Linux auf der Kommandozeile

(Entwurffragment, Version. 0.5)

Vilmar Klemt

Inhaltsverzeichnis

1. Einleitung

Die meisten PC-Nutzer sind mit ihrem Windows-Betriebssystem zufrieden. Jedenfalls sprechen die Verkaufszahlen dafür. Auf den zweiten Blick sieht man aber auch:

- Nahezu alle PCs werden mit Windows ausgeliefert, wobei das Betriebssystem in der HOME-Version mit etwa 50 Euro zu Buche schlägt. Für Alternativen muss man entweder im Internet suchen (PCs ohne Betriebssystem) oder viel Geld ausgeben (Apple).
- Die annähernde Monopolstellung von Microsoft wirkt sich bis in Schulen und PC-Kurse hinein aus, sodass viele Menschen nicht wissen, dass es noch etwas anderes gibt, und das noch dazu gratis.

Die Alternative heißt natürlich LINUX. Allerdings ist LINUX nicht jedermann gleichermaßen anzuraten, wohl aber allen, die

- jung sind oder sich jung fühlen und interessiert sind, ihr Betriebssystem mehr als nur oberflächlich kennenzulernen, um sich gegebenenfalls selbst helfen zu können,
- mehr als einen PC besitzen und ein kleines Heimnetzwerk aufbauen wollen,
- eventuell selbst Software entwickeln oder modifizieren möchten,
- sich im Studium oder als Hobby mit technischen oder wissenschaftlichen Problemen befassen wollen.

Im Unterschied zu Windows gibt es aber bei LINUX verschiedene Varianten (Distributionen), bei `distrowatch.com` findet man derzeit (18. Mai 2017) 289. Die meisten sprechen allerdings spezielle Interessen an, und haben deshalb geringe Verbreitung.
Ich werde mich im Wesentlichen an DEBIAN orientieren, das ich am besten kenne, verspreche aber, auf Kompatibilität mit UBUNTU/MINT zu achten, die ja DEBIAN-Abkömmlinge sind.
Innerhalb der Distributionen gibt es meist noch mehrere Varianten, die sich vor allem in der Wahl der Oberfläche unterscheiden. Gemeinsam ist ihnen heute noch überwiegend das zu Grunde liegende X-Window-System (kurz X oder X11), auf das dann die verschiedenen Window-Manager aufsetzen. Diese unterscheiden sich sehr stark in Bedienung und Funktionsumfang, was bis zu Glaubenskriegen führen kann.

Damit komme ich zu einem der wesentlichsten Merkmale, in denen sich LINUX von Windows unterscheidet. Bis zu Windows 3.11 (Mitte der 90er Jahre) waren die Fenster nur ein DOS-Aufsatz, man konnte solche Rechner in die Konsolenumgebung booten und ohne graphische Fenster arbeiten. Das heutige Windows läßt sich unmittelbar nur mehr von der Oberfläche aus bedienen, sodass den meisten Benutzern die Konsole unbekannt bleibt (dabei ist die heutige POWERSHELL im Vergleich zur überlieferten DOS-Shell sehr leistungsfähig).
Bei LINUX bzw. UNIX hat die Konsole, dh. die Kommandozeile, immer eine entscheidende Rolle gespielt, auch nachdem 1984 das X-Window-System hinzukam. Schließlich war und ist UNIX vor allem ein Betriebssystem für Entwickler ("Programmer's Workbench"), da würde die graphische Oberfläche eher stören.
Auf der Kommandozeilenebene unterscheiden sich die Distributionen zumeist nur wenig, am ehesten noch im Paketmanagement und in der Filesystem-Hierarchie. Und hier sind DEBIAN, UBUNTU und MINT nahezu gleich.

Wie schon in der Titelzeile ausgesagt, soll im Folgenden auf die Bedienoberfläche (Desktop) nur ausnahmsweise eingegangen werden. Vom Leser wird daher erwartet, dass er bereits über ein funktionierendes LINUX verfügt und es im Rahmen des Üblichen bedienen kann. Insbesondere sollte er wissen, wie er Zugang zum Internet bekommt, wie er das System auf dem aktuellen Stand hält und wie er zusätzliche Software installiert.

Weitergehende Linux-Kenntnisse setze ich nicht voraus, empfehle dem Leser aber sehr, die beschriebenen Anwendungsbeispiele nachzuvollziehen und auf der Kommandozeile selbst auszuprobieren (gelegentlich könnten sie bei ihm vieleicht nicht so funktionieren wie bei mir). Darüberhi-

naus sollten bei Kommandos die zugehörigen Manualseiten (-> S. 9) konsultiert (wenn auch nicht unbedingt durchgearbeitet) werden, auch wenn das im Text nicht ausdrücklich angemerkt ist.

Das Ziel des vorliegenden Textes ist, den Leser mit praktischen Beispielen und den zugehörigen Erläuterungen an das Titelthema heranzuführen und ihn so zu eigenem Studium anzuregen. Literaturangaben und Verweise auf Internetseiten sollen dabei helfen.

Auch wenn das Arbeiten auf der Kommandozeile selbst unter Linux nicht sehr populär ist, so gibt es natürlich doch auch dafür umfangreiche Literatur, allerdings ganz überwiegend in englischer Sprache. Dementsprechend sind die meisten Begriffe und Fachausdrücke im englischen Sprachraum entstanden und werden häufig unverändert auch im Deutschen verwendet (z. B. *User, Shell, Prompt*).
Ich werde deshalb häufig die englischen Begriffe in Klammern hinzufügen.

2. Erste Schritte

Wir beginnen unsere Kommandozeilentour mit dem Öffnen eines Terminalfensters. Die meisten Distributionen bringen bereits mehrere entsprechende Terminalprogramme mit, und bieten deren Aufruf von der Benutzeroberfläche aus, etwa unter Applikationen/Terminal-Emulationen, an. Ich verwende `xterm`, das wohl das bekannteste ist und über den gleichlautenden Paketnamen installiert werden kann.
Das Terminal meldet sich auf dem Desktop in Form eines Fensters geringer Größe (z. B. 80×25 Zeichen), das durch einen Klick auf den zweiten Button im rechten oberen Eck auf den gesamten Desktop erweitert weden kann.
Normalerweise wird das ein interaktivesTerminal mit normalen Benutzerrechten sein. Zu erkennen ist dies am Eingabeprompt, einer Zeichenfolge, die meist mit einem Dollar- (`$`) oder Prozent- (`%`) Zeichen endet. Unmittelbar rechts davon erscheint der Cursor, manchmal farbig abgesetzt oder blinkend, der zur Eingabe von der Tastatur auffordert.
Wir haben es hier mit einem im Hintergrund arbeitenden speicherresidenten Programm zu tun, dem Kommandozeileninterpreter, kurz und neudeutsch auch Shell genannt. Es wird durch den Aufruf des Terminalprogramms automatisch in Gang gesetzt, anders wäre Benutzerinteraktion über die Tastatur nicht möglich. Bei Systemen ohne graphische Benutzeroberfläche geschieht Vergleichbares am Ende des Bootvorgangs.

Für Kommandos, die ins System eingreifen, reichen normale Benutzerrechte nicht aus. Anders als unter Windows kann man unter Linux keine Administratorkonten mit beliebigem Namen einrichten. Es gibt hier nur ein Administratorkonto, und das trägt den Namen `root`. Bei Ubuntu ist es standardmäßig deaktiviert. Um ein Kommando aufzurufen, das `root`-Rechte verlangt, gibt man am Eingabeprompt `sudo` *Kommando* ein, und nach der Eingabe des verlangten Passworts wird das Kommando dann ausgeführt.
Mehrfache Eingaben dieser Art sind natürlich umständlich. Man aktiviert dann besser das Administratorkonto durch die Eingabe `sudo passwd`. Man wird dann zunächst nach dem Normalbenutzerpasswort gefragt und dann nach dem neu zu vergebenden `root`-Passwort. Anschließend kann man dann mittels `su -` oder `su -l` dauerhaft in das Administratorkonto wechseln (der Parameter `-l`, oder auch nur `-` sagt, dass wir eine Login-Shell einrichten wollen, d.h. eine Shell, wie sie bei einem normalen Login in Aktion tritt).
Als Eingabeprompt erscheint jatzt (an Stelle von `%` oder `$`) das Zeichen `#`.

Bei Debian wird man bereits während der Installation aufgefordert, ein `root`-Passwort zu wählen, das Administratorkonto ist damit permanent eingerichtet.
Zu beachten ist, dass verschiedene Distributionen hier erheblich voneinander abweichen können, es empfiehlt sich, die jeweiligen Websites zu konsultieren.

Als Administrator kann man mit `adduser` *userid* neue Benutzerkonten einrichten.
Das System fragt dann nach dem Passwort und einigen weiteren Angaben. Für den neuen Benutzer `vitus` sieht dann der Ablauf etwa wie folgt aus (`cd` (change directory) wechselt in ein anderes Verzeichnis):

```
# adduser vitus
adduser vitus
Adding user ‘vitus’ ...
Adding new group ‘vitus’ (1001) ...
Adding new user ‘vitus’ (1001) with group ‘vitus’ ...
Creating home directory ‘/home/vitus’ ...
Copying files from ‘/etc/skel’ ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for vitus
Enter the new value, or press ENTER for the default
        Full Name []: Vilmar Klemt
```

```
Vilmar Klemt
        Room Number []: 5
5
        Work Phone []: 040...
041...
        Home Phone []: 02141...
02141...
        Other []: ...
...
Is the information correct? [Y/n]
```

In der letzten Zeile dieses Protokolls wird man aufgefordert, die Korrektheit des Vorhergehenden mit der Eingabetaste (¡Return¿) zu bestätigen oder durch die Eingabe `n <Return>` th zu verneinen.

