfolge in den Verzeichnissen gesucht, wie sie hier von links nach rechts aufgelistet sind.

- $PS1
  Beschreibt, wie der Prompt der Shell aussehen soll — das ist die Zeichenkette, die angezeigt wird, wenn Sie einen Befehl eingeben können (zum Beispiel „`[c102mr@mycomputer c102mr]$ `“).

- $PWD
  Beinhaltet das aktuelle Verzeichnis.

- $SHELL
  Zeigt an, welche Shell verwendet wird, z.B. „`/bin/bash`“.

- $USER
  Zeigt den Benutzernamen des aktuellen Benutzers.

10.1 Aufgaben

1. Lassen Sie sich die `DISPLAY` Variable anzeigen und versuchen Sie dann, Ihren Wert zu ändern. Was passiert nun, wenn Sie mit `xterm -ls` ein neues X Terminal aufrufen wollen?

2. Ändern Sie die Variable `PS1`. Was geschieht? Lesen Sie dazu in der Hilfeseite der `bash` nach.

3. Interpretieren Sie den Inhalt der Variable `PATH`. Fügen Sie ein neues Verzeichnis hinzu.

4. Wie muss der Inhalt von PATH aussehen, damit auch im aktuellen Verzeichnis nach Programmen gesucht wird?

Die Lösungen finden Sie auf Seite 129.

11 Quoting

C makes it easy for you to shoot yourself in the foot. C++ makes that harder, but when you do, it blows away your whole leg.

(Bjarne Stroustrup)

Die Shell kennt eine Reihe von Sonderzeichen:

`*,?,!,[,],&,>,<,|,(,), ,",',{,},\`

Normalerweise interpretiert sie diese Zeichen (zum Beispiel als Wildcard bei `*`). Es gibt jedoch Möglichkeiten, dies zu verhindern, d.h. die Zeichen vor der Interpretation durch die Shell zu schützen.

Eine Möglichkeit ist es, dem Zeichen einen Backslash \ voranzustellen. Ähnliches haben wir schon beim Editor `vi` gesehen, wenn auch nicht näher besprochen. Wenn Sie beispielsweise eine Datei mit Leerzeichen namens `lies mich` öffnen wollen, interpretiert die Shell das Leerzeichen als Trennzeichen von Programmargumenten — daher ist hier ein Quoting notwendig:

```
less lies\ mich
```

Speziell wenn Sie mehrere Sonderzeichen verwenden, wird das aber mühsam. Eine weitere Methode ist die Verwendung von doppelten Anführungszeichen. Diese schützen alle Zeichen außer

!,",\

Die Datei `liesmich?` könnten Sie also wie folgt anzeigen:

```
less "liesmich?"
```

Am wirksamsten und radikalsten werden die Sonderzeichen mit einfachen Anführungszeichen (`'`) geschützt. Innerhalb dieser erfolgt keine Interpretation der Sonderzeichen (nur von `'` selbst — es wird als Ende des Quotings interpretiert). Das bedeutet aber auch, dass beispielsweise Variablen nicht mehr expandiert werden. Sollten Sie eine Datei `$liesmich` erstellt haben, so können Sie mit einfachen Anführungszeichen schützen:

```
less '$liesmich'
```

Eine spezielle Form des Quotings haben wir weiter oben schon kennengelernt, und zwar beim Befehl `find`. Dieses Programm interpretiert die Zeichen selbst. Daher müssen alle Sonderzeichen geschützt werden; ansonsten werden sie bereits von der Shell interpretiert, z.B.:

```
find . -name '[Ll]i?smi*'
find . -name "[Ll]i?smi*"
find . -name \[Ll\]i\?smi\*
```

Alle drei obigen Beispiele liefern dasselbe Ergebnis: Die Sonderzeichen werden durch Quoting geschützt.

11.1 Aufgaben

1. Was ist der Unterschied (beziehungsweise gibt es einen) zwischen den folgenden Befehlen:

   ```
   ls ?iesmich
   ls "?iesmich"
   ls '?iesmich'
   ```

2. Was bewirkt der folgende Befehl:

   ```
   ls "lies mic"?
   ```

Die Lösungen finden Sie auf Seite 130.

12 Shell Scripts 1

Those who do not understand Unix are condemned to reinvent it, poorly.

(Henry Spencer)

12.1 Einleitung

Die Unix Shells, und damit auch die `bash`, bieten umfangreiche Möglichkeiten, kleine Programme zu schreiben und Aufgaben zu automatisieren. Diese *Shell Scripts* können entweder direkt auf der Kommandozeile eingegeben werden (das haben wir — wenn auch nicht explizit so bezeichnet — schon die ganze Zeit getan) oder werden zuerst in eine Datei geschrieben und dann ausgeführt. Wir werden hier die zweitere Methode verwenden (auch wenn alles Besprochene auf die Kommandozeile ebenso zutrifft).

Dieser Abschnitt ist keine systematische, sondern vielmehr eine logische Einheit: Alle bisher vorgestellten Befehle sind auch hier gültig, ebenso die nachfolgenden. Dasselbe gilt auch umgekehrt, die hier vorgestellten Befehle können ebenso auf der Kommandozeile ausgeführt werden.
Das Folgende dient im Wesentlichen dazu, das Konzept des Shell Scriptings zu verdeutlichen und in den Kontext des bisher Gesehenen zu stellen.

Shell Scripts sind einfache Text Dateien, die über die Shell ausgeführt werden.

Die erste Zeile des Scripts sieht wie folgt aus:

```
#!/bin/sh
```

Diese Zeichenkette, `#!`, bezeichnet man als *Shebang*. Sie zeigt an, dass der nachfolgende Teil mit einem Programm ausgeführt werden soll — hier ist es `/bin/sh`, die Shell.

Danach folgen ganz normale Unix Kommandos, wie wir sie bereits oben verwendet haben. Die meisten Konstrukte haben wir schon kennengelernt. Einige weitere, besonders Kontrollstrukturen, werden wir im Folgenden kennenlernen.

Sie können in ein Script auch Kommentare einbinden. Diese beginnen mit dem Zeichen „#“ und reichen bis zum Ende der Zeile.[45]

Wir wollen mit einem ganz einfachen Shell Script beginnen. Dieses, wie könnte es anders sein, gibt die Meldung `hello world` auf dem Bildschirm aus. Erstellen Sie eine Datei `hello.sh` mit folgendem Inhalt:

```
#!/bin/sh

# das ist ein Kommentar
echo "hello world"
```

[45] Das funktioniert übrigens auch auf der Kommandozeile.

Speichern Sie diese Datei nun und beenden Sie den Editor. Sie können das Script nun ausführen:

```
sh hello.sh
```

Für diese Art der Ausführung hätten wir den *Shebang* nicht benötigt. Wir können aber die Datei ausführbar machen:

```
chmod u+x hello.sh
```

Wissen Sie noch, was das bewirkt? Wenn nicht, lesen Sie oben bei den Datei– und Zugriffsrechten auf Seite 40 nach.
Nun können Sie die Datei direkt als Programm aufrufen:

```
hello.sh
```

Sollte das aktuelle Verzeichnis nicht im Pfad (`PATH`) enthalten sein, wird das Script wie folgt aufgerufen:

```
./hello.sh
```

Erinnern Sie sich noch? „`.`“ steht für das aktuelle Verzeichnis. Obiger Befehl bewirkt also, dass das Script `hello.sh`, das sich im aktuellen Verzeichnis befindet, ausgeführt werden soll.

In Shell Scripts kommt es oft vor, dass ein Befehl länger als der sichtbare Bereich des Editors ist. Sie können die Zeile zwar beliebig lange machen. Der Übersichtlichkeit wegen kann aber hier ein Zeilenumbruch erfolgen. Wird eine Zeile mit einem „\“ beendet, so wird beim Lesen der Datei (oder der Kommandozeile) die nachfolgende Zeile an die aktuelle angehängt.

```
echo "hello world"
```

ist daher identisch mit

```
echo \
"hello world"
```

Das war nicht sonderlich spektakulär. Wir können einmal versuchen, schon verwendete Programme und Methoden in einem Script zu verwenden.

Im Folgenden wird der *Shebang* nicht explizit geschrieben, er ist aber hinzuzufügen; ebenso wird nicht mehr beschrieben, dass eine Datei gespeichert werden und ausführbar gemacht werden muss.

```
ls ~/liesmich
find ~/ -name liesmich
grep "Linux" ~/liesmich
```

Hier werden ganz normale Unix Befehle einfach zu einem Script zusammengefasst (auch wenn es nicht wirklich sinnvoll ist …). Das macht aber noch wenig Sinn. Ein großer Fortschritt ist nicht zu erkennen, irgendwas fehlt …

12.2 test

Wir wollen nun ein Script schreiben, das nachsieht, ob eine Datei vorhanden ist und dann eine entsprechende Meldung ausgibt:

```
test -f ~/liesmich && echo "gefunden" || echo "leider nicht"
```

Dieses Konstrukt haben wir schon oben bei der Kommandoverknüpfung kennengelernt: Abhängig vom *exit status* eines Befehls wird ein folgender ausgeführt oder nicht. Der Status wird übrigens auch in eine Variable, `$?`, geschrieben und ist über diese zugreifbar:

```
test -f ~/liesmich
echo $?
```

`test` überprüft, ob eine Bedingung erfüllt wird. In diesem Fall, aufgrund der Option `-f`, überprüft der Befehl, ob die Datei existiert. Ist das Ergebnis richtig, so ist der Rückgabewert „`0`“, andernfalls „`1`“.

Für `test` gibt es übrigens eine kürzere Schreibweise:

```
[ Testbedingung ]
```

So hätte das vorletzte Beispiel auch so geschrieben werden können:

```
[ -f ~/liesmich ] && echo "gefunden" || echo "leider nicht"
```

`test` Bedingungen können mit „`!`“ negiert werden — etwa um zu überprüfen, ob die Datei `liesmich` *nicht* existiert:

```
[ ! -f ~/liesmich ] && echo "nichtda"
```

Mit `test` kann man auch Ausdrücke verknüpfen. Dieses Beispiel überprüft, ob sowohl die Datei `README` als auch `liesmich` existieren:

```
[ -f ~/liesmich -a -f ~/README ] && echo "beide gefunden"
```

Umgekehrt kann auch geprüft werden, ob einer der beiden Ausdrücke richtig ist:

```
[ -f ~/liesmich -o -f ~/README ] && echo "mind. 1 gefunden"
```

Hier die wichtigsten Parameter von `test` (die gesamte Liste finden Sie wie immer in der man page):

Parameter	Bedeutung
-r *Datei*	Die Datei existiert und ist lesbar
-w *Datei*	Die Datei existiert und ist schreibbar
-x *Datei*	Die Datei existiert und ist ausführbar
-f *Datei*	Die Datei existiert und ist eine reguläre Datei
-d *Verzeichnis*	Das Verzeichnis existiert
-s *Datei*	Die Datei existiert und ist nicht leer
-z *ZK*	Die Zeichenkette ist leer
-n *ZK*	Die Zeichenkette ist nicht leer
ZK1 = *ZK2*	Die beiden Zeichenketten sind gleich
ZK1 != *ZK2*	Die beiden Zeichenketten sind ungleich
n1 op n2	Vergleich von Zahlen

Der letzte Punkt fasst eine Reihe von Operatoren zusammen. Mögliche Werte für *op* sind:

Parameter	Bedeutung
`-lt`	kleiner als
`-le`	kleiner gleich
`-eq`	gleich
`-ne`	ungleich
`-ge`	größer gleich
`-gt`	größer als

Ein einfaches Beispiel:

```
Z1=1
Z2=2

[ "$Z1" -eq "$Z2" ] && echo "die beiden sind gleich"
[ "$Z1" -lt "$Z2" ] && echo "die erste Zahl ist kleiner"
```

Sie sollten Testbedingungen möglichst unter Anführungszeichen stellen, da es sonst zu Problemen kommen kann. Nehmen wir folgendes Beispiel:

```
ZK1="zeichenkette 1"

[ $ZK1 = $ZK2 ] && echo "die Zeichenketten sind gleich"
```

Da `ZK2` nicht gesetzt ist, wird bei der Variablen–Expandierung der `test` Ausdruck zu Folgendem:

```
[ "zeichenkette 1" = ] && echo "die Zeichenketten sind gleich"
```

Das ergibt nun einen *syntax error*, die Shell kann damit nichts anfangen, weil die Zeichenkette, mit der verglichen werden soll, nicht definiert wurde, nicht vorhanden ist:

```
[: ZK1: unary operator expected
```

Hingegen wird dies richtig behandelt, wenn die zu vergleichenden Zeichenketten unter Anführungszeichen stehen. Versuchen Sie's.

12.3 if

Und schon sind wir mitten in dem, was gerne als *flow control* oder *Kontrollstrukturen* bezeichnet wird. Das obige Beispiel war zwar ganz nett, aber ist für komplexere Aufgaben nicht geeignet. Hierzu wird das Konstrukt „`if`“ verwendet. Die allgemeine Syntax lautet:

```
if ⟨Kommandofolge 1⟩
then
      ⟨Kommandofolge 2⟩
```

```
(elif ⟨Kommandofolge 3⟩
      ⟨Kommandofolge 4⟩)
(else
      ⟨Kommandofolge 5⟩)
fi
```

Ein Beispiel:

```
 if [ -f ~/liesmich ]
then
      echo "gefunden"
      ls -l ~/liesmich
elif [ -f ~/README ]
then
      echo "dafür gibt es README"
      ls -l ~/README
else
      echo "nicht gefunden"
fi
```

Das `if` Statement überprüft, ob ein Ausdruck wahr (also „0“) ist. Wenn ja, wird der nachfolgende Teil ausgeführt. Ist er falsch, so wird nun der `elif` Teil (der auch mehrfach vorhanden sein kann) abgearbeitet. Ist dieser (beziehungsweise sind alle diese) auch falsch, dann wird der Block in `else` ausgeführt.
Zum besseren Verständnis des obigen Scripts kann man versuchen, es auf deutsch zu formulieren: „Wenn die Datei `liesmich` existiert, gib `gefunden` aus. Wenn sie nicht existiert, dann schaue, ob die Datei `README` vorhanden ist. Ist diese vorhanden, gib `dafür gibt es README` aus. Ist sie auch nicht vorhanden, dann schreibe `nicht gefunden`.“
Eigentlich braucht man nur den `if` Teil, der Rest ist optional. Ohne die `else` Bedingung würde das Script als so aussehen:

```
if [ -f  /liesmich ]
then
      echo "gefunden"
      ls -l ~/liesmich
elif [ -f ~/README ]
then
      echo "dafür gibt es README"
      ls -l ~/README
fi
```

Wenn man auch noch das `elif` weglässt:

```
if [ -f ~/liesmich ]
then
      echo "gefunden"
```

```
     ls -l ~/liesmich
fi
```

Die `if` Schleife wird oft auch anders geschrieben. Das erste Beispiel würde dann so aussehen:

```
if [ -f ~/liesmich ] ; then
     echo "gefunden"
     ls -l ~/liesmich
elif [ -f ~/README ] ; then
     echo "dafür gibt es README"
     ls -l ~/README
fi
```

Falls Sie nicht mehr wissen, was es mit dem Strichpunkt auf sich hat, lesen Sie bitte oben im Abschnitt Kommandoverknüpfung auf Seite 61 nach. Als kleiner Hinweis: Man hätte das auch so schreiben können:

```
if [ -f ~/liesmich ] ; then
     echo "gefunden" ; ls -l ~/liesmich
elif [ -f ~/README ] ; then
     echo "dafür gibt es README" ; ls -l ~/README
fi
```

Bevor wir zu den weiteren Kontrollstrukturen kommen, noch ein Wort zum Programmierstil:
Wie in allen Programmiersprachen sollte man darauf achten, den Code übersichtlich, strukturiert und auch für andere lesbar zu halten. Das mag zwar bei sehr kurzen Beispielen noch überflüssig erscheinen, wird aber später sehr wichtig — speziell bei verschachtelten Kontrollstrukturen. Gewöhnen Sie sich eine konsistente Einrückung bei Verschachtelung an, strukturieren Sie Ihr Script gut und fügen Sie Kommentare ein, die kurz beschreiben, was gemacht wird.

12.4 case

`if` Strukturen sind zwar sehr nützlich, speziell bei der Überprüfung mehrerer Bedingungen aber etwas umständlich — für diesen Fall gibt es die `case` Anweisung. Die Syntax:

```
case ⟨wort⟩ in
     ⟨m1_1⟩( |⟨m1_2⟩ ))
          ⟨Kommandos⟩
     ( ;; )
     ( ⟨m2_1⟩( |m2_2⟩ ))
          ⟨Kommandos⟩
     ( ;; ))
```

```
      ...
esac
```

Das sieht komplizierter aus als es ist. `case` nimmt eine Zeichenkette und vergleicht sie mit mehreren Möglichkeiten. Für den Fall, der zutrifft, werden Befehle ausgeführt. Ein einfaches Beispiel:

```
ZK="liesmich"

case $ZK in
      liesmich|LIESMICH)
           echo "liesmich gefunden"
      ;;
      READ*)
           echo "beginnt mit READ"
      ;;
      *)
           echo "was anderes"
      ;;
esac
```

In dem Beispiel wird die Variable `$ZK` einfach gesetzt — normalerweise wäre sie wohl durch ein Programm generiert worden (es macht wenig Sinn, eine Variable selbst zu setzen und dann gleich zu überprüfen, worauf man sie gesetzt hat …). Die `case` Struktur prüft in Folge, ob die Zeichenkette `liesmich` *oder* `LIESMICH` ist. Wenn ja, wird mit `echo` eine Meldung ausgegeben. `;;` bewirkt, dass danach ans Ende der `case` Bedingung gesprungen wird, andernfalls würden die anderen Bedinungen auch noch auf ihre Richtigkeit überprüft. Wäre die erste Bedingung nicht richtig, würde in der zweiten geprüft, ob die Zeichenkette mit `READ` beginnt. Schließlich würde die Standard–Bedingung, die auf alles zutrifft, verwendet, falls keine der vorhergehenden zugetroffen hätte.

12.5 while

Mit `while` wenden wir uns nun den sogenannten *Schleifen* zu. Diese führen die in ihnen enthaltenen Kommandos (potentiell) mehrmals aus. Die Syntax von `while`:

```
while ⟨Kommandoliste 1⟩
do
      ⟨Kommandoliste 2⟩
done
```

Ein einfaches Beispiel:

```
zaehler=1
while [ $zaehler -le 10 ]
do
      echo $zaehler
      zaehler=$(($zaehler+1))
done
```

Zuerst wird die Variable `zaehler` gesetzt. Beim Eintritt in die `while` Schleife wird nun überprüft, ob die `test` Überprüfung den *exit code* 0 hat (ob `zaehler` kleiner oder gleich 10 ist). Wenn dem so ist, werden die Befehle in der `while` Schleife ausgeführt. Zuerst wird eine Meldung angegeben, dann wird der Wert der Variable `zaehler` um 1 erhöht — beachten Sie hier die Schreibweise für mathematische Operationen, "`$((`*`ausdruck`*`))`", sie wird auch für Multiplikationen etc. verwendet. Am Ende der Schleife springt die Shell wieder an deren Anfang zurück. Sie überprüft, ob die Bedingung noch zutrifft und führt, wenn das zutrifft, die Schleife wieder aus. Dies geschieht so lange, bis die Bedingung falsch ist (sobald `zaehler` 11 ist). Würden wir den Wert der Variable nicht inkrementieren, würde die Schleife unendlich lange laufen.

