Wulf Alex

Debian GNU/Linux in der Praxis

Konzepte, Werkzeuge, Anwendungen

unter Mitarbeit von Arne Alex und Björn Alex

Stand: 14. Februar 2007

Springer
Berlin Heidelberg New York
Hong Kong London
Milan Paris Tokyo

Korrekturen und Ergänzungen:
`http://www.alex-weingarten.de/debian/`

Wulf Alex
Rieslingweg 14
76356 Weingarten
alex-weingarten@t-online.de

Vorwort

Das Buch wendet sich an Benutzer von PCs unter Debian GNU/Linux, die auf einem fertig eingerichteten System mit der Arbeit beginnen möchten. Die Hobelbank stehe bereit, der Werkzeugschrank sei gut ausgestattet, es soll ans Schaffen gehen, wie man im Südwesten sagt. Dazu müssen sie wissen, welche Aufgaben mit welchen Anwendungen oder Werkzeugen zu erledigen und wie diese wirkungsvoll zu handhaben sind.

Der Gedanke zu diesem Buch reifte bei der Durchsicht des Manuskriptes zur zweiten Auflage des Debian-Buches von PETER H. GANTEN und WULF ALEX aus derselben Reihe des Springer-Verlages. Wir hatten den Eindruck, dass dort manche Themen fehlen oder zu kurz behandelt sind. Als Lösung bot sich eine Aufteilung der Themen an: Grundlagen, Einrichten und Betrieb bei GANTEN + ALEX, die Werkzeuge und Anwendungen hier. Anders ausgedrückt: Jenes Buch richtet sich an System- und Netzverwalter, dieses an Anwender. Für die zweite Auflage wurde der Text aktualisiert und um Kapitel über Programmierwerkzeuge erweitert.

Bedenkt man, dass es weit über tausend Linux/UNIX-Werkzeuge gibt und zu einzelnen Werkzeugen oder Internet-Diensten Bücher von 800 oder mehr Seiten, wird klar, dass umfassendere Arbeiten über Linux/UNIX immer Kompromisse aus Breite und Tiefe sind. Wir haben einerseits eine Auswahl aus der Werkzeugkiste treffen und andererseits auf manche Einzelheit verzichten müssen. Da wir sowohl Anwender wie Verwalter sind und darüber hinaus viele Anwender betreut haben, hegen wir die Hoffnung, einen praxistauglichen Kompromiss gefunden zu haben. *Alles über Debian GNU/Linux* ist kein Buch, sondern ein Bücherschrank.

Die Werkzeuge sind nach Aufgaben kapitelweise zusammengefasst. Innerhalb eines Kapitels ist die Vorgehensweise fast immer gleich: Wir erläutern die Aufgabe samt Grundbegriffen und Konzepten – das ist Wissen auf Dauer – und im Anschluss einige gebräuchliche Anwendungsprogramme. Je tiefer wir in die Einzelheiten dringen, desto kurzlebiger sind die Fakten, obwohl sich in den bald vierzig Lebensjahren von UNIX und fünfzehn von GNU/Linux viele Dinge eingependelt haben. Deshalb zeigen wir, wie man sich bei Bedarf weiter informiert, und verweisen vielerorts auf unsere Informationsquellen, allen voran das Web.

Bei über zehntausend Debian-Paketen und noch einmal der gleichen Menge an Software außerhalb von Debian, aber unter der GNU GPL oder ähnlichen Bedingungen stehen für viele Aufgaben mehrere Werkzeuge zur Auswahl. Der geneigte Leser wird gebeten, sich auch selbst in den Paketlisten auf `http://www.debian.org/distrib/packages` umzusehen und verschiedene Werkzeuge auszuprobieren. Für die Mitteilung der dabei gewonnenen Erfahrungen danken wir im Voraus. Die Vielfalt ist Segen und Fluch zugleich: Wenn Sie in einer einheitlichen Rechnerwelt bleiben, haben Sie es einfacher, alles passt zusammen. Aber Sie haben keine Chance, diese Welt zu verlassen. Die Linux/UNIX-Welt dagegen war, ist und bleibt offen, sowohl bezüglich der Hardware wie der Software. Trotz dieser Fülle an freier Software sind wir vereinzelt auf nicht-freie Produkte eingegangen. Das Bild von den Möglichkeiten unter Debian GNU/Linux wird vollständiger.

Das Buch behandelt *nicht* die Einrichtung und den Betrieb eines Rechners unter Debian GNU/Linux. Das sind Pflichten des Systemverwalters (Administrators, Supervisors, Superusers, Administrateur-système, root); sie sind in GANTEN + ALEX erläutert. Das Verständnis elementarer Begriffe aus der Linux/UNIX-Welt wie Benutzer, Gruppe, Datei, Verzeichnis, Pfad, An- und Abmelden, Speicher wird vorausgesetzt.

Der Text beruht auf Erfahrungen aus fünf Jahrzehnten Umgang mit elektronischen Rechenanlagen und aus Kursen über HP-UX und GNU/Linux für Auszubildende und Studierende. Wir haben auch fremde Hilfe beansprucht und danken Angehörigen der Universität Karlsruhe für Unterlagen, Anregungen und Diskussionen. Darüber hinaus haben wir fleißig das Internet angezapft und viele dort umlaufende Guides, Primers, Tutorials, HOWTOs und Sammlungen von Frequently Asked Questions ausgewertet. Ein besonderer Dank gebührt der deutschen Wikipedia, der Suchmaschine MetaGer, der Klosterbrauerei Andechs, einigen Winzergenossenschaften zwischen Wachenheim und Würzburg, einer ungenannt bleiben wollenden Bremer Kaffeerösterei sowie dem Steinachstübl und der Steinseehütte, die – jeder auf seine Weise – zum Gelingen des Werkes beigetragen haben.

Das Manuskript wurde mit den Texteditoren `vim` und Emacs auf einem PC der Marke *Weingartener Katzenberg Auslese* unter Debian GNU/Linux 3.1 (*sarge*) geschrieben und mit LaTeX 2e formatiert. Die meisten Abbildungen wurden mittels `xwd`, `xfig` und GIMP erstellt.

Dem Springer-Verlag in Heidelberg danken wir für die angenehme Zusammenarbeit, insbesondere Herrn HERMANN ENGESSER, der sich für beide Debian-Bücher eingesetzt und zu ihrem Erfolg beigetragen hat.

So eine Arbeit wird eigentlich nie fertig, man muss sie für fertig erklären, wenn man nach Zeit und Umständen das Möglichste getan hat, um es mit JOHANN WOLFGANG VON GOETHE zu sagen (*Italienische Reise*, Caserta, den 16. März 1787, als er an der *Iphigenie* schrieb).

Weingarten (Baden), den 16. März 2006

Wulf Alex Arne Alex Björn Alex

1

Textverarbeitung

Hier wird (fast) alles erklärt, was man braucht, um einen ordentlichen Text zu erstellen. Anspruchsvolle Textverarbeitung ist eine Stärke von Linux/UNIX und Mitarbeitern.

1.1 Grundbegriffe

1.1.1 Zeichensätze – von ASCII bis Unicode

Wenn es um Texte geht, muss man sich zuerst mit dem Problem der Zeichensätze herumschlagen. Das hat nichts mit Linux/UNIX zu tun, sondern gilt für alle Betriebssysteme.

Der Rechner kennt nur Bits und Bytes (Oktetts, 8 Bits). Die Bedeutung erhalten die Bits durch die Programme. Ob eine Bitfolge in der Menschenwelt eine Zahl, ein Wort oder einen Schnörkel darstellt, entscheidet die Software. Um mit Texten zu arbeiten, muss ein Zeichensatz (E: character set, F: jeu de caractères) vereinbart werden. Dieser besteht aus einer zunächst ungeordneten Menge von Zeichen, auch Répertoire oder Zeichenvorrat genannt, die nur besagt, welche Zeichen bekannt sind. Zu dem Zeichenvorrat der europäischen Sprachen gehören:

- Kleine und große Buchstaben, auch mit Akzenten (Diakritika) usw.,
- Ziffern,
- Satzzeichen,
- Symbole: aus der Mathematik, Euro-Symbol, Klammeraffe (E: commercial at, F: arobace), et-Zeichen,
- Leerzeichen (Zwischenraum, Space), Tabulator (so genannte Whitespaces),
- Steuerzeichen wie Zeilenwechsel (Line Feed), Seitenwechsel (Form Feed), Backspace.

In einem zweiten Schritt wird die Menge geordnet, jedem Zeichen wird eine Position (E: code position, code point, Unicode scalar value) zugewiesen. Nahe liegend ist eine mit null beginnende Nummerierung. Die geordnete Menge ist der Zeichensatz.

In der Regel bekommt jedes Zeichen auch noch einen Namen, hier einige Zeichen aus dem Zeichensatz Latin-1:

```
Nr. 008    bs   Backspace (Rückschritt)
Nr. 055    7    Digit seven (Ziffer sieben)
Nr. 104    h    Latin small letter h (lateinischer
                                    Kleinbuchstabe h)
Nr. 123    {    Left curly bracket (linke geschweifte
                                          Klammer)
Nr. 220    Ü    Latin capital letter u with diaeresis
              (lateinischer Großbuchstabe u-Umlaut)
```

Bekannt ist das Internationale Alphabet Nr. 5 (IA5), auf dem ASCII aufsetzt. Wir wissen aber noch nicht, wie die Zeichen im Rechner und auf dem Bildschirm oder auf Papier dargestellt werden. Auch die Aussprache der Zeichen ist noch zu vereinbaren und nicht einheitlich.

Die Zeichencodierung (E: character encoding) legt fest, wie eine Folge von Zeichen in eine Folge von Bits oder Bytes umzuwandeln ist, und zurück. Der Zeichensatz ist von seiner Codierung zu unterscheiden. Im einfachsten Fall wird die vorzeichenlose, ganzzahlige Positionsnummer als Code für ein Zeichen genommen. Dann stellt sich die Frage, wie solche Zahlen im Rechner dargestellt werden. Der häufigste Fall ist eine Darstellung als duale Zahl mit sieben oder acht Bits. Wir kommen so zur Codetafel. In der linken Spalte stehen die Zeichen, in der rechten die Bits oder umgekehrt. Mit diesem Schritt ist die rechnerinterne Darstellung der Zeichen festgelegt, aber noch nicht deren Aussehen auf Schirm oder Papier, die Glyphen.

Zu Zeiten, als Bits noch knapp und teuer waren, haben die Nordamerikaner eine Codetafel aufgestellt, in der die ihnen bekannten Buchstaben, Ziffern und Satzzeichen zuzüglich einiger Steueranweisungen wie Zeilen- und Seitenvorschub mit sieben Bits dargestellt werden. Das war Sparsamkeit am falschen Platz, aber schon ein Fortschritt gegenüber dem Fernschreib-Alphabet, das nur fünf Bits (oder Löcher auf dem Lochstreifen) vorsah. Mit sieben Bits unterscheide ich $2^7 = 128$ Zeichen, nummeriert von 0 bis 127. Diese Codetafel ist unter dem Namen *American Standard Code for Information Interchange* (ASCII) weit verbreitet und seit 1963 in ISO 646 genormt, gleichzeitig auch als CCITT-Empfehlung V.3. Um Missverständnisse auszuschließen, heißt sie ausführlich 7-bit-US-ASCII. Jeder Rechner kennt sie.

Die ersten 32 Zeichen der ASCII-Tafel dienen der Steuerung der Ausgabegeräte, es sind unsichtbare Zeichen. Ein Beispiel für Steuerzeichen ist das ASCII-Zeichen Nr. 12, Form Feed, das einen Drucker zum Einziehen eines Blattes Papier veranlasst. Auf der Tastatur werden sie entweder in Form ihrer Nummer oder mit gleichzeitig gedrückter <ctrl>-Taste erzeugt. Die Ziffern 0 bis 9 tragen die Nummern 48 bis 57, die Großbuchstaben die Nummern 65 bis 90. Die Kleinbuchstaben haben um 32 höhere Nummern als die zugehörigen Großbuchstaben, was das Umrechnen erleichtert. Der Rest sind Satzzeichen. Im Anhang ist die ASCII-Tafel wiedergegeben. Das Kommando:

```
joe@debian:~$ man ascii
```

bringt sie auf den Schirm.

Textausgabegeräte wie Bildschirme oder Drucker erhalten vom Rechner die ASCII-Nummer eines Zeichens und setzen diese mit Hilfe einer eingebauten Software in das entsprechende Zeichen um. So wird beispielsweise die ASCII-Nr. 100 in den Buchstaben d umgesetzt. Die Ausgabe der Zahl 100 als Zeichenkette erfordert das Abschicken der ASCII-Nr. 49, 48, 48.

Die US-ASCII-Tafel enthält nicht die deutschen Umlaute und andere europäische Absonderlichkeiten. Es gibt einen Ausweg aus dieser Klemme, leider sogar mehrere. Bleibt man bei sieben Bits, muss man einige nicht unbedingt benötigte US-ASCII-Zeichen durch nationale Sonderzeichen ersetzen. Für deutsche Zeichen ist eine Ersetzung gemäß DIN 66003, siehe Anhang **??** *German ASCII* auf Seite **??**, üblich. German ASCII ist als Variante Nr. 21 in ISO 646 registriert. Für Frankreich oder Schweden lautet die Ersetzung anders, siehe die deutsche Wikipedia unter dem Begriff *ISO 646*. Im Ausgabegerät, das die Umsetzung der ASCII-Nummern in Zeichen vornimmt, liegt eine solche Ersatztafel. Deshalb kann ein entsprechend ausgestatteter Bildschirm oder Drucker dieselbe Textdatei einmal mit amerikanischen ASCII-Zeichen ausgeben, ein andermal mit deutschen ASCII-Zeichen. Im ersten Fall erscheinen geschweifte und eckige Klammern, im zweiten stattdessen Umlaute. Werden bei Ein- und Ausgabe unterschiedliche Codetafeln verwendet, gibt es Zeichensalat. Andersherum gesagt: Wenn ich einen Text ausgebe, muss ich die Codetafel der Eingabe kennen und verwenden.

Spendiert man ein Bit mehr, so lassen sich $2^8 = 256$ Zeichen darstellen. Das ist der bessere Weg. Hewlett-Packard hat die nationalen Sonderzeichen den Nummern 128 bis 255 zugeordnet und so den Zeichensatz ROMAN8 geschaffen, dessen untere Hälfte mit dem US-ASCII-Zeichensatz identisch ist. Das hat den Vorzug, dass reine US-ASCII-Texte genau so verarbeitet werden wie ROMAN8-Texte. Diese Codetafel hat sich nicht allgemein durchgesetzt.

Die Firma IBM hat schon früh bei größeren Anlagen (mainframes) den *Extended Binary Coded Decimal Interchange Code* (EBCDIC) mit acht Bits verwendet, der nirgends mit ASCII übereinstimmt. Hätte sich diese Codetafel statt ASCII durchgesetzt, wäre uns Europäern einige Mühe erspart geblieben. Eine globale Lösung wäre es aber auch nicht gewesen.

Bei ihren PCs schließlich wollte IBM außer nationalen Sonderzeichen auch einige Halbgrafikzeichen wie Mondgesichter, Herzchen, Noten und Linien unterbringen und schuf einen weiteren Zeichensatz namens IBM-PC, der in seinem Kern mit ASCII übereinstimmt, ansonsten aber weder mit EBCDIC noch mit Latin-*.

Die internationale Normen-Organisation ISO hat mehrere 8-bit-Zeichensätze festgelegt, von denen einer unter dem Namen *Latin-1* nach ISO-8859-1 Verbreitung gewonnen hat, vor allem in weltweiten Netzdiensten. Seine untere Hälfte ist mit US-ASCII identisch, die obere enthält die Sonderzeichen west- und mitteleuropäischer Sprachen. Der Aufruf:

```
joe@debian:~$ man iso_8859_1
```

zeigt die Tafel. Polnische und tschechische Sonderzeichen sind in *Latin-2* nach ISO 8859-2 enthalten, siehe Anhang *Latin-1* und *Latin-2* ab Seite **??**. Die Zeichensät-

ze Latin-1 bis 4 sind ebenfalls im Standard ECMA-94 beschrieben (ECMA = European Computer Manufacturers Association). Kyrillische Zeichen sind in ISO 8859-5, griechische in ISO 8859-7 festgelegt (logischerweise nicht als Latin-* bezeichnet; Latin-5 ist Türkisch nach ISO 8859-9). ISO 8859-15 oder Latin-9 entspricht Latin-1 zuzüglich des Eurosymbols und weiterer Kleinigkeiten.

Die Latin-Zeichensätze enthalten außer dem gewohnten Zwischenraum- oder Leerzeichen (space) ein in Textverarbeitungen oft benötigtes Zeichen für einen Zwischenraum, bei dem kein Zeilenumbruch erfolgen darf (Latin-1 Nr. 160, no-break space). Im Satzsystem LaTeX wird hierfür die Tilde verwendet, in HTML die Umschreibung (entity) ` `. Dieses Zeichen kommt beispielsweise zwischen Zahl und Maßeinheit oder zwischen den Initialen eines Namens vor.

Auch wenn die Ein- und Ausgabegeräte 8-bit-Zeichensätze kennen, ist noch nicht sicher, dass man die Sonderzeichen benutzen darf. Die Programme müssen ebenfalls mitspielen. Der hergebrachte `vi`-Editor, die `curses`-Bibliothek für Bildschirmfunktionen und einige Email-Programme verarbeiten nur 7-bit-Zeichen. Erst jüngere Versionen von Linux/UNIX mit Native Language Support unterstützen 8-bit-Zeichensätze voll. Textverarbeitende Software, die 8-bit-Zeichensätze verträgt, wird als *8-bit-clean* bezeichnet. Bei Textübertragungen zwischen Rechnern (insbesondere Email) ist Misstrauen angebracht. Die Konsequenz heißt in kritischen Fällen Beschränkung auf 7-bit-US-ASCII, das funktioniert immer und überall. Das Problem verliert an Bedeutung.

Wenn wir das Abendland verlassen, stellen wir fest, dass es drei Gruppen von Schriften gibt, von Exoten wie den Quipus der Inkas abgesehen:

- alphabetische Schriften (E: alphabetic scripts): Lateinisch, Griechisch, Kyrillisch, Arabisch, Hebräisch, mit jeweils wenigen hundert Zeichen,
- Silbenschriften (E: syllabic scripts): Devanagari, Hiragana, Hangul, Thai, mit jeweils wenigen tausend Zeichen,
- Begriffsschriften (E: ideographic scripts): Chinesisch, Japanisch, Koreanisch, Vietnamesisch (CJKV), mit etwa 50000 Zeichen im Chinesischen.

Mehrere Einrichtungen bemühen sich, diese Vielfalt unter einen Hut zu bringen:

- die International Standardization Organization (ISO), die Internationale Normen-Organisation mit Sitz in Genf, an der auch das Deutsche Institut für Normung (DIN) beteiligt ist, `http://www.iso.org/` bzw. `http://www2.din.de/`,
- die International Electrotechnical Commission (IEC), an der die Deutsche Kommission Elektrotechnik Elektronik Informationstechnik im DIN + VDE beteiligt ist, `http://www.iec.ch/` bzw. `http://www.dke.de/`,
- das Unicode Consortium, ein Zusammenschluss führender Hersteller und Institutionen sowie von Einzelpersonen, `http://www.unicode.org/`,
- das World Wide Web Consortium (W3C), `http://www.w3.org/International/`,
- das Internet in Form einiger Requests For Comments.

Eine große, gemeinsame Richtung ist erkennbar, wenn auch im Einzelnen die unterschiedlichen Bezeichnungen und Zuständigkeiten verwirren.

Tab. 1.1: Darstellung des Zeichens A in verschiedenen Standards

Standard	dual	dez
ASCII (7 bit)	1000001	65
ISO 8859-1 (8 bit), UTF-8	01000001	65
UTF-16, UCS-2 (16 bit)	00000000 01000001	65
UTF-32, UCS-4 (32 bit)	00000000 00000000 00000000 01000001	65

ISO und IEC haben mit der Norm ISO/IEC 10646 einen Zeichensatz (Zeichen, Namen und Positionsnummer) festgelegt, der die Zeichen aller historischen, gegenwärtigen und zukünftigen Schriftsprachen der Erde berücksichtigt. Dieser Zeichensatz heißt *Universal Character Set* (UCS), ausführlich *Universal Multiple Octet Coded Character Set*. Unabhängig von der ISO arbeitet das Unicode Consortium seit 1991 an dem gleichen Ziel und hat einen Zeichensatz namens *Unicode* geschaffen.[1] Glücklicherweise haben beide Einrichtungen zusammengefunden. Im Jahr 2003 erschien Unicode in Version 4, die der Norm ISO/IEC 10646:2003 folgt. Der einheitliche Zeichensatz reicht für mehr als eine Million Zeichen. Die Standards von ISO/IEC und Unicode unterscheiden sich in ergänzenden Abschnitten, die teilweise verschiedene Themen aufgreifen. Geht es nur um den Zeichensatz, bedeuten Unicode, UCS und ISO/IEC 10646 dasselbe. In den untersten 256 Zeichen ist Unicode mit Latin-1 identisch, in den untersten 128 Zeichen folglich mit US-ASCII.

Die nächste Frage betrifft die Umsetzung der Zeichen oder ihrer Nummern in Bits und Bytes. Nahe liegend ist eine Darstellung durch 2 oder 4 Bytes, wobei das höchstwertige Byte zuerst übertragen wird (bigendian), wenn nicht anders bestimmt (UTF = Unicode Transformation Format):

- UCS-4 (UTF-32) verwendet eine feste Anzahl von 32 Bits (4 Bytes) und codiert damit bis zu $2^{32} = 4294967296$ Zeichen (Positionen). Der Bedarf an Speicherplatz ist maximal.
- UCS-2 (UTF-16) ist ein Kompromiss, der für die häufigeren Zeichen 2 Bytes (65 536 Positionen) und für die selteneren ein Paar von UCS-2-Codes (4 Bytes) verwendet. Von Java und MS Windows benutzt.

Die Linux/UNIX-Familie bekommt damit ein Problem. In UCS-2- oder UCS-4-codierten Zeichenketten treten häufig Nullbytes (00000000) auf, siehe Tabelle 1.1. Diese haben in der Programmiersprache C und damit auch in Linux/UNIX eine besondere Bedeutung: Sie beenden eine Zeichenkette. Außerdem kennen die herkömmlichen Linux/UNIX-Werkzeuge nur Zeichen mit der Länge von einem Byte. Wir brauchen eine andere Codierung des Zeichensatzes.

[1] The nice thing about standards is that there are so many of them to choose from. GRACE MURRAY HOPPER

Die Lösung des Linux/UNIX-Problems – ausgedacht von KENNETH THOMPSON mit Unterstützung durch ROB PIKE, beide UNIX-Urgestein – besteht in UTF-8, das 1 bis 6 Bytes verwendet und alle UCS-Zeichen codiert. Der Bedarf an Speicherplatz ist minimal, dafür muss mehr gerechnet werden. UTF-8 codiert nach folgender Vorschrift:

- Die US-ASCII-Zeichen Nr. 0 bis 127 (hex 00 bis 7F) bleiben, wie sie sind: ein Byte, höchstwertiges Bit null.
- Alle Zeichen >127 werden als Folgen mehrerer Bytes codiert, wobei in jedem Byte das höchstwertige Bit gesetzt ist. Somit tritt kein ASCII-Byte als Teil eines Zeichens >127 auf.
- Das erste Byte einer Bytefolge, die ein Zeichen >127 darstellt, beginnt mit so vielen Einsen, wie die Folge Bytes umfasst, gefolgt von einer Null, und liegt damit im Bereich von 192 bis 253 (hex C0 bis FD). Die restlichen Bits tragen zur Codierung der Positionsnummer bei.
- Die weiteren Bytes der Folge beginnen stets mit 10, die restlichen Bits tragen zur Codierung der Positionsnummer bei. Die Bytes liegen damit im Bereich von 128 bis 191 (hex 80 bis BF).
- Die Bytes 254 und 255 (hex FE und FF) werden nicht verwendet.
- Jedes Zeichen wird durch seine kürzest mögliche Bytefolge codiert, die Codierung ist umkehrbar eindeutig.

Die Folgen können bis zu sechs Bytes lang sein; die 65000 häufiger vorkommenden Zeichen erfordern jedoch nur bis zu drei Bytes. UFT-8 ist im Web verbreitet und gewinnt in der Linux/UNIX-Welt an Boden. Ein Beispiel: Unser Eszett (ß) hat im Latin-1-Zeichensatz die Nr. 223 und damit auch in UCS oder Unicode. Der Aufruf:

```
joe@debian:~$ unicode -d 223
```

liefert als Ergebnis:

```
U+00DF LATIN SMALL LETTER SHARP S
UTF-8: c3 9f   UTF-16BE: 00df   Decimal: &#223;
SS (SS)
Uppercase: U+00DF
Category: Ll (Letter, Lowercase)
Bidi: L (Left-to-Right)
```

was Folgendes bedeutet:

- Das Zeichen mit der hexadezimal ausgedrückten Position U+00DF (dezimal 223) in Unicode trägt den Namen *LATIN SMALL LETTER SHARP S*, wird in UTF-8 durch hexadezimal *c3 9f*, in bigendian UTF-16 durch hexadezimal *00df* dargestellt.
- Der zugehörige Großbuchstabe (den es nicht gibt) ist identisch mit dem Zeichen selbst.
- Das Zeichen fällt in die Kategorie der Kleinbuchstaben.
- Das Zeichen gehört zu den Schriften, die von links nach rechts geschrieben werden.

Probieren Sie obige Eingabe mit den Nr. 167, 188, 216, 555, 1648, 4711 und 12345 aus. Die UTF-8-Darstellung des Eszetts liest sich dual:

```
c3 9f = 11000011 10011111 = 110 00011  10 011111
```

Von links gelesen ergibt die Anzahl der Einsen bis zur ersten Null die Anzahl der zum Zeichen gehörenden Bytes, hier zwei. Vom ersten Byte sind dann die ab der dritten Eins stehenden Stellen zu nehmen, hier 11. Vom zweiten Byte sind die beiden führenden Bits zu streichen, der Rest ist an die 11 vom ersten Byte anzuhängen, sodass sich 11011111 = 223 dezimal ergibt. Bei solchen Rechnungen ist der Zahlenbasiskonverter `gbase` hilfreich.

Wir haben nun einen globalen Zeichensatz (UCS, Unicode) samt einer Codierung (UTF-8), die ins Konzept von Linux/UNIX passt. Jetzt muss noch die Software mitspielen. Werkzeuge wie `cat` lesen Bytes ein und geben Bytes aus, ohne auf ihre Bedeutung zu achten. Die brauchen nichts von UTF-8 zu verstehen. Aber überall dort, wo die Länge von Zeichenketten (Strings) gefragt ist, müssen die Funktionen und Programme jetzt aufpassen. Die alte Gleichheit Byte = Zeichen = Schritt auf dem Bildschirm gilt nicht mehr. Ein Zeichen kann aus mehreren Bytes bestehen und – bei den CJKV-Sprachen – doppelt breit sein. Bestimmte Zeichen wie selbständige Akzente veranlassen keinen Schritt auf dem Bildschirm, haben also die Breite null. Betroffen sind Programme wie `ls`, `wc` und die Texteditoren. Da der Benutzer keinen Einfluss auf seine Werkzeuge hat (und sich das vorliegende Buch nicht an Programmierer wendet), bleibt nur die Hoffnung, dass die Textwerkzeuge unter Debian GNU/Linux zunehmend UTF-8 beherrschen. Auf der Webseite zum Debian-Paket *vim* aus *sarge* ist jedenfalls von Unicode die Rede; der aktuelle Emacs murmelt etwas von *some Unicode support*. Es gibt auch Editoren, die speziell für UTF-8 und bidirektionale Schreibweise entwickelt worden sind, siehe Abschnitt 1.8 *Editoren für fremde Schriften* auf Seite 50.

UTF-7 nach RFC 1642 ist ein Zugeständnis an die Email-Standards, die verlangen, dass Email nur mit 7 Bits codierte Zeichen enthalten darf. Die US-ASCII-Zeichen bleiben unverändert, Zeichen auf höheren Positionen werden durch ein Plus- und ein Minuszeichen eingerahmt nach UTF-16 und Base64 codiert. Stirbt aus.

Zusammenfassend noch einmal die drei in der Linux/UNIX-Welt gebräuchlichsten Zeichensätze und Codierungen:

- Alt, bewährt, verbreitet, aber beschränkt in seinen Möglichkeiten: 7-bit-ASCII nach ISO 646,
- verbreitet, aber nur für das Abendland ausreichend: ISO/IEC 8859 mit 8 Bits, daraus für West- und Mitteleuropa ISO-8859-1, auch Latin-1 oder IBM Codepage 819 genannt.
- UCS bzw. Unicode mit der Codierung UTF-8, nach ISO/IEC 10646 und Unicode Version 4, ein bis sechs Bytes, zukunftssicher und zunehmend unterstützt.

Wer sich eingehender mit Unicode und UTF-8 beschäftigen möchte, findet einen guten Einstieg bei:

- `http://www.unicode.org/Public/` sowie `http://www.unicode.org/faq/`,

- RFC 3629 *UTF-8, a transformation format of ISO 10646*.
- BRUNO HAIBLE, *The Unicode HOWTO*, `http://www.tldp.org/HOWTO/Unicode-HOWTO.html`,
- MARKUS KUHN, *UTF-8 and Unicode FAQ for Unix/Linux*, `http://www.cl.ac.uk/~mgk25/unicode.html`,
- ROMAN CZYBORRA, *Unicode Transformation Formats: UTF-8 & Co.*, `http://czyborra.com/utf/`,
- ALAN WOOD, *Unicode and Multilingual Support in HTML, Fonts, Web Browsers and Other Applications*, `http://www.alanwood.net/unicode/`,

Die Standards gibt es bei den jeweiligen Organisationen, in der Regel nur gegen Entgelt, da sie sich zum Teil aus dem Verkauf finanzieren. In den Bibliotheken technischer Hochschulen kann man die Standards einsehen.

1.1.2 Fonts

Der Zeichensatz sagt, welche Zeichen bekannt sind, nicht wie sie aussehen. Ein Font (E: font, F: police) legt das Aussehen der Zeichen, die Glyphen, fest. Ursprünglich war ein Font der Inhalt eines Setzkastens. Verwirrend ist, dass einige einfache Font-Formate die Gestaltinformationen verschiedenen Zahlen (Positionen) zuordnen und so der Font darüber befindet, welches Zeichen für welche Zahl ausgegeben wird. Die einzige saubere Lösung ist die Trennung von Zeichensatz, Codierung und Font. Die Vielzahl der Fonts hat technische und künstlerische Gründe.

Bei einem anspruchsvollen Font werden nicht nur die einzelnen Zeichen dargestellt, sondern auch bestimmte Zeichenpaare oder -tripel (Ligaturen). JOHANNES GUTENBERG verwendete 190 Typen, und da war noch kein Klammeraffe und kein Eurosymbol dabei. Der Buchstabe *f* ist besonders kritisch. Erkennen Sie den Unterschied zwischen *hoffen* und *hoffähig* oder *Kaufläche* und *Kaufleute*? Die jeweils ersten Glieder der Paare werden mit Ligatur geschrieben, die zweiten ohne. LaTeX-Fonts kennen f-Ligaturen, wie man sieht, und zwar die Zeichengruppen *ff, fi, fl, ffi, ffl*.

Unter einer Schrift, Schriftfamilie oder Schriftart (E: typeface) wie Times Roman, New Century Schoolbook, Garamond, Bodoni, Helvetica, Futura, Univers, Schwabacher, Courier, OCR (optical character recognition) oder Schreibschriften versteht man einen stilistisch einheitlichen Satz von Fonts in verschiedenen Ausführungen. Die Schriftart soll zum Charakter und Zweck des Schriftstücks und zur Wiedergabetechnik passen. Die Times Roman ist beispielsweise ziemlich kompakt sowie leicht und schnell zu lesen (sie stammt aus dem Zeitungsdruck), während die New Century Schoolbook 10 % mehr Platz benötigt, dafür aber deutlicher lesbar ist, was für Leseanfänger und Sehschwache eine Rolle spielt. Die klassizistische Bodoni wirkt gehoben und ist die Lieblingsschrift von IBM. In einer Tageszeitung wäre sie fehl am Platz. Serifenlose Schriften wie die Helvetica eignen sich für Plakate, Overhead-Folien, Präsentationen, Beschriftungen von Geräten und kurze Texte. Die Computerzeitschrift c't ist mit der Nummer 6/2003 auf einen serifenlosen Font umgestiegen, um sich einen modernen Anstrich zu geben. Die Schriften liegen in

verschiedenen Schriftschnitten (E: treatment) vor: mager, fett, breit, schmal, kursiv, dazu in verschiedenen Größen oder Schriftgraden (E: point size). Die Schriftweite, der Zeichenabstand (E: pitch), ist entweder fest wie bei einfachen Schreibmaschinen, beispielsweise 10 oder 12 Zeichen pro Zoll, oder von der Zeichenbreite abhängig wie bei den Proportionalschriften. Diese sind besser lesbar und sparen Platz, erfordern aber in Tabellen Aufwand. Ein vollständiger Satz von Buchstaben, Ziffern, Satz- und Sonderzeichen einer Schrift, eines Schnittes, eines Grades und gegebenenfalls einer Schriftweite wird heute *Font* genannt.

Im wesentlichen gibt es zwei Kategorien von Font-Formaten: Bitmap-Fonts (Raster-Fonts) und Vektor-Fonts. Bitmap-Fonts speichern die Gestalt eines Zeichens in einer Punktematrix. Der Vorteil besteht in der einfachen und schnellen Verarbeitung. Auf vielen Systemen existieren mehrere Bitmap-Fonts derselben Schrift, optimiert für die am häufigsten benötigten Schriftgrößen. Nachteilig ist die unbefriedigende Skalierbarkeit (Vergrößerung oder Verkleinerung). Das Problem ist das gleiche wie bei Grafiken.

Bessere Systeme verwenden daher Vektor-Fonts, die die Gestalt der Zeichen durch eine mathematische Beschreibung ihrer Umrisse festhalten. Vektor-Fonts lassen sich daher problemlos skalieren. Bei starken Maßstabsänderungen muss jedoch auch die Gestalt etwas verändert werden. Gute Vektor-Fonts speichern deshalb zusätzliche, beim Skalieren zu beachtende Informationen (hints) zu jedem Zeichen.

Die beiden wichtigsten Vektor-Font-Formate sind True Type (TT), hauptsächlich auf Macintoshs und unter MS Windows zu finden, und das von Adobe stammende PostScript-Type-1-Format (PS1), das auch von X11 dargestellt werden kann und daher unter Linux/UNIX verbreitet ist. True Type kam um 1990 heraus und war die Antwort von Apple und Microsoft auf Adobe. Im Jahr 1996 rauften sich Adobe und Microsoft zusammen und schufen das Open Type Format als eine einheitliche Verpackung von PostScript- und Truetype-Fonts.

X11 hat ursprünglich nichts mit der Druckausgabe zu tun. Die Tatsache, dass ein Font unter X11 verfügbar ist, bedeutet noch nicht, dass er auch gedruckt werden kann. Einen gemeinsamen Nenner von X11 und der Druckerwelt stellt das Type-1-Format dar: Fonts dieses Formates können sowohl von X11 auf dem Bildschirm dargestellt als auch als Softfonts (in Dateien gespeicherte Fonts) in PostScript-Drucker geladen werden. Für den Privatanwender, der sich keinen PostScript-Drucker leistet, bietet sich der Weg an, den freien PostScript-Interpreter *Ghostscript* als Druckerfilter zu verwenden. Er wandelt PostScript-Daten in verschiedene Druckersteuersprachen wie PCL um.

Da die Papierformate länglich sind, spielt die Orientierung (E: orientation) eine Rolle. Das Hochformat wird englisch mit *portrait*, das Querformat mit *landscape* bezeichnet[2]. Ferner trägt der Zeilenabstand oder Vorschub (E: line spacing) wesentlich zur Lesbarkeit bei. Weitere Gesichtspunkte zur Schrift und zur Gestaltung von Schriftstücken findet man in der im Anhang angegebenen Literatur und im Netz, zum Beispiel in der FAQ der Newsgruppe `comp.fonts`, auf Papier 260 Seiten,

[2]Woraus man schließt, dass Engländer ein Flachland bewohnende Langschädler sind, während alpine Querköpfe die Bezeichnungen vermutlich andersherum gewählt hätten.

zusammengestellt von NORMAN WALSH (`http://www.nwalsh.com/comp.fonts/FAQ/`). Trinken Sie einen auf sein Wohl und denken Sie darüber nach, wie viel freiwillige und unentgeltliche Arbeit in den FAQs steckt. Es gibt auch ein *Font HOWTO*, genauer betitelt *Optimal Use of Fonts on Linux*, aktuell von 2005.

Jetzt wäre es an der Zeit, über die akustische Darstellung von Texten zu reden, was bei der Sprachsynthese eine Rolle spielt – siehe Abschnitt **??** *Sprachsynthese, Screenreader* auf Seite **??** – aber das verschieben wir aus naheliegenden Gründen in die Zukunft.

1.1.3 Dateiformate (txt, pdf, ps, rtf)

Nicht genug mit Zeichen, Codes und Fonts, ein Textdokument lässt sich auch noch auf mancherlei Weise in einer Datei abspeichern. Der Benutzer bemerkt das, wenn ihm ein Text in einem ungewohnten Format zuflattert. Gewöhnlicher Text in ASCII- oder Latin-Codierung ohne weitere Verzierungen wird oft in Dateien mit der Kennung `txt` gespeichert, was in der Linux/UNIX-Welt jedoch nicht vorgeschrieben ist.

Textdateien aus anderen Welten (DOS, Macintosh)

In Linux/UNIX-Textdateien wird der Zeilenwechsel (E: line break, F: saut de ligne) durch ein newline-Zeichen markiert, hinter dem das ASCII-Zeichen Nr. 10 (LF, Line feed) steckt, das auch durch die Tastenkombination `<ctrl>-<j>` eingegeben wird. In DOS-Textdateien wird ein Zeilenwechsel durch das Zeichenpaar Carriage return – Line feed (CR LF, ASCII Nr. 13 und 10, `<ctrl>-<m>` und `<ctrl>-<j>`) markiert, das Dateiende durch das ASCII-Zeichen Nr. 26 (`<ctrl>-<z>`). Auf Macs ist die dritte Möglichkeit verwirklicht, das Zeichen Carriage return (CR, ASCII Nr. 13, `<cr>`) allein veranlasst den Sprung an den Anfang der nächsten Zeile.

Auf einer Linux/UNIX-Maschine lassen sich die störenden Carriage returns (oktal 15) der DOS-Texte leicht durch folgenden Aufruf entfernen:

```
joe@debian:~$ tr -d '\015' < text.dos > text.unix
```

Die Texteditoren `vi`, `emacs` oder `sed` können das auch, ebenso ein einfaches C-Programm.

Wenn Ihr Text auf einem Bildschirm oder Drucker treppenförmig dargestellt wird – nach rechts fallend – erwartet das Gerät einen Text nach Art von DOS mit CR und LF, der Text enthält jedoch nach Art von Linux/UNIX nur LF als Zeilenende. In einigen Fällen lässt sich das Gerät entsprechend konfigurieren, auf jeden Fall kann man den Text entsprechend ergänzen. Wenn umgekehrt auf dem Bildschirm kein Text zu sehen ist, erwartet das Ausgabeprogramm einen Linux/UNIX-Text ohne CR, die Textdatei stammt jedoch aus der DOS-Welt mit CR und LF. Jede Zeile wird geschrieben und gleich wieder durch den Rücksprung an den Zeilenanfang gelöscht. Viele Linux/UNIX-Pager (`more`, `less`) berücksichtigen das und geben das CR nicht weiter. Auf Druckern kann sich dieses Missverständnis durch Verdoppelung des Zeilenabstands äußern. Kein Problem, nur lästig.

Neben diesen, dem Benutzer auffallenden Unterschieden gibt es auch viele Möglichkeiten der rechnerinternen Darstellung der Zeichen. Damit in einem Netz verschiedene Rechner Texte austauschen können, hat man sich im Internet – im RFC 854 Telnet – darauf geeinigt, im Netz die Zeichen gemäß Network Virtual Terminal ASCII (NVT ASCII) darzustellen, wenn nichts anderes vereinbart ist, und die Umsetzung auf die lokale Darstellung dem Sender bzw. dem Empfänger zu überlassen. Dabei werden die 128 Zeichen des US-ASCII-Zeichensatzes durch ein Byte dargestellt, dessen oberstes Bit (most significant bit) null ist.

PostScript und Portable Document Format (PDF)

Außer dem eigentlichen Text enthält ein besseres Textdokument Formatierungsangaben und Grafik. Um ein derartig zusammengesetztes Dokument für die Ausgabe auf einem Drucker zu beschreiben, hat die Firma Adobe 1985 die Programmiersprache *PostScript* geschaffen, die auch als Seitenbeschreibungssprache bezeichnet wird. Die Entwicklung von Laserdruckern und PostScript hat sich gegenseitig beflügelt. Eine PostScript-Seite besteht aus:

- Linien,
- Text unterschiedlichen Aussehens,
- Bitmap-Grafiken.

Ihre Beschreibung in PostScript ist geräteunabhängig. Im Ausgabegerät (oder kurz davor) werden aus den PostScript-Daten gerätespezifische Daten errechnet. Deshalb kann dieselbe PostScript-Datei einmal einen Laserdrucker steuern, ein andermal einen Schneideplotter zum Ausschneiden eines Schriftzuges oder Firmenlogos aus einer Folie. Außer den Nutzdaten enthalten die Dateien nur Befehle, die aus dem US-ASCII-Zeichensatz zusammengesetzt sind. Es sind lesbare Textdateien ohne Zeilenstruktur. Der Anfang einer PostScript-Datei (Vorspann) sieht so aus:

```
%!PS-Adobe-2.0
%%Creator: dvips(k) 5.86e Copyright Radical Eye Software
%%Title: folien.dvi
%%Pages: 19
%%PageOrder: Ascend
%%BoundingBox: 0 0 596 842
%%EndComments
%DVIPSWebPage: (www.radicaleye.com)
%DVIPSCommandLine: /usr/bin/dvips -o folien.ps folien
%DVIPSParameters: dpi=600, compressed
%DVIPSSource:  TeX output 2004.07.24:1326
%%BeginProcSet: texc.pro
```

und die erste Seite so (eine Vortragsfolie):

```
%%Page: 0 1
```

```
0 0 bop 380 386 a Fu(P)-11 b(a)g(rtik)g(elme\031technik)
703 834 y(\(Kr)1264 845 y(\177)1262 834 y(umelkunde\))
356 1521 y Ft(W.)82 b(Alex,)h(Universit)2252 1530
y(\177)2254 1521 y(at)e(Ka)-7 b(rlsruhe)1266 2085
y(24.)83 b(Juli) e(2004)-97 5702 y Fs({)69 b(T)-17
b(yp)6 b(eset)67 b(b)-6 b(y)69 b(F)-6 b(oil)g Fr(T)1501
5747 y(E)1615 5702 y(X)68 b Fs({)i(W.)f(Alex:)f(Kr)3048
5709 y(\177)3046 5702 y(umelkunde)p
eop
```

Wer sich gut mit PostScript auskennt, könnte mit einem Texteditor die Datei ändern. Bruchstücke des Textes erkennt auch ein ungeübter Leser. Dennoch ist das unmittelbare Schreiben oder Lesen von PostScript-Dateien die Ausnahme. Erzeugt werden die Dateien mittels entsprechender Treiberprogramme aus gewöhnlichen Texten oder Grafiken. Zum Lesen braucht man besondere Leseprogramme, siehe Abschnitt 1.2.3 *Lesen von PostScript-Dateien* auf Seite 16..