Passworte ändern kann man mit dem bereits erwähnten Kommando `passwd`, z. B. mit `passwd` *userid* oder einfach nur `passwd` für den Administrator.

Nun wollen wir uns ein bisschen in den Verhältnissen umsehen, unter denen wir leben.

Alle Daten auf einem PC sind in Form von Dateien (files) gespeichert, die wiederum in Verzeichnissen (Ordnern, directories) zusammengefasst sind. In jedem Verzeichnis können sich beliebig viele Unterverzeichnisse befinden, die wiederum Dateien enthalten können. Umgekehrt hat jedes Unterverzeichnis nur ein Elternverzeichnis (parent directory). So ergibt sich eine Baumstruktur (meist stellt man sich einen auf den Kopf gestellten Baum vor), an dessen Spitze sich das Wurzelverzeichnis befindet, das durch einen Schrägstrich (slash, `/`) bezeichnet wird. Jede Datei und jeder Ordner hat einen Namen, jede reguläre (-> S. 7) Datei hat auch eine in Bytes angebbare Größe. Über den Namen und den Pfad, dh. die vom Wurzelverzeichnis her reichende Kette von Unterverzeichnissen, ist jede Datei eindeutig identifizierbar und auffindbar. Unterverzeichnisse und Dateien werden innerhalb eines Pfades durch Schrägstriche voneinander getrennt dargestellt.
Bemerkung: Bei Windows ist der Rückwärtsschrägstrich (backslash) das Trennzeichen und es gibt im Allgemeinen mehrere Wurzelverzeichnisse (C:, D:, usw.).
Nun wollen wir natürlich wissen, in welchem Verzeichnis wir durch das Öffnen des Terminalfensters gelandet sind.
`pwd` (print working directory) gibt uns den Pfad des Verzeichnisses an, in dem unsere Shell läuft, z. B. mit dem Ergebnis `/home/vitus`.
Mit `cd` (change directory) können wir in einen anderes Verzeichnis (Directory) wechseln. Wenn wir als Administrator (`root`) eingeloggt sind, wird dies zunächst das Directory `/root` sein, das HOME-Directory des Administrators. Bisher und im folgenden habe ich `#` als Prompt-Zeichen für den Administrator gewählt, so wie `$` für den Normal-User. Je nach Konfiguration gibt das System eine längere Zeichenkette (String) aus, die auch den Namen des aktuellen Users, des Rechners (`hostname`, hier `pluto`) und den aktuellen Pfad enthält.
Ein typisches Verlaufsprotokoll sieht also etwa folgendermaßen aus:

```
root@pluto ~ # pwd
pwd
/root
root@pluto ~ # su -l vitus
su -l vitus
vitus@pluto ~ $ pwd
pwd
/home/vitus
vitus@pluto ~ $ cd /root
cd /root
cd:cd:6: permission denied: /root
1 vitus@pluto ~ $ exit
```

```
exit
1 root@pluto ~ # cd /home/vitus
cd /home/vitus
root@pluto /home/vitus # pwd
pwd
/home/vitus
root@pluto /home/vitus #
```

Wir sehen, dass das HOME-Directory, hier `/root` für den Administrator, abgekürzt durch die Tilde (`~`) dargestellt wird, so wie auch das HOME-Directory `/home/vitus`, nachdem wir mit `su -l vitus` zu dem Benutzer `vitus` gewechselt sind. Übrigens sind wir mit `exit` wieder zum Administratorkonto zurückgekehrt, dh. wir haben eine Schale der SHELL-Struktur abgestreift.

An dieser Stelle noch ein Hinweis zur Notation in diesem Text: In der Folge werden bei der Beschreibung von Kommandos die Prompt-Zeichen der Shell (`$` oder `%` für den Normal-User bzw. `#` für den Root-User) weggelassen. In den Protokollen der Terminalausgaben erscheinen sie aber weiterhin.

Wir nehmen jetzt an, dass wir uns wieder als Administrator im Verzeichnis `/root` befinden. Den Inhalt des Verzeichnisses lassen wir uns mit `ls -la` anzeigen:

```
root@pluto ~ # ls -la
ls -la
total 332
drwx------  7 root root   4096 Aug  8 07:37 .
drwxr-xr-x 22 root root   4096 Jul 29 16:36 ..
drwx------  2 root root   4096 Jul 26 18:49 .aptitude
-rw-r--r--  1 root root    570 Jan 31  2010 .bashrc
drwx------  3 root root   4096 Jul 26 19:13 .config
-rw-r--r--  1 root root  11447 Jul 27 10:36 ebuff-menu.el
-rw-r--r--  1 root root  62429 Jul 27 10:36 .emacs
drwx------  3 root root   4096 Jul 29 16:04 .emacs.d
drwxr-xr-x  2 root root   4096 Jul 26 21:13 .kbd
-rw-r--r--  1 root root    140 Nov 19  2007 .profile
drwx------  2 root root   4096 Jul 26 20:55 .ssh
-rw-r--r--  1 root root 139264 Aug  7 20:09 typescript
-rw-------  1 root root    749 Jul 26 21:21 .viminfo
-rw-------  1 root root     51 Aug  2 20:59 .Xauthority
-rw-r--r--  1 root root     32 Jul 26 20:01 .xinitrc
-rw-r--r--  1 root root  38183 Jul 28 18:51 .zcompdump
-rw-r--r--  1 root root     17 Aug  8 07:36 .zdirs
-rw-------  1 root root  19553 Aug  8 07:37 .zsh_history
-rw-r--r--  1 root root      0 Jul 26 18:49 .zshrc
root@pluto ~ #
```

Die Auflistung gliedert sich in neun Spalten. Die erste Spalte umfasst in jeder Zeile zehn Zeichen. Das jeweils erste kennzeichnet die Art der Datei (die letzten drei dieser Kennungen finden wir normalerweise nur in dem Geräte-(Device-)Verzeichnis `/dev` vor):

- `-` reguläre Datei
- `d` Verzeichnis (directory)
- `l` Soft Link (-> S. 10)
- `c` Character Device
- `b` Block Device
- `p` Named Pipe
- `s` Socket

Die letzten neun Zeichen der ersten Spalte geben die der Datei zugeordneten Rechte an. Sie unterteilen sich in drei Tripel, jeweils für den Eigentümer der Datei, die Gruppe (-> später), sowie die übrige Welt. Innerhalb jedes Tripels kennzeichnet das erste Zeichen das Leserecht (`r` read), das zweite das Schreibrecht (`w` write) und das dritte das Ausführbarkeitsrecht, bzw., bezogen auf Verzeichnisse, das Recht, darin zu suchen (`x` execute/search). Ein Minuszeichen (`-`) bedeutet, dass das betreffende Recht nicht gesetzt ist.
Jedem dieser Tripel entspricht ein Oktalwert (3 Bits), dh. eine Zahl zwischen 0 und 7. Dem Bit für das Lesen kommt der Wert 4 zu, dem für das Schreiben der Wert 2 und dem für das Ausführen der Wert 1. Für die Datei `.emacs` (`-rw-r--r--`) entspricht das dem Oktaltripel `644`.
Mit dem Kommando `chmod` kann man die Rechte neu setzen (soweit die eigene Berechtigung dafür aureicht). Mit dem Kommando

```
chmod 755 .emacs
```

könnte man so die Datei `.emacs` (sinnloserweise) exekutierbar machen.
Das Kommando `chmod` bietet auch die Möglichkeit, an Stelle absoluter Setzungen Änderungen von Rechten vorzunehmen. Dabei bedeutet ein Pluszeichen (`+`) ein zusätzliches Recht, ein Minuszeichen ein Wegnehmen eines Rechts. Die Adressaten dieser Änderungen sind je nachdem der Eigentümer (`u`), die Gruope (`g`) oder die übrige Welt (`o`). Das obige Kommando wäre in unserem Fall also äquivalent zu

```
chmod ugo+x .emacs
```

`chmod` ist allerdings viel reichhaltiger und komplizierter als hier dargestellt. Das Kommando `info coreutils` gibt umfassende Auskunft, was hier den Rahmen sprengen würde.
Die zweite Spalte des (`ls -la`)-Listings gibt die Zahl der Hard Links (-> später) an, die auf die Datei verweisen, die dritte und vierte den Eigentümer der Datei und die zugehörige Gruppe, die fünfte die Größe der Datei (exakt aufs Byte genau! Bei Verzeichnissen hat sie eine andere Bedeutung). Die Spalten sechs bis acht geben Modifikationsdatum bzw. -uhrzeit der Datei an, die neunte schließlich den Dateinamen.
Die Eigentümer- und Gruppenzugehörigkeit lässt sich für Dateien und Verzeichnisse mit dem Kommando `chown` ändern, z. B.:

```
chown vitus:vitus .emacs
```

oder z. B. `chown -R vitus:vitus .ssh`

Die Option `-R` macht die Veränderung rekursiv für das ganzen Verzeichnis wirksam.