12.6 until

`until` entspricht funktionell weitgehend dem `while`, ist aber von der Semantik her entgegengesetzt. Der Inhalt der Schleife wird solange ausgeführt, bis die Überprüfung richtig ist (`while` führt die Schleife solange aus, bis der *exit status* des zu überprüfenden Befehles ungleich 0 ist). Die Syntax:

```
until ⟨Kommandoliste 1⟩
do
      ⟨Kommandoliste 2⟩
done
```

Das obige `while` Beispiel würde daher wie folgt aussehen:

```
zaehler=1
until [ $zaehler -gt 10 ]
do
      echo $zaehler
      zaehler=$(($zaehler+1))
done
```

Die Schleife wird solange durchlaufen, bis `zaehler` gleich 11 und damit größer als 10 ist.

12.7 for

`for` dient dazu, eine Schleife mit allen ihr übergebenen Argumenten zu durchlaufen. Ihre Syntax lautet:

```
for ⟨Name⟩ in ⟨Liste⟩
do
      ⟨Kommandoliste⟩
done
```

Obiges Beispiel würde sich als `for` Konstrukt so schreiben (auch wenn es wenig Sinn macht — obige Lösungen sind eindeutig eleganter):

```
for zahl in 1 2 3 4 5 6 7 8 9 10
do
     echo $zahl
done
```

Die `for` Schleife nimmt ihre Argumente (1 bis 10) und weist für jeden Durchlauf eines der Variable (`zahl`) zu. Zuerst enthält `zahl` den Wert 1, beim nächsten Durchlauf den Wert 2 usw.

Mit Hilfe des Befehls `seq` kann man die `for` Schleife aber doch noch elegant schreiben:

```
for zahl in $(seq 1 10)
do
     echo $zahl
done
```

`seq` nimmt 2 Zahlen als Argument. Es gibt die erste Zahl aus und erhöht diese dann um jeweils 1 (und gibt sie aus), bis die zweite Zahl erreicht ist.

Ein etwas realitätsnäheres Beispiel:

```
for item in "$(ls -a ~)"
do
     case $item in
          lies*)
               echo "schon wieder die liesmich ..."
               continue
          ;;
          READ*)
               echo "hatten wir auch schon - jetzt reicht's!"
               break
          ;;
          *)
               echo "$item - mal was Neues"
          ;;
     esac
done
echo "am Ende der Schleife angelangt"
```

Die meisten Konstrukte hier sollten bekannt sein. So etwa das Ausführen von Befehlen mittels `$()`. Der Ausdruck `$(ls ~)` wird durch seine Ausgabe ersetzt. Was bisher noch nicht explizit erwähnt wurde, ist, dass Ausdrücke beliebig verschachtelt sein können. Hier ist es die `case` Bedingung innerhalb der `for` Schleife. Wir treffen auch auf zwei neue Anweisungen.
`continue` veranlasst die Shell, ans Ende des Schleifenrumpfes zu springen. Die Schleife fährt mit dem nächsten Wert fort. Dadurch werden die restlichen Anweisungen innerhalb der Schleife ignoriert. Mit `break` wird die Schleife beendet. Sowohl `continue` als auch `break` können bei `for`, `while` und `until` Schleifen verwendet werden.

12.8 Mehr Variablen

Es gibt eine Reihe von Variablen, die vor allen in Shell Scripts Sinn machen, weshalb Sie hier besprochen werden.

Variable	Bedeutung
`$0`	Name des Shell Scripts
`$`*x*	Inhalt des *x*ten Parameters
`$#`	Anzahl der Parameter
`$@`	Inhalt aller Parameter
`$?`	Endstatus des letzten Kommandos
`$$`	PID des Scripts

Ein Beispiel–Script, das alle diese Variablen verwendet:

```
echo "Dieses Script heißt $0."
echo "Es wurde mit $# Parametern aufgerufen: $@"
echo "Der erste Parameter ist: $1"
echo "Seine PID ist $$"
echo "Das Programm wird sich jetzt selbst beenden ..."
kill $$
echo "Deswegen wird diese Zeile auch nicht mehr angezeigt."
```

Speichern Sie das Script unter dem Namen `testscript.sh` und führen Sie es wie folgt aus:

```
testscript.sh param1 param2 param3
```

Die Ausgabe wird wie folgt aussehen:

```
Dieses Script heißt testscript.sh.
Es wurde mit 3 Parametern aufgerufen: param1 param2 param3
Der erste Parameter ist: param1
Seine PID ist 1719
Das Programm wird sich jetzt selbst beenden ...
Terminated
```

12.9 Funktionen

Funktionen — oder Unterprogramme — stellen eine Möglichkeit dar, den Code besser zu strukturieren und Probleme zu verallgemeinern. Sie können wie normale Programme aufgerufen werden; insbesonders können auch Parameter übergeben werden, die über die zuvor genannten Variablen abfragbar sind. Die Syntax zum Erstellen einer Funktion lautet:

```
funktionsname() {
     Befehle
}
```

Der Aufruf der Funktion erfolgt über

`funktionsname ⟨Parameter⟩`

Ein Beispiel:

```
# Beginn der Funktion
beispielfunktion() {
      echo "Das ist die Funktion $0."
      echo "Folgende Parameter wurden übergeben: $@"
}
# Ende der Funktion

# Aufruf der Funktion
beispielfunktion param1 param2 param3
```

Oft werden häufig verwendete Funktionen in eigenen Dateien abgelegt. Diese können dann von anderen Scripts inkludiert werden. Wenn Sie etwa eine Datei `funktionen.sh` in Ihrem HOME Verzeichnis haben, können Sie diese wie folgt in einem anderen Script inkludieren:

```
. ~/funktionen.sh
```

Damit wird der Inhalt dieser Datei eingefügt und ausgeführt.

12.10 Shell Aliases

Ähnlich wie Funktionen funktionieren *aliases*. Sie sind aber nur für „Kleinigkeiten“ gedacht, größere Aufgaben sollten mit Funktionen gelöst werden. So ist bei *aliases* keine Parameterübergabe möglich. Ein typisches Beispiel für ein `alias` ist:

```
alias ll='ls -l'
```

Damit können Sie `ll` eingeben, und `ls -l` wird ausgeführt. Dieses `alias` können Sie wie folgt wieder löschen:

```
unalias ll
```

12.11 Benutzerinteraktion

Bisher war die Interaktion mit Scripts recht eintönig. Sie haben dem Script bestenfalls einen Parameter übergeben, das Script wurde damit ausgeführt. Sie können während der Ausführung dem Benutzer auch eine Frage stellen (beziehungsweise allgemeiner: auf eine Eingabe, woher auch immer, warten). Das geschieht mit dem Befehl `read`. Ein einfaches Beispiel:

```
echo -n "Geben Sie Ihren Namen ein: "
read name
echo "Sie heißen $name."
```

`read` liest (im einfachsten Fall) von der Eingabe, bis die Eingabetaste gedrückt wird (beziehungsweise bis ein Zeilenende erkannt wird) und speichert diese in einer Variable, in diesem Fall `name`.

Die Option „`-n`“ von `echo` ist hier neu: Sie bewirkt, dass `echo` am Ende der Ausgabe keinen Zeilenumbruch durchführt, sondern auf der aktuellen Zeile fortgesetzt wird.

12.12 Aufgaben

1. Gibt es einen Unterschied zwischen den folgenden Befehlen:

   ```
   test -f ~/liesmich && echo "gefunden"
   [ -f ~/liesmich ] && echo "gefunden"
   ```

2. Wie können Sie überprüfen, ob eine Datei ein Link ist?

3. Wie überprüfen Sie, dass eine Datei kein Link ist?

4. Wie können Sie überprüfen, ob `~/liesmich` eine Datei oder ein symbolischer Link ist?

Die Lösungen finden Sie auf Seite 131.

13 Shell Scripts 2

The UNIX philosophy basically involves giving you enough rope to hang yourself. And then a couple of feet more, just to be sure.

In diesem zweiten Teil zu Shell Scripts werden mit `sed` und `awk` zwei Programme besprochen, die in der täglichen Arbeit mit Scripts von zentraler Bedeutung sind.

Zuvor müssen aber die regulären Ausdrücke behandelt werden. Sie sind für das Verständnis dieser beiden Programme von Bedeutung.
Da reguläre Ausdrücke auch von `grep` unterstützt werden, wollen wir noch einmal auf dieses Programm zurückkommen.

Hier sollten Sie endlich Beispiele sehen, welche die Mächtigkeit von Shell Scripts besser erahnen lassen.

13.1 Regular Expressions

Reguläre Ausdrücke (*regular expressions*) sind eine Form der Mustererkennung. Ähnliches haben wir schon bei der `bash` in Form der Wildcards (was richtiger als *pattern expansion* bezeichnet wird) kennengelernt. Sie sollten die beiden nicht verwechseln, die Unterschiede sind groß. Wir werden versuchen, die regulären Ausdrücke hier relativ formal zu behandeln. Eine umfangreiche Beschreibung finden Sie mit `man 7 regex`.[46]

Reguläre Ausdrücke gehören zu den komplexeren Themengebieten. Für größere Shell Scripts werden Sie diese jedoch häufig benötigen (sie sind auch in vielen Programmiersprachen wie Perl oder Python verfügbar). Wir haben reguläre Ausdrücke auch schon kennengelernt, und zwar bei der Suchfunktion des `vi`. Über reguläre Ausdrücke wurden ganze Bücher geschrieben. Wir werden uns auf das Notwendigste beschränken, gerade genug besprechen, um Programme wie `sed` und `grep` effektiv einsetzen zu können.

Reguläre Ausdrücke bestehen aus einem oder mehreren Teilen. Diese sind durch ein „`|`“ getrennt. Das „`|`“ steht hier für ein *oder*, d.h. nur einer der beiden Teile muss zutreffen.

Jeder Teil besteht aus einr beliebigen Anzahl von Ausdrücken. Hinter jedem Ausdruck kann eines der Zeichen „`*`“, „`+`“ oder „`?`“ stehen; diese werden auch als *bound* bezeichnet.

bound	Bedeutung
`*`	Eine beliebige Anzahl des vorhergehenden Ausdruckes.
`+`	Eine oder mehrere Vorkommen des vorhergehenden Ausdruckes.
`?`	Eines oder kein Vorkommen des vorhergehendes Ausdruckes.

[46] Auf einigen Unix Systemen heißt die Hilfeseite `regexp` und/oder befindet sich in einer anderen Hilfesektion.

Drei Beispiele:

`a*`	Beliebig viele „a“.
`a+`	Beliebig viele, aber mindest ein „a“.
`a?`	Null oder ein „a“.

Hinter dem Ausdruck kann auch in geschwungenen Klammern ein Bereich angegeben werden, der angibt, wie oft der Ausdruck vorkommen soll. Dazu einige Beispiele:

`a{4}`	4 „a“.
`a{2,6}`	2 bis 6 „a“.
`a{2,}`	2 oder mehr „a“.
`a{0,}`	0 oder mehr „a“ (entspricht „`a*`“).
`a{1,}`	1 oder mehr „a“ (entspricht „`a+`“).
`a{0,1}`	0 oder 1 „a“ (entspricht „`a?`“).

Ein Ausdruck kann komplexer sein als ein einzelnes Zeichen — in diesem Fall wird er durch runde Klammern begrenzt, z.B.:

`(abc)`	Die Zeichenkette „abc“.
`(abc)+`	Mehr als 1 aufeinanderfolgendes Vorkommen der Zeichenkette. Z.B. „`abc`“, „`abcabc`“ oder „`abcabcabc`“.

Es gibt auch eine Reihe von Zeichen, die eine besondere Bedeutung haben. Sollen diese ohne deren spezielle Bedeutung angegeben werden, muss davor ein „\“ stehen. Das betrifft folgende Zeichen:

`^.[$()*+?`{\

Das Zeichen „`.`“ bezeichnet ein beliebiges Zeichen und „`.*`“ daher eine beliebige Anzahl beliebiger Zeichen.
„^“ steht für den Zeilenanfang, „`$`“ für das Zeilenende.

`^#`

trifft auf eine Zeile zu, die mit einem „#“ beginnt.

`^$`

bezeichnet eine leere Zeile — nach dem Zeilenanfang folgt sofort das Zeilenende.

Eine in eckigen Klammern geschriebene Zeichenkette bedeutet, dass eines dieser Zeichen vorkommen soll; die Negierung erfolgt durch ein „^“ gleich nach der öffnenden eckigen Klammer.

`[Ll]`	„`L`“ oder „`l`“.
`[^Ll]`	*nicht* „`L`“ oder „`l`“.

Wir haben die Verwendung von eckigen Klammern schon bei der Shell kennengelernt. Wie dort können auch hier Bereiche angegeben werden:

`[a-c]`	„a“, „b“ oder „c“.
`[a-c0-3]`	„a“, „b“, „c“, „0“, „1“, „2“ oder „3“.
`[a0-3c]`	„a“, „0“, „1“, „2“, „3“ oder „c“.

Auf diesen Ausdruck kann wiederum ein *bound* folgen, z.B.:

`[a-c]?`	„a“, „b“, „c“ oder kein Zeichen.
`[a-c]+`	trifft z.B. zu auf: „a“, „ac“, „aba“, „baba“, „cab“.

Es gibt eine Reihe von Zeichenklassen, die innerhalb von eckigen Klammern verwendet werde können. Hier die wichtigsten:[47]

`alnum`	Buchstabe oder Zahl.
`alpha`	Buchstabe.
`blank`	Leerzeichen oder Tabulator.

Diese Klassen werden von „`[:`“ und „`:]`“ begrenzt. Durch die Verwendung in Bereichen ist eine weitere Klammer hinzuzufügen, sodass das Endergebnis wie folgt aussieht:

`[[:space:]]*`	Beliebig viele Leerzeichen oder Tabulartoren.
`[[:alpha:]]`	Ein Buchstabe.

Klassen sind in der Praxis sehr nützlich. Wir haben oben ein Beispiel gesehen, um eine leere Zeile zu beschreiben. Meist sollen aber als leere Zeilen auch jene gewertet werden, die nur Leerzeichen oder Tabulatoren enthalten:

```
^[[:space:]]*$
```

Viele Konfigurationsdateien verwenden (wie Shell Scripts) „#“ als Kommentarzeichen. Eine Zeile, die als Kommentar gewertet wird, muss nicht unbedingt mit „#“ beginnen, es können auch Leerzeichen und Tabulatoren davorstehen:

```
^[[:space:]]*#
```

[47] Auf vielen Unix Systemen sind bei den *regular expressions* die Zeichenklassen nicht verfügbar. Dies ist auch bei den nachfolgenden Programmbeispielen mit `sed` und `grep` zu beachten.

13.2 grep die Zweite

Das unter Linux verwendete `grep` unterstützt reguläre Ausdrücke. Auf anderen Unix Systemen wird diese Funktionalität häufig von `egrep`[48] implementiert. Damit können Sie obige Ausdrücke verwenden.
Das folgende Beispiel zeigt die Datei `/etc/inittab` ohne leere Zeilen und Kommentare an:

```
grep -v "^[[:space:]]*#" /etc/inittab | grep -v "^[[:space:]]*$"
```

Die Option „`-v`“ bewirkt, dass alle Zeilen *außer* jenen, auf die das Muster zutrifft, angezeigt werden.

13.3 sed

`sed` ist ein sehr mächtiges Programm, eine eigene kleine Sprache. Auch über `sed` gibt es eigene Bücher (meist gemeinsam mit `awk`, das weiter unten behandelt wird). Wir wollen hier nur die einfachste Form behandeln — die direkte Ausführung auf der Kommandozeile, ohne Verwendung von Script Dateien.
Die allgemeine Syntax dafür lautet (vereinfacht):

```
sed -e 's⟨delim⟩⟨zu ersetzende ZK⟩⟨delim⟩⟨neue ZK⟩⟨delim⟩(g)'
```

`sed` liest aus einer Datei (oder über eine Pipe von der Standardeingabe) und ersetzt eine Zeichenkette durch eine andere. Dabei werden die Zeichenketten durch ein Trennzeichen (oben als „⟨`delim`⟩“ bezeichnet) getrennt. Dieses ist (nahezu) beliebig, folgende Ausdrücke sind daher identisch:

```
sed -e 's+zu ersetzend+neu+'
sed -e 's/zu ersetzend/neu/'
sed -e 's%zu ersetzend%neu%'
```

Hier wird meist ein Zeichen verwendet, das in keiner der beiden Zeichenketten vorkommt, die obigen Zeichen bieten sich dafür oft an.

Das optionale „`g`“ am Ende bewirkt, dass nicht nur das erste Vorkommen der Zeichenkette in einer Zeile ersetzt wird, sondern alle. Wenn Sie also beispielsweise folgende Zeile in einer Datei haben:

```
linux ist ein Unix Derivat. ... Fedora Core linux.
```

würde nur der folgende Ausdruck beide Vorkommen von `linux` durch `Linux` ersetzen:

```
sed -e 's%linux%Linux%g'
```

`sed` kann mit regulären Ausdrücken umgehen. Das folgende Beispiel ersetzt eine beliebige Anzahl von Leerzeichen und/oder Tabulatoren durch ein Leerzeichen:

```
sed -e 's%[[:space:]]*% %g' /etc/inittab
```

[48] Unter SUN Solaris sind oft zwei `egrep` Varianten installiert. Nur die unter `/usr/xpg4/bin/egrep` unterstützt auch Zeichenklassen.

`sed` beinhaltet eine Funktion, die es erlaubt, Ausdrücke zu gruppieren und wiederzuverwenden. Damit kann man nicht nur den Tippaufwand reduzieren, sondern auch komplexere Ausdrücke formulieren. Die Gruppierung erfolgt durch „\(“ und „\)“.

```
sed -e 's%\(Linux\)%\1, ein Unix Derivat,%g'
```

Hier wird eine Gruppierung für `Linux` erstellt. In einem Ausdruck können beliebig viele Gruppierungen erstellt werden, die nach Ihrem Vorkommen (von links nach rechts) durchnumeriert werden. `Linux` ist hier die erste (und einzige) Gruppierung. Im zweiten, ersetzenden Ausdruck wird nun *Nummer* durch den Inhalt der Gruppe ersetzt, `\1` also durch `Linux`.
Ein Beispiel, das mehrere Gruppen enthält:

```
sed -e 's%^\([[:space:]]*\)#*\(.*\)$%\1\2 # neuer Kommentar%'
```

Dieses Beispiel entfernt alle Kommentarzeichen am Beginn der Zeile und fügt am Ende jeder Zeile einen Kommentar hinzu. Die Leerzeichen vor den Kommentarzeichen „#“ werden in einer Gruppe gespeichert, ebenso der gesamte nachfolgende Text bis zum Zeilenende. Der zweite Teil fügt diese Teile wieder ein (ohne die Kommentarzeichen) und hängt am Ende einen Kommentar an.

Es können auch mehrere `sed` Ausdrücke ausgeführt werden, wenn das Problem z.B. über einen einzelnen Ausdruck nicht lösbar ist.

```
sed -e 's%UNIX%Unix%g' -e 's%LINUX%Linux%g'
```

Hier wird in jeder Zeile `UNIX` durch `Unix` und anschließend `LINUX` durch `Linux` ersetzt.