Encapsulated PostScript (EPS, EPSF) ist eine Untermenge von PostScript zur Beschreibung von Grafiken, die in andere Dokumente eingebunden oder zwischen verschiedenen Anwendungen ausgetauscht werden sollen. Eine EPS-Datei enthält immer genau ein Objekt (Bild, Seite) und am Anfang eine Angabe zur Größe des Objekts (bounding box). Hier der Anfang der EPS-Datei zu Abbildung **??**, die von LaTeX eingebunden wird:

```
%!PS-Adobe-3.0 EPSF-3.0
%%Creator: GIMP PostScript file plugin V 1.12 by Peter
                                                Kirchgessner
%%Title: /server1/wulf/TeX/debian2/bilder/debian.eps
%%CreationDate: Sat Dec 25 22:34:32 2004
%%DocumentData: Clean7Bit
%%LanguageLevel: 2
%%Pages: 1
%%BoundingBox: 14 14 115 138
```

Ein Grafikprogramm wie GIMP kann eine beliebige Grafik im EPS-Format exportieren, die dann von einer Textanwendung importiert wird.

Das *Portable Document Format* (PDF) von Adobe ist eine Weiterentwicklung von PostScript. PostScript-Dateien lassen sich daher einfach mittels eines Programmes wie dem Adobe Distiller oder Ghostscript nach PDF umwandeln. Aus LaTeX-Quellen lässt sich mittels `pdflatex` unmittelbar PDF-Text erzeugen, bei schwierigen Texten (Formeln) der bessere Weg. PDF-Werkzeuge sind für viele Systeme verfügbar, PDF-Dokumente sind zwischen diesen Systemen austauschbar. Ein PDF-Dokument sieht in jedem Betrachter auf jedem System gleich aus, es ist layout-treu. Das ist oft gewünscht, aber nicht immer, beispielsweise dann nicht, wenn ein Dokument barrierefrei gestaltet oder an besondere Ausgabegeräte angepasst werden soll. Bei diesen Aufgaben kommt *tagged PDF* ins Spiel. Das PDF-Dokument enthält in

diesem Fall Strukturangaben in Form von Tags, die dem Ausgabeprogramm Anweisungen zur Strukturierung liefern.

Einen Weg zurück von der PDF-Datei zu den Quelldateien gibt es nur mit Einschränkungen, praktisch eigentlich nicht. Die Druckvorlage zu vorliegendem Buch ist eine PDF-Datei; daraus die Dateien und Verzeichnisse des Autors rekonstruieren zu wollen, ist ein aussichtsloses Unterfangen.

Im Gegensatz zu LaTeX- oder HTML-Quellen und in Übereinstimmung mit PostScript-Dateien wird in PDF-Dateien das Aussehen der Dokumente genau festgelegt. Der Autor bestimmt, was der Leser sieht. Die Erweiterung gegenüber PostScript besteht vor allem darin, dass PDF-Dokumente ebenso wie HTML-Seiten aktiv sein können; es gibt Hyperlinks, Eingabefelder und bewegte Grafiken. Vereinfacht gesagt ist PDF eine Mischung aus PostScript und HTML. Zusätzlich wird PDF deutlich komprimiert und ist daher – von einzelnen Zeichenketten zu Anfang abgesehen – nicht lesbar. Beim Drucken auf Papier geht die Aktivität natürlich verloren, je nach Drucker auch die Farbe. Zum Lesen siehe Abschnitt 1.2.4 *Lesen von PDF-Dateien* auf Seite 17.

Weitere Formate

Das von Microsoft geschaffene Rich Text Format (RTF) soll den Austausch von Texten und Grafiken zwischen verschiedenen Anwendungen und Geräten ermöglichen. Die Spezifikation findet man bei `http://msdn.microsoft.com/library/` und bei `http://latex2rtf.sourceforge.net/rtfspec.html`. Das Format hat innerhalb der Microsoft-Welt eine gewisse Bedeutung erlangt, wird aber nicht als Standardformat verwendet. Außerhalb der Microsoft-Welt hat es sich nicht gegen PostScript, PDF, HTML und LaTeX durchsetzen können. Bei Debian gibt es die Konverter `latex2rtf`, `rtf2latex` und `unrtf`. Letzteres Werkzeug wandelt RTF in einfachen Text, PostScript, LaTeX oder HTML um, siehe Abschnitt 1.5.8 *Werkzeuge für das Rich Text Format* auf Seite 45.

1.1.4 Allgemeines über Texteditoren

Ein Editor (E: editor, F: editeur) ist ein Programm, ein Werkzeug zum Erstellen oder Verändern von Texten, im weiteren Sinn auch von Musiknoten, mathematischen oder chemischen Formeln, Grafiken, Sound oder sonstigen binären Dateien. Das ist eine der ältesten und wichtigsten Arbeiten, die mit Hilfe eines Rechners erledigt werden. Notwendigerweise lässt sich ein Text mit einem Editor auch lesen.

Wird ein bereits vorhandener, abgespeicherter Text verändert, so arbeiten die Editoren – mit Ausnahme der Hex-Editoren – auf einer temporären Kopie des Textes in einem Pufferspeicher (E: buffer, F: mémoire tampon). Am Ende der Arbeit hat man die Wahl zwischen Zurückschreiben oder Verwerfen der veränderten Kopie. Manche Editoren sind so vorsichtig oder lassen sich so einstellen, dass sie auf jeden Fall das Original unter einem leicht veränderten Namen retten. Man hat dann eine

Chance, die Änderungen auch noch später zu bereuen. Es gibt sogar die Möglichkeit, mit Hilfe von Versionskontrollsystemen alle Änderungen seit Anbeginn zurückzunehmen, aber das muss besonders eingerichtet werden. Bei den Textwerkzeugen werden zwei Richtungen unterschieden:

- gewöhnliche Editoren, die den Text ohne besondere Formatierung auf dem Bildschirm darstellen, die also Inhalt und Form trennen und sich für letztere nicht interessieren,
- WYSIWYG-Textprozessoren (What You See Is What You Get[3], auf französisch *tel écran – tel écrit*), welche die Formatierung des Textes selbst sofort durchführen und auf dem Schirm den formatierten Text anzeigen.

WYSIWYG-Textprozessoren wie Abiword, Kword, MS Word oder Wordperfect erscheinen auf den ersten Blick als die intelligenteren und bequemer zu handhabenden Werkzeuge. Sie sind tatsächlich in Büros weiter verbreitet, oft im Rahmen von Office-Paketen, siehe Abschnitt **??** *Office-Pakete* auf Seite **??**. Sowie es jedoch um anspruchsvolle Texte (Bücher, schwierige Formeln, komplexes Layout, mehrere Schreiber in einem heterogenen Netz) geht, kommt man mit der Trennung von Inhalt und Form besser voran. Außerdem birgt WYSIWYG eine Versuchung in sich. Die reichen Mittel aus dem typografischen Kosmetikkoffer namens *Desktop Publishing* sind sparsam einzusetzen, unser Ziel heißt Lesbarkeit, nicht Barock.

Jeder Editor steht vor dem Problem, dass mit einer einzigen Tastatur sowohl der Text wie Editierkommandos eingegeben werden müssen. Die wenigen Editiertasten auf den üblichen PC-Tastaturen reichen bei weitem nicht aus, außerdem sind die Tastaturen verschieden. Auch bei der Lösung dieses Problems haben sich zwei Wege herauskristallisiert:

- Editoren vom `vi`-Typ unterscheiden zwei Eingabe-Modi. Im Kommando-Modus wird jede Tastatureingabe als Kommando verstanden, im Eingabe-Modus als Text.
- Editoren vom Emacs-Typ verwenden als Kommandos Tastenkombinationen wie `<ctrl>-<x>` gefolgt von `<ctrl>-<c>`, die als Text nicht vorkommen.

Den `vi` als Linux/UNIX-Standard-Editor sollte jeder Benutzer in seinen Grundzügen kennen, siehe Abschnitt 1.3.1 *Schnellstart* auf Seite 21. Die Textdateien lassen sich zwischen den Editoren beliebig austauschen, die Werkzeuge verwenden kein besonderes Dateiformat. Man kann einen Text mit dem `vi` beginnen, mit dem `joe` verbessern und mit dem Emacs fertig stellen. Bei WYSIWYG-Textprozessoren ist das nicht uneingeschränkt der Fall, eher schon ein Glücksfall, wenn es funktioniert. Für die tägliche Arbeit bleibe man bei seinem Lieblingseditor. Man kann sich an jeden Editor gewöhnen, nur nicht jede Woche an einen neuen.

Die vorstehenden Zeilen waren vielleicht etwas viel zu einem so einfachen Thema wie der Erzeugung und Wiedergabe von Texten, aber es gibt nun einmal auf der Welt mehr als die sechsundzwanzig Zeichen des lateinischen Alphabets und mehr als

[3] Real Programmers consider WYSIWYG to be just as bad a concept in text editors as it is in women.

ein Textprogramm. Auf längere Sicht kommt man nicht darum herum, sich die Zusammenhänge klar zu machen. Dann versteht man, warum in der Textverarbeitung so viel schief geht, von der künstlerischen Seite ganz abgesehen.

1.2 Werkzeuge zum Lesen und Umwandeln

1.2.1 Einfache Werkzeuge (cat, more, pg, less, lv, bidiv, view)

Das alte Linux/UNIX-Werkzeug `cat` ist das Muster eines Filters: es liest beliebige Dateien von `stdin` und schreibt sie nach `stdout`. Werden dem Kommando Dateinamen als Argument mitgegeben, liest es diese und gibt sie der Reihe nach aus, daher der Name (cat = concatenate = verketten). Es versteht ein paar einfache Optionen. Als Lesewerkzeug hat es bei längeren Texten den Nachteil, dass die Zeilen über den Bildschirm rauschen und nur die letzten stehen bleiben. Sein Hauptanwendungsgebiet sind daher Shellskripte (Programme), nicht der Dialog.

Zum Lesen längerer Texte am Bildschirm wird in der Linux/UNIX-Welt seit altersher das Kommando `more` eingesetzt. Es kann als Filter in einer Pipe oder selbständig mit dem Namen der darzustellenden Datei als Argument aufgerufen werden:

```
joe@debian:~$ more lange_textdatei
```

Nach jeweils einem Bildschirmvoll hält `more` an, sodass man den Text in Ruhe lesen kann. Die Größe des Bildschirms entnimmt das Kommando der Umgebungsvariablen TERM. Mit Hilfe der Eingabetaste schaltet man eine Zeile weiter, mit Hilfe der Leertaste einen Bildschirm. `more` kennt einige einfache Optionen und ebenso einige Kommandos des Editors `vi`. Als Glied einer Pipe kann `more` nicht rückwärts blättern. Die Eingabe von `<q>` beendet das Programm. Eine ähnliche Funktionalität bietet das Werkzeug `pg` (pager).

Das Werkzeug `less` ist eine Weiterentwicklung von `more` im GNU-Projekt mit einer deutlich umfangreicheren Manualseite. Wichtigste Verbesserung ist die Fähigkeit, auch rückwärts blättern zu können. Außer den Kommandos von `more` versteht es eine Reihe von `vi`-Kommandos. Auf Linux-Maschinen ist es das Werkzeug der Wahl zum seitenweisen Lesen von Texten. Ähnlich sieht auch der Viewer `lv` aus, der mehrere Zeichensätze und Sprachen beherrscht, darunter Chinesisch, Japanisch und Koreanisch. Unter dem Namen `lgrep` aufgerufen, stellt er ein fremdsprachiges `grep` zum Suchen nach Zeichenketten bereit.

Geht es um die Darstellung von bidirektionalen Texten – also um aus Deutsch oder Englisch und Arabisch oder Hebräisch gemischten Texten – kommt ein gewöhnliches Terminal in Schwierigkeiten, da es nur eine Schreibrichtung beherrscht, üblicherweise von links nach rechts. Das Filter `bidiv` arbeitet ähnlich wie `cat` und setzt dabei eine logische Schreibrichtung bei Bedarf in eine visuelle um. Weitere Hinweise zu bidirektionalen Texten finden sich im Netz unter den Suchbegriffen *GNU FriBiDi* und *Unicode Bidirectional Algorithm*.

Das Kommando `view` ist der Editor `vi` im Nur-Lese-Modus (readonly). Alle Fähigkeiten des `vi` stehen zur Verfügung, nur Zurückschreiben kann (will) der

Editor nicht. Wer diesen Editor gewohnt ist, greift gern zu `view`, andere Benutzer bleiben besser bei `less`. Auch weitere Editoren kennen einen Nur-Lese-Modus.

1.2.2 Dumper (hexcat, od)

Leseprogramme wie `more` oder `less` fassen die ihnen als Argument übergebenen Dateien als Textdateien auf. Mitunter will man jedoch beliebige Dateien ohne jede Interpretation durch das Werkzeug anschauen oder kopieren, was als dumpen (E: to dump, F: clicher) bezeichnet wird. Das Werkzeug `hexcat` ist einfach zu gebrauchen, kennt keine Optionen und leistet das Wichtigste. Der Aufruf:

```
joe@debian:~$ hexcat bild.jpg | less
```

schreibt den Inhalt der Datei `bild.jpg` in hexadezimaler Form und am rechten Rand soweit möglich als ASCII-Text auf den Schirm. Etwas vielseitiger ist das Werkzeug `od` (octal dumper). Es versteht einige Optionen in verschiedenen Schreibweisen:

```
joe@debian:~$ od -c datei

joe@debian:~$ od -t c datei

joe@debian:~$ od -format=c
```

um die Ausgabe verständlicher zu gestalten, siehe Manual. In vorstehendem Beispiel wird verlangt, die Bytes soweit möglich als ASCII-Zeichen darzustellen wie hier:

```
00000 377 330 377 340  \0 020   J   F   I   F  \0 001
                                              001 001 001
00020 001   U  \0  \0 377 376  \0 027   C   r   e   a
                                                t   e   d
00040   w   i   t   h       T   h   e       G   I   M
                                              P 377 333
00060   C  \0  \b 006 006  \a 006 005  \b  \a  \a  \a
                                            \t  \t  \b
```

aus dem Anfang einer Bilddatei (`jpg`), die – wie man erkennt – unter anderem lesbare Zeichenketten enthält.

1.2.3 Lesen und Umwandeln von bzw. nach PostScript (gv, a2ps)

Zum Lesen von PostScript-Dateien braucht man besondere Leseprogramme, welche die PostScript-Anweisungen in Anweisungen für das jeweilige Ausgabegerät übersetzen. Das Gleiche gilt für `pdf`-Dateien. Das Portable Document Format (PDF), ebenfalls von Adobe, ist eine Weiterentwicklung von PostScript. Beide Formate wurden in Abschnitt 1.1.3 *PostScript und PDF* auf Seite 11 vorgestellt.

GPL Ghostscript ist ein Übersetzer (Interpreter) für PostScript und PDF. Seine Webseite liegt unter `http://www.cs.wisc.edu/~ghost/`. Ghostview ist eine auf X11 beruhende Benutzeroberfläche zu Ghostscript, ebenso `gv`. GSview (nicht bei Debian) leistet ähnliche Dienste, ist aber von Ghostview unabhängig. Um eine PostScript- oder PDF-Datei für einen einfachen Drucker (der nicht selbst PostScript interpretieren kann) aufzubereiten, braucht man Ghostscript als Filter. Zum Lesen einer PostScript-Datei am Bildschirm nimmt man Ghostview oder besser `gv`. Das Debian-Paket *gnome-gv* enthält ein GNOME-Frontend `ggv` für Ghostscript, das Debian-Paket *kghostview* eine Anpassung von Ghostview an den KDE-Desktop.

Das Debian-Paket *psutils* enthält rund ein Dutzend Werkzeuge zur Bearbeitung von PostScript-Dateien, darunter die zunächst überflüssig erscheinenden Konverter `ps2ps` und `eps2eps`. Diese benutzen `gs` bzw. Ghostscript, um die ihnen als Argument übergebenen Dateien zu optimieren und von kleineren syntaktischen Fehlern zu bereinigen. Konverter nach PostScript sind:

- `a2ps` ASCII-Text nach PostScript mit zahlreichen Optionen für die Druckerausgabe,
- `dvi2ps` LaTeX-DVI nach PostScript, daneben kommt mit LaTeX ein Werkzeug `dvips` mit,
- `e2ps` wie `a2ps`, mit Unterstützung für Japanisch,
- `enscript` ASCII-Text nach PostScript, HTML oder RTF,
- `html2ps` HTML (Webseiten) nach PostScript,
- `u2ps`, Paket *gnome-u2ps*, UTF-8-Text nach PostScript.

In der Gegenrichtung gibt es nicht so viele Werkzeuge:

- `ps2ascii`, Paket *ghostscript*, PostScript nach ASCII-Text,
- `ps2eps` PostScript nach Encapsulated PostScript,
- `pstotext` PostScript oder PDF nach Latin-1-Text, alternativ zu `ps2ascii`.

1.2.4 Lesen von PDF-Dateien (Adobe Reader, xpdf, gpdf, kpdf, pdftk, gv, Evince)

PDF-Dateien sind komprimiert und daher deutlich kleiner als PostScript-Dateien. Dazu kommen weitere Fähigkeiten, an die man bei der Entwicklung von PostScript noch nicht dachte. Da Adobe das Lesewerkzeug *Adobe Reader*, Aufruf `acroread`, kostenlos verteilt (`http://www.adobe.de/`) und es auf vielen Rechnern läuft, hat sich das PDF-Format zum Speichern und Austauschen von Dokumenten schnell durchgesetzt. Für Linux/UNIX steht der Adobe Reader 7.0 Basic Version in Englisch, Deutsch oder Französisch als Tarball oder rpm-Paket im Umfang von 40 MB bei Adobe im Netz. Die Einrichtung durch den Verwalter verlief problemlos wie schon bei früheren Versionen. Zum Schluss blieb noch das Anlegen eines symbolischen Links, damit das Werkzeug im üblichen Pfad gefunden wird:

```
debian:/usr/local/bin# ln -s
        /usr/local/Adobe/Acrobat7.0/bin/acroread areader
```

Beim Drucken aus dem Adobe Reader heraus sollte man die PostScript-Einstellung *Große Seiten auf Papiergröße verkleinern* oder *Seitenanpassung* im Normalfall ausschalten, das sie auch dann verkleinert, wenn kein Bedarf besteht (PDF-Ausgabe auf DIN A4, sollte eigentlich passen, wird trotzdem verkleinert, alle Rechnungen zur Größe des Satzspiegels etc. sind für die Katz.).

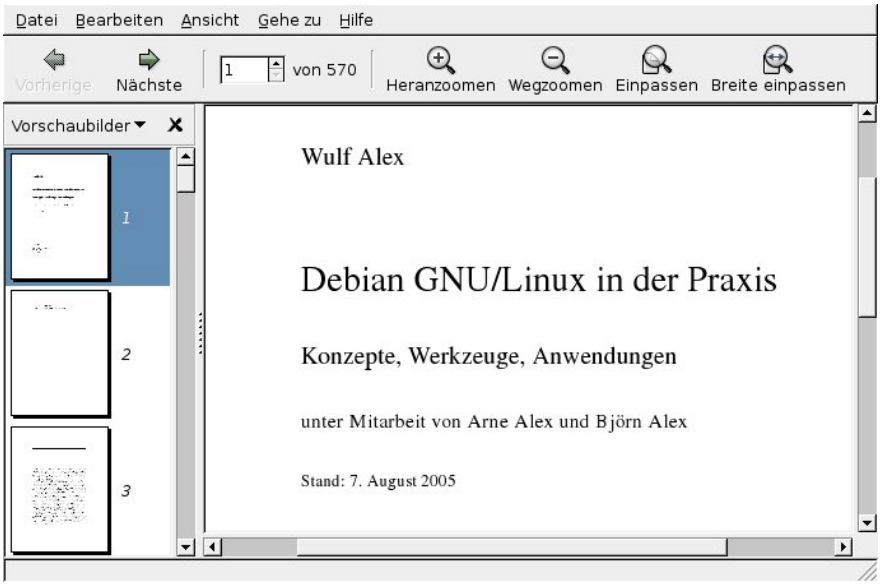

Abb. 1.1: Screenshot des Leseprogramms GNOME Evince

Das Debian-Paket *xpdf* ist ein Oberbegriff für die Pakete *xpdf-common*, *xpdf-reader* und *xpdf-utils*, die zusammen eine Werkzeugkiste zur Handhabung von PDF-Dateien darstellen. Das Kommando `xpdf` ruft einen Betrachter auf, eine Alternative zum Adobe Reader. GNOME `gpdf` setzt auf `xpdf` auf, KDE `kpdf` ist aus `xpdf` abgeleitet. Im Debian-Paket *pdftk* (PDF Toolkit) ist ein vielseitiges Werkzeug zur Bearbeitung von PDF-Dateien enthalten, Heimathafen `http://www.accesspdf.com/pdftk/`, ein deutsches Tutorial von STEFAN LAGOTZKI gibt es bei `http://www.lagotzki.de/pdftk/`. Wie schon erwähnt, kann `gv` außer PostScript auch PDF lesen. Das noch junge Werkzeug Evince aus dem GNOME-Projekt liest PostScript, Encapsulated PostScript, PDF und künftig vielleicht weitere Formate. Es setzt auf Ghostscript und `xpdf` auf. Sein Ziel ist, zahlreiche Einzelwerkzeuge wie `ggv`, `gpdf` und `xpdf` zu erübrigen. Das Werkzeug hinterließ einen angenehmen, aufgeräumten Eindruck, siehe Abbildung 1.1.

1.2.5 Lesen von Word-Dateien und Ähnlichem (catdoc, wordview, antiword, wv, word2x, xlhtml, unrtf)

Ein Debian-Benutzer ist manchmal gezwungen, Textdateien zu lesen, die in einem proprietären Format vorliegen. Am häufigsten handelt es sich dabei um Dateien, die mit MS Word geschrieben sind, kenntlich an der Dateikennung `doc`. Einige der WYSIWYG-Editoren – siehe Abschnitt 1.6 *WYSIWYG-Textprozessoren* auf Seite 46 – können mit Word-Dateien umgehen, wobei unter Umständen komplexe Strukturen verloren gehen. Will man solche Dateien nur lesen, gibt es einfachere Wege.

Das Werkzeug `catdoc` liest mit MS Word oder MS Write geschriebene Texte sowie Textdateien im RTF-Format. Es beherrscht mehrere Zeichensätze; intern verwendet es Unicode. Für die Ausgabe kann es LaTeX-Metazeichen in entsprechende Sequenzen umsetzen, sodass eine Weiterverarbeitung mittels LaTeX erleichtert wird. Es ist jedoch kein Word-to-LaTeX-Konverter, dafür sind die Unterschiede zwischen den beiden Textwelten zu grundlegend. `catdoc` versucht nicht, die Strukturen der eingelesenen Texte zu erhalten oder zu übersetzen.

Das Debian-Paket enthält auch das Werkzeug `xls2csv`, mit dem MS-Excel-Tabellen in Tabellen aus Zeilen mit durch Kommas getrennten Werten (comma separated values) umgewandelt werden. Schließlich bringt das Debian-Paket noch `wordview` mit, eine auf Tcl/Tk beruhende grafische Benutzeroberfläche zu `catdoc`. Da Microsoft mehrmals das Format von Word geändert hat, ist es unwahrscheinlich, dass `catdoc` mit *jeder* Word-Datei klarkommt. Mit Word 8 bzw. Word aus Office 97 erzeugte Dateien sollten keine Schwierigkeiten verursachen.

Das Kommando `antiword` wandelt Word-Dateien in einfachen Text oder PostScript um. Seine Startseite liegt unter `http://www.winfield.demon.nl/`.

Das Debian-Paket *wv* – früher als `mswordview` bekannt – enthält eine Bibliothek und einige Anwendungen, um Word- oder RTF-Dateien in HTML oder LaTeX umzuwandeln. Seine Funktionen werden unter anderem von den WYSIWYG-Textprozessoren Abiword und KWord verwendet. Die Startseite des Projektes findet sich unter `http://wv.sourceforge.net/`.

Nicht bei Debian, aber unter `http://word2x.sourceforge.net/` findet man das Werkzeug `word2x`, das Word-Dokumente in einfachen Text, LaTeX oder HTML umwandelt. Unter `http://pair.mbl.ca/doc2xml/` ist das Werkzeug `doc2xml` erhältlich, ein Tarball in frühem Zustand.

Das Werkzeug `xlhtml` – Startseite `http://chicago.sourceforge.net/xlhtml/` – wandelt Excel- oder PowerPoint-Dateien nach HTML, XML oder CSV um, falls man nicht eine Tabellenkalkulation bzw. ein Präsentationswerkzeug dafür einsetzen möchte.

Das Debian-Paket *unrtf* enthält einen Umwandler von RTF nach ASCII-Text, HTML, LaTeX und PostScript. Weitere Formate sind in Entwicklung. Am weitesten fortgeschritten ist die Umwandlung nach HTML. Näheres auf `http://www.gnu.org/software/unrtf/unrtf.html`.

1.2.6 Optical Character Recognition (clara, gocr)

Optische Zeichenerkennung (E: optical character recognition, OCR, F: reconnaissance optique de caractères) bezeichnet das maschinelle Erkennen von Zeichen und damit Lesen von Texten, beispielsweise im Zusammenhang mit der automatischen Bearbeitung von Schecks. Hierzu wurden Schriften entwickelt, die sowohl von Menschen wie von Maschinen gelesen werden können und insbesondere bei Ziffern die Wahrscheinlichkeit von Verwechslungen minimieren. Auch die neueren Kfz-Kennzeichen verwenden eine solche Schrift. Mit der steigenden Leistungsfähigkeit der Rechner ist der Bedarf an OCR-Schriften jedoch zurückgegangen. Eine wichtige Anwendung von OCR ist die Umwandlung von gescannten oder gefaxten Textvorlagen – die als Rastergrafik vorliegen – zurück in Text. Das Projekt Gutenberg (`http://www.gutenberg.org/`, im deutschen Sprachraum `http://gutenberg.spiegel.de/`), das urheberrechtsfreie Literatur ins Netz stellt, ist ein weltweites, umfangreiches Beispiel für die Zeichenerkennung. Sie finden dort unter anderem Werke von SHAKESPEARE, EINSTEIN, EURIPIDES, FONTANE und KARL MAY.

Das Werkzeug `clara` erwartet Vorlagen im `pbm`- oder `pgm`-Format und setzt X11 voraus. Auf der Webseite `http://www.claraocr.org/` finden sich neben Informationen in Englisch auch einige Textbeispiele zum Ausprobieren, falls man keinen gescannten Text zur Hand hat. Das Programm ist lernfähig. Leider geht seine Entwicklung zur Zeit nur langsam voran.

Das Kommandozeilen-Werkzeug `gocr` verlangt Dateien im `pbm`-, `pnm`-, `pgm`-, `ppm`- oder `pcx`-Format und kommt unter gewissen Voraussetzungen auch mit `png`, `jpg`-, `tiff`- oder `gif`-Dateien zurecht.. Der Gebrauch ist denkbar einfach:

```
joe@debian:~$ gocr datei
```

liest die Grafikdatei `datei` und schreibt das Ergebnis nach `stdout`. Bei einfach aufgebauten Seiten liefert es nach etwas Konfiguration brauchbare Ergebnisse. Seine Webseite ist unter `http://jocr.sourceforge.net/` zu finden; der Name *jocr* ist kein Schreibfehler, *gocr* war bei Sourceforge schon besetzt. Bei Debian liegen noch einige ergänzende Pakete zu `gocr`, darunter ein grafisches Frontend und Dokumentation.

1.2.7 Umcodieren von Zeichen und Zeichensätzen (tr, recode, uniconv, iconv, convmv)

Zur Umsetzung von Zeichen gibt es mehrere Linux/UNIX-Werkzeuge wie `tr` und `sed`. Das erstere ist auch ein Nothelfer, wenn man ein Zeichen in einen Text einbauen möchte, das man nicht auf der Tastatur findet. Will man beispielsweise den Namen *Citroën* richtig schreiben und sieht keine Möglichkeit, das e mit dem Trema per Tastatur zu erzeugen, dann schreibt man *CitroXn* und schickt den Text durch:

```
joe@debian:~$ tr '\130' '\315' < text > text.neu
```

Die Zahlen sind die oktalen Positionen, hier im Zeichensatz ROMAN8. Mit dem Streaming Editor `sed` lassen sich komplexe Ersetzungen bewerkstelligen, siehe Abschnitt 1.9 *Streaming Editor* auf Seite 51.

Aus dem GNU-Projekt stammt ein Filter namens `recode`, dass etwa hundert Codetafeln oder Zeichensätze ineinander umrechnet:

```
joe@debian:~$ recode --help
joe@debian:~$ recode -l
joe@debian:~$ recode ascii-bs:EBCDIC-IBM textfile
```

Man beachte jedoch, dass beispielsweise ein HTML-Text, der mit ASCII-Ersatzdarstellungen für die Umlaute (`&auml;` für a-Umlaut) geschrieben ist, bei Umwandlung nach ASCII unverändert bleibt. Es werden Zeichensätze umgewandelt, mehr nicht. Auch werden LaTeX-Formatanweisungen nicht in HTML-Formatanweisungen übersetzt, dafür gibt es andere Werkzeuge wie `latex2html`. Die Ursprungsdatei wird überschrieben, daher sicherheitshalber mit einer Kopie arbeiten. Nicht jede Umsetzung ist umkehrbar.

Unter dem Namen *uniconv* laufen mehrere, teilweise kommerzielle Produkte. Das Debian-Paket *yudit* – siehe Abschnitt 1.8 *Editoren für fremde Schriften* auf Seite 50 – bringt ein Werkzeug `uniconv` mit, das verschiedene Zeichensätze und -codierungen ineinander umrechnet: UTF-8, UTF-16, ISO 8859-1, ISO 8859-2, ISO 8859-5, CP 1251, ISO 2022 und weitere. Ähnliches leistet das Kommando `iconv` aus dem GNU-Projekt (`http://www.gnu.org/software/libiconv/`. Aufgerufen mit der Option `-l` listen beide Kommandos die ihnen bekannten Zeichensätze und -codierungen auf. Das Werkzeug `convmv` übersetzt Dateinamen nach UTF-8 und hilft beim Umstellen eines Systems auf Unicode.

1.3 Der Editor vi samt Nachwuchs

1.3.1 Schnellstart

Der Name der zu editierenden Datei wird beim Aufruf als Argument mitgegeben:

```
joe@debian:~$ vi datei
```

Der `vi` kennt zwei Modi: den Kommando-Modus[4] (E: command mode) und den Schreib-Modus (E: input mode, editing mode). Tasteneingaben im Kommando-Modus werden als Kommandos aufgefasst, im Schreib-Modus als zu speichernder Text. Kommandos, die mit einem Doppelpunkt (E: colon) beginnen, sind mit `<cr>` abzuschicken; sie werden auf der untersten Zeile angezeigt. Die übrigen Kommandos werden nicht angezeigt und wirken sofort. Mit einem Dutzend `vi`-Kommandos kommt man weit. Was darüber hinaus geht, lernt man bei Bedarf. Alle `vi`-Kommandos nutzt kein Schreiber aus. Das Dutzend (die aktuelle Zeile ist die, in der sich der Cursor befindet):

[4] Genau genommen gibt es zwei Kommando-Modi, was jedoch für den praktischen Gebrauch keine Bedeutung hat.

- <esc> schalte in den Kommando-Modus um,
- o (open) öffne eine neue Zeile zum Schreiben unterhalb der aktuellen Zeile,
- i (insert) füge in aktueller Zeile Text vor der Cursorposition ein,
- a (append) füge in aktueller Zeile Text nach der Cursorposition ein,
- r, R (replace) ersetze genau ein Zeichen bzw. beliebig langen Text,
- x lösche genau ein Zeichen,
- dd (delete) lösche die aktuelle Zeile,
- u (undo) nimm jüngstes Editierkommando zurück,
- nG (go) springe an den Anfang der Zeile n,
- /abcd <cr> suche nach der Zeichenkette abdc vorwärts,
- ?abcd <cr> suche nach der Zeichenkette abdc rückwärts,
- :wq <cr> (write, quit) schreibe die Arbeitskopie zurück und beende den Editor.

Falls Ihre Pfeiltasten (Cursortasten) nicht funktionieren, können Sie sich mit folgenden Buchstabentasten im Kommandomodus helfen:

- h gehe ein Zeichen nach links,
- j gehe eine Zeile nach unten,
- k gehe eine Zeile nach oben,
- l gehe ein Zeichen nach rechts.

Die Zeichen liegen auf der Tastatur in der Mitte beieinander. Diese Alternative wird auch von einigen anderen Programmen genutzt. Wenn Sie jetzt noch wissen, dass man vor die Lösch-Kommandos x und dd sowie die vier Cursorkommandos eine Zahlenangabe setzen darf, die die Kommandos auf entsprechend viele Zeichen bzw. Zeilen ausweitet, beherrschen Sie die Pflicht eines vi-Benutzers. Was folgt, ist Kür.

Während Sie editieren, können Sie Hilfe zu vielen Themen anfordern, indem Sie im Kommandomodus :h thema <cr> (help) eingeben. Zurück aus der Hilfe in Ihren Text gelangen Sie mit :q <cr> (quit).

1.3.2 Übersicht

Der bildschirmorientierte Texteditor vi (visual editor) wurde um 1976 von BILL JOY in Berkeley entwickelt. Von ihm stammt auch die Aussprache *vee-eye*, deutsch *wie-ei*. Der vi löste nach und nach einen einfacheren, heute schon sehr klassischen Editor namens ed ab, der in Zeiten von Druckerterminals und langsamen Datenleitungen seine Berechtigung hatte[5], und ist heute auf jedem Linux/UNIX-System zu finden. vi-ähnliche Editoren sind außer für Linux/UNIX auch für andere Betriebssysteme einschließlich DOS geschrieben worden. Unter Debian GNU/Linux ist meist der nvi (new vi) von KEITH BOSTIC oder der vim (vi improved) von BRAM MOOLENAAR eingerichtet und meldet sich unter dem Aufruf vi, siehe /etc/alternatives/. Unterschiede bemerkt erst ein fortgeschrittener Benutzer. gvim

[5] Ich habe ihn verwendet, als meine Verbindung von daheim zur Universität über einen Akustikkoppler lief. Der Verwalter braucht ihn in verzweifelten Situationen.

ist im Debian-Paket *vim-full* enthalten und ein um eine X11-Oberfläche erweiterter `vim`. Den originalen `vi` gibt es noch bei `http://ex-vi.sourceforge.net/`. Infolge des Alters und der weiten Verbreitung der `vi`-Familie auch auf anderen Systemen findet sich viel Information im Netz und in den Buchregalen. Eine Auswahl:

- Manualseite (`man vi`), nur zum Nachschlagen, nicht zum Lernen,
- Online-Tutorial: aus dem Editor heraus `:help tutor` eingeben, verlassen mit `:q`,
- Vim-HOWTO, zur Zeit (Mitte 2005) in Revision,
- FAQ: `http://www.faqs.org/faqs/editor-faq/vi/part1/` und `part2`,
- WILLIAM JOY und MARK HORTON: *An Introduction to Display Editing with Vi*, `http://docs.freebsd.org/44doc/usd/12.vi/paper.html`,
- *Mastering the VI editor*, `http://www.eng.hawaii.edu/Tutor/vi.html`,
- *The Berkeley VI Editor Home Page*, `http://www.bostic.com/vi/` (es geht um `nvi`),
- `http://freshmeat.net/projects/nvi/`,
- `http://www.vim.org/` mit viel Dokumentation,
- *The VI Lovers Home Page*, `http://thome.com/vi/vi.html`,

Der Editor hat Grenzen. Beim klassischen `vi` von 1987 beträgt die Zeilenlänge maximal 1024 Zeichen, beim `vim` über 2 Milliarden Zeichen (das Manuskript zu vorliegendem Buch umfasst rund 2 Millionen Zeichen). Die maximale Anzahl der Zeilen beträgt 250.000 , beim `vim` über 2 Milliarden. Die maximale Dateigröße liegt bei 0,5 MB, beim `vim` 2 GByte, wobei es Unfug ist, Textdateien so groß werden zu lassen. Zusätzlich begrenzen Arbeitsspeicher einschließlich Swapbereich die Dateigröße. Hat man solche Monsterdateien zu editieren, können Hex-Editoren (Seite 48) weiterhelfen. In den folgenden Abschnitten wird der `vi` systematisch und ausführlich, aber nicht erschöpfend dargestellt.

1.3.3 Konfiguration

Wer mit dem `vi` beginnt, wird die Default-Einstellungen übernehmen. Im Lauf der Zeit kann man über einige Werte nachdenken, um sich die Arbeit zu erleichtern. Die Einstellungen werden als `vi`-Umgebung bezeichnet. Es gibt drei Wege, sie zu beeinflussen:

- durch Setzen einer systemweiten oder benutzerspezifischen Umgebungsvariablen namens `EXINIT`,
- durch eine benutzereigene Datei namens `.exrc` bzw. `.vimrc`,
- während des Editierens durch Eingeben des Kommandos `:set`.

Die augenblicklichen Einstellungen bekommt man während des Editierens mittels des Kommandos `:set all` zu Gesicht. Wir beschränken uns auf die Konfiguration mit Hilfe der Datei `.exrc`. Es kann je nach `vi`-Variante erforderlich sein, den

Editor zum automatischen Lesen dieser Datei durch Setzen einer Option mittels der EXINIT-Variablen in einer Datei wie $HOME/.profile oder $HOME/.bashrc zu bewegen:

```
export EXINIT="set exrc"
```

Äußerst wichtig für den vi ist die Shell-Umgebungsvariable TERM, die ihm sagt, mit welchem Terminaltyp er zusammenarbeitet. Auf Grund der Eintragung unter TERM schaut der Editor entweder in einer Datei /etc/termcap oder /etc/terminfo nach, um die jeweiligen Steuerkommandos herauszulesen. Es müssen also sowohl die TERM-Variable wie die Terminalbeschreibung in termcap oder terminfo stimmen. Für die TERM-Variable ist der Benutzer verantwortlich, für die Terminalbeschreibung der Verwalter. Terminaltyp und TERM-Variable ermittelt man mit folgenden Kommandos:

```
joe@debian:~$ tset -q
```

```
joe@debian:~$ echo $TERM
```

Sie sollten übereinstimmen. Faselt der Editor nach dem Aufruf etwas von *Open Mode*, so kennt er den Terminaltyp nicht und nimmt ein dummes Terminal an. Damit kann man arbeiten, aber nicht glücklich werden. Arbeitet man mittels slogin oder einem ähnlichen Kommando auf einem entfernten Rechner und zeigt sich der vi widerspenstig, gilt der erste Blick der Variablen TERM, die erforderlichenfalls anzupassen ist. Innerhalb der Shell (bash) geschieht dies mittels einer Eingabe wie TERM=xterm, innerhalb des Editors mittels einer Kommandofolge wie:

```
Q set term=xterm <cr> vi <cr>
```

Damit wird zuerst der vi-Modus verlassen und in den ex-Modus geschaltet (Q), dort der Terminaltyp gesetzt (set term=xterm <cr>) und schließlich in den vi-Modus zurückgeschaltet (vi <cr>). Einfacher ist es, schon vor dem Aufruf des Editors die Shellvariable TERM zu überprüfen, siehe oben.

In seinem Home-Verzeichnis oder einem Unterverzeichnis kann ein Benutzer eine Datei .exrc anlegen. Auf diese Weise lassen sich für bestimmte Verzeichnisse – beispielsweise mit Programmquellen – besondere Verhaltensweisen oder Optionen festlegen. In die Datei werden Kommandos geschrieben, wie sie auch während des Editierens eingegeben werden dürfen. Einige sind:

- :set autoindent rücke aktuelle Zeile automatisch wie die vorhergehende ein (kann nerven),
- :set ignorecase unterscheide nicht zwischen Groß- und Kleinschreibung bei der Suche mit regulären Ausdrücken,
- :set number nummeriere die Zeilen auf dem Bildschirm, nicht im Speicher,
- :set readonly rufe den Editor per Default nur zum Lesen auf,
- :set showmatch springe beim Eintippen einer schließenden Klammer kurz zur zugehörigen öffnenden Klammer,
- :set tabstop=4 setze den Tabulatorsprung auf vier Zeichen statt auf acht (Default),

- `:set wrapmargin=n` bricht lange Zeilen auf dem Bildschirm automatisch um, und zwar n Zeichen vor dem rechten Rand.

Die Vorsilbe `no` hat die entgegengesetzte Wirkung:

```
:set nonumber
```

bewirkt, dass die Zeilen auf dem Schirm nicht nummeriert werden (Default).

Der `vim` versteht einige Optionen mehr als der klassische `vi`. Interessant ist die Syntax-Hervorhebung. Der Editor kennt mehrere hundert Dateiformate, darunter Quelltexte gängiger Programmiersprachen sowie LaTeX. Schaltet man die Option ein:

```
:syntax on
```

werden die Schlüsselwörter des jeweiligen Formates farbig hervorgehoben. Die Formatbeschreibungen liegen in `/usr/share/vim/vim63/syntax/`. Man kann sie sich mit `less` ansehen, neue erfinden oder – was eher gefragt sein könnte – aus vorhandenen eigene Varianten ableiten. Schaltet man die Syntax-Hervorhebung ein, sollte man nicht auch noch eine Hervorhebung von Fehlern durch einen Rechtschreibprüfer aktivieren. Das funktioniert zwar, sieht auf dem Bildschirm aber chaotisch aus.

1.3.4 Makros

In der Datei `.exrc` lassen sich auch Makros unterbringen. Das sind kurze Kommandofolgen, die einer einzelnen Taste oder einer Tastenkombination (keychord, key sequence) zugeordnet sind. Wir unterscheiden zwischen Makrobefehlen, die im Kommandomodus einzugeben sind, und solchen, die im Schreibmodus eingetippt werden (ohne vorheriges `<esc>`). Erstere werden durch Zeilen folgender Art in der Datei `.exrc` geschaffen:

```
:map makroname kommandos <cr>
```

Als Makroname kommt bevorzugt ein einzelnes Zeichen in Betracht. Die Buchstaben g und v sind vom `vi` nicht belegt und können bedenkenlos als Makroname herangezogen werden, eventuell auch die Funktionstasten. Definieren wir:

```
:map g ?index<ctrl>v<cr><cr>
```

so führt die Eingabe von `<g>` im Kommando-Modus zum Zurückspringen auf die nächstliegende Zeichenkette *index*. Das erste `<cr>` ist Bestandteil des Kommandos. Um es als Zeichen eingeben zu können, ist ihm die Zeichenkombination `<ctrl>-<v>` voranzustellen. Das zweite `<cr>` schließt das Makro ab.

Im Schreibmodus einzugebende Makros werden folgendermaßen definiert:

```
:map! makroname zeichenkette <cr>
```

Ein Zeichen, dem man auf diese Weise eine besondere Bedeutung verleiht, kann dann nur durch Voranstellen von `<ctrl>-<v>` in seiner ursprünglichen Bedeutung eingegeben werden. Beispiel:

```
:map! C Chemieingenieurwesen <cr>
```

bewirkt, dass die Eingabe von <C> im Schreibmodus zum Erzeugen der Zeichenkette *Chemieingenieruwesen* führt, bei der ich mich fast immer vertippe. Künftig tippe ich nur noch <C>, und die Zeichenkette erscheint. Brauche ich dann in einem Brief an CAIUS JULIUS CAESAR den Buchstaben <C>, muss ich <ctrl>-<v> voranstellen. Die Frage ist, was man häufiger benötigt. Mittels <ctlr>-<v> lassen sich auch das Zeichen <esc> und andere Sonderzeichen buchstäblich eingeben. Die aktuellen Mappings erfährt man mittels `:map`.