Man beachte die Fülle an Information, die hier auf kleinstem Raum geboten wird und vergleiche mit der dürftigen aber graphisch aufgemotzten Darstellung im Windows Explorer!

Für jede Datei (und jeden Ordner) werden so nicht nur die Rechte des Eigentümers sondern auch die der angeführten Gruppe (sowie der übrigen Welt) beschrieben.
Dateien und Verzeichnisse, die mit einem Punkt beginnen, werden nicht angzeigt, wenn man beim Auflisten statt `ls -la` nur `ls -l` angibt. Das gilt auch für die mit einem bzw. zwei Punkten (`.` und `..`) bezeichneten Verzeichnisse, das laufende (current) bzw. übergeordnete (parent) Verzeichnis.

`ls -laR` listet zusätzlich rekursiv den Inhalt der Unterordner auf.

Man kann auch gezielt eine durch ein Muster gekennzeichnete Auswahl von Dateien auflisten lassen,
`ls -l *.c` liefert beispielsweise eine Liste von Quelldateien der Programmiersprache C. Dabei wird man im allgemeinen Wildcards (-> S. 12) verwenden, im angegeben Beispiel steht der Stern (`*`) für beliebig viele beliebige Zeichen.

Solche Listings können schnell sehr umfangreich werden. Es bieten sich zwei Möglichkeiten an, damit umzugehen:

- Man kann die Ausgabe eines Programms (Standard-Output) in eine Datei umleiten mit `ls -laR >` *listing.txt*. Mit `>>` statt `>` wird sie statt dessen an das Ende einer bereits bestehenden Datei angehängt.
- Man kann (jetzt schwelge ich mal in Anglizismen) den Output an einen Pager pipen mit

`ls -la | less`.

Auf Deutsch: Mit dem Pipe-Zeichen `|` wird die Ausgabe eines Programms zur Weiterverarbeitung als Standard-Eingabe an ein anderes weitergeleitet.
Der Pager `less` kann auch als eigenständiges Programm benutzt werden mit `less` *textdatei*. Es wird dann der Inhalt der Datei am Bildschirm ausgegeben, und zwar vom Beginn an soweit er auf eine Bildschirmseite passt. Es stehen einem dann unzählige Kommandos zur Verfügung, für mich sind am wichtigsten

`d`	gehe im Text eine halbe Seite nach unten,
`u`	gehe im Text eine halbe Seite nach oben,
`g`	gehe an den Anfang des Datensatzes,
`G`	gehe an das Ende des Datensatzes,
`CTRL-N`	gehe im Text um eine Zeile nach unten,
`CTRL-P`	gehe im Text um eine Zeile nach oben,
`/`*Zeichenkette*	suche im Text nach *Zeichenkette*,
`?`*Zeichenkette*	suche rückwärts im Text nach *Zeichenkette*,
`q`	beende den Pager.

Es dürfte mittlerweile klar sein, dass sich kaum jemand sehr viele Kommandos mit noch mehr Parametern über längere Zeit zuverlässig merken kann. Es ist daher an der Zeit, auf die wichtigste Hilfe aufmerksam zu machen, zumal sie in jedem LINUX-System vorhanden ist, die Manual-Seiten (manual pages oder kurz manpages).
Um z. B. Details über das weiter oben erwähnte Kommando `ls` zu erfahren, gibt man ein:

`man ls` oder, falls nötig: `man ls | less`

Nun kann man im Text navigieren, wie oben beschrieben, und ihn mit `q` wieder verlassen.

Eine weitere wichtige Informationsquelle ist `info`. Allerdings wird das gleichnamige Paket nicht bei allen Distributionen automatisch mitgeliefert, sodass man es eventuell nachinstallieren muss. Aufgerufen wird es einfach durch Eingabe von `info` in einem Terminalfenster. Man erhält eine Liste von Einträgen, sogenannten Knoten (nodes) präsentiert, in der man nach dem gewünschten Programm suchen muss. Die wichtigsten Navigationskommandos sind die folgenden:

`<TAB>`	gehe zum nächsten Knoten
`<Shift><TAB>`	gehe zum vorhergehenden Knoten
↓	blättere eine Seite vorwärts
↑	blättere eine Seite zurück
`/`*Programm*	suche nach dem gewünschten Programm
`?`*Programm*	suche rückwärts nach dem gewünschten Programm
`<Return>`	folge dem Knoten, öffne neues `info`-Fenster
`l`	kehre auf die vorige `info`-Ebene zurück
`q`	verlasse `info`

`info` ist vor allem deshalb wichtig, weil die für Linux grundlegend wichtigen Dienstprogramme (core utilities), nur mehr hier umfassend dokumentiert werden. Die Manualseiten sind hingegen des öfteren veraltet (das gilt besonders für das auf S. 8 bereits vorgestellte `chmod`).

Diese Dienstprogramme (es sind etwa 100) sind in dem Paket `coreutils` zusammengefasst, das zur Standardausstattung aller Linuxdistributionen gehört. Die `info` entsprechende Dokumention findet man in verschiedenen Formaten (z. B. PDF) unter

`www.gnu.org/software/coreutils/manual/`

Im folgenden sollen die meiner Meinung nach wichtigsten dieser Programme in Kurzform (mit der Empfelung, sich mit `info` weiter zu informieren) vorgestellt werden. Für einige von ihnen wird man gegebenenfalls ROOT-Rechte benötigen.
Die meisten Programme werden auf der Kommandozeile durch eine Vielzahl von durch ein vorangestelltes Minuszeichen gekennzeichneten Parametern gesteuert. Zum Glück wird man normalerweise die meisten davon nicht benötigen, entweder weil sie nur von speziellem Interesse sind oder weil es für sie passend voreingestellte Werte (sogenannte *Defaults*) gibt.

`cd` *Verzeichnis*	(change directory) wechselt in das Verzeichnis *Verzeichnis*
`ls -laR`	listet rekursiv den Inhalt des laufenden Verzeichnisses und aller seiner Unterverzeichnisse auf (einschließlich versteckter Dateien)
`cp -dp` *Datei Ziel*	(copy) kopiert *Datei* unter Beibehaltung von Links und Rechten
`cp -i` *Datei Ziel*	kopiert Datei mit Warnung vor Überschreiben
`cp -R` *Verzeichnis Pfad*	kopiert Verzeichnis rekursiv
`dd`	kopiert Dateien und Partitionen, sowie Ausschnitte derselben auf niedriger Ebene (`->` S. 12)
`rm -i` *Dateien*	(remove) löscht Dateien mit Nachfrage
`rm -R` *Verzeichnis*	löscht Verzeichnis rekursiv
`rmdir` *Verzeichnis*	entfernt leere Verzeichnisse
`shred -u` *Dateien*	entfernt *Dateien* unwiderruflich
`shred` *Partition*	überschreibt *Partition* mehrfach
`mv` *Quelle Ziel*	(move) benennt um oder verschiebt nach Zielverzeichnis; wichtig: Verzeichnisse können nicht über Partitionsgrenzen hinweg verschoben werden (Ersatz: `cp -dpR` und `rm -R`)
`ln -s` *Datei Link*	legt einen Soft-Link auf die Datei (oder den Ordner) *Datei* an. z. B.: `ln -s /etc/alternatives/vi /usr/bin/vi` erzeugt im Verzeichnis `/usr/bin` einen Link auf die Datei `/etc/alternatives/vi`. Ein Aufruf des Editors `vi` ruft dann tatsächlich auf dem Umweg über das im Pfad liegende Verzeichnis `/usr/bin` die Datei `/etc/alternatives/vi` auf. Im Verzeichnis `/usr/bin` wird der Link mit dem Aufruf `ls -l vi` als `vi -> /etc/alternatives/vi` angezeigt.
`mkdir` *Verzeichnis*	erzeugt ein Verzeichnis
`echo` *String*	gibt die gegebenenfalls in einfache oder doppelte Anführungszeichen eingeschlossene Zeichenkette *String* auf dem Terminal aus
`echo $`*Variable*	gibt den Wert der Variablen *Variable* aus Mehr zu `echo` `->` S. 22f.
`cat` *Datei1 ... Datei2 ...*	gibt den Inhalt von Dateien auf dem Terminal (Standard-Output, `->` S. 17) aus (Vorsicht bei Binärdateien); kann um- (`>`) oder weitergeleitet (`|`) werden; Umleitung (`>`*Datei*) resultiert in Verkettung der Ausgangsdateien
`head -n` *m Datei*	listet die ersten *m* Zeilen einer Datei auf (Default 10) (man kann statt `-n` *m* auch einfach `-`*m* schreiben)
`tail -n` *m Datei*	listet die letzten *m* Zeilen einer Datei auf (Default 10) (man kann statt `-n` *m* auch einfach `-`*m* schreiben)
`split -l` *m Datei*	spaltet Datei *Datei* in Teile zu je *m* Zeilen plus Rest auf
`split -b` *m Datei*	spaltet Datei *Datei* in Teile zu je *m* Bytes plus Rest auf
`split -b` *m*`K` *Datei*	spaltet Datei *Datei* in Teile zu je *m* Kilobytes plus Rest auf
`touch` *Datei*	setzt den Zeitstempel von *Datei* auf den momentanen Augenblick oder erzeugt eine neue leere Datei
`pr [`*Optionen*`]` *Datei*	bereitet eine Textdatei zum Ausdrucken vor
`nl -b a` *Datei*	versieht sämtliche Zeilen einer Textdatei mit einer Nummer
`nl -b t` *Datei*	versieht sämtliche nicht leere Zeilen einer Textdatei mit einer Nummer
`cut -c` *m-n Datei*	gibt die Spalten *m* bis *n* einer Textdatei (z. B. Tabelle) auf Standard-Output aus
`cut -d ' ' -f` *m-n Datei*	gibt zeilenweise die durch Leerzeichen getrennten Felder *m* bis *n* einer Textdatei auf Standard-Output aus