`sed` steht für *stream editor*. Es liest eine Zeile von der Standardeingabe und schreibt sie auf die Standardausgabe. Ein Fehler, der dabei von Anfängern gerne gemacht wird:

```
sed -e ⟨Ausdruck⟩ ⟨Datei 1⟩ > ⟨Datei 1⟩
```

Das hat aber fatale Auswirkungen: Sie dürfen nicht gleichzeitig die Datei schreiben, die Sie lesen (d.h. Sie dürfen zwar, aber es ist doch etwas kontraproduktiv …); `sed` liest eine Zeile und überschreibt dann die Datei. Und damit wäre der Rest erfolgreich gelöscht …
Die Lösung dafür könnte so aussehen:

```
sed -e ⟨Ausdruck⟩ ⟨Datei 1⟩ > ⟨Datei 2⟩ && \
mv ⟨Datei 2⟩ ⟨Datei 1⟩
```

Damit schreibt `sed` seine Ausgabe in eine neue Datei. Wenn `sed` erfolgreich ist, wird die entstandene Datei verschoben, sie überschreibt die vorhandene Datei.

13.4 awk

`awk` ist, mehr noch als `sed`, eine eigene Script–Sprache. Wir wollen uns aber wieder auf das Minimum beschränken und `awk` nur dazu verwenden, um auf der Kommandozeile aus einer Zeile bestimmte Zeichen zu extrahieren. Geben Sie dazu ein:

```
echo "ein zwei drei" | awk '{ print $2 }'
```

`awk` arbeitet feldorientiert. Die obige Anweisung veranlasst `awk`, das zweite Feld auszugeben — hier „`zwei`“. Die `awk` Anweisung muss wegen der Klammerung und dem „`$`“ unter einfache Anführungszeichen gestellt werden.
Als Feldtrennzeichen verwendet `awk` standardmäßig Leerzeichen oder Tabulatoren. Mit `-F` können aber auch andere Trennzeichen angegeben werden. Im Folgenden ist der Feldtrenner der Doppelpunkt, es wird wieder „`zwei`“ ausgegeben:

```
echo "eins:zwei:drei" | awk -F: '{ print $2 }'
```

Feldtrennzeichen können auch komplexer sein. Das folgende Beispiel liefert „`rei`“:

```
echo "eins zwei drei" | awk -F"eins zwei d" '{ print $2 }'
```

Sie können der Ausgabe auch eigene Zeichen hinzufügen und mehrere Spalten ausgeben:

```
echo "eins zwei drei" | \
     awk '{ print $1 ", " $2 " und " $3 }'
```

Die Ausgabe liefert:

```
eins, zwei und drei
```

Schließlich noch ein Beispiel „aus dem richtigen Leben“. Wir haben gesehen, dass man einen Befehl über `kill` ⟨`PID`⟩ beenden kann. Manchmal wäre es aber einfacher, einen Prozess über seinen Namen zu beenden. Das folgende Beispiel definiert eine Funktion, die alle Prozesse, die eine bestimmte Zeichenkette beinhalten, beendet (diese wird als erster Parameter übergeben):

```
pskill()
{
     kill $(ps auxw | grep "$1" | grep -v "grep" | \
          awk '{ print $2 }')
}
```

Wird nun `pskill xterm` aufgerufen, so werden alle Prozess beendet, die diese Zeichenkette beinhalten.[49]

13.5 Aufgaben

1. Was macht das folgende Script?[50]

   ```
   FILE=ld.so.conf
   [ -f /etc/$FILE -a ! -f /tmp/$FILE ] && \
        cp /etc/$FILE /tmp
   for dir in /lib /usr/lib /opt/lib ; do
        if [ ! "$(grep "^$dir\$" /tmp/$FILE)" ]
        then
             echo $dir >> /tmp/$FILE
        fi
   done
   ```

[49] Eine bessere Version dieses Programmes ist mit `pkill` bereits unter Linux verfügbar.
[50] Die Datei `/etc/ld.so.conf` ist nur unter Linux verfügbar.

2. Erklären Sie den folgenden `sed` Befehl:

   ```
   sed -e 's+^\( *PATH=.*\)$+# \1+' ~/.bash_profile
   ```

3. Oben wurde eine Funktion `pskill` besprochen. Diese Funktion hat einen Nachteil: Sie liefert einen Fehler, wenn `grep` die Zeichenkette nicht findet. Beheben Sie das (Tipp: Schreiben Sie die Ausgabe von `grep` in eine Variable und überprüfen Sie diese mit `test`).

Die Lösungen finden Sie auf Seite 131.

14 Weitere Hilfsprogramme

A good workman is known by his tools.

(Brooks, „The Mythical Man-Month“)

Spätestens seit Erscheinen des Buches „Der UNIX–Werkzeugkasten“ von Kernigan und Pike wird Unix gerne mit einem Werkzeugkasten verglichen. Es enthält eine Reihe kleiner nützlicher Werkzeuge, mit denen man — werden sie geschickt gemeinsam eingesetzt — große Arbeiten erledigen kann. Im Folgenden sollen einige (wenige) Programme kurz vorgestellt werden. Mehrere haben wir bereits kurz angesprochen, sie werden hier noch einmal etwas genauer erläutert. Es werden wieder nur kurze Beispiele erwähnt. Die ganze Wahrheit finden Sie, wie immer, in den Hilfeseiten der Programme.

Ein Großteil dieser wird vor allem beim Shell Scripting eingesetzt. Einige andere spielen dort zwar keine Rolle, sollen aber in diesem letzten die Kommandozeile behandelnden Abschnitt das bisher Gelesene abrunden und ergänzen.

14.1 type

Der Befehl `type` zeigt Ihnen die Art eines Befehles an: Ob es eine Programmdatei, eine Funktion etc. ist. Hier drei Beispiele:

```
type ls
ls is aliased to 'ls --color=tty'
type type
type is a shell builtin
type xterm
xterm is /usr/bin/X11/xterm
```

Shell Aliases haben wir auf Seite 80 kennenlenen. *Shell Builtins* sind keine eigenständigen Programme, sie sind direkt in die Shell integriert; von diesen haben wir bereits eine ganze Reihe kennengelernt und verwendet, ohne sie also solche zu bezeichnen, zum Beispiel `cd`, `alias` oder `while`. `xterm` ist schließlich eine Programmdatei, die sich im Verzeichnis `/usr/bin/X11` befindet.

14.2 head

`head` zeigt die ersten Zeilen einer Datei an, standardmäßig 10, z.B.:

```
head ~/liesmich
```

Sie können auch wählen, wieviele Zeilen Sie anzeigen wollen. Das folgende Beispiel zeigt 8 Zeilen an:

```
head --lines 8 ~/liesmich
```

14.3 tail

`tail` macht genau das Gegenteil von `head`, es zeigt die letzten Zeilen einer Datei an, z.B.:

```
tail --lines 8 ~/liesmich
```

Mit `tail` können Sie auch eine Datei kontinuierlich anzeigen. Dazu verwenden Sie die Option `-f`. Kopieren Sie dazu die Datei `~/liesmich` nach `~/liesmich.2`.

```
cp ~/liesmich ~/liesmich.2
```

Starten Sie nun `tail`:

```
tail -f ~/liesmich.2
```

Öffnen Sie nun ein neues X Terminal mit `xterm -ls`. Hängen Sie eine neue Zeile an diese Datei an:

```
echo "neue Zeile" >> ~/liesmich.2
```

Wenn Sie dabei einen Blick auf `tail` werfen, sehen Sie, dass die neue Zeile nun angezeigt wird.

14.4 basename und dirname

Mit `basename` können Sie den Namen einer Datei ohne den Pfad anzeigen lassen. So liefert der Befehl

```
basename /bin/vi
```

als Ausgabe „`vi`“ zurück.

Richtig interessant wird das in Shell Scripts:

```
if [ "$#" -ne "1" ]
then
     echo "Verwendung: $(basename $0) [parameter]"
     exit 1
fi
# ansonsten den Befehl ausführen
```

Das Script überprüft, ob es mit genau einem Parameter aufgerufen wurde. Wenn nicht, gibt es eine Meldung zur Verwendung des Programmes aus.
Der Befehl `exit` ist hier neu. Er beendet das Programm. Wird dahinter eine Zahl angegeben, so ist dies der *exit code* (1 sagt, dass ein Fehler aufgetreten ist), ansonsten ist dieser 0. Zur Wiederholung: `$0` beinhaltet den Namen des Scripts (inklusive Pfad). Egal unter welchem Namen und in welchem Verzeichnis Sie nun das Script speichern, es wird immer der Name des Scripts (ohne Pfad) angezeigt.

`dirname` macht genau das Gegenteil — es zeigt den Pfadnamen an, lässt den Dateinamen weg.

14.5 gzip

`gzip`[51] dient zum Komprimieren von Dateien. Wir haben es bereits ohne nähere Erläuterung verwendet: Die Option „z“ bei `tar` verwendet `gzip`, um das Archiv zu komprimieren.

Um eine Datei zu komprimieren, verwenden Sie beispielsweise:

`gzip ~/liesmich`

Die Datei `~/liesmich` wird nun durch die komprimierte Variante, `~/liesmich.gz`, ersetzt.

Mit

`gunzip ~/liesmich.gz`

können Sie diese wieder dekomprimieren.

14.6 sort

`sort` sortiert seine Eingabe alphanumerisch. Beachten Sie, dass standardmäßig Zahlen vor Buchstaben und Groß– vor Kleinbuchsten sortiert werden. In der einfachsten Form sortiert `sort` eine Datei:

`sort ~/liesmich | less`

Das folgende Beispiel nimmt die Ausgabe von `ls -l` und sortiert nach Dateigröße:

`ls -l | sort -k 5 -n`

Wenn Sie nur `ls -l` eingeben, sehen Sie, dass die 5. Spalte die Dateigröße beinhaltet. Mit `-k 5` wird `sort` so aufgerufen, dass es als Sortierkriterium diese Spalte verwendet. Die Option `-n` weist `sort` an, das Sortierkriterium als Zahl zu behandeln. Dadurch wird etwa 2 vor 10 sortiert (bei der normalen Sortierung nach Zeichen stünde 1 vor 2 und damit etwa auch 10 vor 2).

14.7 last

Dieser Befehl zeigt an, wer zuletzt auf einer Maschine angemeldet war. Die Ausgabe könnte wie folgt aussehen:

```
cb13104  pts/7        Thu Apr 19 18:30   still logged in
c102mr   pts/10 red   Thu Apr 19 13:27 - 13:27  (00:00)
csaa5698 pts/7  math1 Wed Apr 11 17:11 - 17:14  (00:02)
c102mr   pts/2        Sun Apr  8 17:27 - 18:08  (00:40)

wtmp begins Sun Apr  1 10:46:26 2001
```

Die erste Spalte zeigt den Benutzer an, die dritte, von welcher Maschine aus der Zugriff erfolgte (falls nicht vom lokalen PC aus). Es folgen dann der Beginn der Verbindung und das Ende sowie die Dauer in runden Klammern.

[51] `gzip` ist auf den meisten Unix Systemen vorhanden.

14.8 uniq

`uniq` nimmt eine sortierte Eingabe (und wird daher fast ausschließlich gemeinsam mit `sort` verwendet) und entfernt doppelte Einträge. Ein Beispiel:

```
last | awk '{ print $1 }' | sort | uniq
```

Dies zeigt Ihnen jene Benutzer an, die in letzter Zeit am System angemeldet waren.

14.9 tr

Auch dieses Programm haben wir bereits kennengelernt. Es ersetzt Zeichen in einer Zeichenkette durch andere. Das folgende Beispiel etwa wandelt alle Großbuchstaben in Kleinbuchstaben um:

```
tr '[A-Z]' '[a-z]' < ~/liesmich
```

Beachten Sie, dass die Bereiche unter einfache Anführungszeichen gesetzt werden müssen, da sie ansonsten von der Shell (und nicht erst von `tr`) interpretiert würden.

Man könnte `tr` auch verwenden, um in einem Verzeichnis alle Dateinamen in reine Kleinbuchstaben zu ändern:

```
for file in *
do
     NN=$(echo $file | tr '[A-Z]' '[a-z]')
     if [ -f "$NN" ]
     then
          echo "$file nicht verschoben:"
          echo "$NN existiert bereits"
     else
          mv $file $NN
     fi
done
```

14.10 wc

`wc` (*word count*) zählt die Zeichen, Wörter und Zeilen seiner Eingabe, z.B.:

```
wc ~/liesmich
```

Die Ausgabe zeigt zuerst die Zeilen, gefolgt von den Wörtern und Zeichen an:

```
    91     455     3925     /home/c102/c102mr/liesmich
```

Mit `wc` kann man auch nur einzelne dieser Zählungen vornehmen:

```
FILE= /liesmich
zeilen=$(wc -l $FILE | awk '{ print $1 }')
woerter=$(wc -w $FILE | awk '{ print $1 }')
zeichen=$(wc -c $FILE | awk '{ print $1 }')
echo "$FILE:"
```

```
echo "$zeilen Zeilen"
echo "$woerter Woerter"
echo "$zeichen Zeichen"
```

14.11 bc

`bc`[52] ist ein „Taschenrechner“ auf Kommandozeilenebene. Nach dem Start mittels `bc` erscheint eine Eingabezeile. Hier können Sie Berechnungen durchführen. Ein Beispiel einer `bc` Sitzung:

```
[c102mr@red-c102 c102mr]$ bc -l
/* Wieviele Nachkommastellen sollen verwendet werden? */
scale = 2

1 + 1
2

3 - 4.2
-1.2

5 * 2.2
11.0

5 / 2.2
2.27

/* "5 hoch 2" */
5 ^ 2
25

/* Wurzel */
sqrt(42.0)
6.48

/* Sinus von 2 */
s(2)
.90

/* natuerlicher Logarithmus */
l(5)
1.60

/* Exponentialfunktion */
e(3)
20.08
```

[52] Auf vielen Unix Systemen ist dieses Programm nicht verfügbar.

quit

Wenn Sie `bc` mit der Option `-l` aufrufen, wird es mit seiner mathematischen Bibliothek gestartet. Dadurch können Sie auch auf zusätzliche mathematische Funktionen wie den Logarithmus zugreifen.
Die erste eingegebene Zeile, `scale = 2`, setzt die Genauigkeit auf 2 Nachkommastellen. Davor steht eine Kommentarzeile, die von `/*` und `*/` begrenzt ist. Danach folgt jeweils ein Ausdruck und in der nächsten Zeile sein Resultat.

Im Gegensatz zur Shell kann `bc` nicht nur mit natürlichen Zahlen umgehen, weshalb es oft in Shell Scripts verwendet wird. Das folgende Beispiel weist der Variable `pi` den Wert von π mit 10 Nachkommastellen zu:

```
pi=$(echo "scale=10; 4*a(1)" | bc -l)
```

14.12 wget

`wget` gehört zwar nicht zum Standard–Lieferumfang der meisten Unix Systeme, ist aber bei praktisch allen Linux Distributionen vorhanden. Damit kann man auf der Kommandozeile Dateien über HTTP oder FTP herunterladen. Das ist nicht nur schneller als viele grafische Programme — `wget` bietet auch zusätzliche Funktionen. Mit der Option `-r` kann man Seiten oder Verzeichnisse rekursiv herunterladen (z.B. inklusive aller Unterseiten, Bilder, Programme etc.). Sollte ein Download einmal abgebrochen worden sein, kann er mit der Option `-c` wiederaufgenommen werden. Es werden dann nur noch die fehlenden Teile, nicht mehr die bereits gespeicherten heruntergeladen. Das funktioniert auch bei teilweise heruntergeladenen Dateien, es wird einfach nur der fehlende Teil ergänzt.
Um etwa die Index Datei der Hauptseite der LFU Innsbruck herunterzuladen, verwenden Sie:

```
wget http://www.uibk.ac.at/index.html
```

14.13 which

`which`[53] zeigt Ihnen an, wo sich ein Programm im Verzeichnisbaum befindet. So liefert beispielsweise

```
which vi
```

als Ausgabe

```
/bin/vi
```

14.14 locate

`locate`[54] funktioniert über eine einfache Datenbank, die von System regelmäßig aktualisiert wird. In dieser können Sie nun nach beliebigen Dateien suchen. Wenn Sie

[53] Auf einigen Unix Systemen sollten Sie stattdessen **`whence`** verwenden.
[54] Dieser Befehl ist Linux–spezifisch und nur auf wenigen anderen Unix Plattformen vorhanden.

beispielsweise

locate fstab

eingeben, werden alle Dateien angezeigt, die im Namen `fstab` enthalten.

14.15 date

`date` gibt das aktuelle Datum und die Uhrzeit aus. Die einfachste Variante

date

liefert als Ausgabe:

```
Fri Apr 20 13:14:24 CEST 2001
```

`date` kann die Ausgabe auch formatieren, beispielsweise nur Tag, Monat und Jahr ausgeben:

date "+%d %m %Y"

Man kann damit auch beliebige Formatierungen vornehmen. Der Befehl

date "+Heute ist der %d. %m. %Y."

ergibt:

```
Heute ist der 20. 04. 2001.
```

14.16 cal

Mit `cal` kann man als Text einen Kalender ausgeben. Ohne Parameter wird einfach nur der Kalender des aktuellen Monats ausgegeben:

```
    Oktober 2004
So Mo Di Mi Do Fr Sa
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31
```

Um den Kalender des gesamten Jahres auszugeben, wird als Parameter das Jahr angegeben:

cal 2004

Will man einen bestimmten Monat, zum Beispiel den März 2001, anzeigen lassen, so werden als Parameter der Monat und das Jahr angegeben:

cal 9 2004

14.17 dos2unix und unix2dos

Microsoft Windows Text–Dateien unterscheiden sich von Unix Text–Dateien durch die Art, wie eine Zeile abgeschlossen wird. Um eine solche Text–Datei nach Unix umzuwandeln, verwenden Sie den Befehl

`dos2unix` ⟨`Datei`⟩[55]

Beachten Sie, dass Sie das nicht auf binäre Dateien — Programme, Bilder etc. — anwenden dürfen, da deren Inhalt sonst zerstört wird!

Sollte `dos2unix` auf einem System nicht vorhanden sein, können Sie dafür übrigens auch `tr` verwenden:

`tr '\r' '' <` ⟨`Datei`⟩

`unix2dos` ist das Pendant zu `dos2unix`. Es wandelt Unix Text Dateien nach Microsoft Windows um.

`unix2dos` ⟨`Datei`⟩

14.18 who und finger

Der Befehl `who` zeigt Ihnen an, welche Benutzer zur Zeit am System angemeldet sind.

`finger`, wenn ohne Parameter aufgerufen, liefert ein ähnliches Ergebnis wie `who`. Allerdings sieht man nicht nur die Benutzerkennung sondern auch den Namen des Benutzers.

Wenn Sie als `finger` Parameter eine Benutzerkennung eingeben, erhalten Sie weitere Informationen zur Benutzerkennung, z.B.:

`finger c102mr`

Die Ausgabe lautet:

```
Login: c102mr                     Name: Michael Redinger
Directory: /home/c102/c102mr      Shell: /bin/bash
Office: ZID, 0512/507-2335
On since Thu Apr 19 18:30 (CEST) on pts/7   49 seconds idle
No mail.
Project:
Linux
Plan:
Linux Benutzereinfuehrung
```

[55] `dos2unix` und `unix2dos` sind nur auf einigen Linux Systemen und nur selten auf anderen Unix Derivaten vorhanden.