Verwandt mit den vorstehenden Makros sind Abkürzungen. Der Unterschied liegt darin, dass eine Abkürzung nur dann erweitert (expandiert) wird, wenn sie als Wort für sich allein steht, nicht jedoch, wenn sie als Teil eines längeren Wortes auftritt. Wollen wir CIW als Abkürzung für Chemieingenieurwesen verwenden, so definieren wir:

```
:abbr CIW Chemieingenieurwesen<cr>
```

und in unserem Text wird die Eingabe `CIW` immer zu *Chemieingenieurwesen* erweitert. Das entspricht unserem Wunsch vermutlich besser als das Makro. Vertippt man sich bei einem Wort häufig in derselben Weise, so lässt sich die falsche Schreibweise als Abkürzung für die richtige definieren:

```
:abbr Karslruhe Karlsruhe<cr>
```

und der Editor verbessert unsere Eingabe stillschweigend. Alternativ wäre Üben mit der Tastatur anzuraten, siehe Abschnitt 1.13.1 *Tipptrainer* auf Seite 76.

1.3.5 Aufrufen und Beenden

Der Editor wird aus der Kommandozeile in einer der folgenden Formen aufgerufen:

```
joe@debian:~$ vi
```

Die Arbeit vollzieht sich in einem Pufferspeicher. Beim Zurückschreiben muss man sich für einen bereits vorhandenen oder neuen Namen entscheiden.

```
joe@debian:~$ vi neue_datei
```

Die Arbeit vollzieht sich in einem Pufferspeicher. Beim Zurückschreiben wird die Datei `neue_datei` angelegt.

```
joe@debian:~$ vi alte_datei
```

Der Editor arbeitet auf einer temporären Kopie von `alte_datei`. Beim Zurückschreiben wird `alte_datei` mit der bearbeiteten Kopie überschrieben. Falls mehrere Benutzer gleichzeitig auf Kopien von `alte_datei` arbeiten, gewinnt der, der als letzter zurückschreibt.

```
joe@debian:~$ vi +144 alte_datei
```

Wie oben, der Editor springt gleich in Zeile Nr. 144. Praktisch, wenn ein Fehlersuchprogramm die Nummer der fehlerhaften Zeile mitgeteilt hat.

```
joe@debian:~$ vi +/muster alte_datei
```

Wie oben, der Editor springt gleich zum ersten Vorkommen des Zeichenmusters.

```
joe@debian:~$ vi datei1 datei2 datei3
```

Der Editor arbeitet auf einer Kopie von `datei1`. Nach Zurückschreiben mittels des Kommandos `:w` holt man durch das Kommando `:n <cr>` eine Kopie der nächsten Datei in den Pufferspeicher. Das Kommando `rew` (rewind) geht in der Dateiliste einen Schritt zurück.

```
joe@debian:~$ view alte_datei
```

Der Editor wird nur zum Lesen (readonly) aufgerufen. Sie können alles mit der temporären Kopie der Datei anstellen, nur nicht zurückschreiben. Eine Alternative zu Pagern wie `more` oder `less`.

Zum Zurückschreiben der temporären Arbeitskopie auf den Massenspeicher dient das Kommando `:w` (write). Man kann und soll es auch während der laufenden Arbeit geben, um Datenverluste bei irgendwelchen Zwischenfällen zu vermeiden. Falls die Datei schreibgeschützt ist oder man den Editor readonly (`view`) aufgerufen hat, muss man `:w!` eintippen, um seinen Willen durchzusetzen. Der Doppelpunkt weist darauf hin, dass es sich hier eigentlich um ein Kommando des Editors `ex` handelt. Tatsächlich sind `vi` und `ex` dasselbe Programm, nur mit unterschiedlichem Verhalten gegenüber dem Benutzer. Das braucht uns jedoch nicht zu interessieren. Wir arbeiten nicht ausdrücklich mit dem `ex`.

Den Editor beendet man mit dem Kommando `:q` (quit). Auch hier kann ein Ausrufezeichen erforderlich werden. Meist kombiniert man Zurückschreiben und Beenden mittels `:wq`.

Wird der Editiervorgang gewaltsam unterbrochen (Stromausfall), so steht die mit dem jüngsten `:w`-Kommando zurückgeschriebene Kopie im Massenspeicher. Aber der `vi` tut von sich aus noch mehr. Er legt im jeweiligen Arbeitsverzeichnis eine Punktdatei mit der Kennung `.swp` an (swap file) – mittels `ls -la` zu sehen – die laufend nachgeführt wird. Der `vim` aktualisiert die Swap-Datei nach der Eingabe von jeweils 200 Zeichen oder nach 4 s Ruhe, konfigurierbar. Die anderen Familienmitglieder des Editors halten sich mit diesbezüglichen Auskünften zurück. Man kann erreichen, dass die Datei nicht im Arbeitsverzeichnis, sondern in einem anderen Verzeichnis, möglichst auf einer anderen Platte, angelegt wird. Dann ist sogar bei einem Plattencrash der Text gerettet. Die Swap-Datei enthält den Text und binäre Informationen, ist also nur eingeschränkt verständlich. Beim ordnungsgemäßen Beenden des Editors wird sie gelöscht, bei Stromausfall bleibt sie erhalten. Rufen wir später den Editor mit dem Namen der während des Stromausfalls editierten Datei als Argument auf, bemerkt der Editor die Existenz der Punktdatei und bietet an, sie als die jüngste Fassung des Textes weiter zu verarbeiten (recover). So gehen im schlimmsten Fall nur wenige Eingaben infolge der Unterbrechung verloren. Einige Versionen des Editors legen vor jedem Zurückschreiben eine Sicherungskopie des vorhergehenden Textes an. Man kann so nach einer Katastrophe auf drei Fassungen des Textes zurückgreifen:

- die Swap- oder Recovery-Datei, die aktuellste Fassung,
- die Textdatei auf dem Massenspeicher mit Stand des jüngsten Zurückschreibens,
- die Sicherungskopie mit Stand vor dem jüngsten Zurückschreiben.

Wir sind dem `vi` schon dankbar gewesen. Bei einem Plattencrash hilft dieses Vorgehen ohne die oben erwähnte Auslagerung der Swap-Datei nichts, da müsste man mit Spiegeln (RAID) oder eigenen Skripts und `cron`-Jobs arbeiten.

Hat man beim Editieren so viel Unsinn gebaut, dass nichts mehr zu retten ist, holt man mit dem Kommando `:vi!` eine frische Kopie der Datei vom Massenspeicher in den Puffer. Man beginnt so mit dem Stand des jüngsten Zurückschreibens von vorn. Ist nur der Bildschirm (nicht die Datei) durcheinander geraten, schreibt das Kommando `<ctrl>-<l>` den Schirm neu. Das kann vorkommen, wenn man während der Arbeit die Fenstergröße ändert.

1.3.6 Navigieren im Text

Beim Schreiben geht ein beträchtlicher Teil der Zeit auf das Navigieren im Text drauf – der möglicherweise auf mehrere Dateien und Verzeichnisse verteilt ist. Nächst dem Eingeben des Textes ist das Navigieren die wichtigste handwerkliche Fähigkeit des Schreibers. Zunächst einige Kommandos zum Bewegen des Bildschirms (Fensters) im Text (das Zeichen `n` steht hier immer für eine Anzahl):

- `n<ctrl>-d` (down) bewege den Text samt Cursor auf dem Bildschirm (scrolle) n Zeilen abwärts,
- `n<ctrl>-u` (up) bewege den Text samt Cursor auf dem Bildschirm (scrolle) n Zeilen aufwärts,
- `n<ctrl>-e` bewege den Text auf dem Bildschirm n Zeilen abwärts, der Cursor bleibt auf seiner Position im Text,
- `n<ctrl>-y` bewege den Text auf dem Bildschirm n Zeilen aufwärts, der Cursor bleibt auf seiner Position im Text,
- `n<ctrl>-f` (forward) bewege den Text samt Cursor um n Seiten (Bildschirme) vorwärts,
- `n<ctrl>-b` (backward) bewege den Text samt Cursor um n Seiten (Bildschirme) rückwärts,
- `<ctrl>-l` schreibe einen verunstalteten Bildschirm neu,
- `z<cr>` schreibe den Bildschirm neu so, dass die aktuelle Zeile oben erscheint,
- `z.` schreibe den Bildschirm neu so, dass die aktuelle Zeile in der Mitte erscheint,
- `z-` schreibe den Bildschirm neu so, dass die aktuelle Zeile unten erscheint,

Nun einige ausgewählte Kommandos zum Bewegen des Cursors:

- `nh` bewege den Cursor um n Zeichen nach links, Default n = 1,
- `nj` bewege den Cursor um n Zeilen nach unten, Default n = 1,
- `nk` bewege den Cursor um n Zeilen nach oben, Default n = 1,
- `nl` bewege den Cursor um n Zeichen nach rechts, Default n = 1,
- `nG` bewege den Cursor nach Zeile n,
- `G` bewege den Cursor zur letzten Zeile,

- `0` (null) bewege den Cursor zum Anfang der aktuellen Zeile,
- `$` bewege den Cursor an das Ende der aktuellen Zeile,
- `nw` bewege den Cursor vorwärts auf den Anfang des n-nächsten Wortes,
- `nb` bewege den Cursor rückwärts auf den Anfang des n-nächsten Wortes,
- die beiden vorstehenden Kommandos können auch mit Großbuchstaben gegeben werden, wobei die Definition von *Wort* etwas anders lautet,
- `n}` bewege der Cursor nach rechts oder unten an den Anfang des n-nächsten Absatzes, gekennzeichnet durch eine Leerzeile,
- `n{` bewege den Cursor nach links oder oben an den Anfang des n-nächsten Absatzes rückwärts,
- `m[a-z]` (m gefolgt von einem Kleinbuchstaben) setze eine Marke im Text,
- `` `[a-z] `` (back quote) bewege den Cursor an die wie vorstehend beschrieben gesetzte Marke,
- `%` falls der Cursor unter einer Klammer (rund, eckig, geschweift) steht, springe zur ergänzenden Klammer.

Sucht man bestimmte Zeichen oder Zeichenmuster im Text, so helfen folgende Kommandos:

- `nfx` bewege den Cursor zum n-nächsten Vorkommen des Zeichens x in der Zeile,
- `nFx` dasselbe rückwärts,
- `;` (Semikolon) wiederhole eines der obigen Kommandos,
- `,` (Komma) wiederhole eines der obigen Kommandos in umgekehrter Richtung,
- `/muster<cr>` suche vorwärts nach dem Zeichenmuster *muster*,
- `?muster<cr>` suche rückwärts nach dem Zeichenmuster *muster*,
- `*` (Stern) suche vorwärts das nächste Vorkommen des Wortes, auf dem der Cursor steht (nur `vim`),
- `#` (Doppelkreuz) dasselbe rückwärts,
- `n` (nochmal) wiederhole die jüngste Suche,
- `N` wiederhole die jüngste Suche in umgekehrter Richtung.

Das Zeichenmuster darf ein regulärer Ausdruck sein, siehe Abschnitt **??** *Reguläre Ausdrücke* auf Seite **??**. Sucht man beispielsweise in einer Liste nach der Zeichenkette *Alex* am Anfang einer Zeile, so lautet das Kommando:

```
/^ Alex
```

Die letzten fünf Suchkommandos werden häufig gebraucht.

1.3.7 Text ändern

Einen Text ändern heißt, Textteile hinzufügen, löschen, ersetzen, verschieben. Diesen Zwecken dienen folgende `vi`-Kommandos:

- `a` (append) füge hinzu im Anschluss an die Cursorposition,
- `A` füge hinzu im Anschluss an die aktuelle Zeile,
- `i` (insert) füge hinzu vor der Cursorposition,
- `I` füge hinzu vor der aktuellen Zeile,

- `o` (open) öffne neue Zeile zum Schreiben unterhalb der aktuellen Zeile,
- `O` öffne neue Zeile zum Schreiben oberhalb der aktuellen Zeile,
- `nx` lösche n Zeichen ab Cursorposition, Default n = 1,
- `nX` lösche n Zeichen links von der Cursorposition, Default n = 1,
- `dnw` (delete) lösche n Wörter ab der Cursorposition, Default n = 1,
- `d$` lösche aktuelle Zeile von Cursorposition bis zum Ende,
- `dG` lösche Text von aktueller Zeile bis zum Textende,
- `d1G` lösche Text von aktueller Zeile bis zum Textanfang,
- `ndd` lösche n Zeilen ab aktueller Zeile vorwärts, Default n = 1,
- `nr` (replace) ersetze n Zeichen ab Cursorposition, Default n = 1,
- `R` ersetze beliebig viele Zeichen ab Cursorposition,
- `ncw` (change) ersetze n Wörter ab Cursorposition, Default n = 1,
- `ncc` ersetze n Zeilen ab aktueller Zeile, Default n = 1,
- `u` (undo) nimm jüngstes textänderndes Kommando zurück,
- `U` stelle die aktuelle Zeile wieder her,
- `:n,m s/regex/ersatz/g <cr>` (substitute) ersetze im Text von Zeile n bis Zeile m das Muster (regulärer Ausdruck) *regex* durch die – eventuell leere – Zeichenkette *ersatz*, und zwar global, nicht nur beim ersten Auftreten in der Zeile.

Das letzte Kommando soll mit einem Beispiel verdeutlicht werden:

```
:1,$ s/129\.13\.118\.15/172.22.132.26/g
```

Die Aufgabe ist, in einem Text überall die IP-Adresse 129.13.118.15 durch 172.22.132.26 zu ersetzen. Nach dem Doppelpunkt werden Anfangszeile (1) und Endzeile ($) genannt. Das zu ersetzende Muster 129.13.118.15 wird stets als regulärer Ausdruck verstanden. In diesen hat der Punkt eine Sonderbedeutung, die ihm durch Voranstellen des Gegenschrägstrichs genommen werden muss. Der Ersatz kann nur eine feste Zeichenkette sein, auch die leere. Wir lernen das Kommando genauer in Abschnitt 1.9 *Streaming Editor* auf Seite 51 kennen.

Wollen Sie die Kontrolle darüber behalten, was ersetzt wird, bietet sich folgendes Vorgehen an:

1. Suchen Sie vom Dateianfang ausgehend mit dem Kommando `/muster` das erste Auftreten des zu ersetzenden Musters,
2. ändern Sie manuell mittels genau eines Editierkommandos wie `cw` das Muster,
3. suchen Sie mittels `n` das nächste Auftreten des Musters,
4. wiederholen Sie mittels `.` (Punkt) das jüngste ändernde Kommando oder übergehen Sie diesen Schritt,
5. wiederholen Sie die beiden vorstehenden Schritte, bis Sie am Ende der Datei angelangt sind.

Wenn Sie *Anna* durch *Otto* ersetzen, denken Sie daran, dass sich auch Personal- und Relativpronomina ändern.

1.3.8 Pufferspeicher und Zwischendateien

Der Editor verwaltet neben dem Pufferspeicher, der die Arbeitskopie des gesamten Textes enthält, mehrere Puffer, in denen Textaussschnitte vorübergehend abgelegt werden. Ebenso können auch Textauschnitte in Dateien herausgeschrieben und von diesen wieder an anderer Stelle eingelesen werden.

In einem namenlosen Puffer befindet sich immer der zuletzt gelöschte Text (Wörter, Zeilen). Jedes neue Löschkommando überschreibt den Puffer. Ferner lässt sich der Puffer gezielt füllen, ohne die jeweiligen Zeilen zu löschen. Dazu dient das yank-Kommando:

- `yw` kopiert ein Wort in den namenlosen Puffer,
- `nY` kopiert n Zeilen in den namenlosen Puffer, Default n = 1.

Den Pufferinhalt schreibt man folgendermaßen an beliebiger Stelle zurück:

- `p` (put) schreibe den Pufferinhalt rechts (bei Wörtern) bzw. unterhalb (bei Zeilen) der Cursorposition zurück,
- `P` schreibe den Pufferinhalt links (bei Wörtern) bzw. oberhalb (bei Zeilen) der Cursorposition zurück.

Der Puffer kann beliebig oft ausgelesen werden. Nützlich, wenn man Zeilen umsortiert oder Zeilen mit geringen Änderungen mehrmals benötigt.

In neun, von 1 bis 9 durchnummerierten Puffern speichert der Editor die durch die neun jüngsten Zeilen-Löschkommandos gelöschten Zeilen. Die Zeilen werden zurückgewonnen, indem man den obigen put-Kommandos Anführungszeichen und die Puffernummer voranstellt:

```
"5p
```

schreibt unterhalb der gerade aktuellen Zeile die beim fünftjüngsten Zeilen-Löschkommando gelöschten Zeilen zurück. Hat man vergessen, in welchem der Puffer sich der gewünschte Text befindet, kann man sich mit nachstehender Kommandofolge durch die Puffer durchhangeln:

```
"1pu.u.u.
```

usw. Der Punkt nach dem undo-Kommando bewirkt hier ein Fortschreiten in den nummerierten Puffern.

Schließlich kennt der Editor noch 26 mit den Buchstaben von a bis z benannte Puffer. Diese füllt der Editor nur auf Anweisung. Dazu sind den entsprechenden Kommandos (yank, delete, change) ein Anführungszeichen und der dem Puffer zugehörige Buchstabe voranzustellen:

- `"a2Y` kopiere die aktuelle und die folgende Zeile in den Puffer a,
- `"b4dd` lösche die aktuelle und drei folgende Zeilen und kopiere sie zusätzlich in den Puffer b,
- `"c2cw` ersetze zwei Wörter und kopiere sie zusätzlich in den Puffer c.

Schreibt man die Namen der Puffer groß, wird der Text an bereits im Puffer befindlichen Text angehängt. Zum Herausschreiben dienen die put-Kommandos mit vorangestellten Anführungszeichen und Puffernamen:

- `"ap` kopiere den Inhalt des Puffers a rechts bzw. unterhalb des Cursors,
- `"bP` kopiere den Inhalt des Puffers b links bzw. oberhalb des Cursors.

Die benannten Puffer bewahren ihren Inhalt, wenn man innerhalb des Editors den zu bearbeitenden Text (die Arbeitskopie) mittels des Kommandos `:e` wechselt. So lassen sich einfach Textausschnitte zwischen Textdateien austauschen.

Alle Puffer werden beim Verlassen des Editors gelöscht. Will man Textausschnitte längerfristig aufbewahren, schreibt man sie in eine Datei. Hierzu markiert man die erste und die letzte Zeile des zu kopierenden Textausschnittes mittels `ma` und `mb` und schreibt den Textausschnitt in eine Datei beliebigen Namens:

```
:'a,'bw datei
```

a und b sind hier keine Puffer, sondern Marken, deshalb stehen Hochkommas. Eingelesen wird eine beliebige Datei unterhalb der aktuellen Zeile mit:

```
:r datei
```

Die Datei bleibt dabei erhalten; sie ist außerhalb des Editors mittels `rm` zu löschen.

1.3.9 Sonstige Kommandos

Wir haben noch einige häufiger gebrauchte vi-Kommandos zu erläutern, die sich keiner der vorgenannten Gruppen zuordnen lassen:

- `J` (join) verbinde die aktuelle Zeile mit der folgenden,
- `.` (Punkt, dot) wiederhole das jüngste Kommando, das die Arbeitskopie verändert hat (funktioniert nicht mit Kommandos, die mit einem Doppelpunkt beginnen),
- `:sh` (shell) stelle vorübergehend eine Shell zur Verfügung (aus-shellen),
- `:!command` führe das Shellkommando `command` aus, beispielsweise `date` oder `ls -l`,
- `:ve` zeige die Versionsnummer des Editors an,
- `<ctrl>-g` zeige den Dateinamen und die Zeilennummer an,
- `:e datei <cr>` (edit) lies die Datei `datei` ein und bearbeite sie. Das setzt voraus, dass die vorangehende Arbeitskopie mit `:w` zurückgeschrieben worden ist,
- `:help` zeige einen Hilfetext an.

Einige Zeichen wie @ oder # können vom Betriebssystem her mit Bedeutungen versehen sein (@ = lösche Zeile, line kill) , die sich auch im Editor auswirken. Das ist eher verwirrend als hilfreich. Wenn es sich nicht abstellen lässt (`man tset`), müssen die Sonderzeichen durch andere Darstellungen ersetzt werden, siehe Abschnitt **??** *Metazeichen* auf Seite **??**.

Arbeitet man mit dem `vim` auf einer Tastatur ohne Umlaute (US-Tastatur) und braucht gelegentlich einen Umlaut, so gibt man `<ctrl>-<k>` ein, ein Fragezeichen erscheint auf dem Schirm, dann tippt man `<:> <a>` bzw. einen anderen Grundlaut ein und erhält den Umlaut. Das Eszett erzeugt man mit `<s> <s>`.

1.3.10 Zusammenarbeit vim – ispell

Es ist wünschenswert, Tipp- und Rechtschreibfehler schon bei der Eingabe zu erkennen und zu verbessern. Das unterbricht zwar den Schreibfluss, kostet aber weniger Zeit als eine nachträgliche Korrektur. Dazu müsste der Editor etwas von Rechtschreibung verstehen. Nun sind die Editoren ohnehin schon recht dicke Brocken, außerdem ist das Rad in Form von Rechtschreibprüfern wie `ispell` schon erfunden, siehe Abschnitt 1.13.4 *Rechtschreibung prüfen* auf Seite 78. Optimal erscheint die Zusammenarbeit eines Editors mit einem Prüfprogramm. Lösungen für `vim` sind zu finden unter:

- `http://vim.sourceforge.net/scripts/script.php?script\_id=195`, von CHARLES E. CAMPBELL,
- `http://home.tu-clausthal.de/~mwra/vim/spell.vim`, ein Vorschlag von RALF ARENS,
- `http://hermitte.free.fr/vim/VS\_help.html`, von LUC HERMITTE.

Ein Versuch mit dem Skript `Ispell.vim` von RALF ARENS als `.vimrc` verlief insofern erfolgreich, als man aus dem Editor heraus `ispell` aufrufen konnte. Kleinere Anpassungen waren wegen der neuen deutschen Rechtschreibung (german -> ngerman), wegen LaTeX und wegen des auf der US-Tastatur nicht vorhandenen, in den Mappings am Ende des Skripts verwendeten Buchstabens ä (-> g) erforderlich. Aber das war noch nicht das Gesuchte. Der Prüfer sollte parallel zum Schreiben mitlaufen (check on the fly).

Um das Skript `vimspell.vim` von LUC HERMITTE einzurichten, gibt man in einer beliebigen Editiersitzung das Kommando `:h add-global-plugin`, tut wie dort geheißen und kopiert das Skript nach `$HOME/.vim/plugin/`. An `.vimrc` ist nichts zu ändern. Eine Dokumentation steht am Ende des Skripts. Beim nächsten Editierversuch – der genau diese Zeilen hier betraf – wurden verdächtige Wörter unterstrichen, on the fly, bei Korrekturen nach einer kurzen Denkpause. LaTeX-Kommandos blieben unbeanstandet. Sieht hoffnungsvoll aus. Jetzt fehlt nur noch die Integration eines LaTeX-Prüfers. Wenn man die Syntax-Hervorhebung eingeschaltet hat, sollte man nicht auch noch verdächtige Wörter markieren lassen, das Erscheinungsbild wird sonst sehr unruhig. Entweder – oder.

Das war etwa die Hälfte der verfügbaren vi-Kommandos. Spezialitäten des `vim` oder `nvi` sind kaum berücksichtigt. Fangen Sie mit wenigen `vi`-Kommandos an, beobachten Sie, welche Vorgänge Sie häufig brauchen und suchen Sie nach dem einfachsten Weg, den Editor zu der gewünschten Tätigkeit zu bewegen. Vermeiden Sie, seltene Aufgaben zu optimieren. Sollte Ihnen das Gebotene nicht reichen, greifen Sie

auf Monografien oder die Referenzliteratur zurück. Wir haben noch andere Themen vor uns.

1.3.11 VimOutliner – ein Outline Prozessor

Ein Outline Prozessor ist ein Werkzeug, um Entwürfe, Umrisse oder Grundzüge von umfangreichen Textdokumenten zu erstellen und zu ändern. Die Arbeit am Buch begann mit einem Entwurf der Kapitel, allerdings noch mit Bleistift und Papier. Als so viel Stoff vorhanden war, dass LaTeX ein Inhaltsverzeichnis schreiben konnte, diente dieses als Outline, in das – immer noch mit Bleistift – Einfälle und Änderungen eingetragen wurden. Ein Outline spiegelt mehr die Gedanken wider – unabhängig von einer Kapitelstruktur – während ein Inhaltsverzeichnis die Einteilung des Manuskriptes darstellt, was nicht dasselbe ist, aber im vorliegenden Fall ausreichend genau übereinstimmt. Falls es zu einer zweiten Auflage kommt, wird sie natürlich mit einem Outline Prozessor erstellt. Seine Aufgaben greifen in die Gebiete von Mindmappern (Abschnitt **??** *Mindmapping* auf Seite **??**) und Jottern oder Notizbüchern (Abschnitt **??** *Notizbuch, Personal Information Manager* auf Seite **??**) über.

Das Debian-Paket *vim-vimoutliner* stellt einige Skripte, Konfigurationsdateien und Werkzeuge bereit, die den Editor `vim` um Fähigkeiten zum Outlining bereichern. Der VimOutliner (VO) ist auf `http://www.vimoutliner.org/` beheimatet, Dokumentation findet sich nach Einrichtung des Pakets in `/usr/share/doc/vim-vimoutliner/`. Manualseiten gibt es nur zu den einzelnen Werkzeugen:

- `otl2docbook`, ein Perl-Skript, das die Ausgabe von VO in das DocBook-XML-Format umwandelt,
- `otl2html`, ein Python-Skript, das die Ausgabe von VO nach HTML umwandelt,
- `otl2pdb`, ein Perl-Skript, das die Ausgabe von VO in eine AddressDB.pdb-Datei für einen Palm-Rechner umwandelt,
- `vo_maketags`, ein Perl-Skript, das eine `vim`-Tag-Datei erzeugt.

Ein Tutorial erhält man, indem man aus dem Editor heraus das Kommando `:help vimoutliner` eingibt. Auf der noch frischen und daher stellenweise inhaltsarmen Website stehen weitere Skripte und Plugins zum Herunterladen bereit, beispielsweise um aus einer Outliner-Datei eine Präsentation im Format von OpenOffice.org Impress zu machen.

1.4 Universalgenie: Emacs

1.4.1 Schnellstart

Um den Emacs aufzurufen, gibt man einfach

```
joe@debian:~$ emacs
```

ein. Wer will, kann noch einen Dateinamen dahinter hängen. Nach spätestens ein paar Sekunden sollte sich ein Fenster öffnen - mit dem Schnellstart ist das bei Emacs so eine Sache, denn üblicherweise ruft man ihn am Anfang einer Sitzung auf und schließt ihn erst wieder, wenn man sich ausloggt. Falls Sie mit einer grafischen Benutzeroberfläche arbeiten, öffnet sich ein Fenster, dessen weitgehend selbsterklärende Menüs und Piktogramme einen schnellen Einstieg erlauben. Andernfalls öffnet man mit <ctrl>-<x> <ctrl>-<f> (zwei Tastenkombinationen hintereinander) erstmal eine Datei, falls nicht schon beim Aufruf von Emacs geschehen. Dabei vervollständigt <Tab> unvollständig eingegebene Verzeichnis- und Dateinamen, ein Entgegenkommen der Bash.

Dann kann man direkt mit dem Schreiben loslegen; `emacs` kennt keine getrennten Kommando- und Eingabemodi wie `vi`. Eine neue Zeile beginnt man mit <cr> (nach dem Cursor) oder <ctrl>-<o> (davor). Kleine Tippfehler korrigieren wir mittels Pfeiltasten und Backspace oder <Del> bzw. <Entf>, ganze Zeilen lassen sich mit <ctrl>-<k> löschen und mit <ctrl>-<y> wieder einfügen. Bei größeren Missgeschicken hilft <ctrl>-<x> <u>, welches das letzte Kommando rückgängig macht. Vermissen Sie in Ihrem Text etwas, können Sie mit <ctrl>-<s> eine Suche vorwärts und mit <ctrl>-<r> dasselbe rückwärts beginnen. Schließlich drücken wir <ctrl>-<x> <ctrl>-<s> zum Speichern des Geschriebenen und <ctrl>-<x> <ctrl>-<c>, um Emacs zu beenden. Verlässt man Emacs, ohne vorher zurückzuschreiben, wird man gefragt, ob man nicht doch abspeichern möchte.

Falls Sie sich in irgend einem Unterkommando verfangen (was früher oder später passiert), kommen Sie mit <ctrl>-<g> oder im schlimmsten Fall <Esc> <Esc> <Esc> wieder heraus. Abschließend seien die recht umfangreichen Hilfsfunktionen von Emacs erwähnt: Drückt man <ctrl>-<h> <a>, kann man einen Begriff eingeben, zu dem Emacs dann Hilfe sucht. Dabei werden Sie meistens mehrere Befehle zu sehen bekommen, die Sie eintippen können, wenn Sie vorher ein <alt>-<x> eingeben. Nach ein paar Mal wird das zu langweilig, und man fragt Emacs per <ctrl>-<h> <w>, auf welcher Taste der jeweilige Befehl zu finden ist. Wollen Sie noch mehr lernen, zeigt Ihnen <ctrl>-<h> <t> eine Einführung in den Umgang mit Emacs an, bei einer deutschen Installation auf Deutsch.

1.4.2 Übersicht

Ursprünglich nur aus einer Sammlung von Makros für einen anderen Editor (daher der Name *Editor MACroS*) bestehend, hat sich der Emacs unter der Federführung von RICHARD STALLMAN zur eierlegenden Wollmilchsau unter den Texteditoren entwickelt. Inzwischen bringt er seinen eigenen Interpreter für eine Programmiersprache mit (LISP) und besitzt spezielle Modi für alle häufiger verwendeten Textformate (insbesondere auch LaTeX). Sie können sogar Mail und News damit lesen, den `vi` emulieren oder Quellcode damit debuggen. Dementsprechend hat er nicht gerade den Ruf, besonders schnell zu laden, wovon diverse Verballhornungen des Wortes *Emacs* wie *Emacs Makes A Computer Slow* zeugen. Der große Unterschied zwischen `vi` und `emacs` besteht darin, dass letzterer nicht über einen getrennten Kommando-

und Eingabemodus verfügt, sondern die Editierkommandos über das gleichzeitige Drücken von <ctrl> oder <alt> und anderen Tasten aufgerufen werden (keycords). Das erfordert eine gewisse Fingerakrobatik, weswegen *Emacs* manchmal auch als *Escape-Meta-Alt-Control-Shift* interpretiert wird.

Der Emacs ist in mehreren Varianten erhältlich, von denen die aus dem GNU-Projekt (http://www.gnu.org/software/emacs/, Debian-Paket *emacs21*) die ursprünglichste ist. Daneben hat sich inzwischen XEmacs (http://www.xemacs.org/, Debian-Paket *xemacs21*), ehemals Lucid Emacs, etabliert. Die Entwicklung von XEmacs wurde vor allem von JAMIE ZAWINSKI und BEN WING vorangetrieben; er verfügt nur über eine grafische Oberfläche, lässt sich also nicht mehr auf einem alphanumerischen Terminal benutzen. XEmacs hat sich bei Version 19 von GNU Emacs aufgrund von Meinungsverschiedenheiten unter den Entwicklern abgespalten und bietet nach eigener Aussage mehr Funktionalität. Seither hat aber auch GNU Emacs wieder nachgelegt, in der aktuellen Version 21.4.1 (Stand Mitte 2005) muss man schon lange suchen, um größere Unterschiede festzustellen. Weiterhin gibt es Portierungen auf andere Systeme einschließlich DOS sowie die Variante Microemacs (http://uemacs.tripod.com/ und http://www.jasspa.com/, nicht bei Debian).

Wer vor dem gewaltigen Emacs erschrickt oder knapp an Speicher und Prozessorleistung ist, kann sein Glück mit *Jonathan's Own Version of Emacs* versuchen, aus den Debian-Paketen *jove* und *xjove* (X11-Frontend zu jove). Der kleine Bruder verfügt über die wesentlichen Fähigkeiten des großen. Umsteigen kann man immer.

Bei Fragen zu Emacs empfiehlt sich zu allererst das Emacs-FAQ, zu erreichen unter http://www.gnu.org/software/emacs/emacs-faq.text, oder das XEmacs-FAQ unter http://www.xemacs.org/FAQ/xemacs-faq.html. Wesentlich ausführlicher gestaltet sich das Emacs-Handbuch (http://www.gnu.org/software/emacs/manual/emacs.html). Weitere Verweise hält http://www.gnu.org/software/emacs/#Help bereit.

1.4.3 Einrichtung und Konfiguration

Unter Debian kann man wie gewohnt mit apt-get install emacs21 oder apt-get install xemacs21 das entsprechende Paket installieren. Falls der Emacs nicht fertig eingerichtet vorliegt, muss man sich selbst darum bemühen. Man holt ihn sich als Tarball per Anonymous FTP oder Web-Browser von http://ftp.informatik.rwth-aachen.de/pub/gnu/, von http://ftp.informatik.tu-muenchen.de/pub/comp/os/unix/gnu/ oder von anderen Servern und richtet das Ganze wie in Abschnitt **??** *tar-Archive* auf Seite **??** beschrieben ein. Aktuell ist Mitte 2005 die Version 21.4.1, sowohl bei GNU wie bei Debian. Unter Debian findet man in /etc/alternatives/emacs einen Hinweis, wo der Editor im Dateisystem ruht.

Die systemweite Konfiguration findet sich im Verzeichnis /etc/emacs/ oder /etc/emacs21/. XEmacs besitzt sein eigenes Verzeichnis /etc/xemacs21, lädt aber normalerweise auch die Einstellungen von GNU Emacs. In diesen Verzeichnissen sollte man mit Änderungen sparsam umgehen, hauptsächlich gehören

hier Einstellungen hin, die die Installation von Emacs selbst betreffen (Verzeichnisnamen, Terminaleinstellungen). Richtig unfreundlich verhalten sich Systemverwalter, die die Standard-Tastenbelegungen ohne Rücksicht auf die Benutzer ändern.

Die benutzerspezifischen Einstellungen lagern in der Datei `$HOME/.emacs`. Spätestens, wenn Sie diese anschauen, werden Sie zum ersten Mal mit dem LISP-Quellcode konfrontiert, mit dem sich Emacs nach Belieben konfigurieren und erweitern lässt. Da sich darüber viele Benutzer beschwert haben, erreicht man inzwischen mit dem Befehl `<alt>-<x> customize <cr>` einen Konfigurationsmodus, der die gewünschten Änderungen in LISP formuliert und wieder speichert. Noch einfacher geht das unter grafischen Oberflächen: Das *Options*-Menü kennt eine ganze Menge verschiedener Einstellungen und bietet mit *Save Options* das Zurückschreiben an.

Trotzdem seien hier ein paar Beispiele angeführt, wie `.emacs` aussehen könnte. Wir verzichten darauf, die Syntax von LISP näher zu beschreiben, da das den Rahmen des Buches sprengen würde. Zuerst müssen wir allerdings das Konzept von Haupt- und Nebenmodi (*major* bzw. *minor modes*) kurz erklären: Jedes Fenster, in dem man eine Datei bearbeitet, befindet sich in genau einem Hauptmodus. Diesen kann man mit `<alt>-<x> neuer-modus <cr>` ändern. Zur Verfügung stehen unter anderem `latex-mode` (für LaTeX-Quellen, Teil von *AUCTex*), `viper-mode` (zur `vi`-Emulation) oder `c-mode` (für C-Programme). Daneben kann man zu einem Hauptmodus einen oder mehrere Nebenmodi aktivieren, z. B. `auto-fill-mode` zum automatischen Zeilenumbruch oder `font-lock-mode` zur Syntax-Hervorhebung, vorausgesetzt, Emacs kennt das jeweilige Dateiformat.

Wenn Sie nun beispielsweise bei jedem Aufruf von Emacs die Zeile und Spalte, in der sich der Cursor befindet, in der Statuszeile angezeigt bekommen wollen, schreiben sie einfach

```
(setq line-number-mode t)

(setq column-number-mode t)
```

in Ihre `.emacs`-Datei. Andere internen Variable von Emacs lassen sich auf gleiche Art und Weise setzen. Falls Sie nur im `tex-mode` und nicht permanent die Syntax-Hervorhebung einschalten möchten, erledigt das

```
(add-hook 'tex-mode-hook (lambda () (font-lock-mode)))
```

für Sie. Auf ähnliche Weise lässt sich der automatische Zeilenumbruch ausschließlich für den Standard-Modus einschalten:

```
(add-hook 'fundamental-mode (lambda () (auto-fill-mode)))
```

Zu guter Letzt lässt sich mit

```
(global-set-key [(meta g)] (quote goto-line))
```

der Befehl `goto-line`, mit dem man in eine bestimme Zeile springt, auf die Tastenkombination `<alt>-<g>` legen. Mit anderen Tasten funktioniert das genauso, für `<ctrl>-<t>` gibt man `(control t)` anstatt `(meta g)` ein. Wenn Sie jetzt

Appetit auf mehr von dieser Sprache mit den unzähligen Klammern bekommen, bietet sich die Einführung *Programming in Emacs Lisp* (`http://www.gnu.org/software/emacs/emacs-lisp-intro/emacs-lisp-intro.html`) an.

1.4.4 Aufrufen und Beenden

Meistens ruft man Emacs einfach mit

```
joe@debian:~$ emacs
```

auf und öffnet mit `<ctrl>-<x> <ctrl>-<f>` eine Datei. Wahlweise schreibt man einen oder mehrere Dateinamen gleich hinter den Befehl:

```
joe@debian:~$ emacs dateiname1 dateiname2 ...
```

Die Dateien brauchen noch nicht zu existieren und werden dann beim ersten Sichern angelegt. Man arbeitet sowieso immer mit einer Kopie im Arbeitsspeicher, nur beim Speichern werden Daten auf dem Massenspeicher verändert. Viele Optionen beim Aufruf kennt Emacs nicht, am interessantesten ist die Möglichkeit, mittels

```
joe@debian:~$ emacs -f lisp-funktion
```

einen Emacs-Befehl sofort auszuführen, z. B. `auto-fill-mode`, um den automatischen Zeilenumbruch sofort einzuschalten.

Ein paar Worte zur Terminologie von Emacs: Mit Puffer wird die Kopie einer Datei im Arbeitsspeicher bezeichnet. Nur diese lässt sich bearbeiten. Was sie zu sehen bekommen, wenn Sie Emacs aufrufen, ist ein Fenster, das einen Puffer anzeigt. Man kann mit `<ctrl>-<x> 2` die Anzeige auch in zwei Fenster aufspalten (oder mehr, aber das wird unübersichtlich). Dann wechselt man mit `<ctrl>-<x> o` dazwischen hin und her. Hat man genug, bringt `<ctrl>-<x> 1` das Fenster, in dem sich der Cursor gerade befindet, wieder auf volle Größe. Haben Sie mehrere Dateien geöffnet, das heißt auch mehrere Puffer, kommt man mit `<ctrl>-<x> b` von einem zum anderen. Zu allem Überfluss bietet Emacs noch die Möglichkeit, innerhalb eines Aufrufs bzw. Prozesses mehrere Fenster der grafischen Benutzeroberfläche zu erzeugen, die dann Frames genannt werden. Das wird aber eher selten gebraucht.

Das Zurruckschreiben eines Puffers auf die Festplatte geschieht mittels `<ctrl>-<x> <ctrl>-<s>`, bei mehreren Dateien erledigt das `<ctrl>-<x> s` auf einmal. Abgesehen davon legt Emacs selbständig Sicherheitskopien der bearbeiteten Dateien an. Diese bekommen an den ursprünglichen Dateinamen einfach eine Tilde angehängt. Außerdem wird jeder Puffer von Zeit zu Zeit automatisch gesichert, was dann Dateinamen der Form `#foobar#` ergibt. Wünschen Sie sich beim Speichern einen anderen Namen als den aktuellen des Puffers, können Sie mit `<ctrl>-<x> <ctrl>-<w>` einen neuen wählen. Sollten Sie es sich einmal ganz anders überlegen, können Sie mit `<ctrl>-<x> k` einen Puffer löschen, auch ohne ihn zurückzuschreiben.

Am Ende des Tages lässt dann `<ctrl>-<x> <ctrl>-<c>` auch Emacs seine Ruhe finden. Wollen Sie jedoch nur einmal kurz was anderes erledigen, schickt

ihn <ctrl>-<z> in den Hintergrund. Das ist besonders nützlich beim Arbeiten auf einem alphanumerischen Terminal, da sich dort nicht so einfach wie unter einer grafischen Oberfläche zwischen verschiedenen Prozessen hin- und herschalten lässt. Aber auch dann erspart diese Methode lange Ladezeiten beim Starten von Emacs.

1.4.5 Navigieren im Text

Wie in jedem Texteditor spielt das Bewegen des Cursors eine zentrale Rolle. Natürlich erreicht man mit den Cursortasten jede beliebige Stelle in einem Text, aber sobald der Text länger als eine Seite wird, geht das langsam auf die Nerven oder führt zu einer Sehnenscheidenentzündung (oder beides). Stattdessen bietet Emacs folgende vier Tastenkombinationen an:

<ctrl>-<f> Bewege den Cursor ein Zeichen nach rechts (forward).
<ctrl>-<b> Ein Zeichen nach links (back).
<ctrl>-<n> Eine Zeile nach unten (next).
<ctrl>-<p> Eine Zeile nach oben (previous).

Leider liegen diese Kommandos auf der Tastatur nicht ganz so dicht beieinander wie beim `vi`; das ist aber immer noch ergonomischer als jedes Mal die Hand aus der üblichen Zehn-Finger-Schreibhaltung zu den Cursortasten hinüber zu bewegen. Weitere häufig benutzte Befehle sind:

<ctrl>-<a> Bewege den Cursor an den Anfang der Zeile,
<ctrl>-<e> ans Ende der Zeile,
<alt>-<f> ein Wort weiter,
<alt>-<b> ein Wort zurück,
<alt>-<a> an den Anfang des Satzes unter dem Cursor,
<alt>-<e> an das Ende des Satzes unter dem Cursor,
<alt>-<}> einen Absatz weiter (getrennt durch eine Leerzeile),
<alt>-<{> einen Absatz zurück,
<alt>-« an den Anfang des Puffers,
<alt>-» an das Ende des Puffers,
<alt>-<x> goto-line <cr> n <cr> springe zu Zeile n (bei XEmacs auch unter <alt>-<g>).

Will man weniger den Cursor bewegen als den angezeigten Textausschnitt, sind folgende Kommandos nützlich:

<ctrl>-<v> Bewege den Textausschnitt um eine Seite weiter,
<alt>-<v> dasselbe eine Seite zurück,
<ctrl>-<l> zentriere die Zeile auf dem Schirm, in der sich der Cursor befindet.