`sort -d` *Datei*	sortiert die Zeilen einer Textdatei lexikalisch
`sort -n` *Datei*	sortiert die Zeilen einer Textdatei numerisch
`pwd`	(print working directory) gibt den absoluten Pfad des aktuellen Verzeichnisses aus
`df -T`	informiert über die Belegung von gemounteten Partitionen
`du -sk .`	gibt die Datenmenge (in kB) im laufenden Verzeichnisbaum an
`printenv`	gibt die gesetzten Umgebungsvariablen und ihre Werte aus
`uname -a`	gibt Informationen über den aktuellen Kernel aus
`whoami`	gibt den gegenwärtigen Benutzernamen aus
`who`	listet auf, wer gerade eingeloggt ist.
`date`	gibt Datum und Uhrzeit aus
`chmod`	setzt die Zugriffsrechte von Dateien und Verzeichnissen ein (-> S. 8)
`chown`	setzt Eigentümer- und Gruppenzugehörigkeit von Dateien und Verzeichnissen (-> S. 8)
`kill` *PID*	eliminiert den Prozess (-> `ps -AF` weiter unten) *PID* (`kill -9` *PID*: bedingungslos)
`killall -r` *String*	eliminiert Prozesse entsprechend dem Textmuster *String*
`stty`	ändert Terminalzeilen-Einstellungen und listet sie auf
`stty sane`	saniert kompromittierte Terminalzeilen-Einstellungen

Es folgen einige weitere wichtige Kommandos, die nicht Teil der `coreutils` sind (näheres zu Netzwerken -> S. 49).

`which -a` *Datei*	listet die Pfade auf, die zu dem Programm *Datei* führen
`ps -AF`	listet, in ausfürlicher Form, alle Prozesse auf
`diff` *Datei1 Datei2*	listet die Unterschiede zweier Textdateien auf
`free -t`	listet Informationen über Arbeits- und Swapspeicher auf
`fdisk -l`	listet die Partitionen aller Datenträger auf
`ifconfig -a`	informiert über die Netzwerkschnittstellen
`ifconfig` *interface* `up`	aktiviert die Schnittstelle *interface* (-> S. 50)
`iwlist scan`	sucht und listet Wireless-Verbindungen
`route`	listet die Routing-Tabelle auf
`ping` *IP-Adresse*	testet die Netzwerkverbindung zu *IP-Adresse*
`apt-get update`	aktualisiert die Paketlisten
`apt-get install` *Paket*	installiert das Paket *Paket*
`apt-get dist-upgrade`	aktualisiert das gesamte System

`COLUMNS=200 dpkg -l "*" | grep "^.i" | less` Listet alle installierten Pakete auf (`COLUMNS=200` ermöglicht extra lange Zeilen)

`mount -t ext3 /dev/sda5 /mnt/data` mountet die 5. Partition von `/dev/sda` (die mit dem Dateisystem `ext3` formatiert ist) in das Verzeichnis `/mnt/data`

`umount /dev/sda5` oder `umount /mnt/data` unmountet `/dev/sda5`

3. tar, dd, gzip, zip, find und grep

3.1 tar

Auf Unix-Systemen ist `tar` (*tape archive*, da es ursprünglich für Backups auf Bandlaufwerken verwendet wurde) das Archivierungssystem der Wahl.
Der Aufruf von `tar` hat die folgende Struktur:

`tar` *Funktion* `[`*Optionen*`] -f` *Archivpfad Datenpfade*

Die wichtigsten Funktionen sind

`-c`	Erzeugen eines Archivs (create)
`-r`	Anhängen an ein existierendes Archiv
`-x`	Extrahieren eines Archivs
`-t`	Auflisten des Inhalts eines Archivs

Die wichtigsten Optionen sind

`-v`	Ausführung wortreich (verbose) protokollieren
`--exclude=`*Muster*	Ausschließen von Dateien
`-h`	Symlinks folgen
`-k`	Alte Dateien nicht überschreiben
`-O`	Extrahieren nach Standard-Output
`-p`	Rechte beibehalten
`-P`	Absolute Pfade erlauben (beginnend mit `/`)
`-C` *Verzeichnis*	Zum Extrahieren in ein anderes Verzeichnis wechseln
`--no-recursion`	Verzeichnis nicht automatisch rekursiv verarbeiten
`--one-file-system`	Beim Erzeugen eines Archivs im lokalen Dateisystem bleiben
`-z`	Komprimieren bzw. dekomprimieren mit `gzip` bzw. `gunzip`

Es gibt noch sehr viel mehr Parameter, `man tar | less` gibt Auskunft.
Die Datenpfade werden normalerweise relativ gewählt.
Sie können Wildcards enthalten, mit folgenden Bedeutungen:

`*`	Beliebig viele (einschließlich 0) Zeichen
`?`	Genau ein beliebiges Zeichen
`[abcd]`	Genau eines der Zeichen `a`, `b`, `c` oder `d`
`[!abcd]`	Genau ein beliebiges Zeichen außer `a`, `b`, `c` oder `d`
`[a-z]`	Ein beliebiger Kleinbuchstabe
`[0-9]`	Eine beliebige Ziffer
`[a-zA-Z0-9]`	Ein beliebiges alphanumerisches Zeichen

Zu beachten ist dabei, dass ein Stern zu Beginn (z. B. `*foo`) mit einem Punkt beginnende Dateien, sogenannte versteckte Dateien, (z. B. `.foo`) ausschließt.
Ein typisches Kommando, um das Home-Verzeichnis zu sichern, sieht also folgendermaßen aus (mehrere aus je einem Zeichen bestehende Parameter kann man zusammenfassen):

```
vitus@pluto ~ $ tar -cvf /mnt/backups/vithome.tar .
```

Der Punkt als Symbol für das laufende Verzeichnis schließt die versteckten Dateien ein.
Wenn man statt des Archivpfades ein Minuszeichen (`-`) angibt, wird der Inhalt der Quelldateien auf Standard-Output (Terminal) ausgegeben. Was auf den ersten Blick sinnlos erscheint, macht einen Backup (z. B. des Home-Verzeichnisses) über das Netzwerk möglich (mehr über `ssh` im Kapitel Netzwerke, `->` S. 54):

```
vitus@pluto ~ $ tar -cvf - . | ssh vitus@192.168.168.216 dd of=/back/vithome.tar
```

3.2 dd

Bei dieser Gelegenheit soll auf des Kopierprogramm `dd` aufmerksam gemacht werden. Seine wichtigsten Parameter sind:

`if=`*InputFile*	Inputdatei oder -gerät (Device)
`of=`*OutputFile*	Outputdatei oder -gerät (Device)

`bs=`*n*	Blockgröße
`count=`*n*	Anzahl der Blöcke
`skip=`*n*	Überspringen von *n* Blöcken auf dem Inputmedium
`seek=`*n*	Überspringen von *n* Blöcken auf dem Outputmedium

Anders als `cp` kann `dd` auch direkt auf Geräte-Dateien zugreifen.
So kann man z. B. (mit Administratorrechten) eine Sicherungskopie des Master-Bootrecods anlegen mit:

```
dd if=/dev/sda of=/backups/mbr.sda bs=512 count=1
```

Eine Datei der Größe 1 GByte, die nur aus Nullzeichen besteht, erzeugt man z. B. mit

```
dd if=/dev/zero of=/mnt/container bs=1024 count=1048576
```

In dieser Datei kann man beispielsweise ein verschlüsseltes Dateisystem einrichten (`cryptsetup`).
Man muss natürlich größte Vorsicht walten lassen, wenn man auf Geräte-Dateien schreibt (z. B. einen neuen Master-Bootrecord anlegen will), man kann sonst in Sekundenschnelle seine ganze Installation zerstören.

3.3 **gzip**

Das Archivierungsprogramm `tar` wird, anders als oben illustriert, meist zusätzlich mit der Option `-z` aufgerufen, wodurch ein komprimiertes Archiv erzeugt wird. Dazu dient das Proramm `gzip`, das man auch direkt aufrufen kann:

`gzip` *Datei*	erzeugt eine komprimierte Datei *Datei*`.gz`
`gzip -d` *Datei*`.gz`	dekomprimiert wieder (wie auch `gunzip`) zu *Datei*
`gzip -c` *Datei*`.gz`	dekomprimiert wie oben, läßt aber die komprimierte Datei unverändert und gibt das Resultat der Dekompression auf Standard-Output aus

Das `tar`-Kommando von weiter oben sieht dann z. B. folgendermaßen aus:

```
vitus@pluto ~ $ tar -cvzf /mnt/back/vithome.tar.gz .
```

Es gibt noch viele weitere Kompressions- bzw. Archivierungsprogramme, z. B. `bzip2`, `lzma`, `7z` und `xz`.