14.19 write

Wenn ein Benutzer angemeldet ist, können Sie ihm eine kurze Nachricht schicken. Dazu geben Sie ein:

`write` ⟨**`Benutzerkennung`**⟩

Darauf erscheint keine neue Eingabeaufforderung. Sie können hier Ihren Text eingeben, den Sie schicken wollen. Wenn Sie fertig sind, drücken Sie Strg – D, um die Nachricht zu versenden.
Beim anderen Benutzer erscheint nun die Nachricht:

```
 Message from ⟨Benutzer⟩@⟨Rechner⟩ on pts/5 at 13:29 ...
⟨Hier der Inhalt der Nachricht⟩
EOF
```

14.20 talk

Auf vielen Systemen ist auch `talk` aktiviert — damit können Sie mit einer Person interaktiv kommunizieren. Wenn Sie

`talk` ⟨**`Benutzerkennung`**⟩

eingeben, ändert sich Ihr Bildschirm. Er ist nun zweigeteilt, und oben sehen Sie folgende Meldung:

```
[Waiting for your party to respond]
```

Beim anderen Benutzer erscheint eine Meldung, dass er `talk` eingeben soll:

```
Message from Talk Daemon@⟨Rechner⟩ at 20:26 ...
talk: connection requested by ⟨Benutzer⟩@⟨Rechner⟩.
talk: respond with: talk ⟨Benutzer]@⟨Rechner⟩
```

Macht er dies, erscheint nun auch bei ihm das zweigeteilte Fenster, und bei beiden wird als Zeichen, dass die Verbindung aufgebaut ist, am oberen Bildschirmrand eine Meldung angezeigt:

```
[Connection established]
```

Im oberen Teil sieht jeder Benutzer seine Eingabe, in der unteren Hälfte erscheinen die Antworten des Gesprächspartners (Abb. 17).

Die Verbindung wird beendet, wenn einer der beiden Strg – C drückt.

14.21 kibitz

Das Programm `kibitz`[56] wir vor allem verwendet, um Hilfestellung bei Problemen zu erhalten. Wenn Sie etwa ein Problem mit einem (Kommandozeilen–)Programm haben und nicht weiter wissen, können Sie einen anderen Benutzer (auch auf einem anderen Unix System, auf dem Sie eine Benutzerkennung haben) um Hilfe bitten (Sie

[56] `kibitz` ist auf vielen Unix Systemen standardmäßig nicht vorhanden.

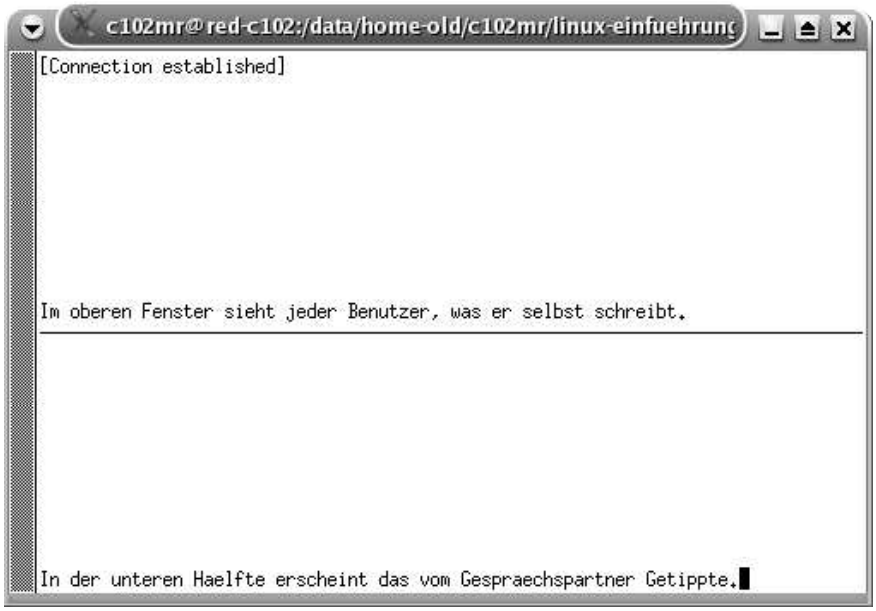

Abbildung 17: Beispiel für eine Unterhaltung mittels `talk`.

sollten ihm das aber vorher ankündigen …). Das wird oft speziell in Kombination mit Telefon–Support eingesetzt:

`kibitz` ⟨**`Benutzer`**⟩

Als Zeichen, dass das funktioniert hat, erhalten Sie als Meldung:

```
asking ⟨user⟩ to type: kibitz -31257
```

Der andere Benutzer sieht nun folgende Meldung in seiner Shell:

```
Message from ⟨Benutzer⟩ on pts/8 at 18:58 ...
Can we talk? Run: kibitz -31257
EOF
```

Wenn dieser dies nun eingibt, erkennen Sie das an folgender Meldung in den Shells beider beteiligter Benutzer:

```
Escape sequence is ^]
```

Nun hat der um Hilfe gebetene Benutzer vollen Zugriff auf die Shell des anderen Benutzers. Jede Interaktion wird von beiden Benutzern gleich gesehen. Wenn Sie `ls` eingeben, erscheint der Befehl und die Ausgabe bei beiden Benutzern.
Die Verbindung wird beendet, wenn einer der beiden Benutzer `exit` eingibt.

14.22 md5sum

Dieses Programm generiert eine sogenamnnte MD5 Prüfsumme (*checksum*) einer Datei (beziehungsweise allgemein einer Zeichenkette). Der Algorithmus weist jeder Datei eine eindeutige Prüfsumme zu, die am Bildschirm ausgegeben wird. Sobald sie

sich in nur 1 Zeichen unterscheidet, ändert sich auch die MD5 Prüfsumme (es konnten bisher nicht 2 unterschiedliche Zeichenketten gefunden werden, welche die selbe Prüfsumme haben). Das folgende Beispiel berechnet die Prüfsumme der `liesmich` Datei:

`md5sum ~/liesmich`

Die Ausgabe des Progammes:

```
0a067672e158db542ac07a6a0c50b086 /home/c102/c102mr/liesmich
```

Ändert man nur eine Kleinigkeit, so ändert sich die Prüfsumme:

```
dc5b4d7527f19c29ebce1353021eb449 /home/c102/c102mr/liesmich
```

Das Programm wird oft verwendet, um die Prüfsumme zu berechnen und dann am FTP oder HTTP Server abzulegen. Wenn die Prüfsumme am Server mit jener der Datei übereinstimmt, die man heruntergeladen hat, kann man sicher sein, dass der Download funktioniert hat.
Auch das Programminstallationssystem `RPM` unter Linux verwendet diese Prüfsumme. Damit kann man feststellen, ob eine Datei (etwa durch einen Eindringling) verändert wurde.

14.23 diff

Mit `diff` können Sie den Unterschied zwischen 2 Dateien anzeigen lassen. Weiter oben, beim Programm `tail`, haben wir eine Datei `~/liesmich.2` erstellt und an diese die Zeile „`neue Zeile`“ angehängt. Wenn Sie nun `diff` mit diesen beiden Dateien aufrufen

`diff ~/liesmich ~/liesmich.2`

erhalten Sie folgende Ausgabe:

```
91a92
> neue Zeile
```

Das bedeutet, dass nach Zeile 91 eine neue angehängt wurde.
Sie könnten diese Ausgabe natürlich auch in einer Datei speichern:

`diff ~/liesmich ~/liesmich.2 > liesmich.diff`

Mit dem Befehl `patch` können Sie nun die `diff` Datei auf eine bestehende Datei anwenden:

`patch liesmich < liesmich.diff`

Damit werden die Unterschiede auf die Datei `liesmich` angewandt, diese enthält nun auch die Änderungen aus `liesmich.2` (und ist damit mit dieser identisch).

`diff` und `patch` sind sehr mächtige Werkzeuge, die vor allen in der Programmierung eingesetzt werden, um Änderungen auszutauschen. Wenn Sie sich damit beschäftigen, werden Sie vielleicht noch ausführlich mit diesen Programmen zu tun haben. Soviel sei noch zum Abschluss gesagt: Sie können `diff` auch rekursiv auf Verzeichnisse anwenden; dafür wird ein Befehl wie folgt verwendet:

`diff -Nur Verzeichnis/ Verzeichnis.original/`

Dies liefert eine Ausgabe mit den Unterschieden zwischen den beiden Verzeichnissen. Die Optionen `N` (*normalized diff*) und `u` (*unified diff*) werden vor allem von Programmierern gerne verwendet. Die so erzeugten `diff` Dateien sind besser lesbar unter Programmierern „Standard“.

14.24 Der Midnight Commander

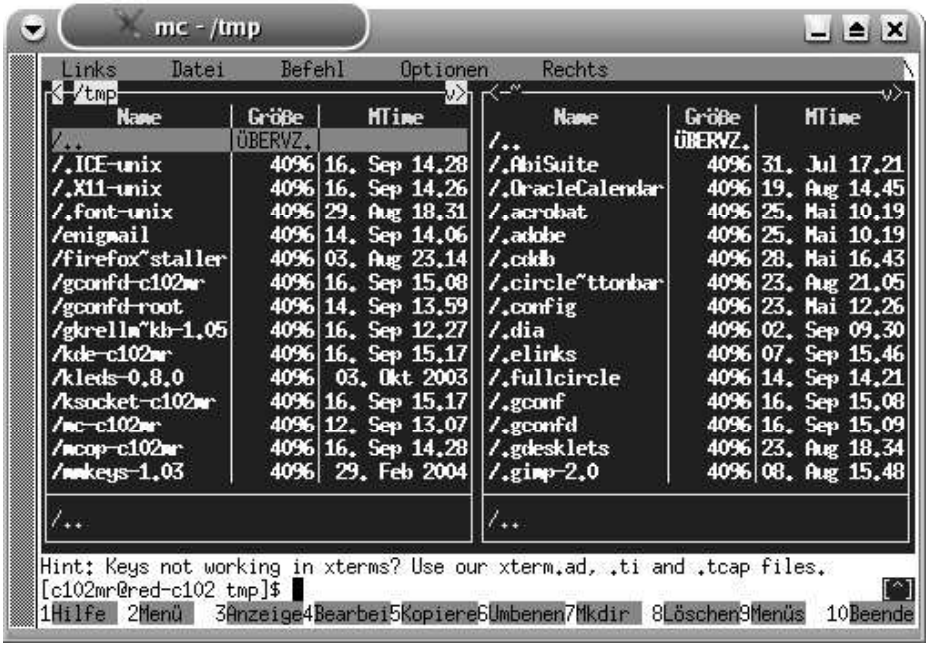

Abbildung 18: Der Midnight Commander, ein Klon des Norton Commanders.

Ein sehr bekanntes und häufig verwendetes Programm ist der Midnight Commander, ein Klon des unter MS–DOS bekannten Norton Commanders, der in vielen Bereichen das Original übertrifft (Abb. 18).[57] Erfahrene Unix Benutzer werden ihn wohl nur selten benutzen, da man, wie immer, auf der Kommandozeile einfach schneller ist. Für Anfänger (aber nicht nur für diese) ist er eine geeignete Wahl auf der Suche nach einem schnellen, übersichtlichen und gleichzeitig flexiblen Dateimanager. Der Midnight Commander **`mc`** ist aber mehr als das: man kann damit u.a. `tar` Archive einsehen, FTP einbinden und vieles mehr.

Der Midnight Commander wird mit `mc` aufgerufen. Sein Bildschirm besteht aus mehreren Teilen:

Ganz oben finden Sie die Menüzeile. Um diese zu aktivieren, drücken Sie F9[58] oder mit der Maustaste auf einen der Menüpunkte. Hier können Sie die verschiedenen Be-

[57] Der Midnight Commander ist auf vielen Linux – und anderen Unix–Systemen nicht installiert.

[58] Wenn Sie mit über `telnet` auf ein Unix System zugreifen und die Funktionstasten nicht funktionieren, können Sie stattdessen im Midnight Commander Esc gefolgt von der Ziffer der Funktionstaste eingeben, also beispielsweise Esc 1 statt F1.

fehle wählen. Neben den Befehlen stehen oft weitere Buchstaben. Das sind Tastaturkürzel, mit denen Sie den Befehl ebenfalls ausführen können.

Darunter befinden sich die Verzeichnisbäume. Deren Aussehen kann übrigens im Menü `Left` bzw. `Right` geändert werden. Hier können Sie mit der Maus oder mit der Taste Einfügen bzw. Insert Dateien markieren, Verzeichnisse wechseln etc. Mit der Tabulatortaste wechseln Sie von einem Baum in den anderen.

Noch weiter unten ist die Shell Befehlszeile. Hier können Sie normale Unix / Linux Befehle eingeben, die dann ausgeführt werden. Direkt oberhalb befindet sich noch eine weitere Zeile, wo Sie immer wieder nützliche Tips finden, wie Sie schneller und effektiver mit dem Midnight Commander arbeiten können.

Den Abschluss bildet die Tastenzeile. Hier steht, was Sie mit welcher Taste machen können. Hier sind allerdings nur die `F` Tasten aufgelistet, F1 bis F10. Um diese anzuwenden, markieren Sie wie oben beschrieben die gewünschten Dateien und drücken dann die entsprechende Taste. Alternativ dazu können Sie auch mit der Maus auf die Taste in der Tastenzeile klicken.

Der Midnight Commander beinhaltet auch eine sehr gute Hilfe, in der Sie zum Bildschirm, zu den Funktionstasten, den Befehlen etc. Informationen erhalten. Lesen Sie diese und arbeiten Sie mit dem Midnight Commander. Zusätzlich noch ein paar nützliche Tricks:

- Wenn Sie auf der Kommandozeile einen Befehl ausführen wollen, der den Dateinamen der gerade markierten Datei enthält, drücken Sie C – x t (C steht für die Strg – oder Ctrl –Taste), d.h. Sie drücken die Control–Taste, halten diese gedrückt und drücken x. dann lassen Sie beide aus und drücken die t Taste. Wenn Sie stattdessen den aktuellen Pfadnamen brauchen, drücken Sie C – x p.
- Um den Inhalt eines `tar` Archives einzusehen, gehen Sie einfach zu diesem und drücken die Eingabetaste. Sie können nun mit den Dateien des Archives (fast) wie mit normalen Dateien arbeiten.
- mit C – o wechseln Sie kurzfristig auf eine normale Shell. Mit einem weiteren C – o kommen Sie von dort wieder zurück in den Midnight Commander.
- Wenn Sie die Ausgabe eines Befehls sehen wollen (z.B. von `ls -l`), drücken Sie M – ! (also die Alt Taste und gleichzeitig Shift und 1) und geben den Befehl dann ein. Der Viewer des Midnight Commanders zeigt Ihnen dann die Ausgabe.
- Um schnell in ein anderes Verzeichnis zu wechseln, können Sie M – c verwenden.
- Um mit FTP zu arbeiten, geben Sie einfach in der Befehlszeile

  `cd ftp://`⟨**`user`**⟩**`@`**⟨**`rechner`**⟩

  ein. Für anonymous FTP können Sie ⟨`user`⟩`@` weglassen.

- Um zu einer bestimmten Stelle in einem Verzeichnis zu springen, drücken Sie C – s und dann die Anfangsbuchstaben der Datei. Den Suchmodus verlassen Sie mit Esc.

14.25 Aufgaben

1. Schreiben Sie eine Funktion, die das Verzeichnis jenes Programmes ausgibt, das als Parameter übergeben wird. Nennen Sie die Funktion `finddir`. Der Aufruf `finddir vi` sollte als Ausgabe `/bin` liefern.

2. Versuchen Sie, die Kompressionsrate von `gzip` zu ändern.

3. Das obige Beispiel bei `uniq` (Seite 92) hat einen Nachteil: es gibt auch eine Leerzeile aus. Außerdem beinhaltet die Ausgabe von `last` als letzte Zeile „`wtmp begins ...`“. Entfernen Sie diese aus der Ausgabe von `last`. Verwenden Sie diese Zeile aber, um zum Schluss das Ergebnis zu formulieren. Die Ausgabe des fertigen Scripts sollte so aussehen:

   ⟨`Anzahl`⟩ `verschiedene Benutzer seit` ⟨`Datum`⟩

4. Bei `tr` (Seite 92) wurde ein Script angeführt, das alle Dateien von Groß– nach Kleinbuchstaben umwandelt. Ändern Sie es so ab, dass daraus ein fertiges Script entsteht. Dieses soll folgende Bedingungen erfüllen:

   - Die Dateien werden als Parameter übergeben.
   - Es wird überprüft, ob die angegebenen Dateien (oder Verzeichnisse) existieren.
   - Wenn der Dateiname nach der Umwandlung mit dem alten identisch ist, wird keine Meldung ausgegeben.
   - Die entstehenden Dateien werden im aktuellen Verzeichnis abgelegt, nicht im ursprünglichen (wird `bin/TEST` übergeben, so resultiert daraus `test`).

Die Lösungen finden Sie auf Seite 132.

15 Bash Startup Dateien

UNIX was not designed to stop its users from doing stupid things, as that would also stop them from doing clever things.

(Doug Gwyn)

Wenn die Shell `bash` gestartet wird, liest sie eine Reihe von Dateien (sofern sie vorhanden sind):

- `/etc/profile`
- `~/.bash_profile`
- `~/.bash_login`
- `~/.profile`

Beim Beenden der Shell wird die Datei `~/.bash_logout` gelesen.

Der Inhalt dieser Dateien wird eingelesen und ausgeführt. Sie können in diesen im Prinzip alles machen, was Sie auch in Shell Scripts und auf der Kommandozeile machen können. Die Datei `/etc/profile` kann nur der Systemadministrator editieren. Für den Benutzer bleiben daher die restlichen Dateien. Normalerweise wird die Datei `~/.bashrc` für Funktionen und `aliases` verwendet, während Umgebungsvariablen oder komplexere oder sogar interaktive Scripts nach `~/.bash_profile` geschrieben werden.

Typischerweise kann der Benutzer hier die Umgebung an seine Bedürfnisse anpassen, etwa Variablen setzen oder `aliases` definieren. Man kann hier auch das Verhalten der Shell einstellen. So bewirkt etwa der Befehl `set -o vi`, dass man nun in der Shell die Tastaturbelegung des `vi` verwendet (mit `set -o emacs` schalten Sie wieder auf die Standard–Tastaturbelegung zurück).

15.1 Aufgaben

1. Erzeugen Sie ein `alias`, das bei der Eingabe von `cd..` den Befehl `cd ..` ausführt.

Die Lösungen finden Sie auf Seite 134.

16 Grafische Applikationen

... it is easy to be blinded to the essential uselessness of them by the sense of achievement you get from getting them to work at all. In other words ... their fundamental design flaws are completely hidden by their superficial design flaws.

(The Hitchhiker's Guide to the Galaxy)

Wir wollen nun zu den grafischen Applikationen zurückkehren und kennenlernen, welche Möglichkeiten sich hier bieten.

Zuvor aber noch eine Präzisierung:

In der Einführung wurde KDE als grafische Oberfläche vorgestellt, was auch richtig ist. Darunter liegt jedoch noch eine andere Schicht, das X Window System (kurz X). Dieses ist für die basalen grafischen Mechanismen zuständig. Es ist auch verantwortlich für die Geräteansteuerung, etwa die Bildschirmauflösung oder die Ansteuerung der Maus. Aber X ist mehr als das: Es ist ein netzwerkbasierendes System. So können Sie etwa auf einem anderen Rechner ein grafisches Programm ausführen, das dann auf Ihrem Rechner dargestellt wird.
Wenn Sie den Ausdruck „unter X“ hören, so bedeutet das meist „auf einer grafischen Oberfläche (unter Unix)“.