Auch die Suchfunktionen kommen bei Emacs nicht zu kurz:

<ctrl>-<s> Inkrementelle Suche vorwärts (search), das heißt Emacs sucht schon beim Tippen des Suchwortes,
<ctrl>-<r> inkrementelle Suche rückwärts (reverse search),

<ctrl>-<alt>-<s> Vorwärtssuche nach einem regulären Ausdruck, siehe Abschnitt **??** *Reguläre Ausdrücke* auf Seite **??**,
<ctrl>-<alt>-<r> Rückwärtssuche nach einem regulären Ausdruck,
<ctrl>-<s> wiederhole eine begonnene Suche in Vorwärtsrichtung,
<ctrl>-<r> wiederhole eine begonnene Suche in Rückwärtsrichtung,
<cr> höre mit der Suche auf und lasse den Cursor dort, wo er sich momentan befindet,
<ctrl>-<g> brich die Suche ab und setze den Cursor an seine ursprüngliche Position.

An dieser Stelle dürfte auffallen, dass viele Kommandos, die sich nur durch den Gebrauch von <alt> anstelle von <ctrl> unterscheiden, auch einen logischen Zusammenhang besitzen: <ctrl>-<f> bewegt den Cursor ein Zeichen weiter, <alt>-<f> ein ganzes Wort. Solche Analogien begegnen uns oft, wenn wir noch mehr Tastenkombinationen von Emacs lernen. Ähnlich besitzen manche Tastenkombinationen ein kontextabhängiges, aber immer logisches Verhalten: <ctrl>-<s> beginnt eine neue Suche, außer man befindet sich gerade schon in einer. Dann will man normalerweise nicht die eben begonnene abbrechen.

1.4.6 Text ändern

Da man bei Emacs im Gegensatz zu vi nicht erst einen Befehl eingegeben muss, um Text schreiben zu können, verfügt er naturgemäß über weniger Kommandos zum Einfügen von Text. Diejenigen zum Löschen sind aber nicht minder zahlreich:

<ctrl>-<o> Füge eine neue Zeile vor dem Cursor ein (open),
<ctrl>-<t> vertausche die beiden Zeichen vor dem Cursor,
<alt>-<t> vertausche die letzten beiden Wörter,
<Backspace> lösche das Zeichen vor dem Cursor,
<ctrl>-<d> lösche das Zeichen nach dem Cursor (delete),
<alt>-<Backspace> lösche das Wort vor dem Cursor,
<alt>-<d> lösche das Wort nach dem Cursor,
<ctrl>-<k> lösche alle Zeichen vom Cursor bis zum Zeilenende (kill). Falls Zeile leer, lösche die ganze Zeile,
<ctrl>-<x> <u> nimm letzte Textänderung zurück (undo).

Weiterhin beherrscht Emacs auch das Finden und Ersetzen von Zeichenketten. Dazu drückt man entweder <alt>-<%>, oder man gibt sogar <alt>-<x> query-replace-regexp <cr> ein, um reguläre Ausdrücke benutzen zu können. Man wird dann nach dem Suchmuster und dem einzusetzenden Text gefragt. Bei einem Treffer antwortet man entweder <y> (yes), um diese Stelle zu ersetzen, <n> (no), um fortzufahren, <!>, um alle restlichen Treffer ohne Nachfrage zu ersetzen, oder <Esc>, um die Suche abzubrechen. Außerdem erweisen sich bei größeren Mengen an zu löschendem Text die im nächsten Abschnitt besprochenen Befehle als nützlich.

1.4.7 Cut & Paste

Mit Cut & Paste bezeichnet man Funktionen zum Ausschneiden und Wiedereinfügen von Textpassagen. Das Kommando <ctrl>-<k> haben Sie schon kennen gelernt, es gibt aber etliche mehr:

<ctrl>-<space> Markierung setzen, dann den Cursor bewegen...
<ctrl>-<w> ...und den markierten Text löschen,
<alt>-<w> markierten Text in einen Zwischenspeicher kopieren, aber nicht löschen,
<ctrl>-<y> zuletzt gelöschten Text aus dem Zwischenspeicher vor dem Cursor einfügen (yank),
<alt>-<y> zuletzt eingefügten Text durch den davor gelöschten Text ersetzen,
<ctrl>-<x> <ctrl>-<x> Cursor und Markierung vertauschen,
<ctrl>-<x> h den ganzen Puffer markieren.

Die Arbeit mit diesen Befehlen ist recht einfach. Falls Sie diese Operationen schon einmal in einem anderen Texteditor ausgeführt haben, sollten sie Ihnen vertraut verkommen. Andernfalls probieren Sie sie mit einer simplen Textdatei aus.

1.4.8 Sonstiges

Die Befehle, die über obige Grundkenntnisse von Emacs hinausgehen, aber von einem der Autoren dieses Buchs häufiger benutzt werden, beinhalten:

<ctrl>-<u> zahl Führt den nächsten Befehl zahl-mal aus. Nützlich insbesondere beim Löschen von Text (etwa <ctrl>-<u> 2 <ctrl>-<k>),
<ctrl>-<x> i fragt nach einer Datei, die dann vor dem Cursor eingefügt wird,
<ctrl>-<x> <ctrl>-<b> listet alle offenen Puffer auf,
<alt>-<q> bricht die Zeilen des Textabschnitts unter dem Cursor automatisch um,
<alt>-<!> führt ein nachfolgend einzugebendes Shellkommando aus,
<alt>-<x> ispell-buffer lässt ispell über den aktuellen Puffer laufen (Rechtschreibung prüfen).

Außerdem seien drei Befehle erwähnt, um Makros aufzunehmen und abzuspielen:

<ctrl>-<x> <(> Makro-Aufnahme beginnen.
<ctrl>-<x> <)> Makro-Aufnahme beenden.
<ctrl>-<x> <e> zuletzt aufgenommenes Makro ausführen (execute).

Jetzt kennen Sie erst einen Bruchteil der in Emacs verfügbaren Kommandos. Nächste Anlaufstellen bei der Suche nach Hilfe bieten Emacs selbst (unter <ctrl>-<h> <ctrl>-<h>) und das Internet (http://www.gnu.org/software/emacs/ sowie http://www.xemacs.org). Es gibt nichts, was es nicht gibt; angefangen beim *Emacs Psychiatrist* im Help-Menü von GNU Emacs über Erweiterungen für exotische Alphabete bis hin zu Emacspeak, einer Sprachausgabe für sehbehinderte Benutzer, siehe Abschnitt **??** *Screenreader* auf Seite **??**.

Falls Sie wider Erwarten etwas nicht finden sollten, haben Sie immer noch die Möglichkeit, Ihre eigene Erweiterung zu schreiben. Der eingebaute LISP-Interpreter steht anderen Programmiersprachen in nichts nach.

Ob Ihnen nun `emacs` oder `vi` besser gefällt, bleibt Ihnen überlassen. Nachdem die Glaubenskriege zwischen den Anhängern der beiden Editoren abgeklungen sind, mehren sich die Stimmen, beide hätten ihre Vorzüge und Nachteile. Falls wir den Eindruck erweckt haben, man könne mit `emacs` mehr bewerkstelligen, lassen Sie sich nicht täuschen: Auch der `vim` lässt sich mittlerweile vollständig programmieren.

1.5 Weitere Texteditoren

1.5.1 Einfachst: nano

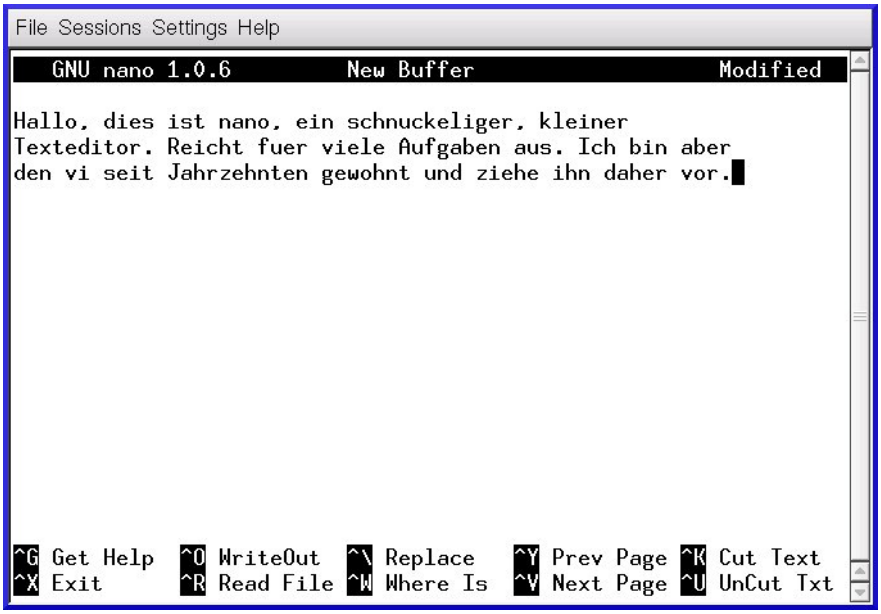

Abb. 1.2: Screenshot des Editors nano

`pico` ist ein kleiner Editor, der usprünglich zu einem Email-Programm gehörte, aber auch selbständig zu gebrauchen ist. Wegen Lizenz-Einschränkungen steht unter Debian nur der Klon `nano` zur Verfügung, ebenso eine abgemagerte Version `nano-tiny`. Wer nur einfache, kurze Texte schreibt, kommt mit ihm aus.

1.5.2 Joe's Own Editor (joe)

Der `joe`[6] von JOSEPH H. ALLEN soll als Beispiel für eine Vielzahl von Editoren stehen, die im Netz herumschwimmen und entweder mehr können oder einfacher zu benutzen sind als die Standard-Editoren. Er ist nicht auf X11 angewiesen und bringt eine eigene Verhaltensweise in normaler und beschränkter Fassung mit, kann aber auch WordStar, `pico` oder den Emacs emulieren (nachahmen), je nach Aufruf und Konfiguration. Diese lässt sich in eine Datei `$HOME/.joerc` den eigenen Wünschen anpassen.

Der `joe` kennt keine Modi. Nach dem Aufruf legt man gleich mit der Texteingabe los. Editorkommandos werden durch control-Sequenzen gekennzeichnet. Beispielsweise erzeugt die Folge `<ctrl>-<k>` und `<h>` ein Hilfefenster am oberen Bildschirmrand. Nochmalige Eingabe der Sequenz löscht das Fenster. Am Ende verlässt man den Editor mittels `<ctrl>-<c>` ohne Zurückschreiben oder mit der Sequenz `<ctrl>-<k>` und `<x>` unter Speichern des Textes. Weitere Kommandos im Hilfefenster oder im Manual.

1.5.3 Der Nirwana-Editor (nedit)

Der `nedit` setzt auf X11 auf und verwendet eine grafische Oberfläche im Stil von Motif. Er ist in vielen Linux-Distributionen enthalten und für weitere UNIXe sowie VMS zu haben. Insbesondere lässt er sich an die Eigenheiten vieler Programmiersprachen anpassen; seine Makro-Sprache ähnelt C. Wer viel programmiert, sollte sich ihn ansehen. Informationen findet man unter `http://nedit.org/`.

1.5.4 Ein Editor für KDE (kate)

Der KDE Advanced Text Editor `kate` ist für alle Arten von Text einsetzbar, beweist jedoch Stärken beim Schreiben von Quellcode wie die Hervorhebung von Syntaxelementen oder das Kollabieren/Expandieren von Funktionen. Er hat mehrere Dokumente gleichzeitig im Griff und lässt sich durch Plugins erweitern. Mit der Option `--help-all` aufgerufen zeigt er alle Optionen an. Vermutlich wird er den KDE-Editor `kwrite` ablösen. Näheres bei `http://kate.kde.org/`.

1.5.5 Ein Editor für GNOME (gedit)

Der aus dem GNOME-Projekt stammende Editor `gedit` hinterließ einen guten Eindruck. Die Konfiguration kann im laufenden Betrieb geändert werden (beispielsweise Schriftgröße und -art). Alle wesentlichen Funktionen eines Editors sind einfach zu erreichen. Auch unter KDE treten keine Probleme auf. Wenn man ohnehin mit einer grafischen Benutzeroberfläche und X11 arbeitet, ist `gedit` eine gute Wahl.

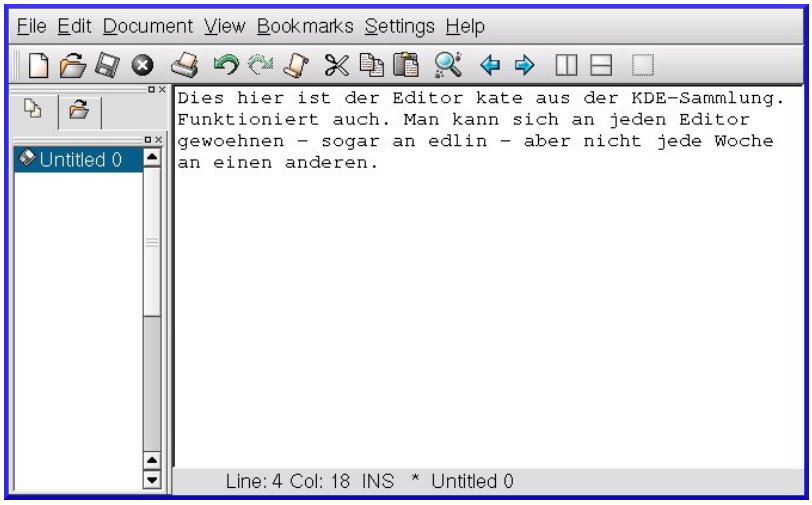

Abb. 1.3: Screenshot des KDE-Editors `kate`

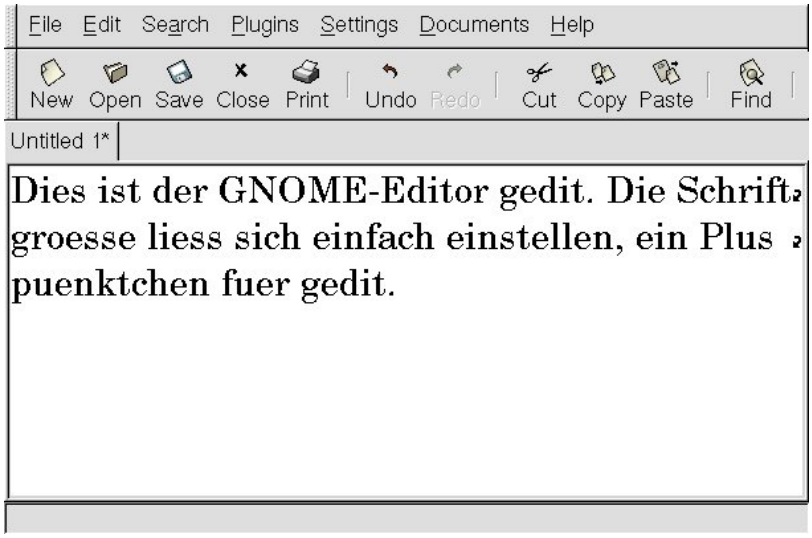

Abb. 1.4: Screenshot des GNOME-Editors `gedit`

1.5.6 aXe – ein einfacher Editor unter X11

Der Editor `axe` ist eine Weiterentwicklung des Editors `xedit`. Er setzt X11 voraus und macht vom X Athena Widget Set Gebrauch.

[6] Leider hat sich ein HTML-Editor aus Frankreich denselben Namen zugelegt.

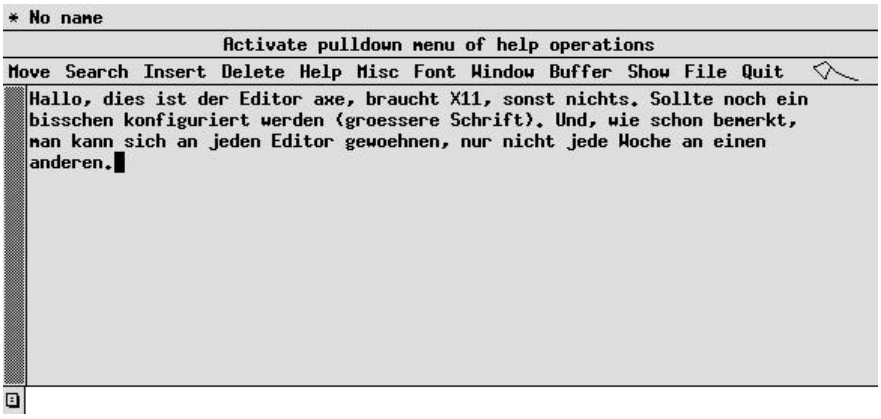

Abb. 1.5: Screenshot des X11-Editors `axe`

Das waren noch längst nicht alle Editoren aus der Debian-Distribution. Außerhalb von Debian finden sich zahllose weitere Texteditoren. Falls der geneigte Leser eine besondere Perle entdeckt, sind wir für eine diesbezügliche Mail dankbar.

1.5.7 jEdit und jed – für Programmierer

Für Programmierer interessant ist der Editor `jedit`, der 130 Programmiersprachen (im weitesten Sinn) und ihre Gepflogenheiten kennt. Er setzt Java 2 voraus und ist nicht bei Debian, sondern samt einem *User's Guide* bei `http://www.jedit.org/` zu haben. Bei Sourceforge liegt ein Debian-Paket, das von `apt-get` erfasst werden kann. Unter anderem beherrscht der Editor Unicode und UTF-8 und bringt eine Vielzahl von Makros und Plugins mit, die sich durch Eigenerzeugnisse ergänzen lassen. Er ist auch für andere Betriebssysteme verfügbar, was in einer heterogenen Umgebung die Zusammenarbeit erleichtert.

Eine Alternative ist der Editor `jed` bzw. `xjed`, der ebenfalls viele beim Programmschreiben angenehme Fähigkeiten besitzt. Bei Debian in sechs Pakete verschnürt zu haben.

1.5.8 Editor für das Rich Text Format (ted)

Unter DOS rief `ted` einen *Tiny Editor* von TOM KIHLKENS auf, 3 kB groß, was sogar damals wenig war. Er kannte zehn Kommandos, die auf den zehn Funktionstasten lagen. Natürlich konnte das Editörle nicht die *Türme von Hanoi* spielen wie der Emacs von heute, aber zum Schreiben reichte er. Friede seiner Asche. Das Debian-Paket *ted-common* in Verbindung mit *ted* oder *ted-gtk* richtet einen Editor mit grafischer Oberfläche ein, der speziell das Rich Text Format (RTF) unterstützt. Beide Varianten brauchen X11, die erste, als stabil bezeichnete dazu die LessTif-Bibliothek, die zweite, weniger stabile GTK+. Von `ted` erzeugte RTF-Dokumente sollten von jedem MS-Windows-Programm akzeptiert werden, umgekehrt kann es bei selteneren

Formatanweisungen zu Schwierigkeiten kommen. `ted` ignoriert, was er nicht versteht, sodass der Inhalt eines Dokumentes auf jeden Fall zu erkennen ist. Der Editor wandelt RTF nach PostScript oder PDF um und lässt sich als MIME-Handler für RTF verwenden. Näheres auf seiner Webseite `http://www.nllgg.nl/Ted/`, wo auch eine deutsche Fassung als Tarball heruntergeladen werden kann.

1.6 WYSIWYG-Textprozessoren

1.6.1 GNOME AbiWord

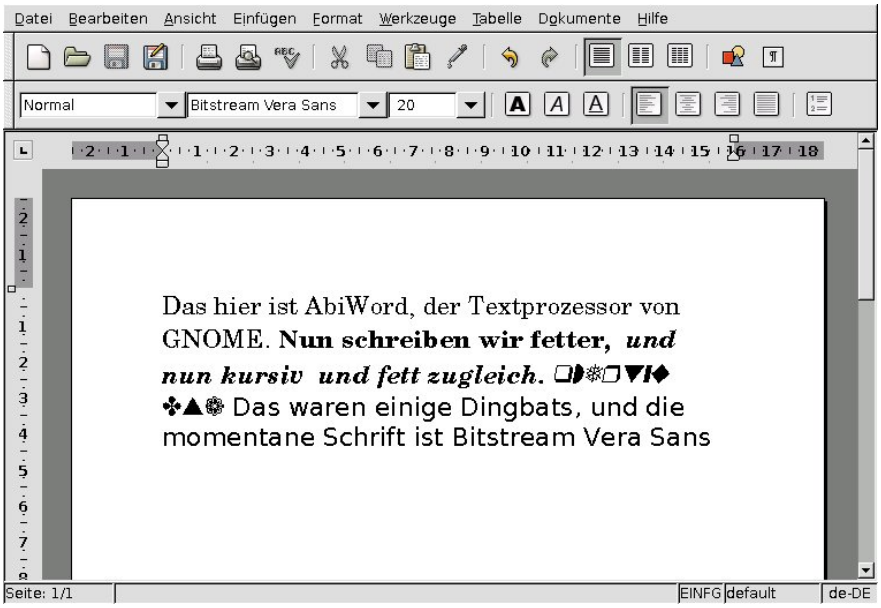

Abb. 1.6: Screenshot des WYSIWYG-Textprozessors AbiWord aus dem GNOME-Office-Paket. Wenn es denn unbedingt WYSIWYG sein soll, keine schlechte Wahl. Woher der Name stammt, wissen die Götter; das Abitur ist nicht zu seinem Gebrauch erforderlich.

Wir kommen nun zu den WYSIWYG-Editoren oder Textprozessoren, die eine bei der Eingabe vorgenommene Formatierung sogleich auf dem Bildschirm wiedergeben. Das ist beim Schreiben von Briefen praktisch, hat aber Grenzen bei anspruchsvollen Texten. Deshalb findet man derartige Editoren vor allem in Office-Paketen, siehe Abschnitt **??** *Office-Pakete* auf Seite **??**. Die Prozessoren lassen sich aber auch einzeln verwenden.

Zum GNOME-Projekt gehört AbiWord (`http://www.abiword.org/`), ein Textprozessor, der alle Funktionen mitbringt, die man im Alltag braucht. Die Beschränkung auf das Wesentliche trägt zur Geschwindigkeit bei. Seine Oberfläche

erinnert an den Marktführer, siehe Abbildung 1.6. Ein Schönheitsfehler ist, dass die Menüs teils deutsch, teils englisch daherkommen. AbiWord ist eine gute Wahl, wenn man nur schreiben will und kein volles Office-Paket braucht. Die Herkunft des Namens ließ sich nicht klären.

1.6.2 KDE KWord

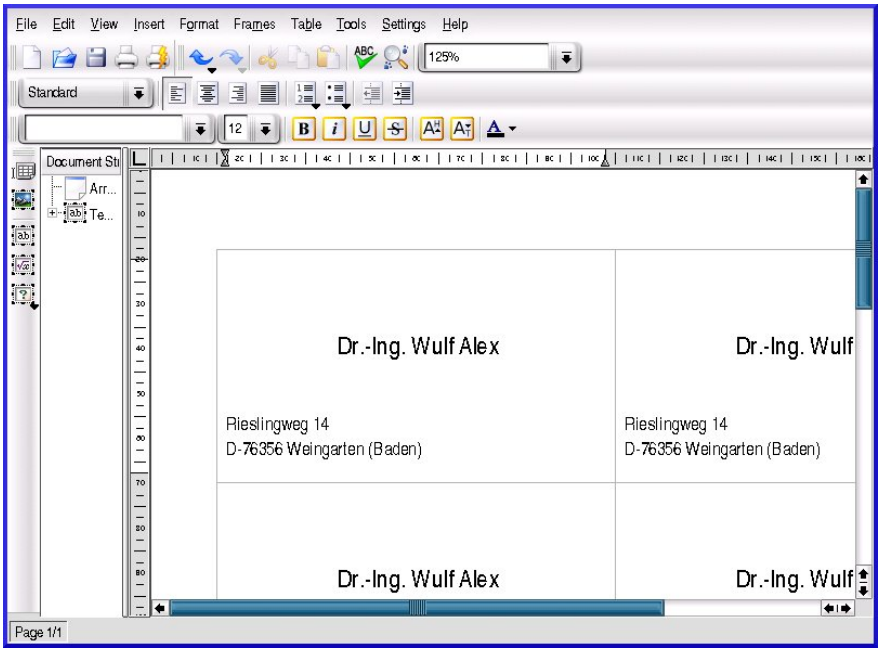

Abb. 1.7: Screenshot des WYSIWYG-Textprozessors `kword` aus dem KDE-Office-Paket beim Gestalten von Visitenkarten

Der Textprozessor KWord aus dem KDE-Office-Paket wartet mit den meisten Features und Buttons auf, siehe Abbildung 1.7, die ihn beim Erstellen von Visitenkarten zeigt. Er fügt sich reibungslos in den KDE-Desktop ein. Sein Look + Feel weicht stärker vom Marktführer ab, was als Vorteil empfunden werden kann.

1.6.3 Open.org writer

Der `oowriter` ist das Textwerkzeug von OpenOffice.org. Man kann es unter seinem Namen aufrufen oder mit `ooffice`. Die Namen aller Einzelwerkzeuge von OpenOffice.org sind Symlinks auf `ooffice`. Öffnet man eine Textdatei oder wählt bei einer Neuanlage den Menüpunkt *Textdokument* aus, meldet sich der Textprozessor. Die Arbeitsweise unterscheidet sich nicht von der anderer Textprozessoren und

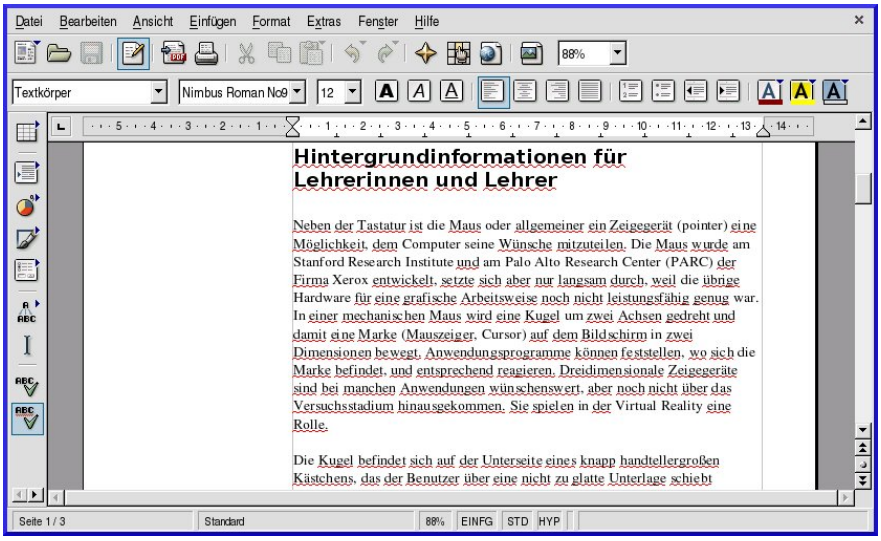

Abb. 1.8: Screenshot des WYSIWYG-Textprozessors `oowriter` aus dem OpenOffice.org-Paket beim Abfassen eines Artikels

ist relativ gut zu verstehen. Das originale Format für Textdateien trägt die Kennung `sxw` und ist ein komprimiertes Archiv von XML-Dateien; die Texte lassen sich aber auch als `pdf`-Datei exportieren oder als Word-, RTF-, ASCII- oder HTML-Datei abspeichern. Abbildung 1.8 vermittelt einen Eindruck vom Aussehen des Bildschirms.

1.6.4 LyX

Der Textprozessor LyX ist ein WYSIWYG-Frontend für LaTeX, so weit das möglich ist. Benutzern, die bisher mit Textprozessoren gearbeitet haben, wird damit der Wechsel zu LaTeX erleichtert. Die Webseite des Werkzeugs liegt unter `http://www.lyx.org/`. LyX kann bestehende LaTeX-Dateien importieren, hat aber bei komplexen Dokumentstrukturen Probleme. Am besten entscheidet man sich zu Beginn eines Projektes für oder gegen LyX. Wer sich mit LaTeX auskennt und einen der Texteditoren beherrscht, gewinnt mit LyX nichts dazu. LyX ist nicht mit dem textorientierten Web-Browser `lynx` zu verwechseln.

1.7 Hex-Editoren (bvi, hexedit, hexcurse, vche)

Text-Editoren fassen ihnen vorgelegte Dateien immer als Text auf. Es gibt aber Aufgaben, bei denen eine Datei Byte für Byte ohne jegliche Interpretation angezeigt und editiert werden soll. Das Werkzeug zu diesem Zweck sind Binär-Editoren. Die meisten zeigen einen Dateiinhalt wahlweise binär, oktal, dezimal, hexadezimal oder soweit möglich als ASCII-Zeichen an, daher auch die Bezeichnung Hex-Editor.

Die beiden Text-Editoren `vim` und `emacs` kennen einen Binär-Modus, Aufruf `vim -b` bzw. `Meta-x` im Emacs. Spezielle Binär-Editoren sind jedoch besser an die Aufgabe angepasst. Neben kommerziellen Produkten wie `vedit` oder `ultraedit` finden sich im Netz auch freie Binär-Editoren in Form des Quellcodes oder als Debian-Paket:

- `bvi`, ein binärer `vi`,
- `hexedit`, äußerlich dem `bvi` ähnlich, aber mit anderen Kommandos,
- `khexit` aus der KDE-Arbeitsumgebung, mit einer netten grafischen Oberfläche, zu finden unter Utilities – Binary Editor,
- `shed`, der *Simple Hex Editor*, siehe `http://shed.sourceforge.net/`,
- `lfhex`, der *Large File Hex Editor*, siehe `http://www.freshports.org/editors/lfhex/`,
- `vche`, der *Virtual Console Hex Editor*, siehe `http://www.grigna.com/diego/linux/vche/`,
- `bed`, der *Menu Driven Binary Editor*, siehe `http://bedlinux.tripod.com/`.

Ihrer Aufgabe gemäß können Binär-Editoren mit extrem großen Dateien umgehen, wo Text-Editoren versagen. Sie arbeiten nicht auf einer Kopie der gesamten Datei, sondern nur auf Ausschnitten.

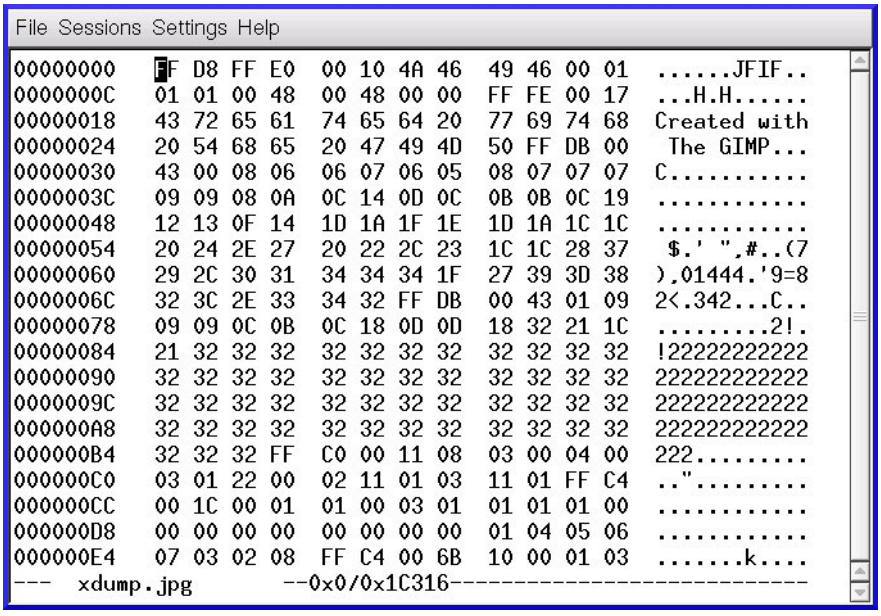

Abb. 1.9: Screenshot des Hex-Editors `hexedit` beim Anschauen einer jpg-Datei

In Abbildung 1.9 sehen wir einen Screenshot des Hex-Editors `hexedit` bei der Untersuchung des Anfangs einer jpg-Datei. Der Schirm des `bvi`-Hex-Editors

sieht genauso aus. Links stehen die hexadezimalen Adressen des ersten Bytes der jeweiligen Zeile, in der Mitte die Bytes in hexadezimaler Schreibweise – pro Byte ein Zeichenpaar von 0 bis F – und rechts deren Übersetzung in ASCII-Zeichen, soweit möglich. Einzelne Bytes lassen sich editieren, vorzugsweise durch Ersetzen. Ohne tieferes Verständnis der Datei kann man sie aber auch leicht ruinieren.

Der Hex-Editor `vche` (Virtual Console Hex Editor) schreibt direkt auf eine virtuelle Konsole (`/dev/vcsa1` oder ähnlich) und läuft daher in seiner Grundversion nicht unter X11 oder über ein Netz (remote). Ein Benutzer braucht Zugriffsrechte auf `/dev/vcsa1`, die er nicht von vornherein hat. Der Editor zeigt auch nicht druckbare Zeichen an und kann Gerätedateien bearbeiten.

1.8 Editoren für fremde Schriften (geresh, katoob, yudit)

Die großen Editoren lernen allmählich, mit UTF-8 und bidirektionalen Texten – zum Beispiel Englisch und Hebräisch gemischt – umzugehen. Es gibt aber schon länger spezielle Editoren für diese Aufgabe. Aus Israel stammt `geresh` mit der Startseite `http://www.typo.co.il/~mooffie/geresh/`. Der Editor arbeitet zeichenbasiert ohne X11, kennt Unicode und unterstützt in erster Linie Hebräisch und Arabisch. Der Begleiter `pgeresh` ist ein Pager, ein Leseprogramm mit denselben Sprachkenntnissen. Die Manualseiten beschränken sich auf das Nötigste; Hilfe bieten die Werkzeuge beim Aufruf mit der Option `--help` oder indem man bei der Arbeit mit der Taste `<F9>` oder `<F10>` in die Menüleiste springt, dort per Pfeiltaste den gewünschten Punkt auswählt und mit `<cr>` bestätigt. Verlassen der meisten Punkte oder des Editors mit `<alt>-<x>`. In `/usr/share/doc/geresh/` findet man ein User Manual, leider nur auf Hebräisch. Abbildung 1.10 zeigt den Schirm mit der Auswahl eines Menüpunktes.

Aus dem arabischen Lager (`http://www.arabeyes.org/`) stammt der Editor `katoob`, der X11 und GTK+ benötigt und ebenfalls mehrere Sprachen und Schriften beherrscht. In die Menüs gelangt man mit der Maus, die Menüsprache ist Deutsch. Als Tastaturen werden US-ASCII, Arabisch und Hebräisch angeboten. Die Manualseite ist kurz, unter dem Menüpunkt *Hilfe* erscheint auch nicht direkt ein Lehrbuch für Arabisch. Der Editor lässt sich als Filter zum Umcodieren von Textdateien gebrauchen. Das User Manual zu `geresh` wird problemlos in hebräischer Schrift mit lateinischen Einlagen angezeigt.

Der Editor `yudit` ist speziell dafür gebaut, unter X11 mit verschiedenen Zeichensätzen zurecht zu kommen, darunter UTF-7 und UTF-8 (`http://www.yudit.org/`). Das Werkzeug eignet sich ebenfalls zum Umcodieren gegebener Texte. Der Editor legt beim ersten Verlassen ein Verzeichnis `.yudit` im Home-Verzeichnis des Benutzers an und darin eine Datei `yudit.properties` mit der Konfiguration.

Für slawische Sprachen und kyrillische Schriften findet man ausführliche Informationen unter `http://www.slovo.info/`, einer Website von CHRISTOPH SINGER, die leider nicht mehr gepflegt wird, jüngster Stand 2000.

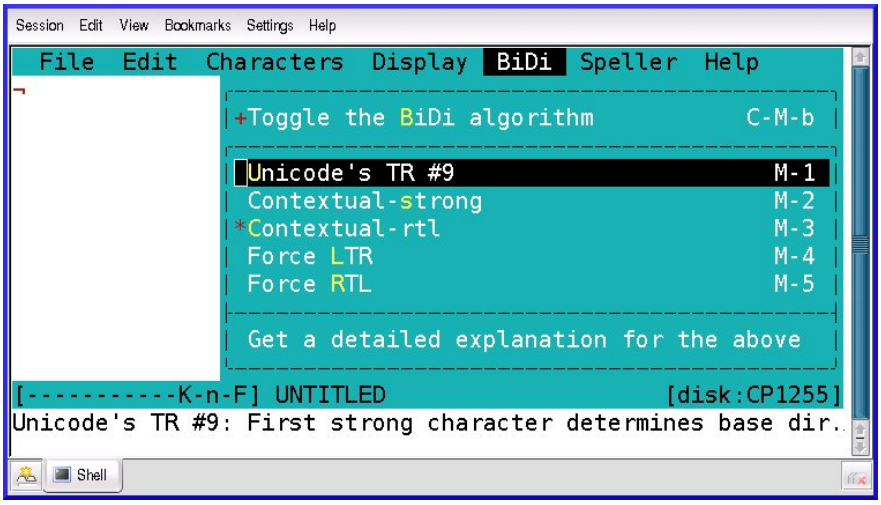

Abb. 1.10: Screenshot des für Hebräisch und Arabisch geeigneten Editors `geresh`

1.9 Streaming Editor (sed)

Der Streaming Editor `sed` bearbeitet eine Textdatei zeilenweise nach Regeln, die man ihm als Option oder in einer getrennten Datei (sed-Skript) mitgibt. Er ist im Gegensatz zu den bisher genannten Editoren nicht interaktiv, er führt keinen Dialog. Die letzte Zeile der zu bearbeitenden Textdatei muss leer sein oder anders gesagt, das letzte Zeichen des Textes muss ein newline-Zeichen (Linefeed) sein.

Die einfachste Aufgabe für den `sed` wäre der Ersatz eines bestimmten Zeichens im Text durch ein anderes (dafür gibt es allerdings ein besseres, weil einfacheres Werkzeug `tr`). Der `sed` bewältigt ziemlich komplexe Aufgaben, daher ist seine Syntax umfangreich. Sie baut auf der Syntax des alten Zeileneditors `ed` auf. Der Aufruf

```
joe@debian:~$ sed 'Kommandos' datei
```

veranlasst den `sed`, die Datei `datei` Zeile für Zeile einzulesen und gemäß den Kommandos bearbeitet nach `stdout` auszugeben. Der Aufruf

```
joe@debian:~$ sed '1d' datei
```

löscht die erste Zeile in der Datei `datei` und schreibt das Ergebnis nach `stdout`. Die Quotes um das `sed`-Kommando verhindern, dass die Shell sich das für den `sed` bestimmte Kommando ansieht und möglicherweise Metazeichen interpretiert. Hier wären sie nicht nötig und stehen einfach aus Gewohnheit. Jokerzeichen in `datei` dagegen werden von der Shell zu Recht interpretiert, sodass der `sed` von der Shell eine Liste gültiger Namen erhält.

Folgender einzeiliger Aufruf ersetzt alle Großbuchstaben durch die entsprechenden Kleinbuchstaben:

Abb. 1.11: Screenshot (Ausschnitt) des für fremde Schriften geeigneten bidirektionalen Editors `yudit`

```
joe@debian:~$ sed 'y/ABCDEFGHIJKLMNOPQRSTUVWXYZ/
abcdefghijklmnopqrstuvwxyz/' datei
```

Das y-Kommando kennt keine Zeichenbereiche, wie sie bei regulären Ausdrücken oder beim Kommando `tr` erlaubt sind, man muss die beiden notwendigerweise gleichlangen Zeichenmengen auflisten. Übrigens ist obiges Kommando ein Weg zur ROT13-Verschlüsselung, indem man die zweite Zeichenmenge mit `n` beginnen lässt. Geht es um den Ersatz eines festen Zeichenmusters oder eines regulären Ausdrucks durch eine feste Zeichenkette, so nimmt man:

```
joe@debian:~$ sed 's/\[a-z][a-z]*{..*}/LaTeX-K/g' datei
```

Im Kommando steht `s` für *substitute*. Dann folgt ein regulärer Ausdruck zur Kennzeichnung dessen, was ersetzt werden soll, hier das Muster eines LaTeX-Kommandos. An dritter Stelle ist der Ersatz (replacement) aufgeführt, hier die feste Zeichenfolge `LaTeX-K`, und schließlich ein Flag, das besagt, den Ersatz global (überall, nicht nur beim ersten Auftreten des regulären Ausdrucks in der Zeile) auszuführen. Das Trennzeichen, der Schrägstrich, zwischen den vier Teilen kann jedes beliebige Zeichen sein, es darf nur nicht in den Teilen selbst vorkommen.

Der `sed` ist mächtig und nützlich, aber lernbedürftig. Erleichtert wird die Arbeit dadurch, dass man ein sed-Skript oder auch die Anweisung in der Kommandozeile Schritt für Schritt aufbauen und testen kann. Im Netz findet sich viel Material dazu, lassen Sie eine Suchmaschine nach `unix AND editor AND sed` suchen.

Merke: Der `vi` oder der Emacs ist ein interaktiver Editor, der Tastatureingaben erfordert und nicht Bestandteil einer Pipe sein oder im Hintergrund laufen kann. Der

sed ist ein Filter, das keine Tastatureingaben verlangt, Glied einer Pipe oder eines Shellskripts sein und unbeaufsichtigt laufen kann.

1.10 Listenbearbeitung (awk)

Das Werkzeug awk ist nach seinen Urhebern ALFRED V. AHO, PETER J. WEINBERGER und BRIAN W. KERNIGHAN benannt und firmiert als programmierbares Filter oder Listengenerator. Es lässt sich auch als eine einfache Programmiersprache für einen bestimmten, engen Zweck auffassen. Der awk bearbeitet eine Textdatei zeilenweise, wobei er jede Zeile – auch Satz genannt – in Felder zerlegt. Eine typische Aufgabe ist die Bearbeitung von Listen. Hier ist er angenehmer als der sed, allerdings auch langsamer. Für die Verwaltung eines kleinen Vereins ist er recht, für das Telefonbuch von Berlin nicht. Unter Debian steht der originale awk im Paket *original-awk* zur Verfügung. In der Regel wird man aber die leistungsfähigere GNU-Version gawk vorziehen.

In einfachen Fällen werden dem awk beim Aufruf die Befehle zusammen mit den Namen der zu bearbeitenden Dateien mitgegeben, die Befehle in Hochkommas, um sie vor der Shell zu schützen:

```
joe@debian:~$ awk 'befehle' dateien
```

Ein awk-Befehl besteht aus den Teilen *Muster* und *Aktion*. Jede Eingabezeile, auf die das Muster zutrifft, wird entsprechend der Aktion behandelt. Die Ausgabe geht auf stdout. Ein Beispiel:

```
joe@debian:~$ awk '{if (NR < 8) print $0}' datei
```

Die Datei datei wird Zeile für Zeile gelesen. Die vorgegebene awk-Variable NR ist die Zeilennummer, beginnend mit 1. $0 ist die ganze jeweilige Zeile. Falls die Zeilennummer kleiner als 8 ist, wird die Zeile nach stdout geschrieben. Es werden also die ersten 7 Zeilen der Datei ausgegeben. Nun wollen wir das letzte Feld der letzten Zeile ausgeben:

```
joe@debian:~$ awk 'END {print $NF}' datei
```

Das Muster END trifft zu, wenn die letzte Zeile eingelesen ist. Üblicherweise gilt die zugehörige Aktion irgendwelchen Abschlussarbeiten. Die Variable NF enthält die Anzahl der Felder der Zeile, die Variable $NF ist das letzte Feld. Nun wird es etwas anspruchsvoller:

```
joe@debian:~$ awk '$1 != prev { print; prev = $1 }'
wortliste
```

Die Datei wortliste enthalte in alphabetischer Folge Wörter und gegebenenfalls weitere Bemerkungen zu den Wörtern, pro Wort eine Zeile. Der awk liest die Datei zeilenweise und spaltet jede Zeile in durch Leerzeichen oder Tabs getrennte Felder auf. Die Variable $1 enthält das erste Feld, also hier das Wort zu Zeilenbeginn. Falls dieses Wort von dem ersten Wort der vorangegangenen Zeile abweicht (Variable

prev), wird die ganze Zeile ausgegeben und das augenblickliche Wort in die Variable prev gestellt. Zeilen, die im ersten Feld übereinstimmen, werden nur einmal ausgegeben. Dieser awk-Aufruf hat eine ähnliche Funktion wie das Linux/UNIX-Kommando uniq. Da Variable mit dem Nullzeichenkette initialisiert werden, wird auch die erste Zeile richtig bearbeitet.