3.4 **zip** und **unzip**

Unter Windows-Benutzern ist `tar` nur wenig bekannt. Wenn es auf Interoperabilität ankommt, ist daher das Packprogramm `zip` einen Blick wert.
Der Aufruf kann sehr komplex sein (`->` die umfangreiche Manualseite), hier beschränken wir uns auf die am häufigsten vorkommende Struktur (die Endung `.zip` kann weggelassen werden):

`zip -`*Optionen Archiv*`.zip` *Dateien* `-i` *Dateien* `-x` *Dateien*

Unter 'Optionen' versteht man hier einen oder mehrere unmittelbar aneinandergereihte einbuchstabige Schalter (`->` weiter unten), während mit 'Dateien' jeweils im allgemeinen Listen von durch Leerzeichen voneinander getrennten Dateien und Ordnern gemeint sind. Dabei kann auch der Stern (asterisk *) als Wildcard verwendet werden, was eventuell Quotierung nötig macht.
Die nach dem Archiv aufgeführten Dateien werden diesem hinzugefügt, gegebenenfalls beschränkt auf die nach dem Schalter `-i` aufgeführten Dateien. Der Schalter `-x` wiederum dient dazu, bestimmte Dateien von der Archivierung auszuschließen.
Ein einfaches Beispiel ist

```
zip -r foo.zip foo
```

(die Option `-r` steht für rekursiv)

Dadurch wird entweder das Archiv `foo.zip` erzeugt, das die Datei `foo`, bzw. das Verzeichnis `foo` enthält, oder es wird dem bereits bestehendem Archiv `foo.zip` die Datei `foo` bzw. das Verzeichnis `foo` mit den darin enthaltenen Dateien und Unterverzeichnissen hinzugefügt, wobei bei auftretenden Namensgleichheiten die Dateien im Archiv überschrieben werden.
Natürlich akzeptiert `zip` noch viele andere Optionen, wie die Manualseiten ausweisen, darunter die folgenden:

`-h`	Gibt Hilfeinformationen aus
`-v`	Wortreiches (verbose) Protokoll (z. B. `zip -rv` *Archiv Verzeichnis*
`-T` *Archiv*	Testet die Integrität des Archivs
`-d` *Archiv Dateien*	Löscht (deletes) Einträge aus dem Archiv
`-u` *Archiv Dateien*	Update (überschreibt Dateien im Archiv nur durch Dateien neueren Datums)
`-e`	Verschlüsselt Dateien/Verzeichnisse mit Hilfe eines Passworts, z. B. `zip -e -r` *Archiv Verzeichnis* (oder `zip -er` *Archiv Verzeichnis*)

Besonders interessant ist der Paramer `-@`, der es `zip` ermöglicht, von Standard-Input zu lesen. Damit kann man z. B. in einer Kommandozeile Dateien nach bestimmten Kriterien suchen und archivieren. Die `zip`-Manpage gibt ein Beispiel für Quelldateien der Sprache C:

```
find . -name "*.[ch]" -print | zip C-Quelldateien.zip -@
```

Mehr zu `find` weiter unten.
Das Partnerprogramm zu `zip` ist `unzip`, mit dem man `zip`-Archive auspackt. Der Aufruf hat die Struktur

`unzip -`*Optionen Archiv*`.zip` *Dateien* `-x` *Dateien* `-d` *Verzeichnis*

Hier bezieht sich die erste 'Dateien'-Liste auf die aus dem Archiv zu extrahierenden Dateien, gegebenefalls gefolgt von einer durch `-x` eingeleiteten Ausschlussliste. Gegenüber `zip` kommt der Schalter `-d` hinzu, der es erlaubt, die Ausgabe in ein gewünschtes Verzeichnis umzuleiten.
Es gibt noch viele weitere Optionen, darunter die folgenden:

`-l` *Archiv*	Listet den Inhalt des Archivs auf
`-v` *Archiv*	Listet den Inhalt des Archivs wortreich (verbose) auf
`-f` *Archiv*	Extrahiert und aktualisiert bereits existierende Dateien
`-u` *Archiv*	Extrahiert und aktualisiert bereits existierende Dateien, extrahiert zusätzlich auch alle anderen
`-p` *Archiv*	Entpackt das Archiv nach Standard-Output
`-c` *Archiv*	Entpackt das Archiv unter Einschluss der Dateinamen nach Standard-Output
`-a` *Archiv*	Konvertiert gegebenenfall extrahierte Dateien, um Interoperabilität, z. B. mit MS-DOS oder WINDOWS, zu erreichen

3.5 find

Ein paar Worte zu dem oben erwähnten `find`. Dieses Programm erlaubt das Auffinden von Dateien, rekursiv ausgehend von einem spezifizierten Punkt im Dateisystem. Eine vom laufenden Verzeichnis ausgehende Suche erfolgt mit

`find . -name "`*Dateiname*`" -print`

Der Dateiname kann die weiter oben beschriebenen Wildcards enthalten (wobei mit `*` auch Punkte am Namensbeginn gefunden werden). Mit `-iname` statt `-name` schaltet man die Unterscheidung zwischen Groß- und Kleinbuchstaben ab.
Ähnliches wie `find` leistet das Programm `locate`. Man legt zunächst mit `updatedb` eine Datenbank für das Dateisystem an. Das kann ziemlich lange dauern, dafür findet `locate` dann die gesuchten Dateien in Windeseile.

3.6 grep

In gewisser Weise komplementär zu `find` arbeitet `grep` (global regular expression print). Es sucht nämlich nach Zeichenketten (Strings) im Text von Dateien. Der Aufruf erfolgt nach dem Schema

`grep` *Optionen* `"`*Zeichenkette*`"` *Ausgangspunkt*

Im Unterschied zu `find` sucht `grep` per Default nicht rekursiv, sondern nur im Ausgagsverzeichnis. Unter den vielen möglichen Optionen seien einige hier aufgeführt.

`-r`	Sucht rekursiv

`-i`	Ignoriert Groß- und Kleinschreibung
`-I`	In Binärdateien wird nicht gesucht
`-a`	Behandelt Binärdateien wie Textdateien
`-v`	Gibt die Zeilen aus, die die gesuchte Zeichenkette *nicht* enthalten
`-l`	Gibt nur die Namen der Dateien aus, in denen Treffer gefunden wurden
`-L`	Gibt nur die Namen der Dateien aus, in denen kein Treffer gefunden wurde
`-n`	Stellt den Ausgabezeien eine Zeilennummer voran
`-A` *n*	Gibt *n* Zeilen nach dem gefundenen Kontext aus
`-B` *n*	Gibt *n* Zeilen vor dem gefundenen Kontext aus
`-E`	Interpretiert die *Zeichenkette* als erweiterten regulären Ausdruck (`->` weiter unten)

`grep` kennt eine Reihe von Zeichen mit spezieller Bedeutung:

`.`	Genau ein beliebiges Zeichen (anders als bei der Shell)
`^`	Beginn der Zeile
`$`	Ende der Zeile
`\<`	Beginn eines Wortes (ein Wort besteht aus Buchstaben und Ziffern)
`\>`	Ende eines Wortes (ein Wort besteht aus Buchstaben und Ziffern)
`[...]`	Ein Zeichen aus der durch ... gekennzeichneten Zeichenkette (`->` S. 12)

Um ein Beispiel anzuführen, das Kommando

```
grep -E -i -r "[mn]+en$" .
```

sucht in Dateien des laufenden Verzeichnisse und seiner Unterverzeichnisse nach Zeichenketten am Zeilenende, die entweder aus einer beliebigen Zahl aufeinander folgender Buchstaben `m` oder aber `n`, gefolgt von der Zeichenfolge `en` bestehen, wobei Groß- und Kleinschreibung keine Rolle spielen soll.

Zusätzlich zu den auf S. 12 beschriebenen Spezifikationen für Zeichenketten gibt es bei grep einige weitere, z. B. `[:alnum:]` , `[:cntrl:]` , `[:punct:]` (`->` Manpage).
Das Folgende bezieht sich im Zweifelsfall auf erweiterte reguläre Audrücke (`grep -E ...`), bei denen die Zeichen `?`, `+`, `|`, `{`, `}`, `(`, und `)` eine besondere Bedeutung haben.
Will man ein solches Zeichen literal verwenden, muss man es in eckige Klammern einschließen oder mit einem Backslash maskieren (schützen).

Wichtig sind auch die Wiederholungsoperatoren, die angeben, wie oft ein ihnen vorangehendes Zeichen oder eine vorangehende Gruppe unmittelbar nacheinander vorkommen soll. Eine Gruppe ist eine in runde Klammern eingeschlossene Zeichenkette (z. B. `(foo)`). Es gelten folgende Regeln:

`?`	Zeichen oder Gruppe kommt nullmal oder eimal vor
`*`	Zeichen oder Gruppe kommt nullmal oder beliebig oft vor
`+`	Zeichen oder Gruppe kommt einmal oder beliebig oft vor
`{`*n*`}`	Zeichen oder Gruppe kommt genau *n* mal vor.
`{`*n*`,}`	Zeichen oder Gruppe kommt *n* mal oder öfter vor.
`{,`*m*`}`	Zeichen oder Gruppe kommt höchstens *m* mal vor.
`{`*n*`,`*m*`}`	Zeichen oder Gruppe kommt mindestens *n* mal und höchstens *m* mal vor.

Der senkrechte Strich (`|`) kennzeichnet eine Oder-Abfrage,

```
grep -E -r "Zeichenkette1|Zeichenkette2" .
```

liefert also einen Treffer, wenn in einer Datei des laufenden Verzeichnisses oder eines seiner Unterordner entweder *Zeichenkette1* oder *Zeichenkette2* gefunden wird.