Nach X folgt der sogenannte Window Manager. Er ist für das Aussehen der Fenster, für deren Verschieben etc. verantwortlich. Der mit KDE gelieferte Window Manager heißt kwin. Es gibt viele andere Window Manager, z.B. sawfish, icewm, fvwm2 oder twm.

Der Window Manager ist aber oft nicht mehr als solcher erkennbar beziehungsweise spielt er mit einer ganzen Reihe von anderen Applikationen zusammen. Die Kombination bezeichnet man als Desktop Umgebung. Die Hauptvertreter unter Linux sind KDE und GNOME (das hier allerdings nicht behandelt wird). Die Applikationen, die zu einer Desktop Umgebung gehören, weisen dasselbe Aussehen, dieselbe Bedienung und Interoperabilität auf (so können Sie etwa mit der Maus Text oder Dateien von einer Applikationen in die andere ziehen).
In einer Desktop Umgebung können Sie aber auch Applikationen, die nicht für diese geschrieben wurden, ausführen; die Bedienung ist nur etwas unterschiedlich — ein Beispiel dafür ist der Open Office.

16.1 konqueror

konqueror ist der Dateimanager von KDE. Über ihn kann man beispielsweise auch Web–Seiten öffnen, auf FTP Seiten oder auf Verzeichnisfreigaben von Windows Rechnern zugreifen.

Der konqueror wird (unter anderem) geöffnet, wenn Sie im KDE Menü Persönlicher Ordner anklicken. Wird der konqueror als Dateimanager und nicht als Webbrowser gestartet, sehen Sie zu Beginn die Dateien und Verzeichnisse in Ihrem HOME Verzeichnis.

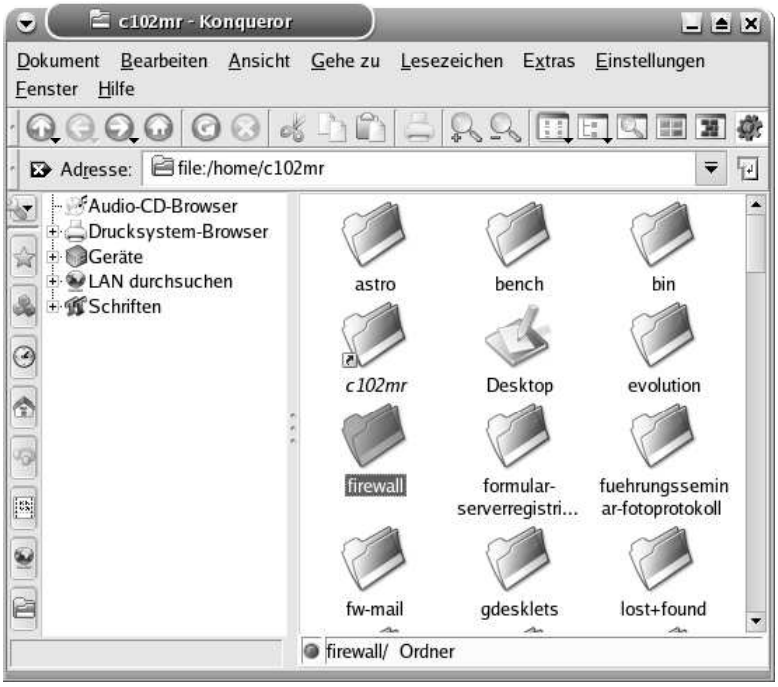

Abbildung 19: Der konqueror als Dateimanager.

Wenn Sie Dateien und Verzeichnisse anklicken[59], werden sie geöffnet. Bei fast allen Dateien weiß der konqueror, wie er sie darstellen soll, er öffnet sie automatisch (allerdings nur zum Betrachten).

Wenn der konqueror nicht weiß, was er mit einer Datei machen soll, erscheint ein Dialog, in dem Sie die entsprechende Applikation auswählen können, (entweder über die Struktur des K Menüs oder durch direkte Eingabe oder Suche der Applikation) (Abb. 20).

Wenn Sie eine Datei bearbeiten wollen, klicken Sie diese mit der rechten Maustaste an. Es erscheint ein Kontextmenü, indem Sie viele Aktionen auf die Datei anwenden können (Abb. 21).

Mit Ausschneiden schneiden Sie eine Datei aus, mit Kopieren kopieren Sie diese. Einfügen fügt eine zuvor ausgeschnittene oder kopierte Datei wieder ein.
Wenn Sie Umbenennen wählen, ändert sich das Aussehen des Dateinamens, er ist blau markiert. Sie können nun den neuen Dateinamen eingeben.
In den Mülleimer werfen und Löschen löschen jeweils die Datei; die erste Methode löscht die Datei allerdings nicht wirklich, sie verschiebt sie nur in den Papierkorb.
Es folgen die beiden Untermenüs Vorschau in und Aktionen. Ersteres öffnet das Dokument zur Ansicht, ohne dass Sie es bearbeiten können, in zweiterem können Sie die Datei in einem Programm zur Bearbeitung öffnen.
In den folgenden Untermenüs können Sie die Datei in ein anderes Verzeichnis kopieren (Kopieren nach), in ein anderes Verzeichis verschieben (Verschieben nach) oder komprimieren oder zu einem Archiv hinzufügen (Komprimieren).

[59] Je nach Einstellung müssen Sie die Datei nur einmal oder zweimal anklicken.

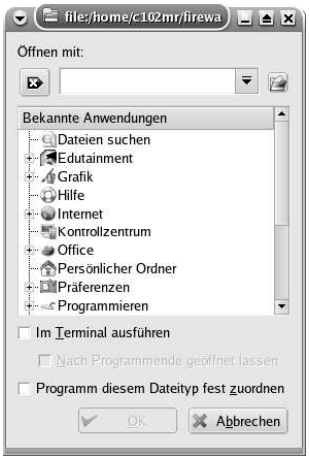

Abbildung 20: konqueror Dialog zum Öffnen einer Datei.

Abbildung 21: Datei–Kontextmenü des konqueror.

Schließlich können Sie mit Eigenschaften beispielsweise die Berechtigungen für die Datei ändern.

Mehrere Dateien markieren Sie, indem Sie die linke Maustaste (aber nicht auf einem Symbol) drücken, gedrückt halten und einen Rahmen aufziehen. Die selektierten Dateien werden nun blau unterlegt, sie können Aktionen auf alle diese Dateien anwenden.

Damit wählen Sie allerdings nur benachbart angeordnete Dateien aus. Stattdessen klicken Sie mit gedrückter Strg –Taste verschiedene Dateien an, um diese gemeinsam zu markieren.

Sie können Dateien über den Menüpunkt Bearbeiten oder über das Kontextmenü der rechten Maustaste kopieren oder verschieben. Alternativ dazu kann man auch 2 konqueror Fenster aufmachen und die Datei(en) von einem in das andere ziehen. Es erscheint ein kleines Popup Menü, in dem Sie auswählen, ob Sie die Datei kopieren, verschieben oder einen Link anlegen wollen. Sie können das auch verwenden, um Dateien auf dem Desktop zu platzieren. Ziehen Sie das Symbol einfach auf den Desktop–Hintergrund, es erscheint wieder das Popup Menü. Das funktioniert übrigens auch für Web– und FTP–Seiten. Damit haben Sie eine sehr einfache Möglichkeit, Dateien her-

unterzuladen (Abb. 22).

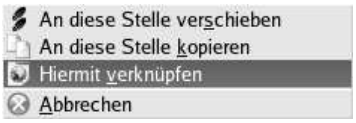

Abbildung 22: KDE Popup Menü zum Kopieren, Verschieben oder zum Anlegen eines Links.

Der konqueror kannn auch als Web–Browser gestartet werden, indem Sie im KDE Menü im Untermenü Internet unter Weitere Applikationen den Punkt Konqueror wählen (Abb 23). Wenn Sie bereits ein konqueror Fenster geöffnet haben, ändern Sie das Aussehen auf das des Web–Browsers, indem Sie aus dem konqueror Menü Einstellungen im Untermenü Ansichtsprofil laden den Punkt Web-Browser wählen. Über dieses Untermenü können Sie jederzeit zwischen den verschiedenen Darstellungsfunktionen wählen, an der Funktionalität ändert sich dadurch nichts.

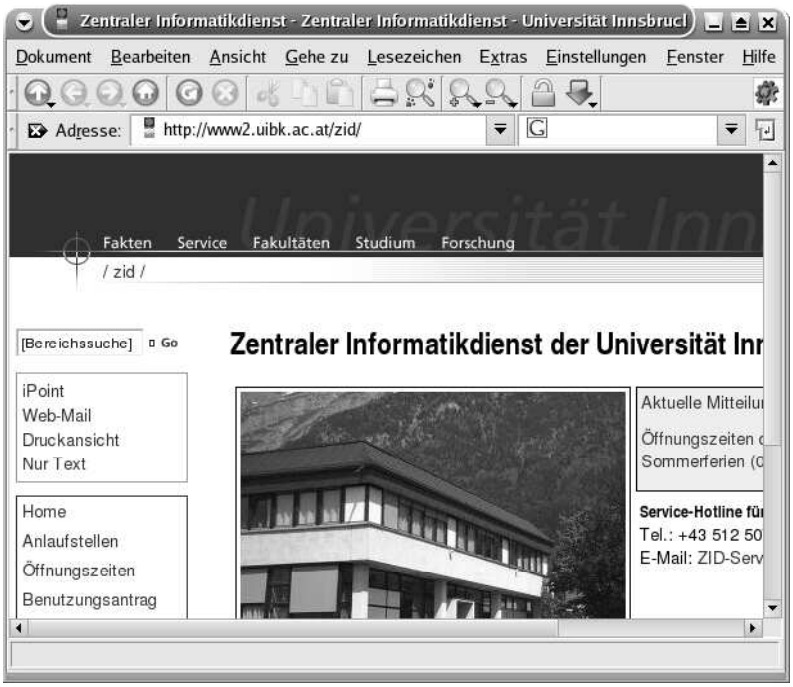

Abbildung 23: Der konqueror als Web–Browser.

Über die URL Eingabezeile des konqueror können Sie auf alle unterstützten Methoden zugreifen. Einige Beispiele:

Beispiel	Beschreibung
`file:/tmp/`	Das Verzeichnis `/tmp`.
`http://www.uibk.ac.at/c102/`	Die Homepage des ZID.
`ftp://ftp.uibk.ac.at`	Der FTP Server der LFU Innsbruck.
`ftp://[benutzer]@mail1.uibk.ac.at`	Benutzer am Mail–Server für Institutsangehörige.
`smb://[benutzer]@info-adm.uibk.ac.at`	Zugriff auf den Management–Server für Web–Seiten und Homepages an der LFU Innsbruck.
`man:less(1)`	Anzeige der Hilfeseite zu `less`.

Darüber hinaus bietet der konqueror Kürzel, um auf Web–Seiten eine Suche durchzuführen (beziehungsweise können Sie diese auch selbst definieren). Beispiele:

Beispiel	Beschreibung
`gg:Linux`	Suche nach „Linux“ in der Suchmaschine Google.
`fm:KDE`	Suche nach „KDE“ im Software–Index von freshmeat.

16.2 ark

ark (*KDE Archiving Tool*) ist ein Archivierungsprogramm ähnlich WinZip unter Windows. Sie starten es über den Befehl `ark` oder aus dem KDE Menü im Untermenü Zubehör über den Punkt Ark. Es wird hier erwähnt, um zu zeigen, dass man mit zuvor auf der Kommandozeile erstellten Archiven auch über eine grafische Applikation operieren kann.

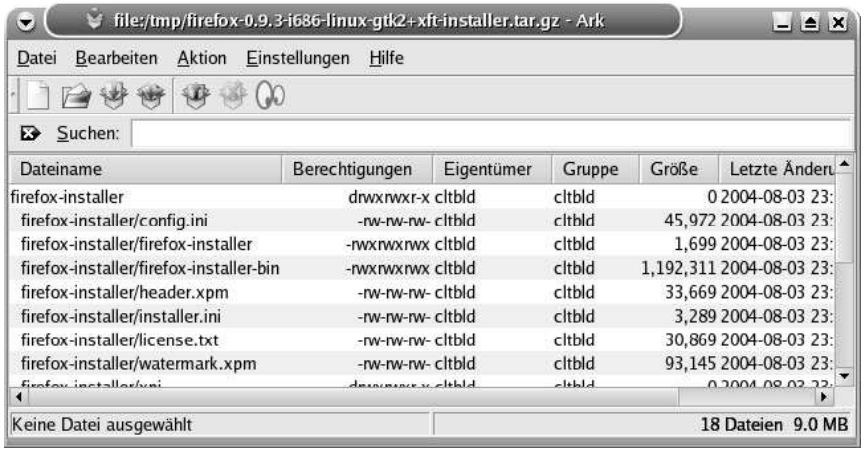

Abbildung 24: Das Archivierungsprogramm ark.

Wenn Sie ein Archiv öffnen wollen, können Sie es einfach aus dem konqueror nach ark ziehen; alternativ wird die Datei auch über das Menü Datei und den Eintrag Öffnen ausgewählt (Abb. 24).

Wenn Sie ein Archiv erstellen wollen, wählen Sie Datei — Neu. Es erscheint ein Dialog, indem Sie Namen und Plazierung des Archives eingeben können. Geben Sie keine Datei–Erweiterung an, wird standardmäßig ein mit `gzip` komprimiertes `tar` Archiv mit der Endung `.zip` erstellt.

Über das Menü Aktion können Sie nun Dateien oder Verzeichnisse hinzufügen oder löschen und Dateien öffnen. Dateien und Verzeichnisse können Sie außerdem hinzufügen, indem Sie diese aus dem konqueror in ein Archiv ziehen.

16.3 KDE Pager

kpager ist ein Programm, das Ihnen eine Miniaturansicht der Desktops und der darauf geöffneten Fenster bietet (Abb. 25). Er ist umfangreicher als der kleine Pager im KDE Panel. Der Aufruf erfolgt über `kpager` oder durch Auswahl von KPager im Untermenü Zubehör des KDE Menüs.

Abbildung 25: Der `kpager`.

Sie können den kpager so konfigurieren, dass er etwa vertikal am rechten Bildschirmrand angeordnet ist; klicken Sie dazu mit der rechten Maustaste in den kpager und wählen Sie KPager einrichten.

Wenn Sie auf einen Desktop klicken, wechseln Sie dorthin. Mit der mittleren Maustaste können Sie auch bequem ein Fenster von einem Desktop auf einen anderen ziehen.

16.4 Kontrollleiste

Die Kontrollleiste (kicker) dient — unter anderem — zum Aufruf von Programmen, vor allem über das KDE Menü. Wir haben es bereits zuvor besprochen.

Sie können ein Programm oder ein Untermenü (oder eine Datei beziehungsweise ein Verzeichnis) in das Panel aufnehmen, indem Sie es einfach aus dem KDE Menü oder aus dem konqueror auf eine freie Stelle der Kontrollleiste ziehen. Bei Verzeichnissen und Untermenüs erscheint ein kleines Menü, das Sie fragt, was Sie mit dieser Verknüpfung machen wollen (Abb. 26). Mit Als Dateimanager-Adresse hinzufügen wird beim Klicken auf das Symbol der konqueror gestartet, mit Als Schnellanzeiger hinzufügen erscheint bei der Auswahl ein Menü, das Ihnen die Programme oder den Inhalt des Verzeichnisses anzeigt (wie die Untermenüs im KDE Menü).

Symbole können Sie auf der Kontrollleiste verschieben, indem Sie des mit der mittleren Maustaste anklicken, diese gedrückt halten, und das Symbol an die gewünschte Stelle verschieben.

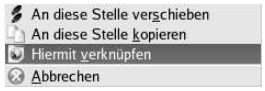

Abbildung 26: Ein Verzeichnis oder Untermenü zur Kontrollleiste hinzufügen.

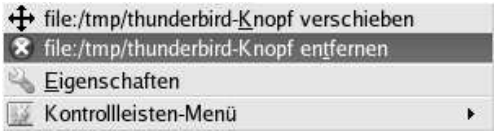

Abbildung 27: Ein Symbol aus der Kontrollleiste entfernen.

Um ein Symbol von der Kontrollleiste zu entfernen, klicken Sie es mit der rechten Maustaste an und wählen den Punkt Entfernen (Abb. 27).

Die Kontrollleiste unterstützt auch sogenannte *Miniprogramme*, Programme, die sich in die Kontrollleiste integrieren und deren Funktionalität ergänzen.
Um ein Miniprogramm hinzuzufügen, klicken Sie auf einen freien Teil der Leiste, wählen im erscheinenden Menü Hinzufügen – Miniprogramm und dort das gewünschte Applet.
Um ein Miniprogramm zu konfigurieren, klicken Sie auf die strichlierte Leiste links neben dem Miniprogramm und wählen hier Einrichten.... Um es zu entfernen, selektieren Sie stattdessen Entfernen.

Zur Konfiguration der Leiste klicken Sie auf einen freien Bereich und wählen im erscheinenden Menü den Punkt Kontrollleiste einrichten... (Abb. 28).

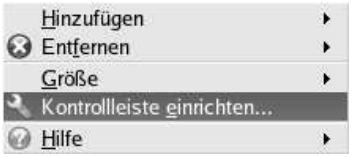

Abbildung 28: Konfigurations–Popup der KDE Kontrollleiste.

16.5 Weitere Applikationen

KDE bietet eine ganze Reihe weiterer, sehr nützlicher Programme. Probieren Sie diese aus. Rufen Sie einfach die Applikation im KDE Menü auf.

Auch die bereits beschriebenen Applikationen bieten weit mehr Funktionalität als hier beschrieben wurde. Rufen Sie dazu die Hilfe der einzelnen Programme auf.

16.6 Aufgaben

1. Ändern Sie die Anzeige des konqueror so, dass nicht nur die Symbole, sondern auch Dateigröße, Änderungsdatum etc. angezeigt werden.

2. Lassen Sie im konqueror auch die versteckten Dateien anzeigen.

3. Wenn Sie eine Datei in den Papierkorb verschieben, bleibt diese erhalten, wird nur dorthin verschoben. Wie können Sie den Papierkorb leeren?

4. Fügen Sie das Miniprogramm Systemüberwachung zur KDE Kontrollleiste hinzu.

5. Was passiert, wenn Sie unter Befehl ausführen den Befehl `man:man` statt `man:man(1)` eingeben?

6. Konfigurieren Sie kpager so, dass er sich horizontal am rechten Bildschirmrand befindet.

7. Ändern Sie das Fensterverhalten, sodass ein Fenster aktiviert wird, wenn Sie die Maus in dieses bewegen (und nicht, wenn Sie mit der Maus darauf klicken). (Sollte das bei Ihnen voreingestellt sein, wählen Sie den umgekehrten Weg.)

8. Führen Sie `less ~/liesmich` unter Befehl ausführen aus. Was müssen Sie machen, damit das auch angezeigt wird?

Die Lösungen finden Sie auf Seite 134.

17 Remote Zugriff

By documenting a design, the designer exposes himself to the criticism of everyone, and he must be able to defend everything he writes. If the organizational structure is threatening in any way, nothing is going to be documented until it is completely defensible.