Wenn die Anweisungen an den awk umfangreicher werden, schreibt man sie in eine eigene Datei (awk-Skript). Der Aufruf sieht dann so aus:

```
joe@debian:~$ awk -f awkscript textdateien
```

awk-Skripte werden in einer Sprache geschrieben, die teils an Shellskripte, teils an C-Programme erinnert. Sie bestehen – wie ein deutscher Schulaufsatz – aus Einleitung, Hauptteil und Schluss. Sehen wir uns ein Beispiel an, das mehrfache Eintragungen von Stichwörtern in einem Sachregister aussortiert und die zugehörigen Seitenzahlen der ersten Eintragung zuordnet:

```
# awk-Script fuer Sachregister

BEGIN { ORS = ""
        print "Sachregister"
      }
      {
        if ($1 == altwort)
                print ", " $NF
        else
                {
                print "\n" $0
                altwort = $1
                nor++
                }
      }
END   { print "\n\n"
        print "gelesen: " NR "  geschrieben: " nor "\n"
      }
```

Quelle 1.1 : awk-Skript für Sachregister

Das Doppelkreuz markiert eine Kommentarzeile. Der Einleitungsblock wird mit BEGIN gekennzeichnet, der Hauptteil steht nur in geschweiften Klammern, und der Schluss beginnt mit END. Die vorbestimmte, awk-eigene Variable ORS (Output Record Separator, das heißt Trennzeichen zwischen Sätzen in der Ausgabe), standardmäßig das Newline-Zeichen, wird mit dem Nullstring initialisiert. Dann wird die Überschrift *Sachregister* ausgegeben.

Im Hauptteil wird das aktuelle erste Feld gegen die Variable altwort geprüft. Bei Übereinstimmung werden ein Komma, ein Leerzeichen und das letzte Feld der aktuellen Zeile ausgegeben, nämlich die Seitenzahl. Die awk-eigene Variable NF enthält die Anzahl der Felder des aktuellen Satzes, die Variable $NF mithin das letzte Feld.

Bei Nichtübereinstimmung (einem neuen Stichwort also) werden ein Newline-Zeichen und dann die ganze Zeile (`$0`) ausgegeben. Anschließend werden das erste Feld in die Variable `altwort` gestellt und die vom Programmierer definierte Variable `nor` um 1 inkrementiert. So wird mit der ganzen Textdatei verfahren.

Am Ende der Textdatei angelangt, werden noch zwei Newline-Zeichen, der Wert der `awk`-eigenen Variablen `NR` (Number of Records) und der Wert der Variablen `nor` ausgegeben. Die Aufgabe wäre auch mit dem `sed` oder einem C-Programm zu lösen, aber ein `awk`-Skript ist der einfachste Weg. Das Werkzeug vermag noch viel mehr.

Eine Besonderheit des `awk` sind Vektoren mit Inhaltindizierung (E: associative array). In Programmiersprachen wie C oder FORTRAN werden die Elemente eines Arrays oder Vektors mit fortlaufenden ganzen Zahlen (Indizes) bezeichnet. Auf ein bestimmtes Element wird mittels des Arraynamens und des Index zugegriffen:

```
arrayname[13]
```

In einem `awk`-Array dürfen die Indizes nicht nur ganze Zahlen, sondern auch beliebige Zeichenketten sein:

```
telefon["Meyer"]
```

ist eine gültige Bezeichnung eines Elementes. Es könnte die Anzahl der Telefonanschlüsse namens Meyer in einem Telefonbuch enthalten.

Eine Alternative zu `awk` ist Perl, eine interpretierte Programmiersprache zur Verarbeitung von Textdateien, die Elemente aus C, `sed`, `awk` und der Shell `bash` enthält. Ihre Möglichkeiten gehen über das Verarbeiten von Texten hinaus in Richtung Shellskripte.

1.11 Formatieren

1.11.1 Inhalt, Struktur und Aufmachung

Ein Schriftstück – sei es Brief oder Buch – hat einen Inhalt, nämlich Text, gegebenenfalls auch Abbildungen, der in einer bestimmten Form dargestellt ist. Bei der Form unterscheiden wir zwischen der logischen Struktur und ihrer Darstellung auf Papier oder Bildschirm, Aufmachung oder Layout genannt. Beim Schreiben des Manuskripts macht sich der Autor Gedanken über Inhalt und logische Struktur, aber kaum über Schrifttypen und Schriftgrößen, den Satzspiegel, den Seitenumbruch, die Nummerierung der Abbildungen. Das ist Aufgabe des Metteurs oder Layouters im Verlag, der seinerseits möglichst wenig an Inhalt und Struktur ändert. *Schreiben* und *Setzen* sind unterschiedliche Aufgaben, die unterschiedliche Kenntnisse und Werkzeuge erfordern. Wer wissen will, was außer den Gedanken des Verfassers hinter einem guten Buch steckt, findet bei JAN TSCHICHOLD Rat.

Der Rechner wird als Werkzeug für alle drei Aufgaben (Inhalt, Struktur, Layout) eingesetzt. Mit einem Editor schreibt man einen strukturierten Text, weitergehende

Programme prüfen die Rechtschreibung, helfen beim Erstellen eines Sachverzeichnisses, analysieren den Stil. Ein Satz- oder Formatierprogramm erledigt den Zeilen- und Seitenumbruch, sorgt für die Nummerierung der Abschnitte, Seiten, Abbildungen, Tabellen, Fußnoten und Formeln, legt die Schriftgrößen fest, ordnet die Abbildungen in den Text ein, stellt das Inhaltsverzeichnis zusammen usw. Ein Satzprogramm wie `groff` oder LaTeX erfüllt hohe Ansprüche und besteht daher aus einem ganzen Softwarepaket. Ein kommerzielles Produkt mit gleichem Ziel ist FrameMaker. Der Textprozessor KWord aus der KDE Office Suite hat Ähnlichkeiten mit FrameMaker und ist bei einem Wechsel das Werkzeug der Wahl.

Der Linux/UNIX-Formatierer `nroff` bzw. `groff` und das Satzprogramm LaTeX (TeX, LaTeX, pdftex etc.) halten Inhalt, Struktur und Layout auseinander. Man schreibt mit einem beliebigen Editor den Text samt Strukturhinweisen und formatiert anschließend. LaTeX verfolgt darüber hinaus den Gedanken, dass der Autor seine Objekte logisch beschreiben und von der typografischen Gestaltung möglichst die Finger lassen soll. Der Autor soll sagen: *Jetzt kommt eine Kapitelüberschrift* oder *Jetzt folgt eine Fußnote*. LaTeX legt dann nach typografischen Regeln die Gestaltung fest. Man kann darüber streiten, ob die Regeln das Nonplusultra der Schwarzen Kunst sind, ihr Ergebnis ist jedenfalls besser als vieles, was Laien einfällt.

Sowohl `nroff`/`groff` wie LaTeX zielen auf die Wiedergabe der Dokumente mittels Drucker auf Papier ab. Mit Hilfe von Programmen wie `gv` oder `xdvi` lassen sich die Ergebnisse auf dem Bildschirm vor dem Ausdrucken beurteilen. Die Hypertext Markup Language HTML, deren Anwendung wir im World Wide Web begegnen, hat viel mit LaTeX gemeinsam, eignet sich jedoch in erster Linie für Dokumente, die auf dem Bildschirm dargestellt werden sollen. Auch sie trennt Inhalt, Struktur und Layout. Leider begreifen das manche Web-Autoren nicht.

Beim Arbeiten mit Formatierern wie LaTeX – gegebenenfalls in Verbindung mit einem Versionskontrollsystem wie CVS – kommen die Schwierigkeiten am Anfang, wenn man eine Reihe von Konzepten und Kommandos lernen muss, wenige Vorlagen hat und viel Zeit in die Strukturierung des Projektes (Verzeichnisbaum) steckt, die sich nicht sofort auf Papier niederschlägt. Beim Arbeiten mit WYSIWYG-Programmen hat man schnell Erfolgserlebnisse, die Schwierigkeiten kommen später, wenn die Manuskripte umfangreicher und komplexer werden, aber dann wird ein Umstieg teuer.

1.11.2 UNIX-Formatierer (nroff, groff)

Der Standard-Formatierer in Linux/UNIX für Druckerausgabe ist `nroff`. Das `n` steht für *new*, da der Vorgänger ein `roff` war, und dieser hieß so, weil damit *run off to the printer* gemeint war. Unter Debian ist `nroff` ein Script, das `groff` aus dem GNU-Projekt aufruft. Die Variante `troff` war ursprünglich ein Formatierer zur Ausgabe auf einem Fotosatzbelichter. Unter Debian ist `troff` der Teil von `groff`, der die eigentliche Arbeit erledigt.

Eine Datei für `nroff` enthält den unformatierten Text und `nroff`-Kommandos. Diese stehen stets in eigenen Zeilen mit einem Punkt am Anfang. Ein `nroff`-Text könnte so beginnen:

```
.po 1c
.ll 60
.fi
.ad c

.cu 1
Ein Textbeispiel

von W. Alex

.ad b
.ti 1c
Dies ist das Beispiel eines Textes, der mit nroff
formatiert werden soll. Er wurde mit dem vi verfasst.

.ti 1c
Hier beginnt der zweite Absatz.
Die Zeilenlaenge in der Textdatei ist unerheblich.
```

Die `nroff`-Kommandos bedeuten folgendes:

- `po 1c` page offset 1 cm (zusätzlicher linker Seitenrand),
- `ll 60` line length 60 characters,
- `fi` fill output lines (für Blocksatz),
- `ad c` adjust center (zentrieren),
- `cu 1` continuous underline 1 line (auch Leerzeichen unterstreichen),
- `ad b` adjust both margins (Blocksatz),
- `ti 1c` temporary indent 1 cm (einmaliger Einzug).

Die Kommandos können wesentlich komplexer sein als im obigen Beispiel, es sind auch Makros, Abfragen und Rechnungen möglich. S. R. BOURNE führt in seinem im Anhang **??** *Zum Weiterlesen* auf Seite **??** genannten Buch die Makros auf, mit denen die amerikanische Ausgabe seines Buches formatiert wurde. Es gibt ganze Makrobibliotheken zu `nroff`.

Da sich Formeln und Tabellen nur schlecht mit den Textbefehlen beschreiben lassen, verwendet man für diese beiden Fälle eigene Befehle samt Präprozessoren, die die Spezialbefehle in `nroff`-Befehle umwandeln. Für Tabellen nimmt man `tbl`, für Formeln `neqn`, meist in Form einer Pipe:

```
joe@debian:~$ tbl textfile | neqn | nroff | col | lp
```

wobei `col` ein Filter zur Behandlung von Backspaces und dergleichen ist.

1.11.3 GNU Texinfo

Das Debian-Paket *texinfo* (gesprochen *tekinfo*) enthält das GNU Documentation System Texinfo, dessen Startseite unter `http://texinfo.org/` oder `http:`

`//www.gnu.org/software/texinfo/` liegt. Das System bezweckt, aus einer Textvorlage (Quelle) sowohl eine Ausgabe auf den Bildschirm (Terminal oder Web-Browser) als auch eine Ausgabe auf einen Drucker zu erzeugen. Eine andere Lösung dieser Aufgabe findet man unter den Stichworten *Standard General Markup Language* (SGML) und *Extensible Markup Language* (XML). Texinfo ist unabhängig von TEX und LATEX, arbeitet mit diesem System aber zusammen. Wir finden Texinfo-Dokumente oft als Ersatz für Manualseiten, siehe Abschnitt **??** *Hilfen* auf Seite **??**. Außerhalb von GNU ist Texinfo wenig verbreitet.

Das Kommando zum Lesen von Texinfo-Dokumenten lautet `info`. Man kann es zunächst einmal anstelle von `man` aufrufen:

```
joe@debian:~$ info ls
```

Dann bekommt man mit allen Möglichkeiten von Texinfo wie Vorwärts- und Rückwärtsblättern die gewohnte Manualseite angezeigt. Mit dem Kommando:

```
joe@debian:~$ info info
```

erscheint eine kurze, strukturierte Einführung auf dem Bildschirm. Bei Schwierigkeiten versuchen Sie es mit der Option `-f` und dem absoluten Pfad der Texinfo-Datei `/usr/share/info/...`. Nachdem Sie den ersten Schirm gelesen haben, bewegen Sie den Cursor mit den Pfeiltasten in die Zeile *Getting Started* und tippen `<cr>`. Darauf hin erscheint die entsprechende Seite. In der obersten Zeile (header) lesen Sie immer, wo Sie sich befinden. Mit den Tasten `<n>` oder `<p>` gelangt man zur nächsten (next) bzw. vorherigen (previous) Seite. Füllt eine Seite mehr als einen Schirm, schiebt man – wie von `less` gewohnt – mit der Leertaste den Text nach oben, der Backspace schiebt nach unten, mittels `<b>` (begin) gelangt man zum Seitenanfang, mittels `<e>` (end) zum Seitenende. Weitere Möglichkeiten erklärt die Infoseite. Eine Alternative zu `info` enthält das Debian-Paket *pinfo*. Auch die Hilfesysteme von GNOME und KDE verstehen das Texinfo-Format. Im Konqueror kann man mittels der Eingabe `info:gzip` als URL die Texinfo-Seite zum Kommando `gzip` betrachten, da automatisch das Filter `info2html` aufgerufen wird. Bei fehlendem Texinfo-Dokument fällt der Konqueror nicht auf das Manual zurück.

Ein Texinfo-Quelltext wird mit einem beliebigen Editor geschrieben. Der Emacs kennt einen Texinfo-Modus, der die Arbeit erleichtert und auch statt `info` zum Lesen verwendet werden kann. Metazeichen sind der Klammeraffe und die beiden geschweiften Klammern. Der Klammeraffe dient auch zum Quoten der Metazeichen, `@@` bedeutet den Klammeraffen buchstäblich. Tabs verursachen Probleme bei der Formatierung, ansonsten dürfen alle druckbaren ASCII-Zeichen im Text vorkommen. Die Formatierkommandos lauten anders als bei `nroff`, TEX oder LATEX, obwohl sie zum großen Teil dieselben Aufgaben haben. Man darf auch nicht vergessen, dass man für zwei verschiedene Leserkreise schreibt: die Buchleser und die Bildschirmleser. Kurze Passagen im Text lassen sich so markieren, dass sie sich nur an einen der beiden Leserkreise wenden, aber zwei gänzlich unterschiedliche Dokumente abzufassen, wäre ein Rückfall in die Zeit vor Texinfo. Die Referenz entnimmt man am besten dem Netz.

Hier ein minimales Beispiel für einen Texinfo-Quelltext (stark gekürzt nach dem Short Sample aus `http://www.gnu.org/software/texinfo/manual/texinfo.html`), nur um einen Eindruck zu gewinnen:

```
\input texinfo
@setfilename sample.info
@settitle Sample Manual

@titlepage
@title Sample Title
@end titlepage

@ifnottex
@node Top
@top Texinfo-Beispiel
@end ifnottex

@menu
* 1. Abschnitt::
@end menu

@node 1. Abschnitt
@chapter 1. Abschnitt

@bye
```

Wer schon mit `nroff` oder LaTeX gearbeitet hat, gewöhnt sich schnell ein. Die online lesbare info-Datei wird von dem Programm `makeinfo` erstellt, das zum Texinfo-Paket gehört. Zur Erzeugung der Papierausgabe ist die Texinfo-Quelldatei durch das TeX-System samt dem Makro `texinfo.tex` zu schicken, die DVI-Datei dann wie unter TeX oder LaTeX gewohnt durch `dvips`, um eine druckbare PostScript-Datei zu erhalten. Zweckmäßig fasst man die Kommandos in einem Makefile zusammen:

```
all :  sample.ps sample.info

sample.info : sample.texinfo
       makeinfo sample

sample.ps : sample.dvi
       dvips -o sample.ps sample

sample.dvi : sample.texinfo
       tex sample.texinfo

clean :
       rm *.aux *.toc *.log
```

und ruft nach dem Editieren nur noch `make` auf. Alternativ erzeugt auch die Eingabe:

```
joe@debian:~$ texi2dvi sample.texinfo
```

eine `dvi`-Datei. Im Netz finden sich Konverter von Texinfo nach HTML, nroff, IPF und RTF.

1.11.4 DocBook

DocBook ist ein Format für Bücher, Artikel und andere Dokumentationen vorzugsweise technischen Inhalts, das in einer Document Type Definition (DTD) beschrieben ist, siehe Abschnitt 1.12.3 *Standard Generalized Markup Language* auf Seite 74. Die DTD (SGML) ist im Debian-Paket *docbook* zu finden, die DTD (XML) im Debian-Paket *docbook-xml*. Das Format ist in der Open-Source-Welt verbreitet. Beispielsweise werden viele HOWTOS in diesem Format erstellt. Ein DocBook-Quelltext hat Ähnlichkeiten mit einem HTML-Dokument, siehe Abschnitt 1.12.2 *Hypertext Markup Language* auf Seite 71. Aus DocBook-Quellen lassen sich als Endprodukte PDF- oder PostScript-Dateien für den Druck oder HTML für das Web erzeugen. Das Debian-Paket *db2latex-xsl* enthält Style Sheets zur Umwandlung von DocBook-XML-Dateien nach LaTeX. Weitere Konverter finden sich im Debian-Paket *docbook-utils*, darunter `docbook2html` und `docbook2pdf`. Der Versuch, einige docbook-Hilfen aus dem KDE-Projekt zu konvertieren, scheiterte leider an Syntaxfehlern. Näheres bei:

- `http://www.oasis-open.org/docbook/`, siehe auch das Debian-Paket *docbook-simple*,
- `http://docbook.org/` von NORMAN WALSH und anderen, siehe auch Debian-Paket *docbook-defguide*,
- `http://www.sagehill.net/docbookxsl/` von BOB STAYTON,
- `http://www.docbook.de/`,
- `http://www.goshaky.com/docbook-tutorial/` von LARS TRIELOFF (in deutscher Sprache).

1.11.5 LaTeX

TeX und LaTeX

TeX ist eine Formatierungssoftware (E: typesetting system), die von DONALD ERVIN KNUTH (`http://www-cs-staff.stanford.edu/~uno/index.html`) um 1980 entwickelt wurde – dem Mann, der seit Jahrzehnten an dem siebenbändigen Werk *The Art of Computer Programming* (TAOCP) schreibt und hoffentlich noch lange lebt. Die Stärke der Software sind umfangreiche mathematische oder technische Texte. Das System optimiert immer ganze Seiten und innerhalb der Seiten die Absätze, nicht einzelne Zeilen wie einfachere Werkzeuge. In TeX steckt viel handwerkliches Wissen aus dem typografischen Gewerbe. Seine Grenzen findet es bei der Gestaltung von:

- Zeitungen mit mehreren Spalten, vielen Überschriften, Fotos, Anzeigen und dergleichen. Für diese Aufgabe braucht man Desktop-Publishing-Programme wie Scribus (Heimathafen `http://www.scribus.net/`, als Debian-Paket verfügbar), Adobe InDesign oder Quark XPress (beide kommerziell).
- Bildbänden oder Fotoalben mit vielen großformatigen Abbildungen. Hier kann man sich helfen, indem man Bild- und Textteil voneinander trennt. Das gesamte Werk lässt sich dann wieder mit TEX einrichten. Wenn jedes Bild samt Unterschrift genau eine oder eine halbe Seite beansprucht, wird es wieder einfach.

Für rein grafische Erzeugnisse wie Plakate gibt es geeignetere Werkzeuge, siehe Abschnitt **??** *Das grafische Atelier* auf Seite **??**.

TEX ist sehr leistungsfähig, verlangt aber vom Benutzer die Kenntnis vieler Einzelheiten, ähnlich wie Programmieren in Assembler. LATEX ist eine Makrosammlung, die auf TEX aufbaut. Die LATEX-Makros von LESLIE LAMPORT erleichtern bei Standardaufgaben und -formaten die Arbeit beträchtlich, indem viele TEX-Befehle zu einfach anzuwendenden LATEX-Befehlen zusammengefasst werden. Kleinere Modifikationen der Standardeinstellungen sind vorgesehen, Sonderwünsche erfordern das Hinabsteigen auf TEX-Ebene. Die Entwicklung seit LAMPORT diente vor allem der Internationalisierung. Zu LATEX sind im Laufe der Jahre zahlreiche Makros hinzugefügt worden, sodass es heute kaum einen Wunsch im Textsatz gibt, der nicht mit LATEX erfüllbar wäre. Das Problem besteht eher darin, das am besten geeignete Makro aus rund 1000 Zusatzpaketen herauszufinden. LATEX eignet sich für alle Arten von Dokumenten, vom Brief bis zum mehrbändigen Lebenswerk. Auch zur Gestaltung von Folien oder Präsentationen lässt sich LATEX heranziehen. Es steht für viele Betriebssysteme kostenfrei zur Verfügung. Alle gängigen Linux-Distributionen bringen LATEX mit. Da TEX samt LATEX ein umfangreiches Programmpaket bilden, stehen eigene LATEX-Distributionen im Netz; eine bekannte ist teTeX von THOMAS ESSER mit der Startseite `http://www.tug.org/teTeX/`. Das Einrichten einer solchen Distribution ist immer noch Arbeit. Debian-Benutzer laden sich einfach mit `apt-get install` die entsprechenden teTeX-Pakete herunter und brauchen sich um nichts weiter zu kümmern.

Im deutschsprachigen Raum werden TEX und LATEX von der *Deutschsprachigen Anwendervereinigung TeX e. V.* (DANTE) mit Sitz in Heidelberg gepflegt. Auf ihrem Webserver `http://www.dante.de/` findet man alles, was zu der Software gehört, insbesondere einen Wegweiser zum *Comprehensive TeX Archive Network* (CTAN) und eine DE-TeX-FAQ-Sammlung, ausgedruckt knapp 200 Seiten. Auch an Einführungen und Anleitungen in diversen Sprachen mangelt es nicht. Wer Geld loswerden will, wird durch eine umfangreiche Bücherliste unterstützt. Als schnelle Einführung ist die bewährte *LATEX-Kurzanleitung* von HUBERT PARTL et. al. hervorzuheben, die auf Deutsch oder Englisch an vielen Stellen im Netz zu finden ist und jeder LATEX-Distribution beiliegt.

Computeralgebrasysteme wie Maple oder Mathematica arbeiten mit LATEX zusammen; aus den Arbeitsblättern lassen sich LATEX-Artikel erzeugen. Diese kann man entweder als selbständige Dokumente behandeln oder nach etwas Editieren (Löschen

des Vorspanns) als Kapitel oder Abschnitte in andere LaTeX-Dokumente einbinden. Man hat damit sozusagen ein LaTeX, das rechnen kann.

Das Adobe-PDF-Format (Portable Document Format) hat sich sehr verbreitet, siehe Abschnitt 1.2 *Werkzeuge zum Lesen und Umwandeln* auf Seite 15. PostScript-Dateien lassen sich in pdf-Dateien umwandeln; besser jedoch ist es, aus LaTeX-Manuskripten mittels `pdflatex` unmittelbar pdf-Dateien zu erzeugen. Zwischen `latex` und `pdflatex` bestehen kleine Unterschiede beim Einbinden von Grafiken in den Text; erforderlichenfalls sind die Grafiken in zwei Formaten (jpg und eps) zur Verfügung zu stellen, GIMP hilft dabei.

Das wichtigste Stilelement von LaTeX ist die Dokumentklasse. Diese bestimmt die Gliederung und wesentliche Teile der Aufmachung. Die Dokumentklasse *book* kennt Bände, Kapitel, Abschnitte, Unterabschnitte usw. Ein Inhaltsverzeichnis wird angelegt, auf Wunsch und mit etwas menschlicher Unterstützung auch ein Sachverzeichnis. Bei der Dokumentklasse *report* beginnt die Gliederung mit dem Kapitel, ansonsten ist sie dem Buch ähnlich. Das ist die richtige Klasse für Dissertationen, Diplomarbeiten, Forschungsberichte, Skripten und dergleichen. Mehrbändige Werke sind in diesem Genre selten. Die Dokumentklasse *article* eignet sich für Aufsätze und kurze Berichte. Die Gliederung beginnt mit dem Abschnitt, die Klasse kennt kein Inhaltsverzeichnis, dafür aber eine Zusammenfassung (Abstract). Nützlich ist auch die Dokumentklasse *foils* zur Formatierung von Folien für Overhead-Projektoren. Hier wird das Dokument in Folien gegliedert, die Schriftgröße beträgt 25 oder 30 Punkte. Zum Schreiben von Briefen gibt es eine Klasse *dinbrief*, mit deren Hilfe sich anspruchsvoll gestaltete, normgerechte Briefe erstellen lassen, auch Privatbriefe.

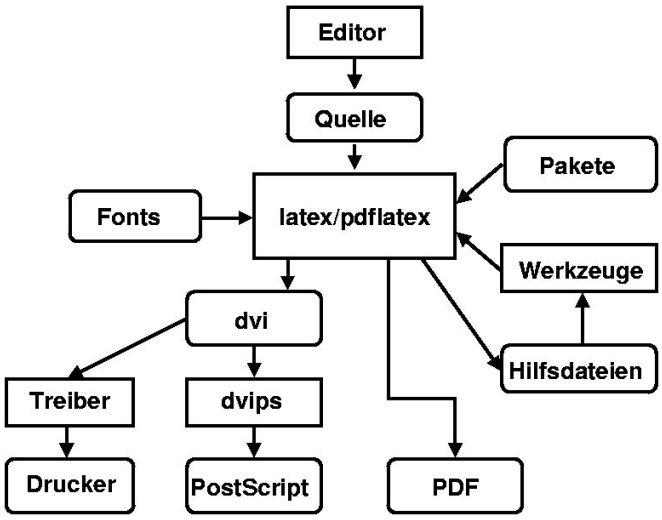

Abb. 1.12: Arbeitsablauf beim Erzeugen eines LaTeX-Dokumentes

In Abbildung 1.12 ist der Arbeitsablauf schematisch dargestellt. Man schreibt seinen Text mit einem beliebigen Editor. Dabei wird nur von den druckbaren Zeichen des US-ASCII-Zeichensatzes zuzüglich `<cr>` Gebrauch gemacht. In den Text eingestreut sind LaTeX-Anweisungen. Der Name der Textdatei (Quelle) muss die Kennung `tex` tragen. Dann schickt man die Textdatei durch den LaTeX-Übersetzer. Dieser erzeugt eine Binärdatei, deren Namen die Kennung `dvi` trägt. Das bedeutet *device independent*, die Binärdatei ist noch nicht auf ein bestimmtes Ausgabegerät hin ausgerichtet. Verwendet man `pdflatex`, entfällt der Zwischenschritt, man erhält unmittelbar eine pdf-Datei. Mittels eines geräteabhängigen Treiberprogramms wird aus der dvi-Datei eine Datei mit der Kennung `bit` erzeugt, die mit einem Linux/UNIX-Kommando wie `cat` durch eine transparente Schnittstelle zum Drucker geschickt wird. Das Programm `dvips` erzeugt aus der dvi-Datei eine PostScript-Datei, die auf einem beliebigen PostScript-Drucker ausgegeben werden kann. Während des Übersetzens schreibt der Compiler mehrere Hilfsdateien – beispielsweise mit Einträgen für den Index oder das Inhaltsverzeichnis – die roh oder nach Bearbeitung durch ein externes Werkzeug dem Compiler beim nächsten Durchgang zugeführt werden. Da der Compiler die Quelle sequentiell durcharbeitet, sind unter Umständen mehrere Läufe erforderlich, bis alle Seitenangaben stimmen. Zu jedem Lauf werden die Hilfsdateien des vorangegangenen Laufs herangezogen.

Zum Schreiben von Privat- oder Geschäftsbriefen verwendet man im deutschen Sprachraum die Dokumentklasse *dinbrief*. Die Klassenbeschreibung heißt `dinbrief.cls` und liegt im Verzeichnis `/usr/share/texmf/tex/latex/dinbrief/`, die zugehörige Dokumentation im Verzeichnis `/usr/share/doc/texmf/latex/dinbrief/`. Im persönlichen Briefverzeichnis legt man sich eine Vorlage (Muster, Schablone, Template) folgender Art an:

```
\documentclass[12pt]{dinbrief}
\usepackage{ngerman, newcent}

% \input{briefkopf}      % sofern vorhanden
\address{\textbf{Dr.-Ing. Wulf Alex}\\
         Latexweg 999\\
         D-12345 Linuxhausen (Baden)\\
         Tel. 01234/567890\\
         Mobiltel. 0160/12345\\
         VoIP 032123456789\\
         Email alex-weingarten@t-online.de}
\backaddress{W. Alex, Latexweg 999, D-12345 Linuxhausen}
% \signature{W. Alex}    % nach Bedarf
\place{Linuxhausen (Baden)}

\begin{document}

\begin{letter}{Superputer GmbH\\
               Debianstra"se 111\\ \\
               \textbf{01234 Linuxberg}}

\subject{Open-Source-Treffen 2006}
```

```
\nowindowrules
\opening{Sehr geehrte Damen und Herren,}

Hier folgt der Text.

\closing{Mit freundlichem Gru"s\\Ihr}

% \ps{Postscriptum}      % nach Bedarf
% \cc{Verteiler}         % nach Bedarf
% \encl{Anlagen}         % nach Bedarf
\end{letter}
\end{document}
```

Quelle 1.2 : LaTeX-Datei `dinbrief.tex` als Vorlage für Briefe

die man kopiert, um einen neuen Brief abzufassen. Nach dem Editieren der Kopie geht es wie mit jedem LaTeX-Dokument weiter. Effektivitätsbewusste Faulpelze schreiben sich ein Makefile dazu:

```
# Makefile fuer Briefe, W. Alex 2003-01-10

SRC = $@     # vordefinierte make-Variable
LATEX  = /usr/bin/latex
PLATEX = /usr/bin/pdflatex
DVIPS  = /usr/bin/dvips

.DEFAULT :   # vordefiniertes Target
        ${PLATEX} ${SRC}.tex
        ${LATEX} ${SRC}.tex
        ${DVIPS} -o ${SRC}.ps ${SRC}

clean :
        rm -f *.aux *.log *.dvi
```

Quelle 1.3 : Makefile zum Erzeugen von Briefen mittels LaTeX

und rufen `make` mit dem Namen der Briefdatei ohne Kennung als Argument auf.

Abbildungen

LaTeX bringt einige Befehle zum Zeichnen mit, die jedoch mehr Frust als Freude hervorrufen. Es gibt Zusatzprogramme (TeXCAD u. a.) zu LaTeX, die es erleichtern, den Text durch Zeichnungen zu ergänzen. Außerdem kann man Grafiken bestimmter Formate – Encapsulated Postscript, jpeg-Fotos und andere – in den Text einbinden. So wurden die Zeichnungen im Buch mittels `xfig` erstellt, siehe Abschnitt **??** *Zeichnen* auf Seite **??**, und mit GIMP in eps- und jpg-Dateien umgewandelt. Abbildungen gehören wie Tabellen zu den gleitenden Objekten, das heißt, LaTeX kann sie im Text im Interesse der Seitenaufteilung verschieben. Dabei gelten einige Regeln:

- Eine Abbildung erscheint frühestens an der Stelle, an der sie im Text genannt wird; verschoben wird nur nach hinten.
- Die Reihenfolge der Abbildungen wird nicht verändert.
- Wenn sich LaTeX gar nicht mehr zu helfen weiß, verschiebt es die Abbildungen an das Ende des Textes (Kapitels).

Bei Manuskripten mit vielen Abbildungen muss der Autor helfend eingreifen, indem er sie möglichst weit vorn in den Text einfügt oder etwas verkleinert. Zur Abbildung gehört auch ihre Unterschrift (E: caption), deren Zeilenanzahl man unter Umständen verringern kann. Im vorliegenden Buch wird das Makropaket `graphicx` verwendet; der Code für das Einfügen einer Abbildung in den Text sieht folgendermaßen aus:

```
\begin{figure}
\centering
\includegraphics[width=\figwidth]{bilder/gimp}
\caption[Screenshot GIMP]{Screenshot des Werkzeugs GIMP}
\label{fig:gimp}
\end{figure}
```

Ohne zwingenden Grund gebe man in der ersten Zeile keine Position (`[htb]`) an, sondern lasse LaTeX alle Freiheit. Die Variable `figwidth` wird in der persönlichen Style-Datei als 0.96-faches der Breite des Satzspiegels definiert. Die Kennung der Grafikdatei `bilder/gimp` ist nicht angegeben; LaTeX sucht sich die passende Datei (`eps` oder `jpg`) heraus.

Tabellen

LaTeX kennt mehrere Formen von Tabellen. Die einfachste ist die fest in den Text eingefügte Tabelle ohne Überschrift. Sie erscheint nicht im Tabellenverzeichnis und wird mittels der `tabular`-Umgebung erzeugt. Mit der `table`-Umgebung definiert man gleitende Tabellen mit Überschrift etc., die sich bei Hinzuziehen des `bigtabular`-Makros auch über mehrere Seiten erstrecken dürfen (im Gegensatz zu Abbildungen).

Formeln

Da TeX ursprünglich entwickelt worden ist, um mathematische Texte zu setzen, ist der Formelsatz von TeX und LaTeX unübertroffen. Auch die Vielfalt an fertigen mathematischen Symbolen übertrifft die meisten Bedürfnisse. Im Anhang werden einige einfache und komplizierte Formeln als Quelle und Ausdruck wiedergegeben, die man als Vorlage für die eigene Arbeit nehmen kann. Hat man oft mit Formeln zu tun, legt man sich eine Sammlung an und erzeugt neue Formeln durch Kopieren und Editieren ähnlicher Formeln.

Will man mit LaTeX gesetzte Formeln in Dokumente anderer Herkunft (Webseiten) übernehmen, geht das meist nur auf dem Weg der Umwandlung in eine Rastergrafik. Einen Weg dazu zeigt STEFAN LAGOTZKI auf `http://www.lagotzki.de/scripte/formeln/`. Er verwendet Ghostscript und ImageMagick, verpackt in ein Shellskript.

Register (Index)

Ein Register, Index oder Sachverzeichnis gehört zu jedem ordentlichen Sachtext von mehr als ein paar Seiten Umfang. Es erleichtert dem Leser den Zugang, vielleicht noch mehr als das Inhaltsverzeichnis. Andererseits zählt das Zusammenstellen eines Registers zu den unbeliebtesten Pflichten. Der Autor – ohnehin schon am Ende seiner Kräfte und vier Wochen im Verzug – geht mit Widerwillen an diese Arbeit, die kein rechtes Erfolgserlebnis beschert. LaTeX greift ihm auch hier unter die Arme.

Im einfachsten Fall schreibt man schon beim Abfassen des Textes vor jeden Absatz Zeilen folgender Art:

```
\index{Stichwort}
```

pro Zeile ein Eintrag. Bei langen Absätzen muss man solche Zeilen auch in die Absätze einfügen, aber stets in eigenen Zeilen, sonst findet man sie schlecht wieder. Ein Nachteil des Herausziehens in eigene Zeilen sei nicht verschwiegen: LaTeX kann zwischen Kommando und dem tatsächlichen Vorkommen des Stichwortes im laufenden Text einen Seitenumbruch einfügen, womit die Seitenangabe im Sachverzeichnis um 1 zu niedrig gerät. Die Stich- oder Schlagwörter werden immer in ihrer lexikalischen Grundform (Infinitiv, Nominativ Singular) genannt, sofern nicht die gebeugte Form einen eigenen Begriff bildet. Bei Personennamen wird der Nachname vorangestellt. Die Einträge dürfen nicht die Zeichen `!`, `@` oder `|` enthalten, weil diese im vorliegenden Zusammenhang eine besondere Bedeutung haben. In die Präambel der Hauptdatei kommt eine Zeile:

```
\makeindex
```

die LaTeX veranlasst, beim Durchgehen des Textes die `\index`-Zeilen samt Seitenzahlen in eine Hilfsdatei mit der Kennung `idx` herauszuschreiben. Weiterhin ist in der Präambel oder in einer gesonderten Datei das Paket *makeidx* einzubinden:

```
\usepackage{makeidx}
```

Das Paket stellt einige für das Registermachen benötigte Befehle bereit. An das Ende der Hauptdatei – meist unmittelbar vor `\end{document}` – kommt die Aufforderung zum Ausgeben des Index:

```
\printindex
```

Weiterhin ist nach ein bis drei LaTeX-Durchläufen – wenn die Seitenzahlen feststehen – das Kommando:

```
joe@debian:~$ makeindex hauptdatei
```

aufzurufen, das die Aufbereitung der Hilfsdatei veranlasst. Abschliessend muss noch einmal `latex` bemüht werden, um den bearbeiteten Index in das Dokument einzufügen. Es empfiehlt sich, auch den Index Korrektur zu lesen, um Ungereimtheiten zu beseitigen.

Der Index lässt sich in mehrfacher Hinsicht ausgestalten. Zu Einträgen können Untereinträge erzeugt werden:

```
\index{Datei!Besitzer}
```

Das Ergebnis ist auf den letzten Seiten des Buches anzuschauen. LaTeX erlaubt sogar Unter-Untereinträge, aber die sind selten erforderlich und ergeben ein unruhiges Schriftbild. Das Einsortieren eines Eintrags an anderer Stelle erzwingt ein Klammeraffe:

```
\index{hostname@\textit{hostname}}
\index{Aland@{\AA}land}
```

Im ersten Fall erscheint der Eintrag in kursiver Schrift (Italics) an der richtigen Stelle, obwohl die Zeichenkette mit einem LaTeX-Befehl beginnt, im zweiten wird die Inselgruppe unter A eingeordnet und nicht unter dem im deutschen Alphabet unbekannten Buchstaben Å und auch nicht wie im Schwedischen üblich hinter Z.

Verweise auf Synonyme oder Erklärungen erzeugt man mittels Eintragungen folgender Art:

```
\index{Computer|see{Rechner}}
\index{IP|see{Internet Protocol}}
```

Das Wörtchen *see* lässt sich durch eine Zeile in der Präambel umdefinieren:

```
\renewcommand*{\seename}{\textbf{$\rightarrow$}}
```

Hat man viele Verweise, packt man sie zweckmäßig in eine eigene Datei namens `verweise.tex`, die man in der Präambel mittels

```
\input{verweise}
```

einbindet. Für das vorliegende Buch haben wir sogar zwischen Synonymen und Abkürzungen unterschieden.

Will man das Register aufteilen in einen Personen- und einen Sachindex, ist das mühelos machbar. Man bindet dazu in der Präambel statt des Paketes `makeidx` das Paket `index` ein und streicht die Zeile `\makeindex`. Ferner definiert man:

```
\newindex{per}{pdx}{pnd}{Personenindex}
\newindex{sac}{sdx}{snd}{Sachindex}
\newcommand{pindex}[1]{\index[per]{#1}}
\newcommand{sindex}[1]{\index[sac]{#1}}
```

Damit stehen die Kommandos `\pindex{}` und `\sindex{}` zum Benennen der jeweiligen Einträge zur Verfügung. Wir haben jetzt zwei Hilfsdateien aufzubereiten:

```
joe@debian:~$ makeindex -o hauptdatei.pnd
hauptdatei.pdx
```

```
joe@debian:~$ makeindex -o hauptdatei.snd
hauptdatei.sdx
```

und müssen am Ende des Dokuments auch zwei Indexdateien einbinden:

```
\printindex[per]
\printindex[sac]
```

Alles Übrige wie gehabt. Natürlich lassen sich auch noch ein Kommandoindex und ein Ortsindex erzeugen. Der Begeisterung sind kaum Grenzen gezogen.

Fehlersuche in LaTeX-Dateien

TeX/LaTeX ist nicht fehlertolerant, sondern gnadenlos. Bei der kleinsten Unstimmigkeit bricht die Übersetzung ab. Andererseits unterlaufen beim Schreiben Tipp- und Syntaxfehler, besonders wenn Mitternacht vorüber ist. Die erste Empfehlung lautet, die zu korrigierenden Textbrocken nicht zu groß werden zu lassen. Dazu unterteilt man einen großen Text in Dateien wie im Abschnitt 1.15 *Organisation eines Textprojekts* auf Seite 87 beschrieben. Die häufigsten Fehler sind:

- fehlende, überzählige oder falsche Klammern,
- Verwechseln von Schräg- und Gegenschrägstrich,
- Metazeichen nicht gequotet, wenn man sie buchstäblich haben will,
- Vertippen bei Kommandos (ergibt ungültige Kommandos),
- `begin irgendwas` geschrieben, das zugehörige `end` vergessen; führt oft zu schwer zu lokalisierenden Fehlern.

Manche Fehler machen sich erst viele Zeilen hinter ihrem Entstehungsort bemerkbar, spätestens am Ende des Dokuments. Tippfehler im eigentlichen Text interessieren LaTeX nicht, dafür sind Rechtschreibprüfer wie `ispell` zuständig.

Syntaxprüfer für TeX/LaTeX-Texte sind `lacheck` und `chktex`, vergleichbar `lint` für C-Programme. Es macht Sinn, beide zu benutzen. Sie arbeiten schneller als der Übersetzer `latex` und warnen auch, wenn ihnen etwas verdächtig erscheint, ohne dass es gleich falsch zu sein braucht. Beide können einzelne Abschnitte prüfen, brauchen also nicht immer das gesamte Dokument. Schreibt man deutsche Umlaute in der Form `"a`, ist es angebracht, diesbezügliche Warnungen von `chktex` per Option zu unterdrücken und auch gleich noch die mittlerweile überflüssigen Warnungen wegen fehlender Italic-Korrektur:

```
joe@debian:~$ chktex -n18 -n6 latexdatei
```

Ihre Grenze erreichen die Werkzeuge bei komplizierten, verschachtelten Makros.

Zum Untersuchen des eigentlichen Textes auf Schreibfehler dienen Rechtschreibprüfer (E: spellchecker) wie `ispell`, siehe Abschnitt 1.13.4 *Rechtschreibung prüfen* auf Seite 78. Das Kommando kennt einen TeX/LaTeX-Modus, in dem es offensichtliche LaTeX-Kommandos überliest. Wenn es auf ein deutsches Wörterbuch eingestellt ist, stört es sich verständlicherweise an englischen Wörtern, an manchen Schreibweisen von Sonderzeichen sowie an dem Einfügen von `\` zum Verhindern

von Ligaturen in Wörtern wie *auffordern*. Die von `ispell` beanstandeten Wörter lassen sich verbessern, übergehen oder in ein persönliches Wörterbuch aufnehmen. Insgesamt gesehen ist der Einsatz von `ispell` anzuraten. Ein paar LaTeX-Fehler findet der Prüfer nebenbei, weshalb man ihn vor einem Syntaxprüfer einsetzen sollte. Zum Verhindern falscher LaTeX-Ligaturen in deutschen Texten dient das kleine Werkzeug `rmligs-german`. Man könnte sich einen Beautifier vorstellen, der sprachenbezogen ein LaTeX-Manuskript auf typografische Unschönheiten abklopft. Der müsste aber auch das Dickicht der Makros durchforsten, was nicht trivial ist.