Ein Beispiel hatten wir bereits weiter oben:

```
COLUMNS=200 dpkg -l "*" | grep "^.i" | less
```

Listet alle installierten Pakete auf (`COLUMNS=200` ermöglicht extra lange Zeilen)

Als Abschluss zum Thema `grep` ist noch zu bemerken, dass man, anscheinend abhängig von der Implementierung, eventuell vor der Suche auf die richtige Lokalisierung achten muss. Ist diese auf Englisch voreingestellt, dann findet man in einer Textdatei unter Umständen (!?) keine deutschen

Umlaute. Um dieses Problem zu beheben, muss man die Umgebungsvariable (-> S. 21) `LANG` auf Deutsch einstellen:

```
export LANG=de_DE.iso8859-1
```

3.7 Bemerkungen zu Kodierungssystemen

Die Unterschiede zwischen den verschiedenen Kodierugssystemen ist natürlich ein komplexes und wichtiges Thema. Am wichtigsten (für die lateinisch schreibende Welt) sind Latin-1 (ISO8859-1), Windows-1252 (auch CP 1252 genannt) und UTF-8. Glücklicherweise sind die beiden ersten praktisch (bis auf die Zeichen zwischen hexadezimal 80 und 9F, bzw. dezimal zwischen 128 und 159) gleich. Im Zusammenhang mit `zip` und `unzip` sind aber die Unterschiede in der Zeilenende-Kennung bei Textdateien wichtig:

Unix/Linux	MSDOS/Windows	MacOS
Linefeed (Nr. 10, LF)	CarriageReturn+Linefeed (Nr. 13+10, CR+LF)	CarriageReturn (Nr. 13, CR)

Das Kodierungssystem Latin-1 (ISO8859-1) umfasst insgesamt 256 Zeichen ($2^8 = 256$). Die obere Hälfte davon stellt eine Erweiterung des ursprünglichen ASCII-Zeichensatzes dar (ASCII = American Standard Code for Information Interchange), dementsprechend stellt nur die untere Hälfte einen weltweit einheitlichen Standard dar (7-Bit ASCII). Im engeren Sinn druckbare Zeichen sind (wenn man das Leerzeichen mit einbezieht) nur der Nummern 32 bis 126. Wenn man also 100% sicher gehen will, sollte man z. B. bei der Email nur diese Zeichen verwenden.
Die untersten 32 Zeichen umfassen das Nullzeichen sowie Steuerzeichen. Man sollte die Bedeutung einiger weniger kennen. Man kann sie am Terminal mit `CTRL-V CTRL-X` (`^V^X`) eingeben, wobei X ein Kleinbuchstabe oder eine Ziffer zwischen 3 und 8 ist (siehe auch S. 40). Das kann nützlich sein, z. B. wenn Standard-Input verlangt wird (-> S. 17).

Nummer	Kürzel	Unix	Bedeutung
3	ETX	^C	Kommandoabbruch (End of Text)
4	EOT	^D	Ende der Eingabe (End of Transmission)
7	BEL	^G	Alarmton (Bell)
8	BS	^H	Backspace
9	HT	^I	Horizontaler Tabulator
10	LF	^J	Zeilenvorschub (Linefeed)
12	FF	^L	Seitenvorschub (Formfeed)
13	CR	^M	Wagenrücklauf (Carriage Return)
17	DC1	^Q	Neustart des Terminal-Outputs (X-ON)
19	DC3	^S	Anhalten des Terminal-Outputs (X-OFF)
26	SUB	^Z	Suspendieren des laufenden Jobs
27	ESC	^[	Escape

Vorsicht: Eingabe von `^D` am Kommandoprompt schließt das X-Terminal augenblicklich und ohne Warnung.
Dieses Verhalten kann man mit `set -o ignoreeof` abschalten (-> S. 18).

4. Standard I/O (input/output) und Umleitung

Einige allgemein wichtige Eigenschaften der Shell sind bisher zu kurz gekommen.

Zunächst: Mit "Shell" soll hier immer die BASH (Bourne Again Shell) gemeint sein, die bei den meisten Linux-Distributionen als Standard eingerichtet wird, sowohl auf der Konsole als auch im X-Terminal. Sie gibt sich durch einen Eingabe-Prompt zu erkennen, an dessen Ende (wie schon auf S. 6 beschrieben, entweder das Zeichen `#` (für den Root-User) oder das Zeichen `$` (manchmal auch `%`, für den Normal-User) erscheint.

Im einfachsten Fall gibt ein Programm, das von hier aus gestartet wird, sein Resultat auf das Terminal aus. Man nennt das Standard-Ausgabe (standard output). Wie wir bereits gesehen haben (-> S. 8), kann dieses Resultat mit Hilfe des Umlenkungszeichens `>` in eine Datei geschrieben, bzw. mittels `>>` an eine Datei angehängt werden.

Nun gibt es analog dazu auch die Standard-Eingabe (standard input), die durch das Zeichen `<` dargestellt wird. Ein Programm, das von Standard-Input liest, erwartet Eingabe von der Tastatur. Das Programm `cat` ist ein Beispiel dafür. Sein üblichster Gebrauch besteht in der Ausgabe einer (typischerweise kleinen) Textdatei auf das Terminal (Standard-Output), also `cat` *Textdatei* .

Man kann `cat` aber auch ohne Parameter aufrufen. Testen Sie dies, indem Sie `cat` eingeben und die Eingabetaste (RETURN) drücken. `cat` erwartet jetzt von Ihnen Texteingabe über die Tastatur mit Abschluss durch RETURN.Sie sehen, dass der eingegebene Text ein zweites Mal erscheint, das Terminal aber nicht freigegeben wird. Sie können dieses Spiel beliebig oft wiederholen, erst die Eingabe von RETURN *und* `<Strg> d` (man schreibt zumeist `CTRL-D`, eigenartigerweise mit großem D, oder auch `^D`) beendet das Spiel und gibt das Terminal wieder frei.

Natürlch ist das keine sinnvolle Anwendung, wohl aber die Erzeugung kleiner Textdateien durch die Methode des Hier-Dokuments (here document), wie im folgenden protokollarisch dargestellt:

```
vitus@pluto ~ $ cat >> textdatei << .
> Dies ist der Text unserer Botschaft.
> Nichts Besonderes, aber man sieht wie es geht.
> Ende des Texts.
> .
vitus@pluto ~ $
```

Auf die Eingabe von `cat >> textdatei << .` (abgeschlossen durch RETURN) erfolgt die Ausgabe des Zeichens `>` am Beginn der nächsten Zeile, womit die Erwartung weiterer Texteingabe angezeigt wird. Die Eingabe eines Punktes am Zeilenbeginn beendet die Texteingabe.
Man kann im obigen Beispiel statt des Punktes auch eine beliebige Zeichenfolge (z. B. `EOF`) wählen, um das Ende der Eingabe festzulegen.
Durch die Eingabe von `cat textdatei` kann man anschließend überprüfen, ob die `textdatei` korrekt erzeugt worden ist.

Ein Beispiel für zweifache Umleitung ist das Umcodieren von Dateien. Im Internet wird vielfach `UTF-8` verwendet (eine typische Zeile in HTML-Dokumenten lautet `<meta charset="UTF-8"/>`). Mit (die Leerzeichen nach `<` und `>` können wegfallen)

`recode u8..lat1 <` *utf8-Datei* `>` *utf8-Datei*`.lat`

kann man eine solche Datei nach Latin-1 umcodieren.

Leitet man die Ausgabe eines Programms mittels `>` *Datei* in eine Datei um, so werden trotzdem eventuelle Fehlermeldungen weiterhin auf das Terminal ausgegeben. Die Fehlerausgabe ist nämlich neben dem Standard-Output und dem Standard-Input ein dritter I/O-Kanal. Man sagt: Die Fehlermeldungen werden auf Standard-Error ausgegeben. Möchte man sowohl Standard-Output als auch Standard-Error in Dateien umleiten, so ersetzt man das obige `>` *Datei* durch `>` *Datei1* `2>` *Datei2*.
Möchte man beides gemeinsam in eine Datei umleiten, so schreibt man statt dessen `&>` *Datei*.

Eine Entsprechung hierfür gibt es auch für das Weiterleitungs- (Pipe) Symbol `|`. Will man die Ausgabe eines Programms an einen Pager weiterleiten, kann es sinnvoll sein, auch den Standard-Error miteinzubeziehen. Dies wird durch die Zeichenfolge `|&` geleistet, oder indem man die Deskriptoren von Standard-Error und Standard-Output zusammanführt, z. B.

`grep -i -a -r "`*Zeichenkette*`" `*Verzeichnis* `|& less` oder
`grep -i -a -r "`*Zeichenkette*`" `*Verzeichnis* `2>&1 | less`

Fehlermeldungen könnten z. B. auftauchen, wenn in *Verzeichnis* Dateien vorkommen, für die kein Leserecht besteht.

Die geschilderten Umlenkmechnismen können natürlich leicht zu versehentlichem Überschreiben von Dateien führen. Das führt uns zu der Möglichkeit, das Verhalten der Shell im Detail durch Variable und Optionen zu steuern. Es gibt nämich die Option `noclobber`, die versehentliches Überschreiben verhindert. Sie wird eingeschaltet durch

`set -o noclobber` und eventuell wieder ausgeschaltet (!) durch `set +o noclobber`.