(Brooks, „The Mythical Man-Month“)

Auf Unix Systeme können Sie auch von anderen Rechnern aus zugreifen. Das beschränkt sich nicht nur auf den Dateizugriff, Sie können auch Programme (selbst grafische) ausführen.
Am ZID der LFU Innsbruck gibt es einen zentralen, ständig verfügbaren Linux Server, der dafür gedacht ist. Sein Rechnername ist zid-gpl.uibk.ac.at.

17.1 FTP

FTP (*file transfer protocol*) dient zum Austausch von Dateien zwischen verschiedenen Rechern. Sie können etwa von Windows Rechnern aus auf Linux zugreifen und die dort gespeicherten Dateien herunterladen.

Oben haben wir bereits gesehen, wie man mit dem KDE Dateimanager konqueror FTP Verbindungen aufbauen kann. Man gibt einfach `ftp://`, gefolgt vom Rechnernamen, ein.
Unter Windows ist oft das Programm WS–FTP verfügbar, ein grafisches Programm zu FTP.

Sowohl unter Windows als auch unter Linux ist eine Kommandozeilenversion von FTP verfügbar. Unter Linux starten Sie diese über die Shell, unter Windows über die MS–DOS Eingabeaufforderung. Geben Sie beispielsweise ein:

`ftp zid-gpl.uibk.ac.at`

Sie werden nun zuerst nach Ihrer Benutzerkennung, dann nach Ihrem Passwort gefragt. Geben Sie diese ein. Nun befinden Sie sich in der FTP Eingabeaufforderung.

Mit **`ls -l`** oder **`dir`** können Sie den Verzeichnisinhalt auflisten lassen, das Verzeichnis wechseln Sie mit **`cd`**. Die Anzeige, in welchem Verzeichnis Sie sich befinden, erfolgt über **`pwd`**. Mit

`get` ⟨**`Datei`**⟩

können Sie eine Datei herunterladen, mit

`put` ⟨**`Datei`**⟩

eine Datei hinaufladen. Mit

`mget` ⟨**`Dateimuster`**⟩

können Sie mehrere Dateien herunterladen, zum Beispiel:

`mget lies*`

mput funktioniert nach demselben Schema. Standardmäßig werden Sie bei `mget` und `mput` bei jeder Datei zur Bestätigung aufgefordert. Um das zu unterbinden, geben Sie davor einmalig **prompt** ein.

Wenn Sie `ftp` unter Windows starten, sollten Sie FTP vor dem Download anweisen, alle Dateien binär zu übertragen. Dazu geben Sie **binary** ein.
Mit **bye** beenden Sie `ftp` wieder.

Hier ein Beispiel für eine FTP Verbindung zum Rechner zid-gpl.uibk.ac.at (die fett dargestellten Zeichen werden vom Benutzer eingegeben, der Rest ist die Ausgabe des Servers):

```
ftp zid-gpl.uibk.ac.at
Connected to zid-gpl.uibk.ac.at.
220 zid-gpl.uibk.ac.at FTP server ... ready.
Name (zid-gpl.uibk.ac.at:c102mr): c102mr
331 Password required for c102mr.
Password: (wird nicht angezeigt)
230 User c102mr logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> get liesmich
local: liesmich remote: liesmich
227 Entering Passive Mode (138,232,1,28,165,227)
150 Opening BINARY mode data connection for liesmich (3936
bytes).
226 Transfer complete.
3936 bytes received in 0.000127 secs (3e+04 Kbytes/sec)
ftp> binary
200 Type set to I.
ftp> prompt
Interactive mode off.
ftp> mget *.ps
local: 20.ps remote: 20.ps
227 Entering Passive Mode (138,232,1,28,195,65)
150 Opening BINARY mode data connection for 20.ps (0 bytes).
226 Transfer complete.
local: a.ps remote: a.ps
227 Entering Passive Mode (138,232,1,28,231,36)
150 Opening BINARY mode data connection for a.ps (36083 bytes).
226 Transfer complete.
36083 bytes received in 0.005 secs (7.1e+03 Kbytes/sec)
local: upload.ps remote: upload.ps
227 Entering Passive Mode (138,232,1,28,232,190)
150 Opening BINARY mode data connection for upload.ps (0 bytes).
226 Transfer complete.
ftp> pwd
257 "/home/c102/c102mr" is current directory.
ftp> put upload.ps
```

```
local: upload.ps remote: upload.ps
227 Entering Passive Mode (138,232,1,28,189,165)
150 Opening BINARY mode data connection for upload.ps.
226 Transfer complete.
ftp> bye
221-You have transferred 57174 bytes in 5 files.
221-Total traffic for this session was 160608 bytes
221-in 5 transfers.
221-Thank you for using the FTP service on zid-gpl.uibk.ac.at.
221 Goodbye.
```

17.2 telnet

Mit `telnet` können Sie eine interaktive Verbindung zu einem anderen Rechner aufbauen. Sie landen auf der Shell, können hier wie gehabt Programme starten.

Um eine `telnet` Verbindung aufzubauen, geben Sie ein:

telnet ⟨**Rechner**⟩

Also beispielsweise:

telnet zid-gpl.uibk.ac.at

Unter Linux erfolgt dies auf der Kommandozeile, unter Windows wählen Sie im Start Menü Ausführen und geben den Befehl ein.

Wenn die Verbindung aufgebaut ist, werden Sie wieder nach Benutzerkennung und Passwort gefragt. War die Anmeldung erfolgreich, befinden Sie sich auf der Kommandozeile.

Ein Beispiel für eine `telnet` Verbindung:

```
telnet zid-gpl.uibk.ac.at
Trying 138.232.1.28...
Connected to zid-gpl.uibk.ac.at.
Escape character is '^]'.


login: c102mr
Password: (wird nicht angezeigt)
Last login: Wed May 9 11:38:06 from red-c102
[c102mr@zid-gpl c102mr]$ pwd
/home/c102/c102mr
[c102mr@zid-gpl c102mr]$ ls
...
[c102mr@zid-gpl c102mr]$ exit
Connection closed by foreign host.
```

Bei `telnet` Verbindungen zwischen Unix Systemen können Sie auch grafische Applikationen starten. Dazu müssen Sie auf Ihrem Rechner (nicht auf dem, zu dem Sie die Verbindung aufbauen) folgenden Befehl eingeben:

xhost +⟨Rechner⟩

⟨Rechner⟩ ist dabei das System, zu dem Sie die Verbindung aufbauen, also beispielsweise:

xhost +zid-gpl.uibk.ac.at

Damit erlauben Sie diesem Rechner, auf Ihrer Oberfläche Fenster zu öffnen. Wenn Sie die Verbindung beenden, können Sie dieses Recht wieder entfernen:

xhost -zid-gpl.uibk.ac.at

Auf dem Rechner, zu dem Sie sich verbunden haben, sollten Sie überprüfen, ob die DISPLAY Variable richtig gesetzt ist:

echo $DISPLAY

Hier sollte nun Ihr lokaler Rechner, gefolgt von „:0“ oder „:0.0“ stehen. Ist das nicht der Fall, müssen Sie die Variable auf den richtigen Wert setzen, um grafische Applikationen starten zu können:

export DISPLAY="⟨Rechner⟩:0"

Nun können Sie mit xterm -ls auf dem remote Rechner überprüfen, ob das funktioniert hat.

17.3 ssh

ssh (*secure shell*)[60] funktioniert ähnlich wie telnet, allerdings wird die Verbindung zwischen den Rechnern verschlüsselt, es ist damit weit sicherer. Sollte ssh auf Ihrem System verfügbar sein, sollten Sie im Zweifelsfall dieses verwenden.

Unter Linux starten Sie ssh durch die Eingabe von

ssh ⟨Rechner⟩

(also beispielsweise ssh zid-gpl.uibk.ac.at).
Am Uni–PC unter Windows starten Sie dieses im Menü Kommunikation über Putty und drücken nach dem Start die Eingabetaste, um den Zielrechner zu wählen.

17.4 Exceed

Exceed ist ein X Server für Windows. Damit können Sie auch unter Windows grafische Applikationen auf Unix Systemen ausführen.

Auf Uni–PCs wählen Sie im Menüs Kommunikation — Exceed und dort den Punkt Xterm Unix (Abb. 29).

Geben Sie hier den Rechnernamen (zum Beispiel zid-gpl.uibk.ac.at), Ihre Benutzerkennung und das Passwort ein und klicken Sie OK. Nun wird ein xterm gestartet, über das Sie wie gewohnt arbeiten können.

[60] ssh ist auf allen neueren Linux Distributionen vorhanden. Auf anderen Unix Varianten muss es (vom Systemadministrator) zusätzlich installiert werden.

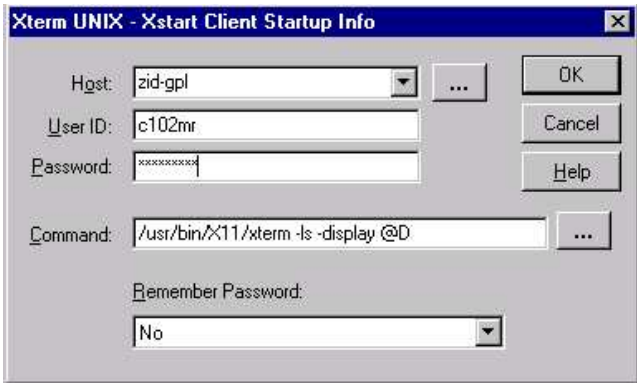

Abbildung 29: Verbindungsdialog beim Aufruf von Xterm Unix unter Exceed.

Auf anderen Windows Systemen müssen Sie sicherstellen, dass Exceed installiert ist. Wählen Sie nun aus dem Menü Exceed und dort Xstart. Nun erscheint ein Konfigurationsdialog wie in Abbildung 30.

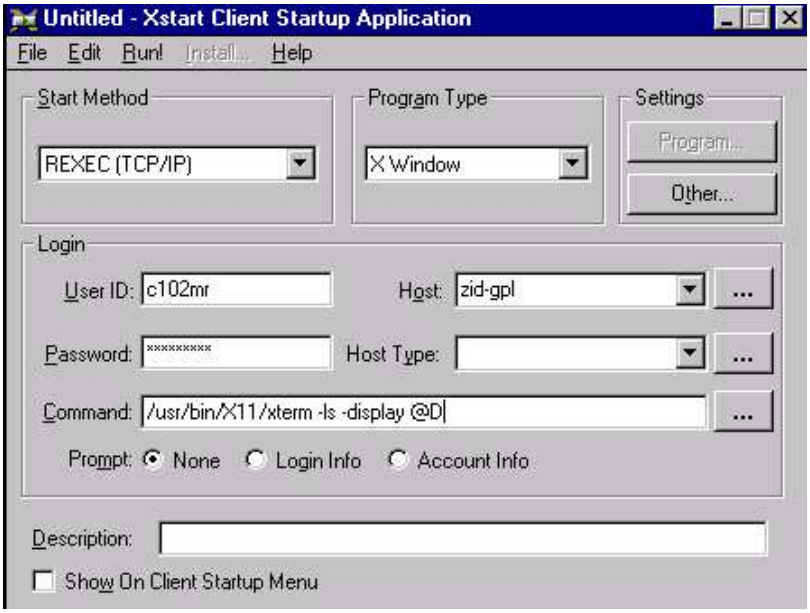

Abbildung 30: Exceed Client Konfigurationsdialog.

Wählen Sie hier als Verbindungsmethode `rexec`[61] und fügen Sie Rechnername, die Benutzerkennung und das Passwort ein. In der Zeile Command tragen Sie Folgendes ein:

```
/usr/bin/X11/xterm -ls -display @D
```

Speichern Sie dieses Konfiguration nun und wählen Sie Run!. In Zukunft können Sie die Verbindung neu aufbauen, indem Sie wieder Xstart aufrufen und diese Konfiguration laden.

[61] Dazu muss `rexec` am Linux System vom Administrator aktiviert worden sein. Sollte dies nicht der Fall sein, können Sie hier andere Verbindungsmethoden testen.

17.5 Aufgaben

1. Bauen Sie zum Rechner zid-luxinst.uibk.ac.at eine anonyme FTP Verbindung auf.
2. Was macht der Befehl `xhost`, wenn Sie ihn ohne Parameter aufrufen?

Die Lösungen finden Sie auf Seite 135.

18 Tipps und Tricks

A good magician never reveals his secret; the unbelievable trick becomes simple and obvious once it is explained. So too with UNIX.

In diesem Abschnitt sollen einige Dinge erwähnt werden, die nicht gut in den vorangegangenen Text hätten oder dort den (hoffentlich vorhandenen) roten Faden gestört hätten.

18.1 KDE Kürzel

- Mit Alt – F2 können Sie den Befehl ausführen Dialog öffnen und so schnell Befehle ausführen. Hier können Sie übrigens auch beliebige URLs eingeben, z.B.

  `http://www.uibk.ac.at/c102/`

  oder

  `file:/tmp/`

  für das Verzeichnis `/tmp`.
  Wenn Sie schnell einmal nach dem Wort „Linux“ auf Google suchen wollen, geben Sie Folgendes ein:

  `gg:Linux`

  Die Hilfeseite zu `man` könnten Sie so aufrufen:

  `man:man(1)`

- Mit Strg – Tab schalten Sie auf den nächsten, mit Strg – Shift – Tab auf den vorigen Desktop.

- Strg – Esc startet den KDE Systemüberwachung. Damit sehen Sie schnell die laufenden Prozesse, können diese auch beenden.

- Mit Strg – Alt – Esc können Sie ein abgestürztes grafisches Programm beenden. Der Mauszeiger verwandelt sich in einen Totenkopf. Wenn Sie nun auf ein Fenster klicken, wird es beendet; das entspricht `kill -9`.

- Im KDE Kontrollzentrum in der Sektion Regionaleinstellungen unter dem Punkt Tastenkürzel finden Sie die aktuellen Tastaturkürzel. Sie können diese hier auch verändern.

- Über den KDE Menü–Editor können Sie für Applikationen Tastaturkürzel setzen, um diese schnell starten zu können. Sie finden diesen im KDE Menü unter Präferenzen.

- Wenn Sie die Alt Taste drücken, können Sie mit der rechten Maustaste Fenster auch verschieben, wenn Sie irgendwo in den Fensterbereich klicken, Sie müssen dann das Fenster nicht an der Fensterleiste verschieben.

- Mit Alt und der linken Maustaste können Sie die Fenstergröße ändern, indem Sie in einen beliebigen Fensterteil klicken, nicht nur über den Fensterrahmen.
- Um ein Fenster nach hinten zu stellen (sodass es etwa andere Fenster nicht mehr verdeckt), drücken Sie die Alt Taste und klicken mit der mittleren Maustaste auf einen beliebigen Teil des Fensters.

18.2 Mittlere Maustaste

Sollte Ihre Maus nur 2 Maustasten haben, so sind die meisten Linux Systeme so konfiguriert, dass Sie die Funktionalität der mittleren Maustaste auch durch gleichzeitiges Drücken der linken und der rechten Maustaste erhalten.

18.3 bash Navigation

- In der `bash` gelangen Sie mit Strg – A an den Zeilenanfang, mit Strg – E an das Zeilenende.
- Mit Strg – K löschen Sie den Rest der Zeile, mit Strg – U von der aktuellen Position bis zum Anfang.
- Mit de Richtungstasten ↑ und ↓ blättern Sie durch die bisher ausgeführten Befehle.
- In einem X Terminal können Sie mit Shift – PageUp und Shift – PageDown die bisherigen Ausgaben durchblättern.
- Im X Terminal können Sie die Strg Taste und die linke, mittlere oder rechte Maustaste drücken. Es erscheinen unterschiedliche Menüs, über die Sie das X Terminal konfigurieren (zum Beispiel die Schriftgröße ändern) oder an Programme Signale schicken können.
- Im `xterm` markieren Sie mit der linken Maustaste Zeichen, Wörter oder Zeilen. Wenn Sie auf ein Wort doppelklicken, so wird dieses markiert.
  Wenn Sie danach mit der rechten Maustaste auf ein nachfolgendes Wort klicken, wird die Markierung bis inklusive dieses Wort erweitert.
  Durch Drücken der mittleren Maustaste wird die Markierung eingefügt. Das funktioniert nicht nur im `xterm`, sondern auch in fast allen anderen grafischen Applikationen.

18.4 Passwort

Mit `passwd` können Sie Ihr Passwort ändern; Sie werden zuerst nach dem alten Passwort, nach der Eingabe zweimal nach dem neuen Passwort gefragt. Sollte Ihr Passwort nicht akzeptiert worden sein, etwa weil Sie nicht zweimal dasselbe Passwort eingegeben oder ein zu einfaches Passwort gewählt haben, bekommen Sie eine entsprechende Meldung angezeigt.

18.5 Dokumentation

Verweise zu weiterführender Dokumentation finden Sie — sowohl allgemein zu Linux als auch speziell zu Linux an der LFU Innsbruck — unter

`http://linuxdoc.uibk.ac.at`

Nicht Linux spezifische Dokumentation (etwa zu ssh oder zur Mail–Thematik) finden Sie unter

`http://www.uibk.ac.at/zid/`

18.6 Remote Zugriff

Wenn Sie remote (über `telnet` oder Exceed) auf ein Linux System zugreifen und den Aufruf einer grafischen Applikation nicht mehr wissen, können Sie die KDE Kontrollleiste mit dem Befehl `kicker` starten. Zum Beenden müssen Sie den `kicker` Prozess mit `ps auxw` heraussuchen und mit `kill` beenden.

18.7 rpm

Wenn Sie wissen wollen, welche Dateien zu einem Programm gehören (zum Beispiel Dokumentation), führen Sie folgenden Befehl aus (hier am Beispiel von `bash`):

`rpm -qf $(which bash)`[62]

So erhalten Sie das sogenannte RPM Paket, über das `bash` installiet wurde. Als Ausgabe erhalten Sie beispielsweise:

`bash-2.04-11`

Lassen Sie die Versionsnummer weg (das, was nach dem vorletzten Bindestrich folgt) und führen Sie damit den folgenden Befehl aus:

`rpm -ql bash | less`

Sie erhalten nun eine Liste der Dateien, die zu diesem Paket gehören.

Nicht alle Dateien (aber ein Großteil) sind Bestandteil eines Paketes. Sollten Sie ein Programm abfragen, das zu keinem Paket gehört, erhalten Sie eine entsprechende Fehlermeldung.

18.8 Konsole

Sie können unter Linux aus der grafischen Oberfläche auf eine textbasierende Konsole umschalten. Dazu drücken Sie Strg – Alt – F1. Mit Alt – F1 bis Alt – F6 haben Sie nun 6 Textkonsolen zur Verfügung, auf denen Sie sich anmelden können. Mit Alt – F7 schalten Sie zurück auf die grafische Oberfläche.

[62] `rpm` ist fast nur auf Linux Systemen verfügbar (und hier nur auf den Systemen, die `rpm` verwenden, nicht aber auf Debian Linux basierenden Systemen).

18.9 Aufgaben

1. Setzen Sie im KDE Menü–Editor für das KDE Hilfezentrum das Tastaturkürzel Strg – Shift – h.
2. Was passiert, wenn Sie bei der Eingabe eines Befehls statt der Eingabetaste Strg – C drücken?
3. Was machen die Befehle `rpm -qi bash` beziehungsweise `rpm -qa`?

Die Lösungen finden Sie auf Seite 136.