Ferner kopiert man zweckmäßig die Hauptdatei (`main.tex` oder ähnlich) in eine Datei `test.tex` und fügt der Präambel eine Zeile folgender Art hinzu:

```
\includeonly{kapitel3/kap}
```

Die Zeile bezieht sich auf eine Zeile im Hauptteil:

```
\include{kapitel3/kap}
```

mit der eine Kapiteldatei eingebunden wird. Der `\includeonly`-Befehl führt dazu, dass trotz mehrerer in das Werk einzubindender Kapitel für Testzwecke nur jeweils ein Kapitel herangezogen wird. In der Datei `test.tex` kann man auch weitere Dinge ausprobieren, ohne die Hauptdatei zu ändern oder zu beschädigen. Beispielsweise kann man das Paket *syntonly* einbinden und im Vorspann mit dem Befehl `\syntaxonly` bewirken, dass nur die Syntax überprüft wird. Erscheint diese fehlerfrei, kommentiert man den Befehl aus und übersetzt.

Der LaTeX-Übersetzer schreibt Meldungen auf den Bildschirm und gleichzeitig in eine Datei `main.log` oder `test.log`. Diese bricht im Fehlerfall mit einer Meldung ab, die nicht immer hilfreich ist, aber wenigstens den Ort des Fehlers eingrenzt. Oft reicht es aus, sich mit einem Editor die Umgebung des Fehlers anzusehen, um ihn zu finden. Die temporäre Reparatur des Fehlers aus dem Übersetzer heraus ist in der Regel nicht zu empfehlen.

Ist die den Fehler enthaltende Datei groß, lassen sich Teile daraus auskommentieren. Einzelnen Zeilen stellt man ein Prozentzeichen voran, ganze Abschnitte rahmt man mit `\iffalse` und `\fi` ein, um sie von der Bearbeitung durch den Übersetzer auszuschließen. `\iffalse` ist eigentlich ein TeX-Kommando zum Basteln von Verzweigungen, erfüllt jedoch hier genau unseren Wunsch. In verzweifelten Fällen haben wir die fehlerhafte Datei auf diese Weise schrittweise halbiert und den Fehler mittels einer Art von binärer Suche eingekreist.

KDE Integrated LaTeX Environment (kile)

Das KDE Integrated LaTeX Environment – früher KTeXmaker – aus den Debian-Paketen *kile* und *kile-i18n* erleichtert das Schreiben von LaTeX-Dokumenten, indem es die verschiedenen Befehle und LaTeX-Umgebungen in Form von rund fünfzig Buttons zur Verfügung stellt und dabei aufpasst, dass Klammern immer paarweise vorkommen. Befehle werden farbig hervorgehoben. Die Fehlersuche wird dadurch erleichtert, dass man gleichzeitig die Protokolldatei des Übersetzerlaufs und den

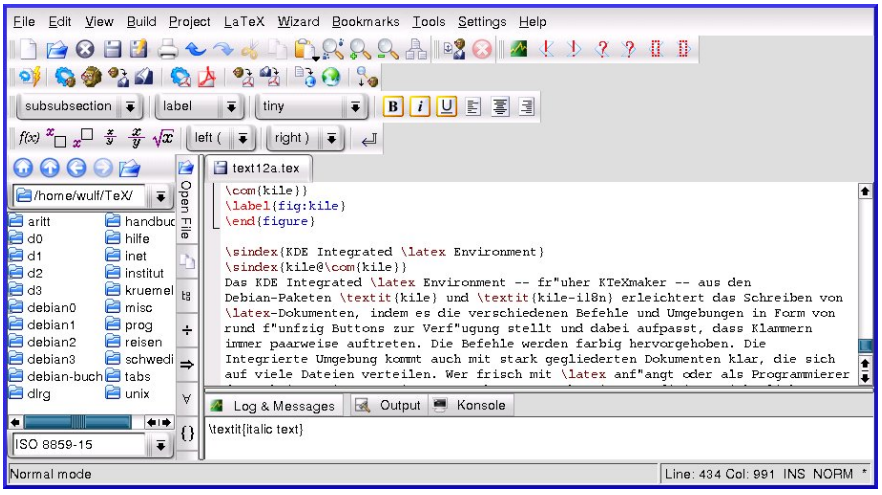

Abb. 1.13: Screenshot des KDE Integrated LaTeX Environments `kile`

Quelltext vor Augen hat. Die integrierte Umgebung kommt auch mit stark gegliederten Dokumenten klar, die sich auf viele Dateien verteilen. Wer frisch mit LaTeX anfängt oder als Programmierer das Arbeiten mit Integrierten Umgebungen gewohnt ist, profitiert von dem Werkzeug. Abbildung 1.13 zeigt ein Beispiel, und zwar genau den vorliegenden Abschnitt, der versuchshalber unter `kile` getippt wurde. Ein bereits laufendes großes Textprojekt auf `kile` umzurüsten, ist weniger zu empfehlen. `kile` ist kein WYSIWYG-Editor, hierzu siehe Abschnitt 1.6.4 *LyX* auf Seite 48.

Metafont

Metafont ist ein Software-System zum Erzeugen von Fonts für TeX/LaTeX. Da der normale Benutzer hoffentlich nicht auf den Gedanken verfällt, eigene Fonts gestalten zu wollen, bräuchte Metafont ihn nicht zu interessieren. Aber man kann mit Metafont Logos, exotische Symbole oder allgemein gesprochen komplexe grafische Pixelbilder (Bitmaps) erzeugen, und das kann schon einmal gefragt sein. Eine Gebrauchsanleitung für Metafont würde den Rahmen des Buches sprengen. Die Metafont-Bibel (zwei Bände) stammt von DONALD E. KNUTH, eine kurze, deutschsprachige Einführung gibt HELMUT KOPKA in einem seiner LaTeX-Bücher (die viel zur Verbreitung von LaTeX im deutschsprachigen Raum beigetragen haben).

1.12 Schreiben für das Web

1.12.1 Was ist Hypertext?

Bei Hypertext und der Hypertext Markup Language (HTML) geht es auch um das Formatieren von Texten oder allgemeiner von Hypermedia-Dokumenten, aber es

kommt noch etwas hinzu. Hypertexte enthalten Hyperlinks, kurz Links (F: lien) genannt und nicht mit den Links im Dateisystem zu verwechseln. Das sind Verweise, die elektronisch auswertbar sind, sodass man ohne Suchen und Blättern zu anderen Hypertexten weitergeführt wird. Man kann die Hyperlinks als aktive Querverweise bezeichnen. Der Begriff wurde Anfang der sechziger Jahre von TED NELSON geprägt. Auf Papier erfüllen Fußnoten, Literatursammlungen, Register, Querverweise und Konkordanzen einen ähnlichen Zweck, ohne elektronischen Komfort. Im ersten Kapitel war von WALLENSTEIN die Rede. Von diesem Stichwort könnten Verweise auf das Schauspiel von FRIEDRICH SCHILLER, die Werke von ALFRED DÖBLIN, RICARDA HUCH, PETER ENGLUND oder auf die Biografie von GOLO MANN führen, die die jeweiligen Texte auf den Bildschirm bringen, in SCHILLERS Fall sogar mit einem Film. In den jeweiligen Werken wären wieder Verweise enthalten, die auf Essays zur Reichsidee oder zur Rolle Böhmens in Europa lenken. Leseratten würden vielleicht auf dem Alexanderplatz in Berlin landen oder bei einem anderen Vertreter der schreibfreudigen Familie MANN. Von dort könnte es nach Frankreich, Indien, Ägypten, in die USA oder die Schweiz weitergehen. Vielleicht findet man auch Bemerkungen zum Verhältnis zwischen Literatur und Politik. Beim Nachschlagen in Enzyklopädien gerät man manchmal ins ziellose Schmökern. Mit Hypertext ist das noch viel, viel schlimmer. So ist jede Information eingebettet in ein Gespinst oder Netz von Beziehungen zu anderen Informationen, und wir kommen zum World Wide Web (WWW, W3).

1.12.2 Hypertext Markup Language (HTML)

Mit dem Web entstand der Bedarf an einer Sprache für Hyperdokumente. LaTeX konnte den Bedarf nicht decken, da es für die Ausgabe auf Papier geschaffen wurde. Also ersannen TIMOTHY BERNERS-LEE und ROBERT CAILLIAU 1989 am CERN in Genf neben den Uniform Resource Locators (URL) und dem Hypertext Transfer Protocol (HTTP) auch noch die Hypertext Markup Language (HTML), in der Hyperdokumente – Webseiten – geschrieben werden. Aktuell ist Version 4.01, die Nachfolgerin XHTML ist jedoch schon da, siehe Abschnitt 1.12.4 *Extensible Markup Language* auf Seite 75. HTML erreicht für Druckerausgaben nicht die Leistung von LaTeX. Beiden Sprachen gemeinsam ist das Konzept der Trennung von Inhalt, Struktur und Darstellung. HTML-Quellen kommen mit dem ASCII-Zeichensatz aus. Sie können externe Grafik- und Sound-Dateien verschiedener Formate einbinden.

Zum Lesen von Hypertexten braucht man HTML-Browser wie Amaya, Netscape, Mozilla, Firefox, Galeon, Opera, Mosaic, Konqueror oder den MS Internet-Explorer. Leider halten sich die wenigsten Browser an den gültigen HTML-Standard. Sie erkennen nicht alle standardkonformen HTML-Konstrukte und bringen eigene Vorstellungen mit. Wenn man HTML-Dokumente für die Öffentlichkeit schreibt, sollte man daher seine Erzeugnisse mit verschiedenen Browsern betrachten und überdies mit mehreren Werkzeugen testen.

Ein einfaches HTML-Dokument, das bei weitem nicht alle Möglichkeiten von HTML ausreizt, ist schnell geschrieben:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN">

<HTML>

<HEAD>
<TITLE>Institut fuer Hoeheres WWW-Wesen</TITLE>
</HEAD>

<BODY BGCOLOR="#ffffff">

<IMG SRC="http:../logowww.jpg" alt="logo">

<H3>
Fakult&auml;t f&uuml;r Internetwesen
</H3>

<HR>

Geb&auml;ude 30.70 <BR>
Telefon +49 721 608 9999 <BR>

<H4>
Leiter der Verwaltung
</H4>
Dipl.-Ing. Schorsch Meier

<H4>
Werkstattleiter
</H4>
Alois Wuchtbrumme

<HR>
Zur <A HREF="http://www.uka.de/">Universit&auml;t</A>
<HR>

http://www.mvm.uni-karlsruhe.de/index.html<BR>
J&uuml;ngste &Auml;nderung 2004-11-25
<A HREF="mailto:webmaster@mvm.uka.de">Webmaster</A>

</BODY>
</HTML>
```

Quelle 1.4 : Quelltext eines einfachen HTML-Dokuments (Webseite)

Die erste Zeile ist HTML-Kommentar, beschreibt den Typ des Dokuments und kann vom Checker `tidy` erzeugt werden, falls man sie nicht selbst abfasst. Das eigentliche Dokument wird durch `<HTML>` und `</HTML>` eingerahmt. In seinem Inneren finden sich die beiden Teile `<HEAD>` und `<BODY>`. Die Formatanweisungen `<H3>` usw. markieren Überschriften (Header). Sonderzeichen werden entweder durch eine Umschreibung (entity) wie `&auml;` oder durch die Nummer im Latin-1-Zeichensatz wie `&#228;` dargestellt. `<BR>` ist ein erzwungener Zeilenumbruch

(break), <HR> eine waagrechte Linie (horizontal ruler). Am Ende sollte jedes Dokument seinen Uniform Resource Locator (URL) enthalten, damit man es wiederfindet, sowie das Datum der jüngsten Änderung und die Email-Anschrift des Verantwortlichen.

Zum Schreiben von HTML-Dokumenten (Webseiten) braucht man kein besonderes Programm; der einfachste Editor, der US-ASCII beherrscht, reicht aus. Einige Spezialeditoren können jedoch bei komplizierten Dokumenten die Arbeit erleichtern. Dazu zählen:

- `amaya`, der Referenz-Editor und -Browser des W3-Konsortiums, `http://www.w3.org/Amaya/`,
- `bluefish`, `http://bluefish.openoffice.nl/`,
- Phase5, `http://www.qhaut.de/`, setzt `wine` voraus,
- Arachnophilia, `http://www.arachnoid.com/`, setzt Java voraus,
- `tkhtml`, `http://www.hwaci.com/sw/tkhtml/`, setzt Tcl/Tk voraus,
- `nvu`, `http://nvu.com/`, abgeleitet aus dem Mozilla Composer und gepflegt von der Firma Linspire, unter der GPL,
- Netscape bzw. Mozilla Composer, `http://www.mozilla.org/`.

Hingegen sind die von Word, Access, Frontpage oder älteren Versionen des Composers erzeugten HTML-Dokumente oft nicht mit dem HTML-Standard konform und unnötig groß. Man sollte sie erst nach sorgfältiger Prüfung veröffentlichen. Die kommerziellen HTML-Editoren SoftQuad HotMetal, Macromedia Dreamweaver und Adobe GoLive sind nicht für Linux verfügbar.

Vor einem verbreiteten Fehler sei gewarnt. Die HTML-Kommandos beschreiben eine Struktur, nicht das Aussehen. Viele Kommandos lassen sich missbrauchen, was in manchen Zusammenhängen gut geht, in anderen nicht. Das
-Element erzeugt einen Zeilenumbruch, mehrere aufeinander folgende
-Elemente also Leerzeilen. Diese kann ich auch mittels mehrerer <P>-Elemente hervorrufen, die eigentlich zum Einrahmen von Absätzen oder Paragrafen gedacht sind. Abgesehen von der unterschiedlichen Syntax (Attribute etc.) kann ein Browser unter einem Paragrafen etwas anderes verstehen als einen Zeilenwechsel, man weiß das nie sicher. Beliebt ist der Missbrauch von Tabellen zur Darstellung von mehrspaltigem Text. Spätestens beim Lesen des Dokumentes mit einem zeilenweise arbeitenden Screen-Reader geht das voll daneben. Man nehme also immer das HTML-Element, das den eigenen Wunsch logisch genau wiedergibt, und nicht eines, das unter den gegenwärtigen, zufälligen Umständen den gewünschten Effekt auf dem Bildschirm erzeugt.

HTML-Dateien tragen die Kennung `html` oder `htm`, letztere in der auf Dateinamen nach dem 8.3-Muster beschränkten Welt (DOS und Nachfolger). Dokumente können Anweisungen an den Web-Server enthalten, die dieser bei einer Anfrage nach dem Dokument vor dem Senden der Antwort ausführt. Diese Server Parsed Documents oder Webseiten mit Server Side Includes tragen oft die Kennung `shtml`, aber das ist nicht zwingend. Beispiele sind Zähler oder bei jeder Anfrage aktualisierte Tabellen oder Datumsangaben. Im Gegensatz dazu stehen Dokumente mit Anweisungen an den Client (Browser).

Es ist dringend anzuraten, eine neue oder veränderte Webseite durch mindestens ein Prüfwerkzeug untersuchen (validieren) zu lassen. Das Kommando `tidy` ist ein einfach anzuwendender Syntaxchecker, der auf Wunsch auch gleich noch den Quelltext in eine ansprechende Form bringt (`http://www.w3.org/People/Raggett/tidy/` oder `http://sourceforge.net/projects/tidy/`). Der Aufruf:

```
joe@debian:~$ tidy -ium alex.html
```

rückt in der HTML-Datei `alex.html` die Zeilen ähnlich wie bei einer Programmquelle ein, schreibt alle Tags in Versalien und modifiziert die Originaldatei. Dabei werden kleine Fehler korrigiert und größere beanstandet. Eine zweite Möglichkeit besteht darin, die Webseite `http://validator.w3c.org/` um die Validierung zu bitten. Bei `http://jigsaw.w3.org/css-validator/` kann man etwaige Cascading Style Sheets (CSS) untersuchen lassen. Schließlich prüft `http://bobby.watchfire.com/bobby/` Webseiten auf Zugänglichkeit. Das Werkzeug `linklint` stellt fest, ob Hyperlinks funktionieren (`http://www.linklint.org/`). Darüber hinaus sollte man sich sein Produkt mit mindestens zwei unterschiedlichen Browsern ansehen. Das kostet Zeit, aber nur so ist zu gewährleisten, dass die Mehrzahl der Surfer keine Schwierigkeiten mit der Seite haben. Und das wollen Sie doch, oder?

Wer sich näher mit der Gestaltung von HTML-Dokumenten und Webseiten befassen will, sollte unbedingt die Website `http://de.selfhtml.org/` von STEFAN MÜNZ besuchen. Dort liegt ein umfassendes und ausgereiftes Werk in deutscher Sprache zu dem Thema. Aktuell ist die Version 8.1; das zip-Archiv zum Herunterladen umfasst 8,5 MB. Band 1 der zweibändigen Buchausgabe ist 1300 Seiten stark. Kürzer ist die *HTML-Einführung* des durch seine LaTeX-Kurzeinführung bekannten Autors HUBERT PARTL. Darüber hinaus sind im Netz weitere Anleitungen und die ausführliche, verbindliche Spezifikation von HTML verfügbar, diese bei `http://www.w3c.org/` aktuell in der Version 4.01.

1.12.3 Standard Generalized Markup Language (SGML)

Texte, Bilder und Tabellen werden von HTML gut unterstützt. Zum Schreiben von mathematischen Formeln sind zwar im HTML-Standard Wege vorgesehen, die sich an LaTeX anlehnen, die gängigen HTML-Browser geben jedoch die Formeln nicht wieder, sodass manche Autoren getrennte LaTeX- und HTML-Fassungen ihrer Manuskripte herstellen (müssen). Aus dieser unbefriedigenden Lage schafft einen Ausweg die Standard Generalized Markup Language (SGML) nach ISO 8879, eine Sprache (Metasprache) zur Beschreibung von Sprachen wie HTML oder LaTeX, deren Anfänge in den sechziger Jahren[7] von CHARLES GOLDFARB bei IBM entwickelt wurden. Zugespitzt lässt sich SGML als die Mutter aller Markup-Sprachen bezeichnen. Man verfasst einen Text mit einer Formatierungssprache (Markup Language) eigener

[7] Die sechziger Jahre waren ein sehr fruchtbares Jahrzehnt. Viele Dinge, die heute in der Computerei Allgemeingut sind, wurden in dieser Zeit erdacht.

Wahl und vielleicht sogar eigener Zucht. In einem zweiten Dokument namens Document Type Definition (DTD) beschreibt man dazu in SGML, was die Konstrukte der Markup Language bedeuten. Eine weiterverarbeitende Anwendung muss die Sprachdefinition verstehen, das heißt SGML beherrschen, und kann dann den Inhalt des Dokumentes verarbeiten. Ein SGML-Browser erzeugt aus Text und DTD nach Wunsch ASCII-, LaTeX- oder HTML-Vorlagen für die jeweiligen Zwecke. Dieses allgemeine Konzept läuft proprietären WYSIWYG-Textprozessoren zuwider. Es ist zunächst aufwendiger, aber langfristig effektiver. Einzelheiten wie immer im Netz, Einstieg bei Wikipedia, Suchwort *SGML*, und `http://www.w3.org/MarkUp/SGML/`.

1.12.4 Extensible Markup Languages (XML, XHTML)

SGML ist für manche Aufgaben ein zu schweres Werkzeug. Andererseits möchte man gelegentlich mehr machen, als HTML erlaubt. Hier hilft die Extensible Markup Language (XML) mit Möglichkeiten, eigene Tags (Formatieranweisungen) zu definieren. Man könnte XML als eine abgespeckte SGML bezeichnen, als eine Teilmenge. Ehe man an XML oder SGML geht, sollte man ein einwandfreies HTML-Dokument und Style Sheets (CSS) schreiben können, was nicht bedeutet, alle Feinheiten von HTML zu beherrschen. Obwohl sich XML-Dokumente mit jedem Editor schreiben lassen, gibt es auch spezielle XML-Editoren wie `kxmleditor` aus dem KDE-Projekt. Bei `http://www.conglomerate.org/` entsteht ein XML-Editor, der unter der GPL verfügbar ist. Einen Umwandler von XML nach mehreren anderen Formaten enthält das Debian-Paket *xmlto*. Einzelheiten ebenfalls im Netz, Einstieg bei Wikipedia, Suchwort *Extensible Markup Language*, und `http://www.w3.org/XML/`. Auch STEFAN MÜNZ (siehe oben) erklärt XML. Bei allem, was mit XML zu tun hat, ist gegenwärtig viel im Fluss.

Die Extensible Hypertext Markup Language (XHTML) ist die Weiterentwicklung von HTML 4.01 und dabei, diese abzulösen. Die Sprache wird mittels XML anstelle von SGML beschrieben. Die Umwandlung von HTML-Dokumenten nach XHTML ist einfach; vor allem sind strengere syntaktische Vorgaben zu befolgen. Ein Web-Browser, der XHTML noch nicht versteht, zeigt XHTML-Dokumente wie gewöhnliche HTML-Dokumente an, sodass ein Übergang im Web gewährleistet ist. Ein deutschsprachiges Tutorial zu XHTML von JAN WINKLER liegt unter `http://www.html-world.de/program/xhtml_ov.php`.

1.12.5 Web-Entwicklungsumgebung (quanta)

Quanta Plus ist eine Web-Entwicklungsumgebung für KDE. Sie hat nichts mit kommerziellen Produkten ähnlichen Namens gemein. Die Manualseite hilft kaum weiter; die wesentliche Dokumentation liegt im DocBook-Format im Verzeichnis `usr/share/doc/kde/HTML/en/quanta/` und ist im KDE Help Center unter den *Application Manuals*, Punkt *Development* zu finden. Der Browser Konqueror sollte mit dem URL `help:/quanta/` die Dokumentation anzeigen, kam aber über das Auflisten des Verzeichnisses nicht hinaus, vermutlich weil ihm ein nicht näher bekanntes Plugin fehlte. Eine Konvertierung der DocBook-Dateien

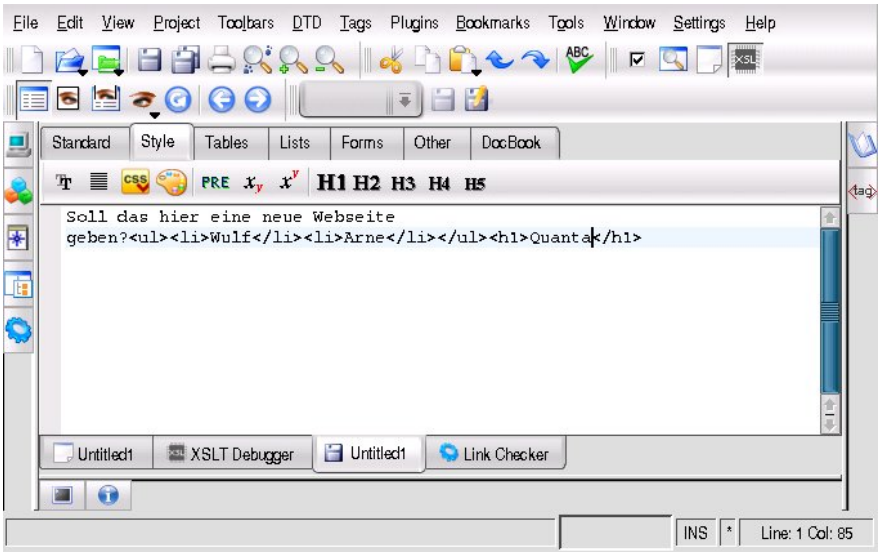

Abb. 1.14: Screenshot der Web-Entwicklungsumgebung Quanta Plus aus dem KDE-Projekt, ein schwerer Hammer für umfangreiche Web-Projekte

nach HTML oder PDF scheiterte an Syntaxfehlern. Die beste Hilfe findet man auf `http://quanta.kdewebdev.org/`.

Das Werkzeug ist gedacht zum Gestalten eines Projektes aus vielen Webseiten durch eine Gruppe von Entwicklern. Als einfacher HTML-Editor für den Einzelkämpfer im stillen Kämmerlein ist es zu umfangreich. Bevor die erste Webseite das Licht der Welt erblickt, ist das Projekt anzulegen, sind globale, projektbezogene und persönliche Vorlagen zu erstellen, Zugriffsrechte zu vergeben usw. Das übt man möglichst an einem kleinen Versuchsprojekt.

1.13 Ergänzende Textwerkzeuge

1.13.1 Tipptrainer (ktouch, tipptrainer, tuxtype)

Beim Arbeiten mit Tastaturen soll man alle Finger benutzen. Das Schreiben geht flotter, und die Last verteilt sich gleichmäßiger. Wer sich die Kunst des Schreibens mit zehn Fingern selbst beibringen oder auch nur üben will, greift zu einem Tipptrainer. Das KDE-Werkzeug KTouch erlaubt die Wahl einer Sprache und einer Tastatur und gibt dann Zeichenfolgen vor, die der Übende abschreiben soll, wobei jeder Fehler beanstandet und für die Statistik gezählt wird. Abbildung 1.15 zeigt das Aussehen. Eine Alternative enthält das Debian-Paket *tipptrainer*, zu Hause im PingoS-Projekt (`http://www.pingos.org/`). Kinder werden eher von dem Kommando und Debian-Paket *tuxtype* angesprochen, auf dessen Schirm Buchstabenfische vom Him-

mel fallen, leider nur mit englischem Text, ansonsten nett gemacht. Weitere Tipptrainer bietet Sourceforge unter dem Suchbegriff *typing tutor* an.

Abb. 1.15: Screenshot des KDE-Tipptrainers KTouch

1.13.2 Vergleichen (diff, cmp, ed)

Für einzelne Aufgaben der Textverarbeitung gibt es Spezialwerkzeuge in Linux/UNIX. Häufig gebraucht werden das bereits erläuterte `grep(1)` (= global regular expression print). Mittels `diff` werden die alte und die neue Version einer Datei miteinander verglichen. Bei entsprechendem Aufruf wird eine dritte Datei erzeugt, die dem historischen Editor `ed` als Kommandoskript (ed-Skript) übergeben werden kann, sodass dieser aus der alten Version die neue erzeugt. Gebräuchlich zum Aktualisieren von Programmquellen. Schreiben Sie sich eine kleine Textdatei `alt`, stellen Sie eine Kopie namens `neu` davon her, verändern Sie diese und rufen Sie dann `diff` auf:

```
joe@debian:~$ diff -e alt neu > edscript
```

Fügen Sie mit einem beliebigen Editor am Ende des `edscript` zwei Zeilen mit den `ed`-Kommandos `w` und `q` (write und quit) hinzu. Dann rufen Sie den Editor `ed` mit dem Skript auf:

```
joe@debian:~$ ed - alt < edscript
```

Anschließend vergleichen Sie mit dem simplen Kommando `cmp` die beiden Versionen `alt` und `neu` auf Unterschiede:

```
joe@debian:~$ cmp alt neu
```

Durch den `ed`-Aufruf sollte die alte Version genau in die neue Version überführt worden sein, `cmp` meldet nichts. Das Werkzeug `diff3` vergleicht drei Textdateien zeilenweise.

1.13.3 Sortieren (sort)

Über das Sortieren sind Bücher geschrieben worden, ein bekanntes von DONALD E. KNUTH. Für einfache Sortieraufgaben steht das Werkzeug `sort` zur Verfügung. Per Option teilt man ihm mit, ob alphabetisch oder numerisch, aufsteigend oder absteigend und nach welchen Feldern ein Text zeilenweise sortiert werden soll. Das Kommando wird oft als Glied einer Pipe eingesetzt, siehe das Shellskript `frequenz` auf Seite **??**.

Zur Übung schreiben wir eine unsortierte, zweispaltige Liste mit Familiennamen und Telefonnummern. Die Datei namens `liste` soll auch einige mehrfache (identische) Einträge enthalten, beliebig verstreut. Dann bearbeiten wir die Datei mit `sort` und schauen uns das Ergebnis auf dem Schirm an:

```
joe@debian:~$ sort liste

joe@debian:~$ sort -u liste

joe@debian:~$ sort -r liste

joe@debian:~$ sort --key=2 liste
```

Der erste Aufruf sortiert die Liste mit den Defaulteinstellungen, also die Zeilen alphabetisch aufsteigend. Der zweite macht dasselbe und wirft dabei mehrfache Zeilen raus (u = unique). Der dritte sortiert in umgekehrter Folge (absteigend, r = reverse), der vierte schließlich sortiert nach dem zweiten Feld (Telefonnummer), wobei die Feldzählung mit 1 beginnt und als Trennzeichen das Leerzeichen oder TAB angenommen wird. Man kann sogar angeben, ab welchem Zeichen im Feld der Sortierschlüssel beginnen soll, und wo er endet. Die zahlreichen Optionen erlauben die Anpassung an fast jede Sortieraufgabe. Das Locale sollte von `sort` beachtet werden.

1.13.4 Rechtschreibung prüfen (ispell, aspell, acheck)

Das alte UNIX-Werkzeug `spell` untersuchte Texte auf Verstöße gegen die nordamerikanische Rechtschreibung. Das aus dem GNU-Projekt stammende jüngere Werkzeug `ispell` gibt sich international. Als Debian-Paket findet sich noch ein Werkzeug `aspell`, das behauptet, intelligenter zu sein als `ispell`, eine Kombination aus `ispell` und Metaphone. Beide bringen keine Wörterbücher mit; diese sind getrennt einzurichten, beispielsweise für Bretonisch oder Bengali. Auch

ein deutsches Wörterbuch gibt es. Beim Korrigieren lernen sie mit Unterstützung des Benutzers neue Wörter und speichern sie in einem persönlichen Wörterbuch (`$HOME/.ispell_default` bzw. `$HOME/.aspell.de.pws`). Die persönlichen Wörterbücher lassen sich editieren; man sollte sich aber darauf beschränken, offensichtliche Fehleinträge zu löschen. Grafische Frontends gibt es auch, um überschüssige Prozessorkraft zu verbraten. Die Werkzeuge lassen sich in andere Programme wie Mail User Agents oder den Emacs einbinden, sodass sie aus diesen heraus aufgerufen werden können.

Falls `ispell` mit einem deutschen Wörterbuch als Default eingerichtet ist, untersucht der Aufruf:

```
joe@debian:~$ ispell kapitel5.tex
```

die Datei `kapitel5.tex` auf Verstöße gegen die deutsche Rechtschreibung, wobei das Programm auf Grund der Dateikennung `tex` automatisch von einer LaTeX-Datei ausgeht und Formatanweisungen überliest. Hat man die Debian-Pakete `aspell` und `aspell-de` eingerichtet, so lässt sich mittels:

```
joe@debian:~$ aspell check --lang=german -t
kapitel5.tex
```

die gleiche Untersuchung wie vorstehend durchführen. `aspell` zeigt bei Korrekturvorschlägen etwas mehr Phantasie als `ispell`, kommt aber ohne weitere Konfiguration mit den deutschen Umlauten in LaTeX nicht zurecht. Kein Rechtschreibprüfer erkennt Fehler, die das Verständnis eines Satzes erfordern (Maße vs. Masse, Tücher vs. Tuche, falscher Artikel, falsche Verbform, fehlendes Wort, falsche Wortfolge).

Das Werkzeug `acheck` zusammen mit dem Paket *acheck-rules* prüft die Lokalisation (Schreibweise von Datum etc.) eines Textes nach gegebenen Regeln und findet auch einige grammatische oder stilistische Fehler. Mit der Option `-s` zieht es gleichzeitig `aspell` zur Rechtschreibprüfung heran. Da sein Autor Franzose ist, gibt es auch einen französischen Regelsatz.

1.13.5 Weitere Werkzeuge (cut, paste usw.)

Weitere Werkzeuge, deren Syntax man im Manual nachliest, sollen hier nur tabellarisch aufgeführt werden:

- `bfs` big file scanner, untersucht große Textdateien auf Muster,
- `col` filtert Backspaces und Reverse Line Feeds heraus,
- `comm` common, vergleicht zwei sortierte Dateien auf gemeinsame Zeilen,
- `cut` schneidet Spalten aus Tabellen heraus,
- `expand/unexpand` wandelt Tabs in Leerzeichen um und umgekehrt,
- `fold` faltet lange Zeilen (bricht Zeilen um),
- `hyphen` findet Zeilen, die mit einem Trennstrich enden,
- `nl` number lines, nummeriert Zeilen,
- `paste` mischt Dateien zeilenweise,
- `ptx` permuted index, erzeugt einen permutierten Index,

- `rev` reverse, mu nelieZ trhek,
- `rmnl` remove newlines, entfernt leere Zeilen,
- `ssp` entfernt mehrfache leere Zeilen,
- `tr` translate, ersetzt Zeichen,
- `uniq` findet wiederholte Zeilen in einer sortierten Datei,
- `vis` zeigt eine Datei an, die unsichtbare Zeichen enthält,
- `wc` word counter, zählt Zeichen, Wörter, Zeilen.

Die Liste lässt sich durch eigene Werkzeuge beliebig erweitern. Das können Programme oder Shellskripte sein. Hier ein Beispiel zur Beantwortung einer zunächst anspruchsvoll erscheinenden Fragestellung mit einfachen Mitteln. Ein Sachtext soll nicht unnötig schwer zu lesen sein, die Sachzusammenhänge sind schwierig genug. Ein grobes Maß für die Lesbarkeit eines Textes ist die mittlere Satzlänge. Erfahrungsgemäß sind Werte von zehn bis zwölf Wörtern pro Satz für deutsche Texte zu empfehlen. Wie kann eine Pipe aus Linux/UNIX-Werkzeugen diesen Wert ermitteln? Schauen wir uns das Vorwort an. Als erstes wären die LaTeX-Konstrukte herauszuwerfen. Hierfür gibt es ein Programm `detex` bzw. `delatex`, allerdings nicht bei Debian, sondern als Tarball bei `http://www.cs.purdue/edu/homes/trinkle/detex/`. Leider brach die Übersetzung mit zwei Parserfehlern ab, die nicht auf die Schnelle behoben werden konnten. Auch ein Werkzeug `untex` soll es geben. Wir lassen die Konstrukte drin. Dann sind Zeilenwechsel durch Leerzeichen zu ersetzen sowie Ziffern und einige Satzzeichen zu entfernen – Werkzeug `tr`. Schließlich muss jeder Satz in einer eigenen Zeile stehen. Wir ersetzen dazu alle Punkte durch Line Feeds. Ein Fehler entsteht dadurch, dass Punkte nicht nur ein Satzende markieren, aber bei einem durchschnittlichen Text ist dieser Fehler gering. Schicken wir den so aufbereiteten Text durch das Werkzeug `wc`, so erhalten wir die Anzahl der Zeilen gleich Anzahl der Sätze, die Anzahl der Wörter (wobei ein Wort eine maximale Zeichenkette begrenzt durch Leerzeichen, Tabs oder Line Feeds ist) und die Anzahl der Zeichen im Text. Das Werkzeug `tee` ist nicht notwendig und schreibt nur den veränderten Text zur Kontrolle in eine temporäre Datei. Die Pipe sieht so aus:

```
cat $1 | tr '\012' '\040' | tr -d '[0-9],;"()' |
tr '.' '\012' | tee $1.tmp | wc
```

Quelle 1.5 : Shellscript zur Stilanalyse

Die Anzahl der Wörter geteilt durch die Anzahl der Sätze liefert die mittlere Satzlänge. Die Anzahl der Zeichen durch die Anzahl der Wörter ergibt die mittlere Wortlänge, infolge der Leerzeichen am Wortende erhöht um 1. Auch das ist ein Stilmerkmal. Die Ergebnisse für das Vorwort sind 45 Sätze, 712 Wörter und 5197 Zeichen, also eine mittlere Satzlänge von 15,8 Wörtern und eine mittlere Wortlänge ohne Leerzeichen von 6,3 Zeichen pro Wort. Zählt man von Hand nach, kommt man auf 36 Sätze; 9 Punkte markieren nicht ein Satzende. Man müsste das Satzende genauer definieren. Die nächste Verbesserung des Verfahrens wäre, nicht nur die Mittelwerte, sondern auch die Streuungen zu bestimmen. Hierzu wäre der `awk` zu bemühen oder

gleich ein C-Programm zu schreiben. Das Programm liefert nur Zahlen; ihre Bedeutung erhalten sie, indem man sie zu Erfahrungswerten in Beziehung setzt. Soweit sich Stil durch Zahlen kennzeichnen lässt, hilft der Rechner; wenn das Verständnis von Wörtern, Sätzen oder noch höheren Einheiten verlangt wird, ist heutige Software überfordert.

Es soll ein Linux/UNIX-Kommando `style` geben, das den Stil eines englischen Textes untersucht und Verbesserungen vorschlägt. Dagegen ist das Kommando `diplom`, das nach Eingabe eines Themas und einer Seitenanzahl eine Diplomarbeit schreibt – mit `ispell` und `style` geprüft – noch nicht ausgereift.

1.14 Verschlüsselung

1.14.1 Aufgaben der Verschlüsselung

Auf einem Linux/UNIX-System kann der Verwalter auf jede Datei zugreifen, auf MS Windows mit Einschränkungen auch. Das Netz ist mit einfachen Mitteln unauffällig abzuhören. Will man seine Daten vor Unbefugten schützen, hilft nur Verschlüsseln. Man darf aber nicht vergessen, dass bereits die Analyse des Datenverkehrs einer Quelle oder eines Ziels Informationen liefert. Wer ganz unbemerkt bleiben will, muss sich mehr einfallen lassen als nur eine Verschlüsselung.

Eng verwandt mit der Verschlüsselung (E: encryption, F: cryptage, chiffrement) ist die Authentifizierung oder Authentisierung (E: authentication, F: authentification). Diese Aufgabe behandeln wir im Abschnitt **??** *Electronic Mail* auf Seite **??**, weil sie dort eine Rolle spielt. Hier geht es nur darum, einen Text oder auch andere Daten für Unbefugte unbrauchbar zu machen; für Befugte sollen sie natürlich brauchbar bleiben.

Das Ganze ist heute eine Wissenschaft und heißt Kryptologie. In den letzten Jahrzehnten hat sie einen stark mathematischen Einschlag bekommen. Sie bietet einen gewissen Unterhaltungswert, insbesondere die Kryptanalyse, der Versuch, Verschlüsselungen zu knacken.

Die zu verschlüsselnden Daten nennen wir Klartext (E: plain text, F: texte en clair), die verschlüsselten Daten Geheimtext (E: cipher text, F: cryptogramme).

1.14.2 Symmetrische Verfahren

Im einfachsten Fall wird jedes Zeichen des Klartextes nach einer Regel durch ein anderes Zeichen desselben Alphabets ersetzt. Die einfachste Regel ist die Verschiebung um eine feste Anzahl von Stellen im Alphabet, beispielsweise um $+3$ Stellen. Aus A (Zeichen Nr. 1) wird D (Zeichen Nr. $1 + 3$). Dieses Verfahren soll CAIUS JULIUS CAESAR benutzt haben. Er vertraute auf die Dummheit seiner Gegner. Zum Entschlüsseln des Geheimtextes nimmt man dasselbe Verfahren mit -3 Stellen. Wählt man eine Verschiebung um 13 Stellen, so führt bei einem Alphabet mit 26 Zeichen eine Wiederholung der Verschlüsselung zum Klartext zurück. Dieses Verfahren ist unter dem Namen ROT13 bekannt und wird im Netz verwendet, um einen Text

– beispielsweise die Auflösung eines Rätsels – zu verfremden. ROT26 gilt als die schwächste aller Verschlüsselungen des lateinischen Alphabets. Man kann die Verfahren raffinierter gestalten, indem man Zeichengruppen verschlüsselt, Blindzeichen unter den Geheimtext mischt, die Algorithmen wechselt usw.

Seit 1970 unterscheidet man zwei Gruppen von Verschlüsselungsverfahren:

- Symmetrische Verfahren (Private-Key-V.),
- Asymmetrische Verfahren (Public-Key-V.).

Dazu kommen für bestimmte Aufgaben noch Einweg-Hash-Verfahren. Symmetrische Verfahren arbeiten schneller als asymmetrische. Bei den symmetrischen Verfahren kennen Sender und Empfänger neben dem Algorithmus sowohl den Chiffrier- wie den Dechiffrierschlüssel. Beide Schlüssel sind identisch oder voneinander ableitbar. Da der Algorithmus kaum geheim zu halten ist, beruht die Sicherheit auf dem Schlüssel, der nicht zu simpel sein darf und geheim bleiben muss. Das Problem liegt darin, den Schlüssel vom Sender zum Empfänger zu schaffen. Das geht nur über einen vertrauenswürdigen Kanal, also nicht über Email. Treffen Sie Ihren Brieffreund gelegentlich bei Kaffee und Kuchen, können Sie ihm einen Zettel mit dem Schlüssel zustecken. Wohnen Sie in Karlsruhe, Ihr Briefpartner in Fatmomakke, wird der Schlüsselaustausch aufwendiger. Ein weiteres Problem liegt in der Anzahl der benötigten Schlüssel beim Datenverkehr unter mehreren Beteiligten. Geht es nur darum, die eigenen Daten vor dem Superuser zu verbergen, ist kein Schlüsselaustausch nötig und daher ein symmetrisches Verfahren angebracht.

Die Verschlüsselung nach dem weit verbreiteten Data Encryption Standard (DES) gehört in diese Gruppe, zur Ver- und Entschlüsselung wird derselbe Schlüssel benutzt. DES wurde von IBM entwickelt und 1977 von der US-Regierung als Standard angenommen. Es gilt heute nicht mehr als sicher, Triple-DES ist besser. Weitere Mitglieder dieser Gruppe sind IDEA, Blowfish und CAST5.

Unter Linux/UNIX stehen ein Kommando `crypt` sowie eine gleichnamige C-Standardfunktion zur Verfügung, die ein nicht sehr ausgefeiltes symmetrisches Verfahren verwenden (DES oder MD5). Die Passwörter in `/etc/passwd` werden damit verschlüsselt. Jünger und vielseitiger ist `mcrypt` (`http://mcrypt.sourceforge.net/`). Man ver- und entschlüsselt mittels des Kommandos:

```
joe@debian:~$ crypt < eingabe > ausgabe
```

Das Kommando fragt nach einem Schlüssel. Dieser wird für beide Richtungen eingesetzt. Der Klartext ist erforderlichenfalls gesondert zu löschen (physikalisch, nicht nur logisch, das heißt zu überschreiben). Die Crypt Breaker's Workbench enthält alles Nötige, um diese Verschlüsselung zu knacken (`http://axion.physics.ubc.ca/cbw.html`).

1.14.3 Asymmetrische Verfahren

Die asymmetrischen Verfahren verwenden zum Verschlüsseln und Entschlüsseln zwei völlig verschiedene, nicht voneinander ableitbare Schlüssel. Benutzer A hat sich ein Paar zusammengehöriger Schlüssel gebastelt, den ersten zum Verschlüsseln,

den zweiten zum Entschlüsseln, wie, werden wir noch sehen. Den ersten Schlüssel gibt er öffentlich bekannt, daher Public Key. Jeder kann ihn benutzen, zum Beispiel Benutzer B, der A eine vertrauliche Mail schicken möchte. Was einmal damit verschlüsselt ist, lässt sich nur noch mit dem zweiten Schlüssel entschlüsseln, und den hält Benutzer A geheim. Er teilt ihn niemandem mit, daher Private Key.

Jetzt kann es nur noch passieren, dass ein Benutzer C unter Missbrauch des Namens von B an A eine beleidigende Mail schickt und B darauf hin mit A Krach bekommt. Veröffentlicht A den Dechiffrierschlüssel und behält den Chiffrierschlüssel für sich, kann er chiffrierte Texte versenden, die jeder entschlüsseln und lesen kann, wobei die Texte nur von A chiffriert sein können. Das ist die Lösung des Authentifizierungs-Problems, auf das wir im Abschnitt **??** *Email, Listen* auf Seite **??** eingehen.

Wie kommt man nun zu einem derartigen Schlüsselpaar? Ein Weg beruht auf der Tatsache, dass man leicht zwei ganze Zahlen großer Länge miteinander multiplizieren kann, sogar ohne Rechner, während die Zerlegung einer großen Zahl (um die vierhundert dezimale Stellen entsprechend etwa 1000 Bits) in ihre Primfaktoren mit den heute bekannten Algorithmen und Rechnern aufwendig ist, jedenfalls wenn gewisse Voraussetzungen eingehalten werden. RON RIVEST, ADI SHAMIR und LEONARD ADLEMAN haben auf diesem Gedanken aufbauend das verbreitete RSA-Verfahren entwickelt.

Man wähle zufällig zwei große Primzahlen p und q, zweckmäßig von annähernd gleicher Länge. Ihr Produkt sei $n = pq$. Weiter wähle man eine Zahl e so, dass e und $(p-1)(q-1)$ teilerfremd (relativ prim) zueinander sind. Eine vierte Zahl d berechne man aus:

$$d = e^{-1} \; mod \; ((p-1)(q-1)) \tag{1.1}$$

Die Zahlen e und n bilden den öffentlichen Schlüssel, die Zahl d ist der private, geheime Schlüssel. Die beiden Primzahlen p und q werden nicht weiter benötigt, müssen aber geheim bleiben (löschen).

Wir sehen den Klartext K als eine Folge von Ziffern an. Er wird in Blöcke K_i kleiner n aufgeteilt. Die Geheimnachricht G besteht aus Blöcken G_i, die sich nach

$$G_i = K_i^e \; mod \; n \tag{1.2}$$

berechnen. Zur Entschlüsselung berechnet man

$$K_i = G_i^d \; mod \; n \tag{1.3}$$

Einzelheiten und Begründung hierzu siehe die Bücher von FRIEDRICH L. BAUER oder BRUCE SCHNEIER, ersterer ein deutscher Computerpionier, letzterer ein bekannter Fachmann für Computer-Sicherheit. Nun ein Beispiel aus einem Buch von F. L. BAUER. Wir wählen einen Text aus lateinischen Buchstaben samt Zwischenraum und ersetzen die Zeichen durch die Nummern von 00 bis 26. Er bekommt folgendes Aussehen:

$$K = 051818011805000821 \ldots \tag{1.4}$$

Wir wählen:

$$p = 47 \qquad q = 59 \qquad n = p * q = 2773 \tag{1.5}$$

und teilen den Klartext in vierziffrige Blöcke kleiner n auf:

$$K_1 = 0518 \qquad K_2 = 1801 \qquad K_3 = 1805 \ldots \tag{1.6}$$

Zur Bestimmung von e berechnen wir:

$$(p-1)(q-1) = 46 * 58 = 2668 \tag{1.7}$$

Die Zahl 2668 hat die Teiler 2, 4, 23, 29, 46, 58, 92, 116, 667 und 1334. Für e wählen wir 17, teilerfremd zu 2668. Dann ergibt sich d zu:

$$d = 17^{-1} \; mod \; 2668 \tag{1.8}$$

Diese vielleicht unbekannte Schreibweise ist gleichbedeutend damit, ein Paar ganzer Zahlen d, x so zu bestimmen, dass die Gleichung:

$$d * 17 = 2668 * x + 1 \tag{1.9}$$

erfüllt ist. Die Zahl $d = 157$ ist eine Lösung mit $x = 1$. Gezielt ermittelt man Lösungen mittels des Erweiterten Euklidischen Algorithmus. Nun haben wir mit n, e und d alles, was wir brauchen und gehen ans Verschlüsseln:

$$G_1 = K_1^e \; mod \; n = 0518^{17} \; mod \; 2773 = 1787 \tag{1.10}$$

und entsprechend für die weiteren Blöcke. Gleiche Klartextblöcke ergeben gleiche Geheimtextblöcke, was bereits eine Schwäche ist. Zum Entschlüsseln berechnet man:

$$K_1 = G_1^d \; mod \; n = 1787^{157} \; mod \; 2773 = 518 \tag{1.11}$$

und so weiter. Die Arithmetik großer Ganzzahlen ist für Rechner kein Problem. Man kann sie sogar in Silizium gießen und erhält schnelle Chips zum Ver- und Entschlüsseln, ohne Software bemühen zu müssen. Da n und e öffentlich sind, könnte man durch Zerlegen von n in seine Primfaktoren leicht den privaten Schlüssel d ermitteln, aber das Zerlegen großer Zahlen ist nach heutigem Wissensstand aufwendig.

Es gibt weitere asymmetrische Verfahren wie das von TAHER ELGAMAL. Wird das Dokument symmetrisch verschlüsselt und der dazu erforderliche Schlüssel asymmetrisch verschlüsselt mitgeteilt, spricht man von hybriden Verfahren. Auf `http://www.rsa.com/` findet sich Material zur Vertiefung des Themas. Eine zehnteilige FAQ-Sammlung zur Kryptografie liegt im Netz.

1.14.4 Angriffe (Kryptanalyse)

Angriffe auf verschlüsselte Daten – wissenschaftlich als Kryptanalyse, sonst als Cracking bezeichnet – gehen möglichst von irgendwelchen bekannten oder vermuteten Zusammenhängen aus. Das kleinste Zipfelchen an Vorkenntnissen kann entscheidend sein[8]. Die Wahrscheinlichkeit, dass ein Benutzer seinen nur gering modifizierten Benutzernamen als Passwort verwendet, ist leider hoch. Damit fängt man an. Das

[8] Beim Knacken von Enigma spielte eine Rolle, dass der Gegner wusste, ein Buchstabe wurde niemals durch sich selbst verschlüsselt.

Ausprobieren aller nur möglichen Schlüssel wird *Brute Force Attack* genannt und ist bei kurzen Schlüsseln dank Rechnerhilfe auch schnell von Erfolg gekrönt. Das Faktorisieren kleiner Zahlen ist ebenfalls kein Problem. Aber selbst bei großen Zahlen, die für einen einzelnen Rechner – auch wenn er zu den schnellsten gehört – eine praktisch unlösbare Aufgabe darstellen, kommt man in kurzer Zeit zum Ziel, wenn man die Leerlaufzeiten von einigen Tausend durchschnittlichen Rechnern für seinen Zweck einsetzen kann. Das ist ein organisatorisches Problem, kein mathematisches, und bereits gelöst, siehe `http://www.distributed.net/rc5/`. Das ganze Nachdenken über sichere Verschlüsselung erübrigt sich bei schlampigem Umgang mit Daten und Schlüsseln. Der Benutzer ist erfahrungsgemäß das größte Risiko.

1.14.5 Einrichten des GNU Privacy Guards (GnuPG)

Um mit dem Softwarepaket GNU Privacy Guard zu arbeiten, braucht man die Debian-Pakete *gnupg* und *gnupg-doc* (Verwalter). Unter `/usr/share/doc/gnupg/` und `/usr/share/doc/gnupg-doc/` findet sich genügend Lesestoff für ein Wochenende, insbesondere das *GNU Privacy Handbook* in mehreren Sprachen – auch Deutsch – und Formaten, rund 60 Seiten. Um auszuprobieren, ob der Guard eingerichtet ist, ruft man das Manual mit `man gnupg` auf.

Als erstes erzeugen wir uns ein Schlüsselpaar mittels der Eingabe:

```
joe@debian:~$ gpg --gen-key
```

wobei wir die Fragen mit den Defaultwerten beantworten. Gegen Ende der Generierung wird nach dem bürgerlichen Namen des Benutzers gefragt, worauf etwas wie:

```
"Wulf Alex"
```

mit Anführungszeichen anzugeben ist. Dann will das Programm noch die Emailanschrift wissen:

```
alex-weingarten@t-online.de
```

und einen nicht zu langen Kommentar, beispielsweise ob es sich um die privaten oder dienstlichen Schlüssel handelt. Abschließend ist eine Passphrase – ein langes Passwort, etwas kürzer als ein Roman – einzugeben, die man sich unter Ausnutzung aller ASCII-Zeichenarten einfallen lässt, aufschreibt und an einem sicheren Ort verwahrt. Wir brauchen die Passphrase später unter anderem beim Verschlüsseln von Mails. Gelegentlich wird eine Passphrase auch Mantra genannt. Bei Erfolg erscheint eine Meldung, die man sich anschaut, aber nicht merkt. Im Verzeichnis `$HOME/.gnupg` ist das Ergebnis gespeichert, von dem man möglichst bald eine Sicherungskopie (Backup) zieht. Zugriffsrechte 700 für das Verzeichnis, 600 für die Dateien. Ein Verlust des privaten Schlüssels in `secring.gpg` führt dazu, dass man die Sicherheitsmechanismen nicht mehr benutzen kann, beispielsweise verschlüsselte Mails nicht mehr entziffern kann. Gelangt er in falsche Hände, stehen dem Missbrauch Tür und Tor offen. Den privaten Schlüssel und das unten erläuterte Widerrufszertifikat sollte man auf CD und auf Papier speichern und sicher aufbewahren. Nun können wir ein bisschen herumspielen:

```
joe@debian:~$ gpg --list-sigs
joe@debian:~$ gpg --list-secret-keys
joe@debian:~$ gpg --fingerprint
joe@debian:~$ gpg --edit-key "Wulf Alex"
joe@debian:~$ gpg --edit-key alex
```

Aus letzterem Kommando kommt man mit <q> (quit) heraus. Zur Weitergabe des öffentlichen Schlüssels ist dieser zu exportieren:

```
joe@debian:~$ gpg --export --armor
```

Den ausgegebenen Zeichensalat darf man jedermann anbieten, beispielsweise indem man ihn auf der eigenen Webseite veröffentlicht.

Umgekehrt brauchen wir die öffentlichen Schlüssel unserer Freunde, die samt unserem eigenen in der Datei `$HOME/.gnupg/pubring.gpg` (Schlüsselbund, E: key ring) gesammelt werden. Angenommen, wir hätten eine Datei namens `key.gpg` mit einem fremden öffentlichen Schlüssel bekommen, dann ist diese zu importieren:

```
joe@debian:~$ gpg --import key.gpg
```

Öffentliche Schlüssel lassen sich auch von einem Keyserver im Netz wie `http://wwwkeys.de.pgp.net/` herunterladen. Gibt man dort als Suchstring `peterh.ganten` ein, erhält man eine mehrzeilige Antwort. Die Zeilen mit anderen Zeitgenossen beglaubigen den voranstehenden Schlüssel und interessieren uns nicht im Einzelnen. Durch Anklicken eines Schlüssel-Identifiers bringt man den Schlüssel auf den Schirm, wo er uns nicht viel nützt. Mit dem Schlüssel-Identifier und dem Kommando:

```
joe@debian:~$ gpg --keyserver wwwkeys.de.pgp.net
--recv-keys identifier
```

importiert man den Schlüssel in die Datei `pubring.gpg`, den öffentlichen Schlüsselbund.

Will man seinen eigenen öffentlichen Schlüssel auf den genannten Keyserver hochladen, so erzeugt man vorsichtshalber noch ein Widerrufszertifikat:

```
joe@debian:~$ gpg --gen-revoke "Wulf Alex"> revoke
```

und fügt in der Datei `gpg.conf` gegen Ende folgende Zeile hinzu:

```
keyserver x-hkp://pgpkeys.pca.dfn.de
```

Dann verschickt man den Schlüssel mittels:

```
joe@debian:~$ gpg --send-keys
```

am einfachsten, bevor man fremde öffentliche Schlüssel heruntergeladen hat. Andernfalls ist der eigene Name in Anführungszeichen als Argument mitzugeben. Anschließend kann man den eigenen Schlüssel wie oben beschrieben suchen. Jetzt sind nur noch die Anwendungen von den neuen Möglichkeiten zu benachrichtigen. Ein Beispiel findet sich in Abschnitt **??** *Email, Listen* auf Seite **??**.

1.15 Organisation eines Textprojektes

1.15.1 Verzeichnisstruktur

Einfache Texte wie Briefe oder Übungsblätter erfordern keine besondere Verzeichnisstruktur. Ihre Anzahl kann eine flache Verzeichnishierarchie zweckmäßig erscheinen lassen, mehr nicht. Ein Textprojekt (Diplomarbeit, Forschungsbericht, Buch) hingegen ist nur mit einer durchdachten Verzeichnisstruktur im Griff zu behalten, zumal wenn mehrere Autoren beteiligt sind oder mehrere Auflagen zu erwarten sind. Damit Sie nicht unsere Fehler wiederholen, schildern wir ausführlich, wie wir ein Textprojekt strukturieren. Sie können von unserem Vorschlag abweichen, aber dann sollten Sie wissen, warum Sie es anders machen wollen. Der Aufbau einer solchen Struktur kostet am Anfang Zeit – etwa einen Arbeitstag – aber eine ordentliche Struktur spart insgesamt Zeit und hilft, Fehler zu vermeiden. Auch die Fehlersuche vereinfacht sich, wenn Text und Zutaten in handliche Portionen aufgeteilt sind. Das Umstrukturieren eines bereits laufenden Projektes kostet viel Mühe und bringt neue Fehler hinein, deshalb vorher nachdenken. Wir nehmen an, dass mit LaTeX gearbeitet wird, unter anderen Formatierern oder Textprozessoren bleiben die Überlegungen gleich.

Für das Textprojekt ist ein eigenes Verzeichnis anzulegen, eventuell sogar in Form eines Home-Verzeichnisses eines fiktiven Benutzers, unter dem die tatsächlichen Autoren zusammengefasst sind. In dieses Verzeichnis – nennen wir es `projekt` – wird für jedes Kapitel ein Unterverzeichnis angelegt, unabhängig von der Länge der Kapitel. Die Unterverzeichnisse erhalten kurze (4 bis 8 Zeichen), aussagekräftige Namen. Man kann die Verzeichnisse auch mit `kap1` bis `kapn` bezeichnen und aussagekräftige Namen dazu per Symlink erzeugen. Zu den Kapitelverzeichnissen kommen einige weitere Verzeichnisse hinzu:

- ein Verzeichnis namens `Haupt` oder `Main`, in dem sich übergeordnete Dateien finden,
- ein Verzeichnis `bilder` für Abbildungen,
- ein Verzeichnis `sonstig` für Vorwort und ähnliches Drumherum,
- bei Bedarf ein Verzeichnis `anhang`,
- ein Verzeichnis `Mails` für das Projekt betreffende Mails,
- ein Verzeichnis `tmp` für temporäre Daten.

Das Bilder-Verzeichnis könnte man auch jeweils als Unterverzeichnis der Kapitelverzeichnisse anlegen, aber bei nicht zu zahlreichen Abbildungen erscheint obiger Vorschlag zweckmäßiger. Man hat es beim Bearbeiten der Bilddateien leichter. Wenn es mehr als hundert Bilddateien werden, ist die kapitelweise Aufteilung vorzuziehen, da umfangreiche Verzeichnisse beispielsweise beim Arbeiten mit Versionskontrollsystemen wie CVS unhandlich sind.

In jedem Kapitelverzeichnis wird eine Kapiteldatei angelegt sowie für jeden Abschnitt (section) eine Abschnittsdatei, Namensgebung wie gehabt. Dazu kommt bei Bedarf jeweils ein Unterverzeichnis `problem` für das Problem-Management (Fehler, TODO-Listen etc.). Eine Abschnittsdatei könnte die Pfadbezeichnung

`projekt/kap2/sec3.tex` tragen, wenn wir die phantasielose Namensgebung wählen. In diesem Fall ließe sich sogar die Struktur auf Vorrat halten und für jedes neue Projekt kopieren. Die Nummern sind Schall und Rauch; sie legen nicht die endgültige Reihenfolge fest.

Das Verzeichnis `Haupt` (besser wäre `0haupt`, damit das Verzeichnis beim Auflisten mittels `ls` auf jeden Fall an vorderster Stelle steht) enthält in erster Linie:

- die Hauptdatei des Projektes wie `projekt.tex`,
- eine Testdatei `test.tex`,ähnlich wie die Hauptdatei, zum Testen einzelner Kapitel,
- ein Makefile für das Projekt,
- eine LaTeX-style-Datei für das Projekt (`projekt.sty`),
- weitere LaTeX-style-Dateien, die eigens für das Projekt beschafft wurden und nicht in den allgemeinen LaTeX-Verzeichnissen enthalten sind,
- eine Datei mit schwierig zu trennenden Wörtern (`trenn.tex`),
- bei Bedarf eine Datei mit Synonyma für den Index,
- für jedes Kapitelverzeichnis einen Symlink der Art `kap3 -> ../kap3`,
- nach einem `make`-Lauf einige temporäre LaTeX-Hilfsdateien sowie das Ergebnis (`projekt.ps` und/oder `projekt.pdf`).

Die Symlinks sind erforderlich, weil die Kapiteldateien in der Hauptdatei mit Bezug auf die Hauptdatei eingebunden werden. Eine andere Lösung wäre, die Kapitelverzeichnisse als Unterverzeichnisse auf derselben Ebene wie die Hauptdatei anzulegen. Wir halten unseren Vorschlag für übersichtlicher.

1.15.2 Unentbehrlich: make

Werkzeuge wie `make` werden als Builder bezeichnet und fassen die Kommandos zusammen, die aus den Teilen das Endergebnis erzeugen. Hauptanwendung ist das Übersetzen (Compilieren) von Programmquellen, aber auch bei Textprojekten ist `make` beinahe unentbehrlich. Einige Aufgaben könnte man auch mit Shell- oder Perl-Skripten bewältigen, aber `make` ist besser an die Aufgabe angepasst und in seinen Anfängen einfacher zu erlernen. Die Anweisungen für `make` stehen in einer Datei namens `Makefile` im Verzeichnis `Haupt`.

Sehen wir uns das Makefile an, das zur Erzeugung der Vorlage zum Buch verwendet wurde, etwas vereinfacht (PostScript-Versionen weggelassen, nur ein Index):

```
# Makefile fuer Debian-Buch
# W. Alex, 2004-10-05

######## Makros, je nach System

SHELL=/bin/sh
MAKE=/usr/bin/make
PLATEX=/usr/bin/pdflatex
MAKEINDEX=/usr/bin/makeindex -c -g -l
```

```
CP=/bin/cp
GREP=/bin/grep
RM=/bin/rm -f
WWW=/var/www/debian/

######## Targets

all  : test buch www
test : test.pdf
buch : buch.pdf
www  : web

######## Testversion einzelner Kapitel erzeugen

# Fuenf LaTeX-Durchlaeufe erforderlich,
# damit Index im Inhaltsverzeichnis erscheint.

test.pdf :
        $(PLATEX) test.tex
        $(PLATEX) test.tex
        $(PLATEX) test.tex
        $(MAKEINDEX) -s svind.ist -o test.ind test.idx
        $(PLATEX) test.tex
        $(GREP) chapter test.toc > test.ovr
        $(PLATEX) test.tex

######## Testversion aller Kapitel (Buch) erzeugen

buch.pdf :
        $(PLATEX) buch.tex
        $(PLATEX) buch.tex
        $(PLATEX) buch.tex
        $(MAKEINDEX) -s svind.ist -o buch.ind buch.idx
        $(PLATEX) buch.tex
        $(GREP) chapter buch.toc > buch.ovr
        $(PLATEX) buch.tex

######## Ergebnis ins lokale Web stellen

web :
        $(CP) buch.pdf $(WWW)debian2.pdf

######## Aufraeumen

xclean: clean cleanp

clean :
        $(RM) *.log
        $(RM) *.lof
        $(RM) *.idx

# usw., alle Hilfsdateien loeschen
```

```
cleanp :
        $(RM) *.pdf
```

Quelle 1.6 : Makefile zu vorliegendem Buch, etwas vereinfacht

Das Doppelkreuz leitet eine Kommentarzeile ein. Der erste Block enthält Makros, das sind Abkürzungen für Kommandos samt Optionen oder für Pfade, die nachfolgend verwendet und buchstäblich ersetzt werden. Das hat den Vorteil, dass man etwaige Änderungen nur am Anfang vorzunehmen braucht, wenn beispielsweise der Webspace woanders liegt. Das erste Ziel (target), hier und meist mit dem Namen *all*, wird erzeugt, wenn man `make` ohne Argument aufruft. Die Unterziele haben hier nur wegen der Übersicht nochmals eigene Definitionen, man kann sie so auch leichter erweitern. Der dritte bzw. vierte Block erledigen die eigentliche Arbeit. Nach dem Ziel folgen die Kommandos, wobei jede Zeile mit einem TAB beginnen muss. Im fünften Block wird das Ergebnis `buch.pdf` in den lokalen Webspace kopiert. Der Rest sind Aufräumarbeiten, wobei das Ziel `clean` üblicherweise alle Hilfsdateien löscht, das Ziel `xclean` auch die Ergebnisse, sodass nur die Quelldateien übrig bleiben. Diese beiden Ziele werden nicht unter `all` genannt, da man vielleicht eine Hilfsdatei untersuchen will. Es ist gebräuchlich, das Löschen nur auf ausdrücklichen Wunsch durchzuführen. Ein solches Makefile ist schnell geschrieben, vor allem, wenn man schon eine Vorlage hat, und leicht zu ändern. Es erspart viel Mühe.

Mit Makefiles kann man auch andere Dinge treiben, beispielsweise die *Türme von Hanoi* spielen. Das Spiel ist auf Seite 110 erläutert. Hier ein Makefile, das einer Vorlage von AMIT SINGH (`http://www.kernelthread.com/hanoi/`) nachempfunden ist:

```
# Tuerme von Hanoi, Aufruf: make -f makehanoi N=4

ifndef $(N)
    N = 3
endif

ifndef $(INITIALIZED)
    MAKEFILE = makehanoi
    F = 1
    U = 2
    T = 3
    INITIALIZED = 1
endif

hanoi:

ifeq ($N,1)
        @echo "$(F) -> $(T)"
else
        @$(MAKE) -s -f $(MAKEFILE) F=$(F) U=$(T) \
        T=$(U) N=$(shell expr $(N) - 1)
```

```
        @echo "$(F) -> $(T)"
        @$(MAKE) -s -f $(MAKEFILE) F=$(U) U=$(F) \
        T=$(T) N=$(shell expr $(N) - 1)
endif
```

Quelle 1.7 : Makefile, das die *Türme von Hanoi* spielt

1.15.3 Versionsverwaltung (rcs, cvs, subversion, arch)

Revision Control System (RCS)

Wenn zu befürchten steht, dass sich ein Text (Manuskript, Programmquelle, Konfigurationsdatei, Webseite) über längere Zeit hindurch entwickelt, oder wenn mehrere Autoren daran arbeiten, muss man Ordnung in dem Projekt halten. Außer der oben erwähnten Verzeichnisstruktur helfen dabei Versionsverwaltungen wie SCCS, RCS, CVS oder Subversion. Die vorgenannte Reihenfolge entspricht der historischen Entwicklung, der Leistungsfähigkeit und der Komplexität der Versionsverwaltungen. Am weitesten verbreitet ist gegenwärtig das Concurrent Versions System (CVS). Solche Systeme

- führen Buch über die Änderungen an den Dateien,
- ermöglichen, ältere Versionen wiederherzustellen, ohne dass diese vollständig gespeichert werden,
- verhindern oder harmonisieren gleichzeitige schreibende Zugriffe mehrerer Autoren auf dieselbe Datei.

Versionsverwaltungen können nicht:

- ein Projekt von sich aus sinnvoll gliedern (modellieren),
- Termine verfolgen,
- Fehler verfolgen (bug tracking),
- Dateien testen,
- die Kommunikation zwischen den Beteiligten ersetzen.

Sowie es um mehr als Texte oder Programme für den einmaligen Gebrauch geht, sollte man `make` und eine Versionsverwaltung einsetzen. Der anfängliche Mehraufwand – der sich bei kleinen Projekten in Grenzen hält – macht sich langfristig bezahlt. Geht bei einem Textprojekt die Fehlerverwaltung über das einfache Korrigieren hinaus – was bei Software die Regel ist – ist der Einsatz eines Bug Tracking Systems wie GNATS zu erwägen (`http://www.gnu.org/software/gnats/`). Im Umfeld von Webservern trifft man häufig auf den Begriff *Distributed Authoring and Versioning* (DAV, speziell WebDAV), der eine Erweiterung einer einfachen Versionskontrolle von Webdokumenten bezeichnet.

Die klassische Versionsverwaltung unter UNIX ist das Source Code Control System (SCCS), das aber durch das von WALTER F. TICHY entwickelte Revision Control System (RCS) so gut wie überall abgelöst worden ist. Das RCS aus dem GNU-Projekt steht in einem gleichnamigen Debian-Paket zur Verfügung, dazu gibt es noch

ein Project Revision Control System (PRCS) als Frontend für ein Bündel von Werkzeugen zur Projektverwaltung und ab *sarge* ein Advanced Revision Control System im Paket `darcs`. Informationen finden sich bei `http://www.gnu.org/software/rcs/rcs.html` und `http://www.cs.purdue.edu/homes/Trinkle/RCS/`. Das *CVS-RCS HOWTO* (2003) bei TLDP ist gegenwärtig zwecks Bearbeitung zurückgezogen; das *RCS mini-HOWTO* (1997) ist verfügbar.

Versionskontrollsysteme stellt man sich am einfachsten als einen Schrank vor, in dem die Dateien aufbewahrt und mit Zusatzinformationen über die Ausleihe versehen werden. Nehmen wir an, wir hätten in einem der Unterverzeichnisse des Projektes eine Textdatei `datei.tex` mittels eines Editors erzeugt und gefüllt. Dann sind die nächsten Schritte:

- Mit dem Kommando `ci datei.tex` (check in) stellen wir die Datei in das RCS ein. Dieses ergänzt die Datei um Versionsinformationen und macht eine nur lesbare (444) RCS-Datei namens `datei.tex,v` daraus. Die Kennung ist durch ein Komma abgetrennt, ungewöhnlich. Die ursprüngliche Datei löschen wir, falls nicht automatisch geschehen. Ab jetzt gehört die Datei dem RCS. Sieht man sich die Datei `datei.tex,v` mit `less` an, findet man am Anfang die RCS-Zusatzinformationen.
- Mit dem Kommando `co datei.tex` (check out, Dateiname ohne v) erzeugen wir eine nur lesbare Kopie der RCS-Datei. Die Datei `datei.tex,v` bleibt erhalten, die Kopie kann man mit allen Linux/UNIX-Werkzeugen betrachten oder bearbeiten, nur das Zurückstellen in RCS mittels `ci` verweigert das System.
- Mit dem Kommando `co -l datei.tex` – die Option bedeutet *lock* – holen wir uns eine les- und schreibbare Kopie der RCS-Datei in das Arbeitsverzeichnis. Dabei wird die RCS-Datei für weitere Zugriffe dieser Art gesperrt, bis wir unsere Kopie wieder zurückgestellt haben. So wird verhindert, dass zwei Benutzer gleichzeitig die Datei verändern. Sie erinnern sich; Der Letzte gewinnt. Die Kopie können wir mit allen Linux/UNIX-Werkzeugen bearbeiten, nur umbenennen wäre ein schlechter Einfall.
- Beim Zurückstellen mittels `ci datei.tex` haben wir Gelegenheit, einen kurzen Kommentar in die Versionsinformationen zu schreiben, beispielsweise Art und Grund der Änderung. Mit `less` entdecken wir die Änderungen am Ende der Datei.

Falls wir uns mit `co -l datei.tex` eine Kopie zum Editieren geholt und damit gleichzeitig das Original für weitere Zugriffe zwecks Schreibens gesperrt haben, anschließend die Kopie mittels `rm` löschen, so haben wir nichts mehr zum Zurückstellen. In diesem Fall lässt sich die Sperre mit `rcs -u datei.tex` aufheben. Auf die Linux/UNIX-Kommandos zur Dateiverwaltung sollte man verzichten und nur mit den RCS-Kommandos arbeiten. Das Aufbrechen des Schrankes von der Rückseite her ist möglich, aber dann braucht man keinen Schrank. Der Einsatz von RCS verlangt von den Beteiligten Disziplin.

RCS speichert die jüngste Version vollständig und zu den vorangehenden Versionen nur die Unterschiede (Differenzen, Deltas). Auf diese Weise wird Speicherplatz gespart, ohne auf das Zurückgehen zu älteren Versionen verzichten zu müssen.

In den Textdateien kann man einige RCS-Variable unterbringen, beispielsweise im Kommentar, aber auch in der Definition von Zeichenketten-Konstanten. Die wichtigsten sind:

- `$Id$` Diesen Bezeichner ergänzt RCS um den Namen der RCS-Datei, die Versionsnummer, das Datum des Zurückschreibens, den Autor, einen Status (z. B. experimentell) und den Namen des Benutzers, der die Datei gesperrt (gelockt) hatte. So stehen auch im Text die wichtigsten RCS-Informationen.
- `$Header$` wie vorstehend, nur dass der volle Pfad der Datei genannt wird, meist überflüssig.
- `$Log$` der beim Zurückschreiben als Kommentar eingegebene Text. Auf diese Weise lässt sich die Entwicklung der Datei verfolgen.

Da sich Autoren meist um das Protokollieren dieser Informationen drücken, bietet das RCS eine bequeme, mit einem sanften Zwang verbundene Möglichkeit, die Daten sogar direkt im Text (Manuskript, Quelle) festzuhalten. Das Ergebnis sieht bei einem in RCS eingestellten Makefile so aus (abgeschnitten):

```
# Makefile fuer Skriptum
# $Id: text15.tex,v 1.12 2006/04/03 18:40:19 wulf Exp $
# $Log: text15.tex,v $
# Revision 1.12  2006/04/03 18:40:19  wulf
# Kleiner Ergaenzungen und Korreekturen
#
# Revision 1.11  2005/08/21 18:30:40  wulf
# Endfassung
#
# Revision 1.10  2005/08/01 10:16:12  wulf
# 1. Korrektur fertig
#
# Revision 1.9  2005/07/18 20:06:27  wulf
# Zwischenstand
#
# Revision 1.8  2005/07/08 15:33:26  wulf
# Kapitel text fertig
#
# Revision 1.7  2005/05/21 20:43:44  wulf
# Zwischenstand, Kapitel text fertig
#
# Revision 1.6  2005/05/08 11:54:33  wulf
# Zwischenstand
#
# Revision 1.5  2005/04/24 20:56:39  wulf
# *** empty log message ***
#
# Revision 1.4  2005/03/26 17:05:35  wulf
```

```
# Aufgeraeumt.
#
# Revision 1.3  2005/03/26 15:08:27  wulf
# Sektion Lokalisierung rausgeworfen.
```

In LaTeX-Dateien muss die Entwicklungsgeschichte nach `\end{document}` kommen oder mittels `iffalse - fi` eingerahmt werden, da RCS nicht das LaTeX-Kommentarzeichen (Prozentzeichen) kennt.

Das System beherrscht Verzweigungen, Releases, eigene Zugriffsrechte und andere Dinge, deren Erläuterung zu weit führen würde. Der durchschnittliche Benutzer schöpft die Möglichkeiten von RCS ebenso wenig aus wie die von `make` oder die der großen Editoren.

Concurrent Versions System (CVS)

Das RCS hat aus heutiger Sicht Mängel, die sich bei größeren, im Netz verteilten Projekten bemerkbar machen. Deshalb wurde das Concurrent Versions System (CVS) entwickelt, das weit verbreitet ist und sich vom RCS in drei Punkten unterscheidet:

- Es kann mit Verzeichnissen und Unterverzeichnissen umgehen, nicht nur mit Dateien,
- es ist nach dem Client-Server-Modell aufgebaut; die Clients können auf anderen Rechnern laufen als der Server, wobei eine Verbindung nur für das Hoch- oder Herunterladen der Daten benötigt wird, nicht für das Arbeiten,
- es erlaubt, gleichzeitig mehrere Arbeitskopien zum Schreiben auszuchecken. Konflikte werden beim Zurückschreiben (Einchecken) entdeckt und teils automatisch, teils mit Hilfe der Autoren gelöst.

Da CVS ursprünglich auf RCS aufbaute, sind die Kenntnisse, die man beim Arbeiten mit RCS erworben hat, nicht verloren. CVS ist im gleichnamigen Debian-Paket in der Abteilung *Development* enthalten, wo auch ergänzende Pakete liegen. Weiteres findet sich bei `http://www.cvshome.de/` oder `.org/`, darunter eine hundertseitige, englische Anleitung von PER CEDERQVIST. Ein Frontend (Client) mit grafischer Benutzeroberfläche ist LinCVS. Einzelheiten unter `http://www.lincvs.org/` und `http://lincvs.sourceforge.net/`. Die Einrichtung des Tarballs mit den Binaries für die Qt-Lib 3.3 auf einem PC unter *sarge* verlief mühelos. Das Anlegen eines CVS-Projektes erfordert etwas Nachdenken und Zeit, das Arbeiten damit ist einfach[9]

Vorausgesetzt, auf dem Server und den Arbeitsplätzen ist CVS eingerichtet, besteht der erste Schritt im Anlegen des Daten-Depots (Repository) auf dem Server. Zu diesem Zeitpunkt sollte die Verzeichnisstruktur des Projektes grob festliegen, da Verschieben oder Umbenennen von Verzeichnissen oder Dateien zusätzlichen Aufwand bedeutet. Falls Sie mehrere Projekte unter CVS stellen wollen, ist meist ein

[9] Das Debian-Buch von GANTEN + ALEX ist unter CVS entstanden. Der Server stand in Bremen, ein Client bei Karlsruhe.

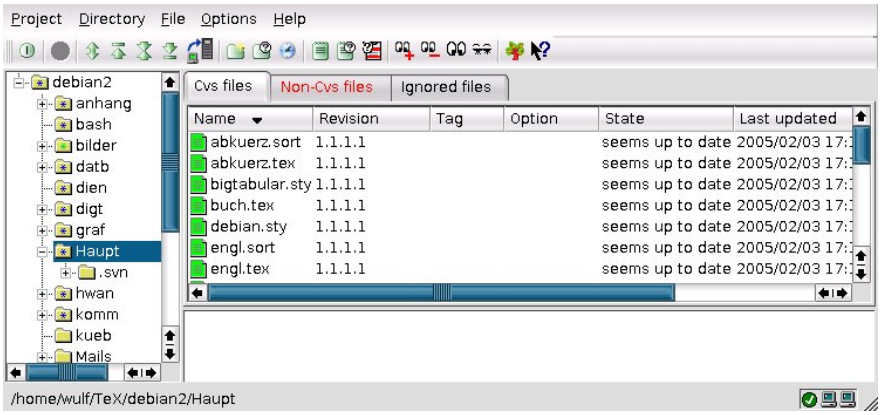

Abb. 1.16: Screenshot der grafischen Oberfläche LinCVS für das Concurrent Versions System (CVS)

Repository mit einem Verzeichnis je Projekt geschickter als ein eigenes Repository für jedes Projekt. Es kann vorkommen, dass ein Benutzer (Client) an mehreren, völlig getrennten CVS-Projekten auf verschiedenen Servern beteiligt ist. Das erfordert eine sorgfältige Konfiguration auf Seiten des Benutzers, insbesondere hinsichtlich der Authentifizierung. Den Anfang macht der Verwalter auf der Servermaschine:

```
debian:~# cvs -d /home/debian init
```

Damit wird ein Repository namens `/home/debian` angelegt; es könnte auch im `/var`-Zweig liegen. Dann richtet der Verwalter mittels des Kommandos `adduser` einen Benutzer und eine Gruppe `debian` ein und übereignet ihnen mit dem Kommando `chown -R` das neue Verzeichnis samt Inhalt. Schließlich fügt er ebenfalls mittels `adduser` alle Benutzer des Repositories der Gruppe `debian` zu. Das Weitere sollte von einem der berechtigten Benutzer ausgehen.

Der begibt sich auf seinem Arbeitsplatz-Rechner in das Verzeichnis, welches das vorläufige Projekt beherbergt (die Unterverzeichnisse seines Projektes), räumt noch einmal auf und importiert die Verzeichnisse und etwaige Dateien in das CVS auf dem Server:

```
joe@debian:~$ cvs -d :ext:wulf@server:/home/debian
import debian2 Debian2 Start
```

Das CVS-Kommando importiert unter dem Namen des Benutzers `wulf` über das Netz (Zugriffsmethode `:ext:`) auf den Server `server` in das Wurzelverzeichnis `/home/debian` von CVS, neues Unterverzeichnis `debian2` alle Daten aus dem Arbeitsverzeichnis. Die beiden Argumente `Debian2` und `Start` spielen jetzt keine Rolle, müssen jedoch sein. Der Benutzer wird nach seinem Passwort auf dem Server und nach einem kurzen Kommentar gefragt. Dann beginnt die Übertragung der Daten. Anschließend findet man auf dem Server im Verzeichnis `/home/debian` neben dem mittels `init` angelegten Verzeichnis `CVSROOT` ein Verzeichnis `debian2` mit allen Daten.

Nun holen wir uns die Daten vom Server zurück auf den Arbeitsplatz. Dazu gehen wir ein Verzeichnis höher und benennen das Projektverzeichnis um (Löschen wäre verfrüht). Mittels des Kommandos:

```
joe@debian:~$ cvs -d :ext:wulf@server:/home/debian
checkout debian2
```

erzeugen wir lokal das eben in den Server importierte Verzeichnis `debian2` neu. Es sollte mit seiner umbenannten Vorlage übereinstimmen, zuzüglich eines Verwaltungsverzeichnisses `CVS` in jedem Verzeichnis und abzüglich etwaiger Symlinks, die erneut anzulegen sind. Ist das der Fall, löschen wir die umbenannte Vorlage. Wir arbeiten künftig nur noch mit der Arbeitskopie aus dem CVS-Repository auf dem Server. Das `checkout`-Kommando brauchen wir nur einmal.

Die Arbeit unter CVS beginnt mit dem Editieren einer beliebigen Datei. Sind wir der Meinung, dass sie reif zum Hochladen in das Repository ist, geben wir im Arbeitsverzeichnis, ohne Argument:

```
joe@debian:~$ cvs commit
```

(E: to commit = anvertrauen, einliefern). Die restlichen Daten ergänzt der lokale CVS-Client aus den Verwaltungsverzeichnissen. Rufen wir dann das Kommando:

```
joe@debian:~$ cvs status
```

auf, so erfahren wir den Zustand der Dateien aus unserem Arbeitsverzeichnis und sehen, dass die Revisonsnummer der eben editierten und hochgeladenen Datei um 1 hochgezählt worden ist. Ist unsere Arbeitskopie einer Datei älter als die Vorlage im Repository, lautet das Kommando:

```
joe@debian:~$ cvs update
```

Haben wir lokal eine Datei oder ein Verzeichnis neu angelegt und wollen sie in das CVS aufnehmen, sind zwei Kommandos erforderlich:

```
joe@debian:~$ cvs add dateiname
```

```
joe@debian:~$ cvs commit
```

Auf diese Weise lässt sich verhindern, dass lokal erzeugte pdf-Dateien oder LaTeX-Hilfsdateien den Weg in das Repository nehmen; wir addieren sie nicht dazu. Ein anderer Weg mit Hilfe einer Datei `.cvsignore` wird unten beschrieben. Nach dem Löschen einer lokalen Datei lautet das Subkommando zum Entfernen aus dem Repository `remove`; es muss ebenfalls von einem `commit` begleitet werden. Die weiteren Möglichkeiten des CVS entnimmt man der Literatur aus dem Netz.

Die von CVS im Repository abgelegten Dateien tragen wie RCS-Dateien die Kennung `,v` und sind wie diese aufgebaut. Liegt das Repository irgendwo im Netz, hat man keine anderen Zugriffswege als über CVS-Kommandos. Solange alles funktioniert, interessiert den Benutzer das Repository wenig. Innerhalb der eigenen Verzeichnisse finden sich CVS-Unterverzeichnisse mit den Dateien:

- **`Entries`** Liste der in CVS eingestellten Unterverzeichnisse und Dateien aus dem jeweiligen Verzeichnis,
- **`Repository`** Name des Repository-Verzeichnisses für das jeweilige Projekt (im Schrank),
- **`Root`** Netzanschrift des Wurzelverzeichnisses des CVS-Systems (der Schrank).

um deren Pflege man sich nicht zu kümmern braucht.

Die Versionsnummer einer Datei hängt davon ab, wie oft sie eingecheckt wurde. Das führt dazu, dass die Nummern der Dateien auseinander laufen. Will man sich auf einen bestimmten Zustand des Projektes beziehen, müsste man für jede Datei die jeweilige Versionsnummer angeben. Diesem Missstand helfen symbolische Revisionen (Tags) ab. Man gibt – zweckmäßig im obersten Verzeichnis des Projektes – das Kommando:

```
joe@debian:~$ cvs tag Release-Feb-2005
```

Damit werden zu dem Namen der symbolischen Revision die jeweiligen Versionsnummern gespeichert. Man kann nun jederzeit auf den Zustand des gesamten Projektes zum Zeitpunkt der Vergabe des Tags zugreifen, beispielsweise um von dort eine Entwicklung abzuzweigen. Kommandos wie `checkout` kennen entsprechende Optionen. In ähnlicher Weise kann man sich auch auf einen bestimmten Zeitpunkt beziehen, ohne Tags angelegt zu haben.

Wer mit LinCVS arbeitet, findet ein Fenster wie in Abbildung 1.16 auf Seite 95 vor. In der obersten Zeile verbergen sich hinter den Worten Project, Directory, File, Options und Help Menüs mit den Kommandos, wie wir sie eben in der Kommandozeile eingegeben haben, und weiteren. Darunter folgt dasselbe mit Piktogrammen für die des Lesens Unkundigen. Links im Fenster wird die Verzeichnisstruktur dargestellt. Den Hauptteil des Fensters nimmt die Auflistung der Dateien im Arbeitsverzeichnis ein, eingeteilt in CVS-Dateien mit Revisions-Nummer, Tag, Status etc., Nicht-CVS-Dateien und von CVS zu ignorierende Dateien. Das Ganze lässt sich konfigurieren und macht einen aufgeräumten Eindruck; die Sprache ist Englisch.

Ein Benutzer kann in seinem Home-Verzeichnis eine Datei `.cvsrc` mit CVS-Kommandos anlegen, die immer wieder mit denselben Optionen aufgerufen werden, und sich so etwas Arbeit sparen. Ebenso kann er in einer Datei `.cvsignore` in seinem Home- oder jedem Unterverzeichnis Dateinamen angeben. die von den CVS-Kommandos `import`, `update` und `release` ignoriert werden sollen. Die Dateinamen dürfen die Jokerzeichen der Shell enthalten. Bei einem LaTeX-Projekt empfiehlt es sich, die zahlreichen temporären Hilfsdateien wie `*.aux` in `.cvsignore` aufzunehmen, unter Umständen auch die Enderzeugnisse im pdf- oder ps-Format. Bei einem Software-Projekt wären Objektdateien Kandidaten für diese Liste.

Subversion

Die Versionsverwaltung Subversion (SVN) behebt einige Schwächen von CVS und ist in ihrer Benutzung sehr ähnlich, sodass ein Umstieg leicht fällt. Es gibt auch ein

Werkzeug `cvs2svn` zum Konvertieren eines Repositories. Die Vorteile von Subversion gegenüber CVS sind (mangels eigener Erfahrung aus der deutschen Wikipedia entnommen):

- Umbenennen oder Verschieben von Dateien oder Verzeichnissen ist direkt möglich,
- Zugriffsrechte können ebenfalls verwaltet werden,
- die Behandlung binärer Dateien ist erleichtert,
- Commits sind wie in Datenbanken atomar, das heißt, sie werden entweder ganz oder gar nicht durchgeführt,
- Revisionsnummer werden nicht dateiweise, sondern immer für das ganze Projekt erhöht (ob das in der Praxis ein Vorteil ist, bleibt auszuprobieren),
- der Zugriff über das Netz ist schneller und flexibler.

Als Nachteile stehen dem ein höherer Massenspeicherbedarf und eine höhere Komplexität gegenüber. Letztere trifft jedoch nur den Verwalter, nicht die Benutzer. Auch ist SVN noch nicht so verbreitet wie CVS, aber jeder fängt einmal klein an. Einzelheiten in der Wikipedia und bei `http://subversion.tigris.org/`. Ein Online-Buch über Subversion liegt auf `http://svnbook.red-bean.com/`.

Eine grafische Oberfläche für Subversion unter Verwendung der Qt-Bibliothek bieten die Debian-Pakete *esvn* und *esvn-doc*, beheimatet auf `http://esvn.umputun.com/`.

Die Software *websvn* (`http://websvn.tigtis.org/`) ist in PHP geschrieben und ermöglicht den Zugriff auf Subversion-Repositories. Man kann alles lesen oder herunterladen, was im Repository liegt, es ist aber keine grafische Oberfläche für Subversion.

SVK ist eine dezentralisierte Versionsverwaltung, die auf Subversion oder CVS aufsetzt und in Perl geschrieben ist. Näheres bei `http://svk.elixus.org/`. Das Projekt entwickelt sich noch lebhaft.

Arch

Das Versions-Kontrollsystem Arch (zur Zeit nur in *oldstable* enthalten) soll ebenfalls CVS ersetzen. Es arbeitet mit verteilten Repositories. Näheres bei `http://wiki.gnuarch.org/`. Interessant sind auf den Webseiten die Vergleiche mit anderen Versions-Kontrollen. Selbst wenn man bei CVS bleiben will, erfährt man hier Einiges über die Mitbewerber. CVS hat zweifellos Schwächen, aber die besseren Systeme sind noch ziemlich jung. Einen Vergleich mehrerer Versionsverwaltungen findet man auch unter `http://better-scm.berlios.de/comparison/comparison.html`.

2

Spiele und Spielereien

Hier werden einige unter Debian GNU/Linux verfügbare Computerspiele und mehr oder weniger ernsthafte Spielereien vorgestellt.

2.1 GNU Chess (gnuchess, xboard)

Spiele stellen hohe und vielseitige Anforderungen an Hard- und Software. Wer nicht bloß spielen will und sich für die Hintergründe interessiert, ist mit dem *Linux Gamer's HOWTO*, aktuell von 2004, gut bedient.

GNU Chess ist ein erprobtes Schachprogramm, aktuell in Version 5, das mit einem Textbildschirm (curses-Bibliothek) zufrieden ist. Es sieht aber hübscher aus, wenn man die grafische Benutzerschnittstelle `xboard` hinzunimmt, die auf X11 aufsetzt. Alternativ dazu die Debian-Pakete *eboard* plus *eboard-extras-pack1*. Man braucht die Debian-Pakete *gnuchess*, *gnuchess-book* und *xboard* bzw. *eboard* aus der *sarge*-Distribution. Aus einem Terminalfenster (xterm, Konsole, gnome-terminal) wird das Spiel folgendermaßen gestartet:

```
joe@debian:~$ xboard -size small
```

Anstatt `small` passt oft `mediocre` oder `average`, weitere Größen und Optionen siehe `man xboard`. Ein Handbuch zu GNU Chess liegt unter `/usr/share/doc/gnuchess/MANUAL`. Man kann allein gegen die Maschine spielen, zu zweit auf einem Rechner oder im Netz. Es ist auch unterhaltsam zu beobachten, wie die Maschine gegen sich selbst spielt. Das Programm bietet vielfältige Möglichkeiten zum Mitschreiben, Analysieren und Üben.

Eine Alternative zu `gnuchess` enthält das Debian-Paket *crafty* aus dem Non-free-Bereich mit mehreren Paketen *crafty-books-** aus dem Contrib-Bereich. Neuigkeiten und Quellen bei `http://www.crafty-chess.com/`.

Weitere Informationen bei `http://www.gnu.org/software/chess/chess.html` und `http://www.tim-mann.org/chess.html`. Auf den Webseiten finden sich auch interessante Hinweise auf andere Schachprogramme. Zu Schach allgemein siehe `http://www.unix-ag.uni-kl.de/~chess/`,

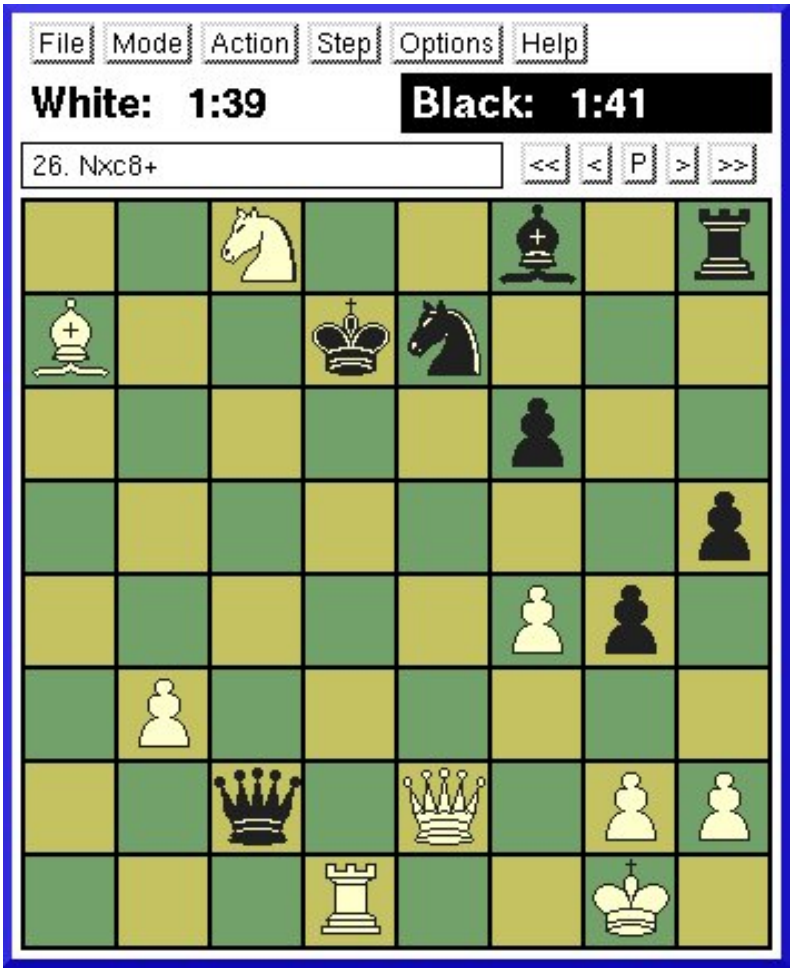

Abb. 2.1: Screenshot von GNU Chess mit `xboard`

`http://www.schachbund.de/` und `http://www.fide.com/`, speziell zu Computerschach: `http://www.computerschach.de/`. Und falls Sie einmal einen richtigen Computer beim Schachspielen erleben wollen, ist `http://www.research.ibm.com/deepblue/` die passende Anschrift.

2.2 Brettspiele (gtkboard), Go (gnugo, cgoban)

Das Debian-Paket *gtkboard* bietet unter dem gleichnamigen Kommando eine Reihe von Brettspielen an, teils ausgereift, teils noch nicht spielbar. Abbildung 2.2 zeigt ein Puzzle aus dem Paket, Abbildung 2.3 den Schirm eines Mastermind-ähnlichen Spiels, bei dem die Verteilung der am rechten Rand gezeigten Farben hinter den am

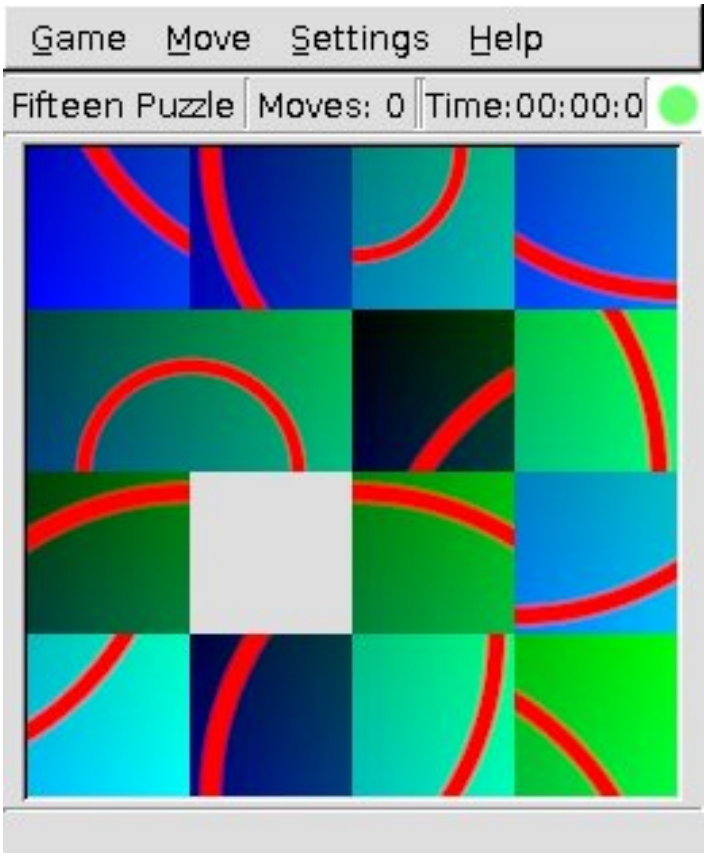

Abb. 2.2: Screenshot eines Puzzles aus dem Debian-Paket *gtkboard*, auf dem Schirm farbig

oberen Rand schwarz dargestellten Kreisen zu raten ist. Die Spielereingabe ist am unteren Rand zu sehen. Ein ähnliches Spiel ist in dem Paket *gnomermind* enthalten.

GO ist ein altes strategisches Brettspiel aus China für zwei Spieler. Deutschsprachige Informationen findet man bei `http://www.degob.de/` und wie immer in der Wikipedia. Das Kommando `gnugo` startet ein GO-Spiel auf einem Textbildschirm (ohne X11). Nimmt man das Paket `cgoban` hinzu und hat X11 laufen, wird das Spiel hübscher.

2.3 Freeciv – ein Civilization-Klon

FreeCiv ist ein freier Klon des Spieles *Civilisation II* der Firma Microprose. Informationen finden sich in `/usr/share/doc/freeciv` und auf `http://www.freeciv.org/`. Es ist ein Strategiespiel, auch für mehrere Spieler über ein Netz.

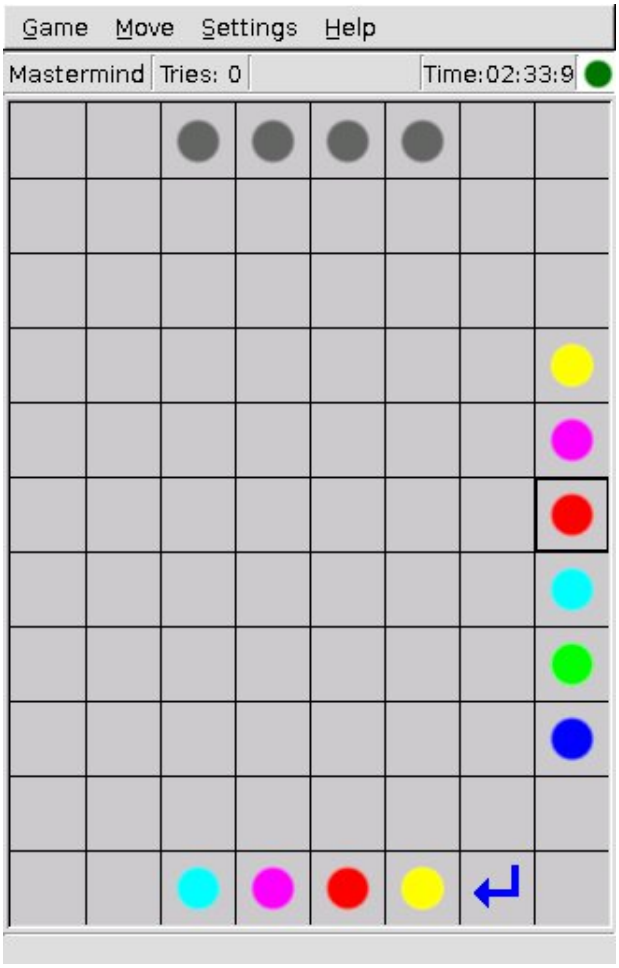

Abb. 2.3: Screenshot eines Mastermind-Spiels aus dem Debian-Paket *gtkboard*

Es kann auf einem lokalen Server gespielt werden; im Internet stehen aber auch eine Handvoll Metaserver bereit, auf denen jedermann mitspielen kann.

Das Spiel setzt sich aus einem Server (`civserver`) und einer beliebigen Anzahl von Clients (Spielern, `civclient`) zusammen. Die Clients benötigen X11, der Server nicht. Das Debian-Paket für den Server heißt `freeciv` (*woody*) bzw. `freeciv-server` (*sarge*). Für die Klienten stehen je nach verwendeter Grafikbibliothek zwei Debian-Pakete zur Verfügung: `freeciv-gtk` oder `freeciv-xaw3d` für *woody* bzw. `freeciv-client-gtk` oder `freeciv-client-xaw3d` für *sarge*.

Beim Spielen im Netz durch eine Firewall hindurch ist unter Umständen darauf zu achten, dass der verwendete Port freigeschaltet ist (port forwarding). Im übrigen

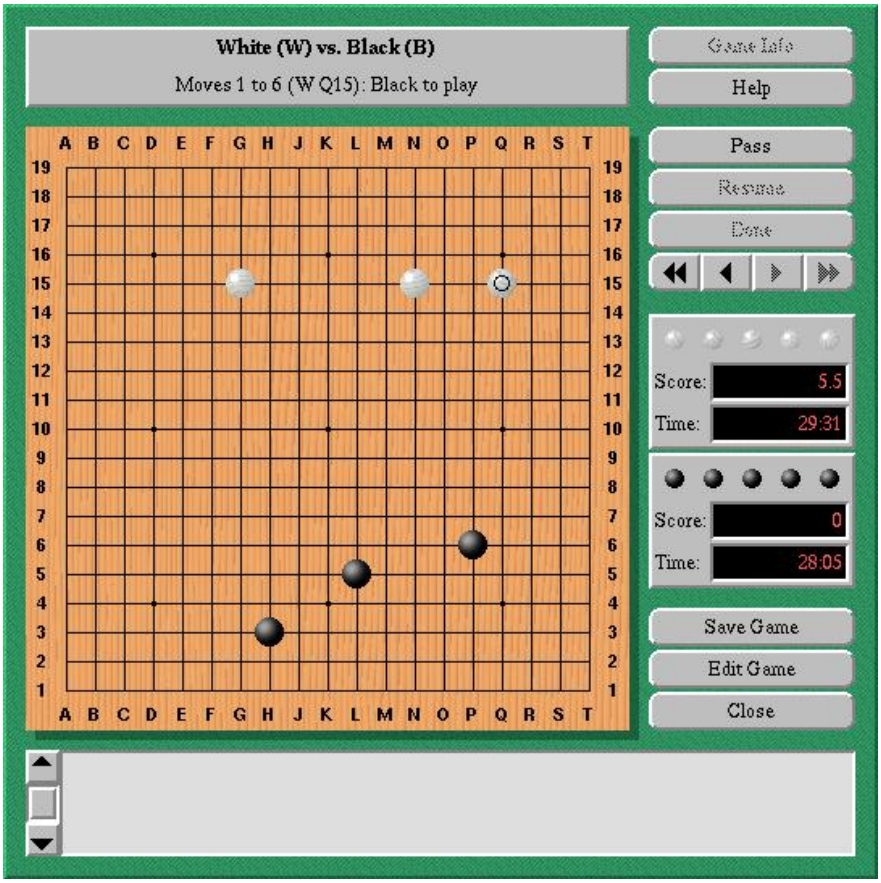

Abb. 2.4: Screenshot von `gnugo` (GO-Spiel) mit `cgoban`

ist das Spiel in vielerlei Hinsicht konfigurierbar. Das Ziel ist, mit friedlichen und kriegerischen Mitteln zur mächtigsten Zivilisation im Spiel zu werden.

2.4 TuxRacer – für Wintersportler

Das Spiel TuxRacer belegt 15 MB auf der Platte. Es geht darum, den Pinguin Tux auf einer Abfahrtspiste in einer dreidimensionalen Winterlandschaft möglichst schnell ins Ziel zu bringen, Hindernissen (Slalomstangen, Bäume) auszuweichen und nebenbei noch herumliegende Heringe einzusammeln. Das Programm kennt keine Kommandozeilen-Argumente. Ein Blick in die Datei `$HOME/.tuxracer/options` und vorsichtiges Editieren kann weiterhelfen. Per Default wird TuxRacer mittels der Cursor-Tasten gesteuert. Die Escape-Taste bricht das Spiel ab.

Abb. 2.5: Screenshot des Spieles FreeCiv. ALEXANDER DER GROSSE am Beginn seiner Laufbahn. Nicht ganz der historischen Wahrheit entsprechend, da er ein bereits großes Reich von seinem Vater erbte.

TuxRacer bringt Sound mit, allerdings berichten viele Benutzer von Problemen. Bei einem Start aus dem KDE-Menü (*Games -> Debian -> Arcade -> TuxRacer*) kam der Sound sofort, bei einem Start aus der Kommandozeile nicht. Abhilfe schaffte die Änderung der Option `set audio_frequ_mode` von 22 auf 44 kHz.

Bis zur Version 0.61 war TuxRacer frei im Sinne der GPL. Eine aktuelle Information von HOLGER THIELE findet man unter `http://www.hthiele.de/` – auch zu anderen Spielen unter Linux – die freie Version bei `http://tuxracer.sourceforge.net/`, Kommentare bei `http://www.linuxgames.com/`, `http://www.pro-linux.de/spiele/tuxracer.html` oder `http://happypenguin.org/`, die freie Weiterentwicklung OpenRacer bei `http://openracer.worldforge.org/`. Eine zweite freie Abwandlung namens PlanetPenguin Racer ist auf `http://projects.planetpenguin.de/racer/` zu Hause. Einen Strecken- oder Kurseditor und weitere Abfahrtsstrecken liegen auf `http://www.erinacom.de/tux/` oder `http://tuxracer.fubaby.com/`.

Die kommerzielle Fortsetzung, gegenwärtig in Version 1.1, von den Sunspire Studios wird von Amazon oder Ixsoft angeboten, der Hersteller hat sich aus dem Web verabschiedet.

Abb. 2.6: Screenshot des Spieles TuxRacer, an heißen Sommertagen sehr erfrischend. Mit der Farbe von Tux stimmt etwas nicht.

2.5 Pingus – ein Lemmings-Klon

Pingus ist ein freier Lemmings-Klon, eine Art von Puzzle mit Hindernissen. Es belegt 20 MB auf der Platte. Die Manual-Seite ist kurz, die Info-Seite gibt mehr her, ausführliche Informationen findet man bei `http://pingus.seul.org/`. Zu dem Spiel gehört ein Level-Editor, mit dem man eigene Level gestalten kann. Die Entwicklung geht langsam voran, aber sie lebt.

2.6 Flight Gear – ein anspruchsvoller Flugsimulator

Das Debian-Paket *flightgear* stellt unter dem Aufruf `fgfs` einen Flugsimulator zur Verfügung:

```
joe@debian:~$ man fgfs
joe@debian:~$ fgfs -help
joe@debian:~$ fgfs
```

Die Dateien belegen im Grundausbau 200 MB auf der Platte; das *World Scenery Set* umfasst drei DVDs mit zusammen 12 GB. Einzelne Gegenden können aus dem

Abb. 2.7: Screenshot des Startbildes von Pingus, einem Lemmings-Klone

Netz heruntergeladen werden. CPU, Arbeitsspeicher und Grafikkarte sollten nicht zu schwach ausgelegt sein. Ferner empfiehlt es sich, ergänzend zur Tastatur ein etwas realistischeres Eingabegerät zu gebrauchen. Vielleicht findet sich bei Ebay ein alter Link-Trainer, den man mit einer USB-Schnittstelle aufrüsten kann ... Das Spielen zu mehreren in einem lokalen Netz wird unterstützt (Multiplayer Protocol). Alles Weitere bei `http://www.flightgear.org/`.

Bei dem Simulator geht es nicht darum, feindliche Flugobjekte abzuschießen. Er will vielmehr das Verständnis für die Physik und Technik des Fliegens fördern und kommt daher auch mit verschiedenen Modellen für die Flugdynamik.

2.7 Das Planetarium auf dem Desktop (kstars)

Die Astronomen mögen verzeihen, dass wir ein ernsthaftes wissenschaftliches Gerät zu den Spielzeugen rechnen. Das Programm `kstars` wird entweder in KDE unter dem Menüpunkt *Applications* oder in einer Kommandozeile aufgerufen:

```
joe@debian:~$ kstars -help-all

joe@debian:~$ kstars
```

Als erstes vergewissere man sich über Zeit (LT, Local Time) und Ort der Beobachtung. Beides lässt sich einstellen. Für die Abbildung 2.9 auf Seite 108 haben wir

Abb. 2.8: Screenshot des Flugsimulators Flight Gear, hier der Blick aus einer Cessna 310.

MEZ und Heidelberg gewählt. Weitere deutsche Städte lassen sich aus einer Liste von 2500 Orten auswählen; zudem kann man die Liste erweitern.

Ein weiteres Planetarium mit vielen Möglichkeiten ist Stellarium (`http://www.stellarium.org`) aus dem gleichnamigen Debian-Paket. Ebenfalls astronomische Darstellungen bieten die Programme `ssystem` (Solar System) und `openuniverse` aus den gleichnamigen Debian-Paketen. Beide bringen einen netten Demo-Modus mit.

2.8 MegaHAL – ein Gesprächs-Simulator

Das Programm `megahal` ist ein naher Verwandter des bekannten Programms `Eliza`, ein von JOSEPH WEIZENBAUM im Jahr 1965 entwickelter Chatbot, der einen Psychotherapeuten mimt. Wer Näheres über Eliza wissen will. suche nach *eliza weizenbaum acm*. Hintergrund zu MegaHAL ist in `/usr/share/doc/megahal/paper.txt` zu finden, einer kurzen Einführung. Der Text ist auch von `http://megahal.alioth.debian.org/` zu beziehen, nebst weiterem Material.

Wenn man sich nur selbst mit MegaHAL vergnügen will, ruft man am besten das Shellscript `/usr/bin/megahal-personal` auf, das ein Verzeichnis `.megahal` im Home-Verzeichnis des Benutzers anlegt und füllt. Dort speichert

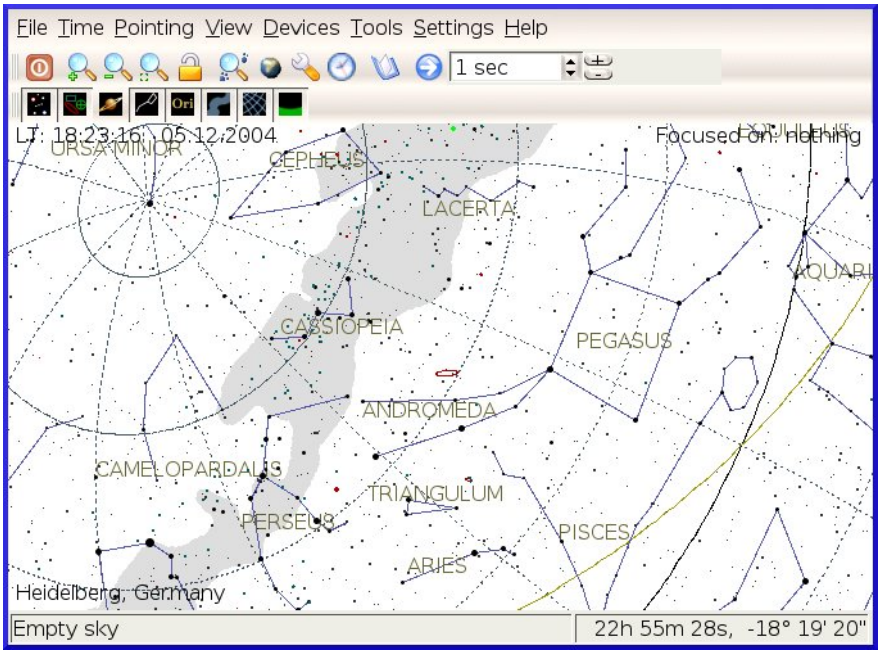

Abb. 2.9: Screenshot des Programmes Kstars, eines Planetariums auf dem Desktop. Hier der Himmel über Heidelberg am 5. Dez. 2004 gegen 18.20, in Bildmitte der Andromeda-Nebel. Als Darstellungsart wurde *Sternenkarte* gewählt.

MegaHAL seine Erfahrungen aus vergangenen Gesprächen mit dem Benutzer, es lernt:

```
joe@debian:~$ megahal-personal
```

Anfängliche Klagen über fehlendes Hirn (`megahal.brn`) übergeht man, das wächst von allein. MegaHAL startet mit einer Begrüßung; dann kann man antworten. Die Antwort darf Zeilenwechsel enthalten. Erst ein zweifacher Zeilenwechsel beendet die Eingabe, Von Geburt aus redet MegaHAL englisch, aber wenn man stur bleibt und immer nur deutsch mit ihm redet, stellt es sich langsam darauf um. Während des Zwiegesprächs können Kommandos eingegeben werden, die mit einem Doppelkreuz (command prefix) beginnen:

- #QUIT MegaHAL beenden und Gelerntes speichern,
- #EXIT MegaHAL beenden, ohne Gelerntes zu speichern,
- #HELP die verfügbaren Kommandos anzeigen und kurz erläutern.

Es ist interessant zu verfolgen, wie MegaHAL lernt, vor allem, wenn man deutsch mit ihm redet, weil es in dieser Sprache keine Vorkenntnisse mitbringt. Tippfehler lernt es genau so eifrig wie schwierige Fremdwörter. Wird man grob zu MegaHAL, ändert es seine Ausdrucksweise ebenfalls rasch. Nach einigen Stunden Unterricht beginnt MegaHAL, beinahe intelligente Sätze zu äußern. Hängt vom Lehrer ab.

Eine Verwandte von MegaHAL namens Nicole (Nearly Intelligent Computer Operated Language Examiner) liegt unter `http://nicole.sourceforge.net`. Sie ist nicht bei Debian zu haben. Die im Netz eingerichtete Nicole mit Web-Interface zeigte sich nicht gesprächsbereit, weder unter Galeon noch unter Opera.

2.9 Fraktale (kfract, xaos, xfractint)

Die eigenartigen Bilder, die Fraktal-Generatoren liefern, kennt jeder. Die Mathematik dahinter zu beschreiben, würde den Rahmen des Buches sprengen. Am besten liest man bei BENOIT MANDELBROT nach, einem französischen Mathematiker, der die Fraktale populär gemacht hat.

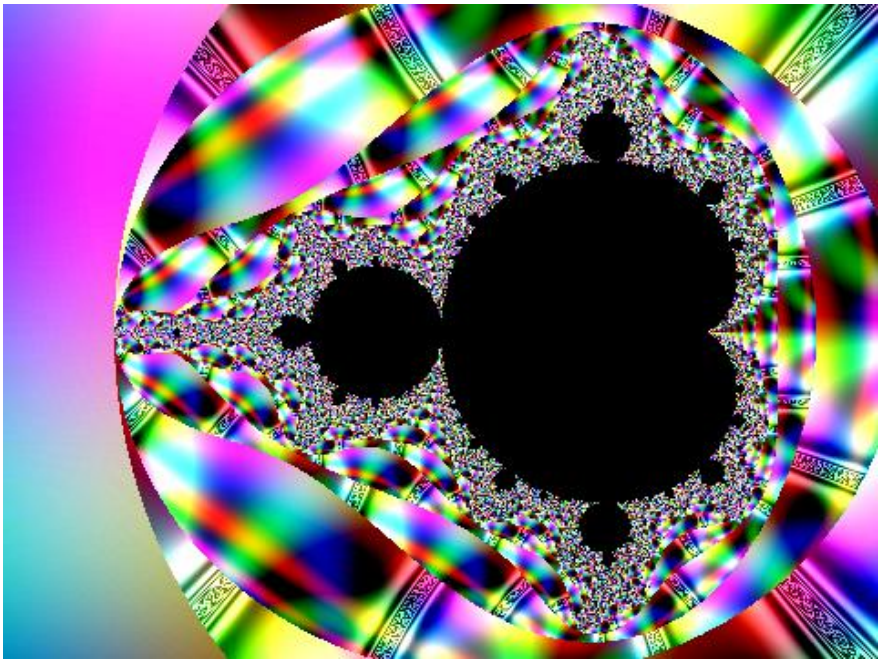

Abb. 2.10: Screenshot eines Apfelmännchens (Fraktals), erzeugt mit Xaos, auf dem Bildschirm farbig

Der KDE Fraktal-Generator `kfract` wird aus der Kommandozeile aufgerufen oder als Menüpunkt *Graphics -> Fractal Generator*. Er zeichnet sofort ein Apfelmännchen in ein eigenes Fenster. Über das Menü *Options* lassen sich Zahlenwerte eingeben, jedoch nicht der Typ (die zu Grunde liegende Gleichung) oder die Farben ändern.

Xaos zeichnet ein Apfelmännchen auf den Bildschirm, Klickt man mit der linken Maustaste in das Bild, kommt es näher. Die rechte Maustaste verschiebt das Bild in

die Ferne. Mit der Taste <h> holt man sich eine Hilfe ins Fenster, darunter eine gut gemachte, animierte Einführung in Fraktale in mehreren Sprachen. Mit der Tastenfolge <h> <h> <1> <2> <h> <h> <2> <2> holt man sich eine deutschsprachige Einführung in Fraktale auf den Schirm. Abbrechen mit <q> oder <esc>.

Der Fraktal-Generator `xfractint` – ursprünglich von IBM für den PC entwickelt – ist für X11 umgeschrieben worden. Nach unserem Eindruck stimmen die in der Online-Hilfe (<h>) angegebenen Tastenzuordnungen nicht ganz zu dem Verhalten des Programms. Beenden mit <esc>. Näheres bei `http://www.fractint.org//`.

2.10 Türme von Hanoi

Die *Türme von Hanoi* sind ein Spiel für eine einzelne Person, bei dem ein Stapel unterschiedlich großer Scheiben von einem Turm auf einen zweiten Turm gebracht werden soll, ein dritter Turm als Zwischenlager dient, mit einem Zug immer nur eine Scheibe bewegt werden und niemals eine größere Scheibe über einer kleineren liegen darf. Das Spiel wurde 1883 von dem französischen Mathematiker FRANÇOIS EDUOUARD ANATOLE LUCAS erdacht. Es ist ein beliebtes Beispiel für rekursive Lösungswege; suchen Sie einmal im Web nach *Towers of Hanoi* und *recurs* sowie ihren deutschen Entsprechungen.

Das Spiel gibt es nicht als Debian-Paket, aber wir können es uns selbst schreiben. Tippen Sie mit einem Editor das Programmbeispiel 2.1 in eine Datei namens `hanoi`, setzen Sie deren Zugriffsrechte mittels `chmod 750 hanoi`, und rufen Sie es mittels `./hanoi` aus einem Textfenster auf. Diese Art von Programmen nennt man Shellskript. Im obigen Shellskript ist die Anzahl der Scheiben auf 16 begrenzt, weil mit steigender Scheibenzahl die Zeiten lang werden (Anzahl der Züge minimal $2^n - 1$, bei 10 Scheiben also 1023). Bei 16 Scheiben braucht selbst ein schneller Rechner etwas Zeit. Sie dürfen die Grenze im Skript aber beliebig erhöhen, bis Ihr Rechner streikt.

```
# Shellscript hanoi (Tuerme von Hanoi)
# Aufruf hanoi n mit n = Anzahl der Scheiben
# max. 16 Scheiben, wegen Zeitbedarf

# Funktion, rekursiv (selbstaufrufend)

function fhanoi
{
        typeset -i x=$1-1
        ((x>0)) && fhanoi $x $2 $4 $3
        echo "von Turm $2 nach Turm $3"
        ((x>0)) && fhanoi $x $4 $3 $2
}

# Hauptscript
```

```
case $1 in
[1-9] | [1][0-6])
        echo "Tuerme von Hanoi"
        echo "Start Turm 1, Ziel Turm 2, mit $1 Scheiben"
        echo "Bewege die oberste Scheibe"
        fhanoi $1 1 2 3;;
*) echo "Argument zwischen 1 und 16 erforderlich"
   exit;;
esac
```

Quelle 2.1 : Shellskript Türme von Hanoi

Das Hauptskript ruft die Funktion `fhanoi` mit vier Argumenten auf. Das erste ist die Anzahl der Scheiben, die weiteren Argumente sind Start-, Ziel- und Zwischenturm. Die Funktion `fhanoi` setzt die ganzzahlige Variable `x` auf den um 1 verminderten Wert der Anzahl, im Beispiel also zunächst auf 2. Diese Variable begrenzt die Rekursionstiefe. Ist der Wert des ersten Argumentes im Aufruf bei 1 angekommen, ruft sich die Funktion nicht mehr auf, sondern gibt nur noch aus. Die Zeile:

```
((x>0)) && fhanoi $x $2 $4 $3
```

ist in der Bourne-again-Shell so zu verstehen:

- berechne den Wert des booleschen Ausdrucks `x > 0`,
- falls TRUE herauskommt, rufe die Funktion `fhanoi` mit den jeweiligen Argumenten auf, wobei `$2` das zweite Argument ist usw.

Schreiben wir uns die Folge der Funktionsaufrufe untereinander, erhalten wir:

```
fhanoi 3 1 2 3
      fhanoi 2 1 3 2
            fhanoi 1 1 2 3 -> print 1 2
      print 1 3
            fhanoi 1 2 3 1 -> print 2 3
print 1 2
      fhanoi 2 3 2 1
            fhanoi 1 3 1 2 -> print 3 1
      print 3 2
            fhanoi 1 1 2 3 -> print 1 2
```

Die Ausgabe des Skripts für $n = 3$ sieht folgendermaßen aus:

```
Tuerme von Hanoi
Start Turm 1, Ziel Turm 2, mit 3 Scheiben

Bewege die oberste Scheibe
von Turm 1 nach Turm 2
von Turm 1 nach Turm 3
von Turm 2 nach Turm 3
von Turm 1 nach Turm 2
```

```
von Turm 3 nach Turm 1
von Turm 3 nach Turm 2
von Turm 1 nach Turm 2
```

Für $n = 1$ ist die Lösung trivial, für $n = 2$ offensichtlich, für $n = 3$ überschaubar. Bei größeren Werten muss man systematisch vorgehen. Ein entscheidender Moment ist erreicht, wenn nur noch die unterste (größte) Scheibe im Start liegt und sich alle übrigen Scheiben im Zwischenlager befinden, geordnet natürlich. Dann bewegen wir die größte Scheibe ins Ziel. Der Rest ist nur noch, den Stapel vom Zwischenlager ins Ziel zu bewegen, eine Aufgabe, die wir bereits beim Transport der $n - 1$ Scheiben vom Start ins Zwischenlager bewältigt haben. Damit haben wir die Aufgabe von n auf $n - 1$ Scheiben reduziert. Das Rezept wiederholen wir, bis wir bei $n = 2$ angelangt sind. Wir ersetzen also eine vom Umfang her zu schwierige Aufgabe durch eine gleichartige mit geringerem Umfang so lange, bis die Aufgabe einfach genug geworden ist. Das Problem liegt darin, sich alle angefangenen, aber noch nicht zu Ende gebrachten Teilaufgaben zu merken, aber dafür gibt es Rechner. Mit der Entdeckung eines Lösungsweges (Algorithmus), der mit Sicherheit in kürzest möglicher Zeit zum Ziel führt, ist der Charakter des Spiels verloren gegangen, es ist nur noch ein Konzentrations- und Gedächtnistest.

Was noch zu tun bliebe, wäre, dem Programm eine hübsche grafische Oberfläche zu schneidern, aber das ist mehr Arbeit, als das Programm selbst zu entwerfen, und verdeutlicht den Preis, den wir für bequeme Benutzeroberflächen zahlen.

2.11 Game of Life

Das Game of Life (GOL) wurde 1970 von dem britischen Mathematiker JOHN HORTON CONWAY erdacht. Eine gute Übersicht bieten die deutsche oder die englische Wikipedia. Den zugehörigen Artikel von MARTIN GARDNER im *Scientific American* und anderes findet man unter `http://www.ibiblio.org/lifepatterns/` und `http://www.ibiblio.org/pub/Linux/science/ai/life/`, eine aufbereitete Sammlung von Links bei `http://www.radicaleye.com/lifepage/`, eine FAQ bei `http://cafaq.com/lifefaq/`. Ein fertiges Programm namens `xlife` von JON BENNETT enthält das gleichnamige Debian-Paket aus der Abteilung *Games*, zwei weitere Programme gibt es als Tarball im Netz:

- GtkLife von SUZANNE BUTTON `http://ironphoenix.org/tril/gtklife/` oder `http://freshmeat.net/projects/gtklife/`,
- Life von DAVID BELL `ftp://sunsite.unc.edu/pub/Linux/games/amusements/life/dblife-6.0.tgz`

Das Game of Life ist ein zweidimensionaler zellulärer Automat, eine Art von Schachbrett, auf dem nach folgenden Regeln gespielt wird:

- Jedes Feld (Zelle) ist entweder tot (in Abbildung 2.11 weiß) oder lebendig (schwarz).

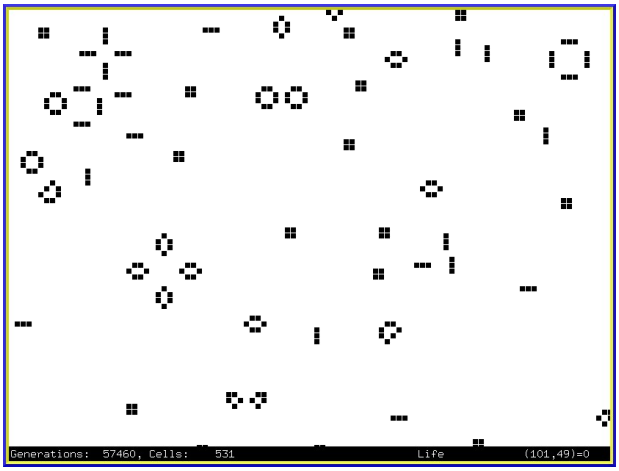

Abb. 2.11: Screenshot von `xlife`, einem *Game of Life*

- Jede neue Generation von Feldern wird entsprechend den beiden nachfolgenden Regeln aus der vorangehenden Generation gebildet.
- Ein lebendes Feld mit 2 oder 3 lebenden Nachbarn (punkt- oder linienförmig benachbart, jedes Feld hat 8 Nachbarn) lebt weiter, andere Felder sterben.
- Ein totes Feld mit genau 3 lebenden Nachbarn beginnt zu leben.

Der Reiz des Spiels liegt in der Entwicklung der Muster. Man kann auch die Regeln variieren oder mit sechseckigen Feldern oder in höheren Dimensionen spielen. Das Spiel hat außer seinem Unterhaltungswert eine tiefere Bedeutung in der Informatik (Entscheidungsproblem).

2.12 Wortspielereien

Ein Anagramm ist ein Wort oder eine Wortgruppe (Phrase), die aus einem anderen Wort oder einer anderen Phrase durch Vertauschen (Permutieren) der Zeichen entsteht. Das ist für den Rechner eine Kleinigkeit. Die Anzahl der möglichen Permutationen von n Zeichen ist $n!$. Da in Anagrammen erleichterte Regeln für das Einfügen von Leer- oder Satzzeichen bestehen, ist die Anzahl der Anagramme größer. Beispiele: *Geburt – Erbgut*, *Klaus – Lukas*, *Strolche – rechtlos* oder *Briefmarke – Fabrikmeer – Imkerfarbe – Krambriefe – Markfieber*. Zum Ermitteln von Anagrammen braucht ein entsprechendes Programm ein Wörterbuch (dictionary), mit dessen Hilfe es aus der Menge aller möglichen Vertauschungen die sprachlich zulässigen herausfiltert. Die geistreichen muss der Benutzer selbst heraussuchen.

Das Programm `an` sucht per Default in einem Wörterbuch `/usr/dict/words`, das vermutlich nicht eingerichtet ist. Eine einfach formatierte deutsche Wörterliste findet sich unter `ftp://ftp.cs.tu-berlin.de/pub/doc/wordlists/germanl.Z`. Wenn man die Datei mit dem alten Linux/UNIX-Werkzeug

`uncompress` oder mit `gunzip` verdünnt hat, bleibt noch die Frage der Umlaute, die man eventuell mit Hilfe des Streaming-Editors `sed` angehen muss. Immerhin lieferte `an` nach dem Kopieren von `germanl` nach `/usr/bin/words` als Anagramm zu *alex* das Wort *axel*.

Ein zweiter Anagramm-Generator ist das Programm `wordplay` aus dem gleichnamigen Debian-Paket. Es bringt seine Default-Wörterliste `/usr/share/games/wordplay/words721.txt` mit, die entsprechend ihrer Herkunft nur amerikanische Wörter enthält. Die Liste ist lesbar, einfach aufgebaut und wird auch vom Programm `an` verstanden. Der Aufruf

```
joe@debian:~$ wordplay copenhagen
```

liefert unter anderem *CHANGE OPEN* zurück. Besteht das Eingangsargument aus mehreren Wörtern, sind sie mit Gänsefüßchen oder Single Quotes einzurahmen, damit die Shell sie als Ganzes übergibt.

Eine besondere Form des Anagramms sind Palindrome oder Kehrwörter. Das sind Wörter, die – gegebenenfalls mit anderer Bedeutung – von rechts und links gelesen werden können: Anna, Annasusanna, Otto, Rentner, Lagerregal. Weiteres zu Wortspielen bei `http://anagramme.de/`, `http://www.gnudung.de/`, `http://www.palindromes.org/` und wie immer in der Wikipedia.

Scrabble ist ein bekanntes Gesellschaftsspiel (`http://www.scrabble.de/`, `http://www.scrabble.com/`), das mit Kreuzworträtseln verwandt ist. Die *sarge*-Distribution enthält ein Debian-Paket dieses Namens. Das Spiel wird von der Kommandozeile ohne X11 gespielt. Durch Eingabe von `help` erfährt man Weiteres, `quit` beendet das Spiel. Die gegenwärtige Version bringt ein amerikanisches Wörterbuch mit; an Optionen kennt sie nur die Spielstärke. Das Programm verträgt noch Entwicklung.

Geht es um Kreuzworträtsel, so sind zwei Aufgaben für den Rechner zu unterscheiden:

- Erzeugen (Generieren, Konstruieren) von Kreuzworträtseln,
- Lösen oder Hilfe beim Lösen.

Bei Debian findet sich noch nichts Geeignetes, aber bei `http://sourceforge.net/` wird man mit dem Suchbegriff *crossword* fündig. Im deutschen Sprachraum kann man sein Glück bei `http://www.tea.ch/de/` versuchen, zwar ein kommerzielles Angebot, aber auch mit einigen kostenlosen (nicht *freien* im Sinne von Debian) Päckchen.

Wir sind am Ende. Während des Schreibens hat sich die Anzahl der Debian-Pakete um ein paar tausend vermehrt. Debian GNU/Linux lebt. Nicht alle Zweige grünen, aber doch so viele, dass aus der zarten Pflanze von 1993 ein stattlicher Baum geworden ist.