Das Kommando `set -o` listet sämtliche durch den Parameter `-o` eingeleiteten Optionen und ihre Werte auf. Probieren Sie es bitte aus!
Will man trotz der Option `noclobber` überschreiben, so muss man das Zeichen `>` durch `>|` ersetzen.

Das kommando `set` kennt außer `-o` noch eine ganze Reihe anderer Parameter. Welche das sind und was sie bedeuten, erfährt man aber nicht etwa durch eine Manualseite (die gibt es nicht) sondern mit `help set`.
Information darüber, welche von diesen gesetzt sind, ist in der Variablen `$-` enthalten, wird also durch den Aufruf `echo $-` auf dem Terminal ausgegeben.

5. Hintergrund-Jobs und Prozesse

Wenn man, z. B. von der Shell aus, ein Programm aufruft, so startet man damit einen Job. Jedem Job entspricht mindestens ein Prozess (eventuell auch mehrere). Überdies laufen stets mehrere Prozesse, die nicht vom Benutzer, sondern vom Betriebssystem gestartet worden sind.
Da Linux von Anfang an als Multi-User- und Multi-Tasking-System ausgelegt worden ist, können mehrere Jobs, sei es von einem Benutzer gestartet, sei es von mehreren, gleichzeitig ablaufen.
Wenn man nur ein Terminal geöffnet hat, kann man einen zweiten Job nur starten, wenn man den ersten in den Hintergrund schiebt und einen dritten nur, wenn auch der zweite im Hintergrund läuft. Dies erreicht man durch Anhängen eines Ampersands (`&`) an das Ende der Eingabezeile. Nur so wird das Terminal für weitere Tastatureigaben wieder freigegeben. Hintergrund-Jobs sind besonders nützlich, wenn es um Aufgaben geht, die viel Zeit in Anspruch nehmen (z. B. umfangreiche Datentransfers), aber kaum Terminalausgaben verursachen und deren Erledigung nicht unmittelbar dringend ist.
Bei einem Programm, das ein eigenes Fenster öffnet, bedeutet "Hintergrund" (Aufruf mit angehängtem `&`) lediglich die Freigabe des Ausgangsterminals; es selbst bleibt im Vordergrund (aktives Fenster) und ist somit selbstverständlich für Interaktionen verfügbar. In der Icon-Leiste (meist am unteren Rand des Bildschirms) erscheint es als hervorgehobener Eintrag (wie gegebenenfalls auch das jeweils aktive Terminalfenster). Durch Mausklick auf diesen kann man das Programm in den Hintergrund schicken (ebenso durch Eingabe von `CTRL-Z` auf der Tastatur) und durch erneuten Mausklick wieder in den Vordergrund holen.
Beispiel: `emacs&` (-> S. 36).
Linux ermöglicht einem überdies, die Priorität zu bestimmen, mit der ein Job abgearbeitet werden soll. Das Programm, das dies erledigt, heißt `nice`, was wohl heißen soll, dass man zu verschiedenen Programmen, bzw. Prozessen, verschieden nett sein kann. Die Manualseite sagt uns, dass die höchste Priorität `-20`, die niedrigste `+19` ist. Als leicht ausfürbares Beispiel wollen wir eine große Datei erzeugen, die nur aus Nullzeichen besteht (-> S. 13). Das kann durchaus sinnvoll sein, und Linux stellt eine Device zur Verfügung, die genau diese Nullzeichen liefert (zu Devices kommen wir hoffentlich später noch ausführlicher). Wir geben also z. B. ein (bitte ausprobieren):

```
nice -n 12 dd if=/dev/zero of=/tmp/zerofile bs=512 count=2000000 &
```

Am Erscheinen eines neuen Eingabeprompts sieht man, dass das Terminal sofort wieder freigegeben wird. Information über alle Hintergrund-Jobs bekommt man durch

`jobs -l` und über alle Prozesse mit dem Namen `dd` durch
`ps -af | grep "\<dd\>"`

Hat man nur einen Hintergrund-Job, so kann man ihn mit dem Kommando `fg` wieder in den Vordergrund holen, hat man mehrere, so geht es mit `fg %`*Job-Nummer* oder mit `fg` *Prozess-ID*.
Das Protokoll des Ablaufs sieht z. B. folgendermaßen aus:

```
root@augias test # nice -n 19 dd if=/dev/zero of=zerofi bs=512 count=6000000&
[1] 689
root@augias test # jobs -l
[1]+   689 Running                 nice -n 19 dd if=/dev/zero of=zerofi bs=512 count=6000000 &
root@augias test # ps -af | grep "\<dd\>"
root       689 32083  0 20:41 pts/19   00:00:00 dd if=/dev/zero of=zerofi bs=512 count=6000000
root       693 32083  0 20:41 pts/19   00:00:00 grep \<dd\>
root@augias test # fg
nice -n 19 dd if=/dev/zero of=zerofi bs=512 count=6000000
6000000+0 records in
6000000+0 records out
3072000000 bytes (3,1 GB) copied, 350,418 s, 8,8 MB/s
root@augias test #
```

Man sieht, dass `jobs -l` nicht nur die Job-Nummer (eine kleine einstellige Zahl) sondern auch die Prozess-ID ausgibt, die man auch mit `ps -af` erhält.

Manchmal will man aber auch einen Job (bzw. einen Prozess) vorzeitig beenden. Im allgemeinen gelingt dies, wenn man die Berechtigung hat, mit `kill` *Prozess-ID*. Auf diese Weise wird ein Prozess ordnungsgemäß beendet, dh. es können auch die damit verbundenen Aufräumungsarbeiten durchgeführt werden. Gelegentlich ist dies aber nicht möglich. Dann kann man zu dem brutaleren Mittel `kill -9` *Prozess-ID* greifen.
Das `kill`-Kommando ist übrigens viel mächtiger als hier angedeutet, es geht um an Prozesse gesendete Signale. Man kann sie sich durch `kill -l` auflisten lassen.
Die wichtigsten sind (mit den zugehörigen Nummern) HUP (hangup, 1), INT (interrupt, 2), QUIT (3), KILL (9), SEGV (segmentation violation, 11), TERM (terminate, 15), CONT (continue, 18), STOP (stop, 19), STP (suspend, 20).
Die Manualseiten und vor allem Wikipedia (Unix signal) geben ausführlichere Auskunft.
`interrupt` entspricht übrigens der Tastatur-Eingabe `CTRL-C` und `suspend` der Eingabe von `CTRL-Z`. `kill` ohne Angabe einer Nummer bedeutet `kill -15`.
Man kann Prozesse auch beenden, wenn man ihren Namen (oder einen Teil davon) kennt, z. B. den Firefox durch `killall -r fire`.

6. Variable und Quotierung

Wir haben weiter oben (-> S. 18) gesehen, dass die Shell durch Optionen (z. B. `noclobber`) gesteuert wird. Eine ähnliche Rolle spielen auch Variable, insbesondere die Umgebungsvariablen. Diese und ihre Werte kann man sich mit `printenv` ausgeben lassen, besser noch mit `printenv | sort`, um sie gleich alphabetisch sortiert zu erhalten. Die meisten dieser Variablen wird man nicht verändern wollen, immerhin haben wir ein Beispiel dafür auf S. 16 gesehen, wo es nötig war, die Sprachvariable `LANG` abzuändern.
Den Wert einer einzelnen Variablen kann man sowohl mit `printenv` als auch mit `echo` abfragen, nämlich mit

`printenv` *Variable* und, aber etwas anders, mit `echo $`*Variable*

Wenn eine Variable als Aufrufparameter eines Kommandos (z. B. `echo`) verwendet wird, muss man ihr ein `$`-Zeichen voranstellen, nicht dagegen, wenn man sie neu definiert oder abändert, z. B.:

`export` *Variable*`=`*Wert*	setzt *Variable* auf *Wert*
`export` *Variable*`=''`	setzt *Variable* auf den Null-String
`unset` *Variable*	macht die Variable undefiniert

Durch `set -o nounset` erreicht man übrigens, dass im Falle des Ansprechens einer undefinierten Variablen eine Fehlermeldung erfolgt.
Man kann den Variablennamen auch in geschweifte Klammern einschließen: `${`*Variable*`}`. Manchmal ist das sogar notwendig, um den Variablennamen vom folgenden Programmtext abzugrenzen.
Dagegen wird `echo $(`*Kommando*`)` (mit runden Klammern) verwendet, um das Ergebnis eines Kommandos auszugeben. Anders als beim direkten Aufruf des Kommandos erscheint das Ergebnis in Form von durch einfache Leerzeichen getrennten Feldern in einer einzigen Zeile. Daher wird diese Schreibweise vor allem in Skripten angewandt.
Wird der Aufruf, aber in doppelte Anführungszeichen eingeschlossen, `echo "$(`*Kommando*`)"`, liefert er das Ergbnis in getrennten Zeilen.