19 Lösungen zu den Aufgaben

That's the whole problem with science. You've got a bunch of empiricists trying to describe things of unimaginable wonder.

(Calvin and Hobbes)

19.1 Abschnitt 3 (Seite 22)

1. Bei dieser Aufgabe sollten Sie versucht haben, die Größe, Position und das Aussehen von Fenstern zu ändern. Weitere Einstellungsmöglichkeiten finden Sie wie folgt:

   - Ändern des Aussehens der Oberfläche durch Drücken der rechten Maustaste am Arbeitsplatz und Wahl von Arbeitsfläche einrichten.
   - Viele weitere Einstellungsmöglichkeiten finden Sie im KDE Menü unter Kontrollzentrum.

   Sie sollten auch versucht haben, sowohl über das KDE Menü als auch die KDE Kontrollleiste Programme zu starten.

   Hilfeseiten öffnen Sie mit `man` ⟨`Sektion`⟩ ⟨`Befehl`⟩ auf der Kommandozeile oder im KHelpcenter durch Auswahl der Hilfeseite in der entsprechenden Sektion beziehungsweise durch Eingabe der URL `man:`⟨`Befehl`⟩`(`⟨`Sektion`⟩`)`. Für Benutzerkommandos sollte die Sektion immer 1 sein.

19.2 Abschnitt 4 (Seite 38)

1. **`cd /bin`** wechseln in das Verzeichnis `/bin`; dies ist eine absolute Pfadangabe. **`cd bin`** wechselt, wenn Sie sich in `/usr` befinden, nach `/usr/bin`; dies bezeichnet man als relative Pfadangabe. Würden Sie sich in `/usr/local` befinden, würde `cd bin` nach `/usr/local/bin` wechseln.

   „**`.`**“ bezeichnet das aktuelle Verzeichnis. Mit **`cd .`** bleiben Sie daher im aktuellen Verzeichnis.

   „**`..`**“ bezeichnet das übergeordnete Verzeichnis. Befinden Sie sich in `/usr/bin`, wechselt cd .. daher nach `/usr`.

   `cd ~/bin` wechselt in das Verzeichnis `bin` in Ihrem HOME Verzeichnis. „**`~`**“ steht immer für Ihre HOME Verzeichnis.

2. Wenn Sie in ein nicht vorhandenes Verzeichnis wechseln, zum Beispiel mit **`cd ~/notthere`**, bekommen Sie ein Fehlermeldung wie folgende:

   ```
   bash: cd: /home/c102mr/notthere: No such file or directory
   ```

3. Mit **cd ~** wechseln Sie in Ihr HOME Verzeichnis und erstellen dort mit **mkdir junk** dieses Verzeichnis. Kopieren Sie beispielsweise die Datei `liesmich` dort hinein:

   cp ~/liesmich ~/junk

   Mit **rmdir junk** können Sie dieses Verzeichnis nicht löschen, da `rmdir` nur leere Verzeichnisse löscht; Sie erhalten eine Fehlermeldung.

   Mit **rm -r ~/junk** können Sie das Verzeichnis rekursiv, inklusive Inhalt, löschen.

4. Mit **mkdir junk** erstellen Sie wiederum dieses Verzeichnis. Mit **cp liesmich liesmich.1** erstellen Sie eine Kopie der Datei `liesmich`.

   Mit

   mv liesmich.1 liesmich.kopie
   mv liesmich.kopie junk

   würden Sie diese zuerst umbenennen und dann nach `junk` verschieben.

   Schneller geht das mit einem einzigen Befehl:

   mv liesmich.1 junk/liesmich.kopie

   Mit

   cp liesmich liesmich.1
   cp liesmich liesmich.2
   cp liesmich liesmich.3

   erstellen Sie 3 Kopien der Datei `liesmich` Wenn Sie den folgenden Befehl ausführen:

   mv liesmich.1 liesmich.2 liesmich.3

   erhalten Sie eine Fehlermeldung. Wenn Sie mehrere Dateien oder Verzeichnisse bei `mv` angeben, muss das letzte ein Verzeichnis sein; dann werden die voranstehenden dorthin verschoben:

   mv liesmich.1 liesmich.2 liesmich.3 junk

5. Die Hilfeseite zu `ls` rufen Sie mit **man ls** auf.

   -A steht für „almost all“: Es werden alle Dateien, inklusive der mit „**.**“, angezeigt — allerdings nicht die speziellen Verzeichnisse „**.**“ und „**..**“.

   -h ist eine nur unter Linux, nich aber auf anderen Unix Systemen, verfügbare Option. Dadurch werden die Dateigrößen in Kilobyte und Megabyte angezeigt.

   -r sortiert die Dateien in umgekehrter Reihenfolge; standardmäßig werden die Dateien alphabetisch angeordnet, durch `-r` wird die Reihenfolge umgedreht.

   Wenn Sie ein Verzeichnis, nicht aber dessen Inhalt anzeigen wollen, verwenden Sie die Option `-d`, also hier:

   ls -ld /tmp

6. Der Befehl `cat` gibt Zeichenketten aus. In diesem Fall nimmt er als Eingabe die beiden Dateien `/etc/profile` und `/etc/services` und gibt sie in die Datei `~/spr` aus. Die beiden Dateien werden also aneinandergehängt, der Inhalt wird in die neue Datei geschrieben. Würde die zu erstellende Datei bereits bestehen, würde sie überschrieben.

7. Dieser Befehl gibt folgende Kriterien für `ls` an:

   - Die Datei in `/etc` beginnt mit „`m`“.
   - Es folgt ein „`a`“ oder ein „`o`“.
   - Danach kommen beliebige Zeigen, gefolgt von „`.conf`“.
   - Es folgt entweder „`ig`“ oder nichts.

   Die Ausgabe könnte wie folgt aussehen:

   ```
   /etc/man.config
   /etc/modules.conf
   ```

8. Der erste Befehl durchsucht rekursiv und ohne Beachtung von Groß– oder Kleinschreibung alle Dateien und Verzeichnisse im Verzeichnis `/etc/`, die mit einem `s` beginnen, nach der Zeichenkette `xxx`. Fehler werden nach `/dev/null` geschrieben (das ist der „Müllkübel“, alles, was dorthin geschrieben wird, verschwindet). Die Ausgabe wird an das Programm `less` übergeben und von diesem dargestellt.

   Der zweite Befehl unterscheidet sich insofern, als Fehler nicht nach `/dev/null` geschrieben werden, sondern durch `2>&1` die Fehlermeldungen dieselbe Ausgabe wie die Standardausgabe verwenden, beide werden an `less` übergeben. Wenn Fehler auftreten, werden sie auch in der Darstellung durch `less` sichtbar.

   Würde man die dritte Variante verwenden, würde die normale Ausgabe verworfen, es würden nur die Fehler ausgegeben.

9. Mit `touch` können Sie auch das Änderungsdatum einer Datei ändern. Für die Datei `~/liesmich` würden Sie folgenden Befehl verwenden:

   ```
   touch -t 0403311235 ~/liesmich
   ```

10. Mit `find` können Sie nicht nur nach Dateinamen suchen, sondern auch nach Dateitypen. Folgender Befehl listet alle Verzeichnisse auf, die sich unter Ihrem HOME Verzeichnis befinden:

    ```
    find ~ -type d
    ```

11. Dazu wird die Option `-newer` von `find` verwendet:

    ```
    find ~ -newer ~/Desktop
    ```

12. Mit

    `tar cvf mein.archiv.tar liesmich Desktop`

    erstellen Sie ein komprimiertes Archiv, das die Datei `liesmich` und das Verzeichnis `Desktop` inklusive aller darin enthaltenen Dateien und Verzeichnisse beinhaltet.

    Mit

    `tar rvf mein.archiv.tar /etc/fstab`

    fügen Sie nun die Datei `/etc/fstab` hinzu.

    Um aufzulisten, welche Dateien im Archiv vorhanden sind, verwenden Sie folgenden Befehl:

    `tar tvf mein.archiv.tar`

19.3 Abschnitt 5 (Seite 44)

1. Die Zugriffsrechte für Ihr HOME Verzeichnis erhalten Sie mit

   `ls -ld ~`

   Normalerweise sind diese wie folgt gesetzt:

   `rwx --- ---`

   Das bedeutet, dass niemand in Ihr HOME Verzeichnis wechseln kann. Egal, welche Rechte Sie für die Dateien und Verzeichnisse darunter vergeben — niemand anderer als Sie kann darauf zugreifen.

2. Mit

   `mkdir /tmp/verzeichnis`

   erstellen Sie in `/tmp` ein Verzeichnis `verzeichnis`.

   Mit

   `chmod 711 /tmp/verzeichnis`

   geben Sie sich für dieses Verzeichnis alle Rechte, allen anderen aber nur das Recht, dorthin zu wechseln. Da sowohl das `read` als auch das `write` Attribut fehlen, können die Benutzer weder die darin enthaltenen Dateien oder Verzeichnisse auflisten noch neue erstellen.

19.4 Abschnitt 6 (Seite 50)

1. Mit **`cp /dev/null ~`** kopieren Sie die spezielle Datei `/dev/null` in Ihr HOME Verzeichnis, es entsteht eine leere Datei `null`.

   Wenn Sie eine Datei nach `/dev/null` kopieren, landet diese im „Müllschlucker“, sie verschwindet (die Datei `/dev/null` wird nicht überschrieben — das würde passieren, wenn Sie eine „normale“ Datei auf eine bereits vorhandene kopieren).

2. Wenn Sie die Ausgabe eines Befehles nach `/dev/null` umleiten, zum Beispiel:

   `ls ~ > /dev/null`

   „verschwindet“ diese — sie wird nicht angezeigt, und auch in `/dev/null` wird die dorthin geschickte Ausgabe nicht gespeichert.

3. Mit **`mmd A:dir1`** legen Sie auf der Diskette ein Verzeichnis an, mit **`mmd A:dir1/dir1-1`** darin ein Unterverzeichnis. Der Befehl

   `mcopy ~/liesmich A:dir1/dir1-1/`

   kopiert die Datei `~/liesmich` dorthin. Sie können den ganzen erstellten Verzeichnisbaum mit `mdeltree` löschen:

   `mdeltree A:dir1`

4. Kopieren Sie mit **`mcopy ~/liesmich A:`** wiederum diese Datei auf die Diskette. Der Befehl `mmove` benennt sie nun um:

   `mmove A:liesmich A:liesmich.2`

5. Das Verzeichnis `/usr/share/doc` beinhaltet einen Großteil der Dokumentation (mit Ausnahme der `man` Seiten). Wenn Sie in dieses Verzeichnis wechseln und **`ls -d bash*`** ausführen, sehen Sie, welche Verzeichnisse mit Dokumentation zur `bash` vorhanden sind, zum Beispiel:

   ```
   bash-2.04
   bash-doc-2.04
   ```

   Die Hilfeseiten befinden sich unter:

   `/usr/share/man`[63]

   Da die Hilfe zur `bash` in der Sektion 1 zu finden ist, lautet der vollständige Verzeichnisname:

   `/usr/share/man/man1`

19.5 Abschnitt 7 (Seite 55)

1. Geben Sie im command mode „`:help`“ ein (diese Information finden Sie sowohl in der Hilfeseite als auch, wenn Sie `vi` ohne eine Datei aufrufen).

2. Hierzu gibt es nicht allzu viel zu sagen — versuchen Sie’s einfach.

3. Halten Sie dazu die Antworten zu den folgenden Abschnitten schriftlich fest. Machen Sie gelegentlich Sicherheitskopien, um im Falle einer groben Fehlbedienung nicht alles zu verlieren.

[63] Auf manchen Linux – und den meisten Unix Systemen finden Sie diese unter `/usr/man`.

19.6 Abschnitt 8 (Seite 62)

1. In `top` bekommen Sie durch Eingabe von „?“ eine kurze Bedienungsanleitung (die Sie mit einer beliebigen Taste wieder beenden können).

   Geben Sie in `top` den Buchstaben u (*user*), gefolgt von Ihrer Benutzerkennung ein; nun werden nur mehr Ihre Prozesse angezeigt. Durch nochmaliges Drücken von u und dem Drücken der Eingabetaste werden wieder die Prozesse aller Benutzer angezeigt.[64]

2. `top` sollte einer der Prozesse sein, die im Moment am meisten CPU brauchen, sodass Sie ihn weit oben in der Liste finden. Ganz links sehen Sie die Prozessnummer (PID) von `top`. Drücken Sie k (*kill*) und geben Sie auf der erscheinenden Eingabeaufforderung die PID ein. Wenn Sie gefragt werden, welches Signal Sie schicken wollen, drücken Sie einfach die Eingabetaste. Dadurch wird das Signal 15 geschickt, der Standardwert (übrigens auch bei `kill`: `kill` ⟨`PID`⟩ ist daher identisch mit `kill -15` ⟨`PID`⟩).

3. Starten Sie mit **`xterm -ls`** ein neues X Terminal. Dessen Prozess suchen Sie nun:

   `ps auxw | grep xterm | grep -v grep`

   Die Ausgabe könnte so aussehen:

   ```
   root 10121 0.0 1.8 4996 2408 ?  S  17:18  0:00 xterm -ls
   root 10169 0.0 1.9 5068 2520 ?  S  17:23  0:00 xterm -ls
   ```

   Der letzte aufgelistete Prozess ist Ihr neues `xterm`. Beenden Sie diesen nun:

   `kill 10169`

4. Einige Beispiele:

Name	Zahl	Beschreibung
SIGKILL	9	Beendet einen Prozess (bedingungslos).
SIGTERM	15	Schickt dem Prozess eine Meldung, sich zu beenden.
SIGTSTP	18	Suspendiert den Prozess.

   Sie können mit einem Prozess mit `kill` ein Signal schicken. Dabei können Sie den Namen oder die Zahl des Signals verwenden. Folgende Beispiele sind daher identisch:

   `kill -9 10169`
   `kill -KILL 10169`
   `kill -SIGKILL 10169`

[64] Auf vielen Unix Derivaten bekommt man mit u und + wieder alle Benutzer angezeigt.

5. Sehen Sie dazu die Antwort der vorigen Frage.

6. Starten Sie beispielsweise **`less /etc/fstab`**. Wechseln Sie nun in anderes `xterm` und suchen Sie dort den Prozess heraus:

   `ps auxw | grep less | grep -v grep`

   Die Ausgabe könnte wie folgt aussehen:

   ```
   c102 1062 0.0 0.6 1840 808 pts/1 S 18:49 0:00 less /etc/fstab
   ```

   Schicken Sie diesem nun das Signal:

   `kill -TSTP 1062`

   Im anderen Fenster verschwindet nun die Ausgabe von `less`, es wird eine Meldung ausgegeben:

   ```
   P[1]+     Stopped     less /etc/fstab
   ```

   Diese Meldung haben wir schon kennengelernt, und zwar beim Drücken der Tastenkombination Strg – Z: dieses entspricht dem Signal `SIGSTP`. Mit **`fg`** können Sie also `less` wiederaufnehmen.

7. Dieser Befehl liefert den Status 0. `ls` beendet mit einem Fehler, da die angegebene Datei nicht existiert. Durch das `||` wird nun `true` gestartet. Dieses Programm macht nichts, beendet sich nur mit Status 0. Da dies der letzte ausgeführte Befehl der Kette ist, ist dies auch der endgültige *exit status*.

8. Geben Sie dazu ein:

   `history | grep man`

   Die Ausgabe könnte wie folgt aussehen:

   ```
   1034 man ls
   1046 man tar
   1090 man vi
   1091 man top
   1096 man 7 signal
   1107 man true
   ```

19.7 Abschnitt 9 (Seite 65)

1. Wenn Sie nur **`vmstat`** eingeben, wird eine Zeile ausgegeben, die den Durchschnitt seit dem Systemstart angibt. Mit **`vmstat 1`** bekommen Sie eine durchgehende Ausgabe, die Sie mit Strg – C abbrechen. Dies bewirkt, dass einmal pro Sekunde der Systemstatus ausgegeben wird (die erste Zeile ist wieder der Durchschnitt seit dem Systemstart). Geben Sie **`vmstat 1 10`** ein, wird ebenfalls einmal pro Sekunde eine Ausgabe geliefert, aber nach 10 Ausgaben beendet sich `vmstat`

```
procs                  memory  swap    io    system      cpu
r b w   swpd free buff cache si so bi bo   in   cs us sy id
0 0 0 92660 3072 2620 42296  3  1  2  1   88   82  4  1 96
1 0 0 92660 3076 2620 42296  0  0  0  0  154  776  2  4 94
0 0 0 92660 3072 2620 42296  0  0  0  0  150  776 14  2 84
1 0 0 92660 3068 2620 42296  0  0  0  0  209  102  8  3 89
3 0 0 92656 3152 2620 42212 48  0 12  0  465  210 17 14 69
```

Die ersten drei Spalten spezifizieren die Prozesse des Systems. Die ersten drei Spalten geben an, wieviele Prozess darauf warten, ausgeführt zu werden (`r`, oder wieviele Prozesse blockiert sind (`b`).
Es folgen Angaben zum Arbeitsspeicher und zum Swap. Bei `si` und `so` sehen Sie, ob gerade auf den Swap zugegriffen wird. Sind die Werte hier hoch, hat das System Probleme. Es folgen Angaben zum Zugriff auf Blockgeräte (vor allem die Festplatte) und über das System. Die letzten drei Spalten geben die CPU Auslastung an — die letzte etwa, wieviel Prozent der CPU im Moment nicht beschäftigt sind. Ist dieser Wert 0, so ist das System voll ausgelastet.

2. In `/proc/pci` finden Sie eine Reihe von Hardware-Informationen zum System. Die folgenden beiden Zeilen, ein Ausschnitt aus dieser Datei, zeigen etwa den Typ der Grafik– und der Netzwerkkarte:

   ```
   VGA compatible controller: ATI Mach64 GB (rev 92).
   Ethernet controller: 3Com 3C905B 100bTX (rev 36).
   ```

19.8 Abschnitt 10 (Seite 67)

1. Mit **`echo $DISPLAY`** geben Sie den Wert der Variable `DISPLAY` aus. Dieser ist, wenn Sie direkt an einem Linux Rechner arbeiten und nicht beispielsweise über `telnet` verbunden sind `:0.0`.

   Ändern Sie diesen auf einen beliebigen Wert:

   `DISPLAY=1`

   Wenn Sie nun mit **`xterm -ls`** ein neues X Terminal zu öffnen versuchen, bekommen Sie eine Fehlermeldung:

   ```
   xterm Xt error: Can't open display: 1
   ```

   Die `DISPLAY` Variable setzt fest, wo das Fenster geöffnet werden soll. Da dies nun auf einen falschen Wert gesetzt ist, kann kein Fenster geöffnet werden. Mehr dazu finden Sie im Abschnitt 17.

2. Setzen Sie `PS1` auf einen anderen Wert, zum Beispiel:

   `PS1="hallo: "`

   Nun ändert sich die Anzeige der Eingabeaufforderung — statt Ihrer Benutzerkennung, dem Rechnernamen und dem aktuellen Verzeichnis wird nun „`hallo:` “ angezeigt. Das Aussehen dieser Anzeige wird durch die Variable `PS1` festgelegt.

Sie können das nun wieder auf einen „normaleren“ Wert ändern:

PS1="[\u@\h \W]\\$ "[65]

Eine genaue Dokumentation, was das bedeutet und was Sie hier setzten können, finden Sie in der `man` Seite der `bash`. Suchen Sie dort nach dem Abschnitt PROMPTING.

3. Die Variable `PATH` beinhaltet den Pfad, in denen nach Programmen gesucht wird. Damit müssen Sie nicht das Programm inklusive Pfad sondern nur das Programm eingeben. Ein Beispiel, wie der Wert der Variable aussehen könnte:

   `/bin:/usr/bin:/usr/bin/X11:`

   In diesen Verzeichnissen, die durch einen Doppelpunkt getrennt werden, wird nach den Programmen gesucht. Der Doppelpunkt am Ende wird übrigens dazu verwendet, dass auch im aktuellen Verzeichnis nach dem Programm gesucht wird.

   Um nun Ihr HOME Verzeichnis aufzunehmen, würden Sie Folgendes eingeben:

   PATH=$PATH:~

   Dadurch wird `PATH` auf sich selbst plus den nachfolgenden Teil gesetzt, also um den nachfolgenden Teil erweitert. Der neue Wert würde nun zum Beispiel so lauten:

   `/bin:/usr/bin:/usr/bin/X11::/home/c102mr`

   Um das Verzeichnis aufzunehmen, in dem Sie sich gerade befinden, würden Sie verwenden:

   PATH=$PATH:$(pwd)

4. Damit auch im aktuellen Verzeichnis nach Programmen gesucht wird, muss das aktuelle Verzeichnis „.“ im Pfad enthalten sein, also zum Beispiel:

   PATH=/bin:/usr/bin:/usr/bin/X11:~/bin:.

19.9 Abschnitt 11 (Seite 69)

1. Beim ersten Beispiel wird „?“ nicht maskiert, es behält seine spezielle Bedeutung (steht für ein beliebiges Zeichen). Die beiden anderen Zeichen maskieren dieses, es würde also wirklich nach Dateien gesucht, die mit einem Fragezeichen beginnen.

2. Dieser Befehl listet alle Dateien auf, die mit „`lies mic`“ beginnen und danach ein beliebiges Zeichen haben.

   Würde man die Anführungszeichen weglassen, würde nach der Datei `lies` sowie einer Datei, die mit `mic` beginnt und danach ein beliebiges Zeichen hat, gesucht.

[65] Bei anderen Shells ist diese Zeile entsprechend der Dokumentation anzupassen.

Würde man das schließende Anführungszeichen nach dem Fragezeichen schreiben, würde dieses maskiert, es würde nach einer Datei gesucht, die am Ende ein Fragezeichen hat (statt einem beliebigen Zeichen).

Diesen Befehl hätte man übrigens auch so schreiben können:

```
ls lies\ mic?
```

19.10 Abschnitt 12 (Seite 81)

1. Nein, es gibt keinen Unterschied. Dies sind die beiden möglichen Schreibweisen für `test`.

2. Um beispielsweise zu überprüfen, ob `/usr/bin/X11` ein symbolischer Link ist, können Sie Folgendes verwenden:

   ```
   test -L /usr/bin/mformat && echo "Ist ein Link."
   ```

   Der folgende Befehl ruft `ls` mit der Option `-L` auf, falls es sich um einen Link handelt. Dadurch werden nicht die Daten des Link, sondern der Datei, auf die er zeigt, angezeigt. Andernfalls wird `ls` ohne diese Option ausgeführt.

   ```
   FILE=/usr/bin/mformat
   [ -L "$FILE"] && ls -lL $FILE || ls -l $FILE
   ```

3. Das erreichen Sie, indem Sie einfach mit dem Rufzeichen die `test` Anweisung aus dem vorigen Besipiel negieren (`&&` wurde durch `||` ersetzt, um das Ergebnis besser darzustellen, dadurch ist das Beispiel eine doppelte Verneinung):

   ```
   [ ! -L /usr/bin/mformat ] || echo "Ist ein Link."
   ```

4. Kombinieren Sie dazu mit `-o` zwei `test` Überprüfungen. Durch `-o` muss einer der beiden zutreffen (beachten Sie die Zeilenfortsetzung mithilfe von „\“):

   ```
   [ -f /usr/bin/mformat -o -L /usr/bin/mformat ] && \
   echo "Ist eine Datei oder ein Link."
   ```

19.11 Abschnitt 13 (Seite 87)

1. Zuerst wird die Variable `FILE` gesetzt.

   Ist die Datei `/etc/ld.so.conf` vorhanden, die Datei `/tmp/ld.so.conf` aber nicht, wird `/etc/ld.so.conf` nach `/tmp` kopiert.

   Nun wird eine `for` Schleife aufgerufen. Für `/lib`, `/usr/lib` und `/opt/lib` wird in einer `if` Bedingung jeweils überprüft, ob diese in der Datei vorhanden sind (sie müssen alleine auf einer Zeile stehen — der Zeilenanfang wird gefolgt von dem Ausdruck, gleich danach kommt das Zeilenende).

   Ist dies nicht der Fall, wird die Zeile mit `echo` an die Datei angehängt.

2. Als Trennzeichen von `sed` wird hier „+“ verwendet.

   Der Ausdruck sucht in der Datei `~/.bash_profile` beliebig viele Leerzeichen am Zeilenbeginn, gefolgt von `PATH=` und beliebigen anderen Zeichen bis zum Zeilenende. Vor diesen Ausdruck wird nun „#“ gestellt. Im ersten Ausdruck wird für den gesamten Ausdruck (ohne Zeilenbeginn und Zeilenende) eine Gruppierung erstellt, die dann beim ersetzenden Ausdruck wieder eingefügt wird.

3. Die Funktion `pskill` fängt den beschriebenen Fehler ab, wenn Sie sie wie folgt formulieren:

```
pskill()
{
      PROC=$(ps auxw | grep "$1" | grep -v "grep" | \
          awk '{ print $2 }')
      if [ "$PROC" ]
      then
           kill $PROC
      else
           return 1
      fi
}
```

   Zuerst wird die Suche (mit dem übergebenen Parameter) ausgeführt. Ist diese erfolgreich, ist der Inhalt der Variable `PROC` nicht leer. Damit ist die `test` Bedingung erfolgreich, `kill` wird ausgeführt. Andernfalls beendet sich die Funktion mit dem *exit status* 1 (bei Funktionen wir `return` verwendet — `exit` würde bewirken, dass das gesamte Script beendet wird; so wird nur der Status der Funktion selbst gesetzt).

   Für Wagemutige: Sie können die Funktion so abändern, dass mehr als ein Parameter übergeben werden kann. Verwenden Sie dazu eine `for` Schliefe über `$@`.

19.12 Abschnitt 14 (Seite 102)

1.
```
finddir() {
     FULL=$(which $1 2> /dev/null)
     if [ "$FULL" ]
     then
          DIR=$(dirname "$FULL" 2> /dev/null)
          if [ "$DIR" ]
          then
               echo $DIR
          else
               return 1
          fi
     else
```

```
        return 2
    fi
}
```

Zuerst wird mit `which` der volle Name (inklusive Pfad) des übergebenen Parameters gesucht.

Wurde dieser gefunden, wird versucht, mit `dirname` das Verzeichnis aufzulösen. Bei Erfolg wird es mit `echo` ausgegeben.

Tritt ein Fehler auf (wenn eine Variable leer ist), wird mit `return` ein *exit code* größer als 0 zurückgeliefert.

Sowohl bei `which` als auch bei `dirname` werden die Fehler verworfen, sodass sie gegebenenfalls nicht angezeigt werden.

Für Wagemutige: Sie können die Funktion so abändern, dass mehr als ein Parameter übergeben werden kann. Verwenden Sie dazu eine `for` Schliefe über `$@`.

2. Sie können die Kompressionsrate durch Angabe von $-\langle$`Rate`$\rangle$ angeben, wobei 1 die niedrigste und 9 die höchste Kompressionsrate ist. Bei hoher Kompressionsrate dauert die Komprimierung zwar etwas länger, die entstehende Datei ist jedoch kleiner.

3.
```
LAST=$(last)

COUNT=$(echo "$LAST" | grep -v "^$" | \
    grep -v "wtmp begins" | awk '{ print $1 }' | \
    sort -u | wc -l | sed -e 's+ *++')

DATE=$(echo "$LAST" | grep "wtmp begins" | \
    awk -F"wtmp begins " '{ print $2 }')

if [ "$COUNT" -a "$DATE"]
then
    echo "$COUNT verschiedene Benutzer seit: $DATE"
else
    exit 1
fi
```

Zuerst wird die gesamte Ausgabe von `last` in die Variable `LAST` geschrieben.

Es werden aus der Ausgabe die leeren Zeilen sowie jene, die „`wtmp begins`“ beinhaltet, herausgefiltert. Mit `awk` wird nun die erste Spalte ausgegeben — hier stehen die Benutzer. Das wird nun mit `sort` sortiert, mit `uniq` werden doppelte Einträge ausgeschieden. Das Ergebnis wird mit `wc` gezählt (`-l` bewirkt, dass nur die Zeilenanzahl ausgegeben wird). Da `wc` in seiner Ausgabe Leerzeichen beinhaltet, werden diese mit `sed` entfernt. Das Ergebnis — die Zahl der unterschiedlichen am System bisher angemeldeten Benutzer — ist schließlich der Wert der Variable `COUNT`.

Für die Variable `DATE` wird aus der Variable `LAST` nun die Zeile mit `wtmp begins` herausgesucht. `awk` gibt dann das Datum aus, indem es „`wtmp begins` “ als Spaltentrenner nimmt und die zweite Spalte (hier steht das Datum) ausgibt.

Sind sowohl `COUNT` als auch `DATE` nicht leer, wird schließlich das Ergebnis ausgegeben, andernfalls beendet sich das Script mit *exit status* 1.

4.
```
for file in $@
do
     if [ ! -f $file ]
     then
          echo "$file exisitiert nicht.">&2
          break
     fi
     NN=$(basename $file | tr '[A-Z]' '[a-z]')
     if [ "$NN" ]
     then
          [ ! -f $NN ] && cp $file $NN
     fi
done
```

Alle übergebenen Parameter sind in der Variable „`$@`“ gespeichert. In einer `for` Schleife wird nun jeder dieser Parameter behandelt.

Der Parameter muss eine Datei sein. Ist dies nicht der Fall, so wird eine Fehlermeldung ausgegeben (hier haben wir einen neuen Fall — der Standardfehler wird nicht zur Standardausgabe umgeleitet, sondern umgekehrt), und mit `break` wird zur nächsten Iteration der `for` Schleife gewechselt.

Andernfalls wird der Dateiname (ohne Pfad!) in Kleinbuchstaben umgewandet und in der Variable `NN` gespeichert.

Ist `NN` nicht leer, so wird überprüft, ob die zu erstellende Datei noch nicht vorhanden ist; stimmt dies, so wird die Datei kopiert (nacht `$NN` — das heißt, die Datei wurde in Kleinbuchstaben umgewandelt, der Pfad–Teil verworfen, sodass die Datei ins aktuelle Verzeichnis kopiert wird).

19.13 Abschnitt 15 (Seite 103)

1. Dazu tragen Sie in die Datei `~/.bashrc` in Ihrem HOME Verzeichnis folgende Zeile ein:

   ```
   alias cd..='cd ..'
   ```

   Beim nächsten Start einer Shell ist die Änderung aktiv.

19.14 Abschnitt 16 (Seite 110)

1. Wählen Sie dazu im konqueror das Menü View, dort das Untermenü View Mode und den Eintrag Detailed List View.

2. Wählen Sie im konqueror das Menü Ansicht, und dort den Eintrag Versteckte Dateien anzeigen.

3. Um den Mülleimer zu leeren, klicken Sie mit der rechten Maustaste auf dessen Symbol am Arbeitsplatz. Im erscheinenden Menü wählen Sie Mülleimer leeren.

4. Klicken Sie mit der rechten Maustaste auf eine freie Stelle der Kontrolleiste. Wählen Sie im erscheinenden Menü Hinzufügen, dann das Untermenü Miniprogramm und schließlich Systemüberwachung.

   Nun ist das Miniprogramm auf der Kontrollleiste sichtbar. Wenn Sie mit der mittleren Maustaste auf den strichlierten Bereich links neben dem Applet klicken, können Sie es verschieben. Wenn Sie auf diesen Bereich mit der rechten Maustaste klicken, erscheint ein Kontextmenü, in dem Sie mit Entfernen das Miniprogramm wieder entfernen können.

5. In diesem Fall erhalten Sie — da es eine Hilfeseite `man` in mehreren Sektionen gibt — eine Auswahl, welche der Seiten aus diesen Sektionen Sie sehen wollen. Da die Hilfeseiten für Benutzerkommandos sich in Sektion 1 befinden, wählen Sie diesen Eintrag.

6. Starten Sie den Pager mit kpager. Drücken Sie in einem freien Bereich des Pagers die rechte Maustaste. Es erscheint ein Kontextmenü, in dem Sie mit KPager einrichten den Konfigurationsdialog des Pagers öffnen können. Wählen Sie hier Senkrecht und schließen Sie den Dialog. Ändern Sie nun über die Fensterbegrenzung die Größe der Kontrollleiste so, dass die Arbeitsplatzeigenschaften ordentlich aussieht (das Verhältnis von Höhe zu Breite passt) und verschieben Sie den Pager an den rechten Rand.

7. Rufen Sie im KDE Menü den Punkt Kontrollzentrum auf. Wählen Sie hier links die Kategorie Arbeitsfläche und dort Fenstereigenschaften. Hier können Sie mittels Aktivierung zwischen den Verhalten wählen. Bei Aktivierung unter Mauszeiger wird ein Fenster aktiv, sobald Sie die Maus dorthin bewegen, die Tastatureingabe erfolgt nun in diesem Fenster. Bei Aktivierung nach Klick müssen Sie ein Fenster anklicken, damit es aktiv wird (das ist etwa auch das Verhalten unter Windows).

8. Mit Befehl ausführen werden standardmäßig grafische Applikationen gestartet, die sich selbst um ihre Darstellung kümmern. `less` muss jedoch in einem Terminal gestartet werden, sonst sehen Sie keine Ausgabe. Klicken Sie dazu auf Einstellungen, wählen hier In Terminal ausführen und führen den Befehl aus. Er wird nun in einem Terminal gestartet.

19.15 Abschnitt 17 (Seite 117)

1. Sie können dazu unter Linux den konqueror verwenden. Geben Sie als URL ein:

   `ftp://zid-luxinst.uibk.ac.at`

Wenn Sie dies über das Kommandozeilenprogramm `ftp` machen, geben Sie ein:

`ftp ftp://zid-luxinst.uibk.ac.at`

Geben Sie als Benutzerkennung `anonymous` und als Passwort Ihre Mail–Adresse an.

2. Wird `xhost` ohne Parameter aufgerufen, so wird eine Liste der Rechner ausgegeben, die berechtigt sind, auf Ihrer Oberfläche ein Fenster zu öffnen (diese haben Sie zuvor mit `xhost +[Rechner]` hinzugefügt).

19.16 Abschnitt 18 (Seite 121)

1. Öffnen Sie den KDE Menü–Editor, indem Sie im KDE Menü unter Präferenzen den Punkt Menü–Editor wählen. Stattdessen können Sie auch den Befehl `kmenuedit` ausführen.

   Wählen Sie hier links den Punkt Hilfe. Rechts sollte nun neben Aktuelles Tastenkürzel der Eintrag Keine stehen. Klicken Sie dies an, geben Sie ein neues Tastaturkürzel ein und bestätigen Sie es mit OK.

2. In diesem Fall wird die Eingabe abgebrochen, es erscheint eine neue Eingabezeile. Der Befehl wird zwar in der vorigen Zeile angezeigt, wurde jedoch nicht ausgeführt.

3. `rpm -qi bash` liefert eine Beschreibung des Paketes `bash`.

   `rpm -qa` listet alle RPM Pakete auf — das heißt sämtliche Software, die über das Fedora Core Linux Paketsystem installiert wurden.

20 Literatur

***Linus**: I guess it's wrong always to be worrying about tomorrow. Maybe we should think only about today.*
***Charlie Brown**: No, that's giving up. I'm still hoping that yesterday will get better*

Wenn Sie diesen Text (inklusive der Beispiele) durchgearbeitet haben, sollten Sie in der Arbeit schon ziemlich weit kommen. Der Text hat aber einige (beabsichtigte) Schwächen:

- Die Programme wurden nur sehr oberflächlich erwähnt. Nur bei wenigen wurden einige der unzähligen Parameter erwähnt.
- Die Mechanismen hinter den beschriebenen Phänomenen (etwa, wie und auf welcher Ebene eine Pipe funktioniert) wurden nicht angesprochen.
- Systemadministratoren und solche, die es werden wollen, haben hier wohl nichts (Diesbezügliches) erfahren.
- Der nächste logische Schritt wäre eine Einführung in die Programmierung. Einige der hier besprochenen Mechanismen werden dort erst wirklich sinnvoll.

Daher sollten Sie, wenn es Ihnen noch immer Spaß macht, einige weitere Bücher und Texte lesen, um diese nicht oder wenig behandelten Bereiche abzudecken.

Es folgt eine kurze Auswahl einiger Dokumente, die Ihnen hier weiterhelfen können.

- Im Rahmen des *Linux Documentation Projects* sind zahlreiche HOWTOs und Guides entstanden, beispielsweise *Securing and Optimizing Linux Red Hat Edition*; zu finden unter `http://linuxdoc.uibk.ac.at`.
- Unter `http://linuxdoc.uibk.ac.at` finden Sie auch einen Verweis auf Dokumentation, die für die LFU Innsbruck spezifisch ist.
- Unter `http://rute.sourceforge.net/` finden Sie einen umfangreichen Text, der neben dem hier Behandelten auch auf für Systemadminstratoren relevante Bereiche eingeht.
- Sehr empfohlen sei das Buch Der UNIX-Werkzeugkasten von Kernigan und Pike. Das Unix System wird hier mit seinen Komponenten sehr systematisch besprochen. Auch Programmierer kommen hier nicht zu kurz.
- Für den Heimanwender, der auch sein System betreuen muss, ist das Buch Linux von Michael Kofler sehr zu empfehlen. Inhaltlich bestehen gewisse Parallelen zu diesem Text, jedoch wird auch viel mehr auf die grafische Oberfläche und deren Applikationen eingegangen. Auch der „kleine“ Systemadministrator kommt hier nicht zu kurz. Mit über 1000 Seiten bietet das Buch mehr als genug Lesestoff. Mehr Information finden Sie unter `http://www.kofler.cc/linux.html`.

- Zum Thema Systemsicherheit und –Administration finden Sie einen ausgezeichneten Text unter `http://linuxdoc.uibk.ac.at/LDP/LDP/lasg/`.

Schließlich, obwohl nicht unbedingt zum Thema Dokumentation passend, sollen auch zwei Ressourcen erwähnt werden, die sich auf der Suche nach Software für Linux sehr bewährt haben:

- freshmeat.net (`http://freshmeat.net`) ist wohl *der* zentrale Anlaufplatz bei der Suche nach Linux–Applikationen. Der Applikations–Index beinhaltet eine große Menge an Produkten (vor allem nicht–kommerzielle). Die Hauptseite bietet eine Liste der täglichen Software–Neuerscheinungen.
- Unter `http://SAL.KachinaTech.COM/index.shtml` finden Sie einen Index wissenschaftlicher Software für Linux. Hier werden auch viele Produkte erwähnt, die nicht auf `freshmeat` zu finden sind.

Abbildungsverzeichnis

Index