Damit kommen wir zu den Zeichen, die für die Shell eine besondere Bedeutung haben. Wir wollen sie im folgenden Spezialzeichen nennen.
Das Kommando `echo` gibt im einfachsten Fall exakt den Text auf dem Terminal aus, den wir auf der Kommandozeile eingegeben haben. Allerdings reduziert es mehrere aufeinanderfolgende Leerzeichen jeweils auf ein einziges. Bestimmte Zeichen in der Eingabezeile werden aber interpretiert, oder wie man auch sagt, expandiert. Eingabe von `echo $LANG` liefert z. B. `en_US.UTF-8` oder `de_DE.iso88591`.
Ein anderes Beispiel ist die Verkettung mehrerer Eingabezeilen mit Hilfe von Backslashes am Zeilenende. Das folgende Protokoll illustriert dies:

```
vitus@pluto /usr/local/linux/bash $ echo       abcdefghij        klmnop    123\
>       4567890       \
> ABCDEFGHIJ
abcdefghij klmnop 123 4567890 ABCDEFGHIJ
vitus@pluto /usr/local/linux/bash $
```

Der Backslash ist vor allem als Escape- (Maskierungs-)Zeichen wichtig. Wird er einem Spezialzeichen vorangestellt, so nimmt er diesem seine spezielle Bedeutung, dh. `echo \$LANG` liefert `$LANG`.
Die Zeichen der besonderen Art spielen vor allem in Skript-Dateien eine wichtige Rolle. Bevor wir aber dazu kommen, müssen wir noch ein paar Konzepte behandeln. Zunächst eine Liste der Spezialzeichen (Nach Newham und Rosenblatt [3]):

`~`	HOME-Verzeichnis
`` ` ``	Kommando-Substitution durch Backquotes (veraltet)
`#`	Kommentar
`$`	Variablenausdruck
`&`	Hintergrund-Job

*	Wildcard für Zeichenkette
(	Beginn einer Subshell
)	Ende einer Subshell
\	Escape- bzw. Quotierungszeichen
|	Weiterleitungszeichen (Pipe)
[	Beginn einer Wildcard-Zeichengruppe
]	Ende einer Wildcard-Zeichengruppe
{	Beginn eines Variablennamens oder einer Kommandogruppe
}	Ende eines Variablennamens oder einer Kommandogruppe
;	Trennzeichen für Shell-Kommandos
'	Starkes Quotierungszeichen
"	Schwaches Quotierungszeichen
<	Standard-Input-Umleitung
>	Standard-Output-Umleitung
/	Pfadnamenverzeichnis-Separator
?	Wildcard für ein einzelnes Zeichen
!	Logisches NICHT (NOT) oder Beginn einer History-Substitution

Es gibt noch eine andere Möglichkeit (neben dem Escape-Zeichen \), Zeichen von der Interpretation durch die Shell auszunehmen, nämlich die Quotierung. Schließt man eine Zeichenkette zwischen einfache Anführungszeichen (singlequote) ein, so werden alle Zeichen wörtlich genommen, dh. nicht von der Shell interpretiert. Die Singlequote selbst darf allerdings innerhalb der Zeichenkette nicht vorkommen.
In einer zwischen doppelten Anführungszeichen eingeschlossenen Zeichenkette verlieren die meisten Spezialzeichen ihre spezielle Bedeutung, einige werden hingegen interpretiert (expandiert), nämlich $, `, ", \, NEWLINE und unter gewissen Umständen !. Will man sie wörtlich dargestellt sehen, muss man ihnen einen Backslash (\) voranstellen.
Zur Illustration ein Testprotokoll ohne Worte:

```
vitus@pluto ~ $ echo Ein   kleiner            Test!
Ein kleiner Test!
vitus@pluto ~ $ echo Ein   kleiner            Test   ${LANG}
Ein kleiner Test en_US.UTF-8
vitus@pluto ~ $ echo 'Ein   kleiner            Test   ${LANG}'
Ein   kleiner            Test   ${LANG}
vitus@pluto ~ $ echo Ein   kleiner            Test   \${LANG}
Ein kleiner Test ${LANG}
vitus@pluto ~ $ echo Ein   kleiner            Test   \\${LANG}
Ein kleiner Test \en_US.UTF-8
vitus@pluto ~ $ echo Ein   kleiner            Test   ${LANG}    \
> mit Fortsetzung
Ein kleiner Test en_US.UTF-8 mit Fortsetzung
vitus@pluto ~ $ echo Ein   kleiner            Test   $(ls)
Ein kleiner Test Desktop Downloads dwhelper ebuff-menu.el text
vitus@pluto ~ $ echo "Ein   kleiner            Test   $(ls)"
Ein   kleiner            Test
Desktop
Downloads
dwhelper
ebuff-menu.el
text
vitus@pluto ~ $
```

An dieser Stelle soll noch erwähnt werden, dass das Kommando `echo` durch bis zu drei Parameter gesteuert wird:

`-n` Es wird kein abschließendes Newline-Zeichen ausgegeben
`-e` Einige Zeichenkombinationen mit Backslash werden interpretiert
`-E` Es werden keine speziellen Backslash-Kombinationen interpretiert (default)

Die wichtigsten der genannten Backslash-Kombinationen sind die folgenden:

`\b` Backspace (`Ctrl-H`, Ascii-Zeichen Nummer 8)
`\f` Seitenvorschub (`Ctrl-L`, Ascii-Zeichen Nummer 12)
`\n` Neue Zeile (`Ctrl-J` , Ascii-Zeichen Nummer 10)
`\t` Tabulator (`Ctrl-I`, Ascii-Zeichen Nummer 9)

Beispiel: `echo -e "`*String1*`\n`*String2*`"` gibt die beiden Zeichenketten auf getrennten Zeilen aus.

7. Lokalisierung, Kodierungssysteme, Zeichensätze und Tastatureinstellungen

Die Struktur des Textes, der auf dem Bildschirm ausgegeben wird, wenn man auf der Konsole oder in einem Terminalfenster ein Kommando oder ein Programm aufruft, kann von einer Reihe von Einstellungen abhängen.
Da ist zum einen die Lokalisierung, dh. die Sprache, in der z. B. Fehlermeldungen, aber auch die Manualseiten (falls in der gewünschten Sprache vorhanden) ausgegeben werden. Die Standardsprache (default language) wird bereits bei der Linux-Installation abgefragt und festgelegt. Man findet die zugehörigen Einstellungen in der Datei `\etc\default\locale`.
Mit dem Aufruf von

```
dpkg-reconfigure locales
```

erhält man eine Liste von Sprachen präsentiert, mit der Option, einige davon zwecks Generierung der zugehörigen Systemdateien auszuwählen. Anschließend wird man noch aufgefordert (wie bei der Linux-Installation), die Standardsprache festzulegen. Man lernt dabei, dass außer der Sprache auch die Kodierung festgelegt wird, typischerweise

`UTF-8` oder `ISO-8859-1` (besser bekannt als `Latin-1`)

Das Kodierungssystem `ISO-8859-15` (auch bekannt als `Latin-9`) unterscheidet sich von `Latin-1` nur geringfügig. Es enthält unter anderem das Euro-Zeichen, hat sich aber trotzdem nicht recht durchsetzen können.
Die Kodierungen aus der Reihe `ISO-8859-`n benutzen den ASCII-Zeichensatz, dh. die untere Hälfte der durch ein Byte darstellbaren 256 Zeichen sowie die obere Hälfte der letzteren für die Darstellung länderspezifischer Zeichen, die nicht im ASCII-Zeichensatz enthalten sind. Der Bedarf an solchen Zeichen ist natürlich viel größer als 128, sodass jeweils eine Auswahl getroffen werden muss. `Latin-1` enthält die westeuropäischen Zeichen, also z. B. die deutschen Umlaute und das scharfe ß.
`UTF-8` behält ebenfalls den ursprünglichen ASCII-Zeichensatz bei, kodiert aber die länderspezifischen und andere spezielle Zeichen mit zwei bis vier Bytes. Für die Länder mit lateinischer Schrift reichen allemal zwei Bytes, womit sich (siehe den UTF-8-Artikel in der Wikipedia) 63488 Zeichen darstellen lassen. Offenbar wird `UTF-8` immer mehr zum Standard, insbesondere auch im Internet.

Natürlich hindert einen die voreingestellte Sprache nicht, temporär auf eine andere Sprache umzuschalten. Mit Bezug auf `grep` wurde auf S. 16 bereits die Möglichkeit erwähnt, durch Neusetzen der Umgebungsvariablen `LANG` für die gesamte Terminalsitzung auf eine andere Sprache umzustellen. Will man nur für ein einzelnes Kommando die Ausgabe in einer anderen Sprache erreichen, so stellt man einfach ein `LANG=`*Sprache* voran. Ist Deutsch voreingestellt, so erhält man z. B. mit den zwei Kommandos

```
LANG=en_US man lpr
man lpr
```

nacheinander die Manualseite des Druckkommandos `lpr` in englischer und in deutscher Sprache angezeigt.

Das bisher Gesagte hat sich lediglich auf Sprache und Kodierung bezogen, nicht aber auf die benötigten Zeichensätze (fonts). Beschränkte man sich auf Englisch, müsste man sich darum vielleicht nicht weiter kümmern, aber nicht alle Zeichensätze bringen die deutschen Umlaute richtig auf den Bildschirm.
Überdies muss man sowohl auf die Ausgabe in der Konsole (dh. ohne X-Window-System) als auch im X-Terminal-Fenster achten.
Auf der Konsole wird der Zeichensatz durch das Kommando `setupcon` (aus dem Paket `console-setup`) eingestellt, welches die nötigen Daten aus der Datei `/etc/default/console-setup` liest. Mit der Originaleinstellung bin ich nicht glücklich geworden. Die Manualseite `console-setup` hilft bei der Einstellung der Parameter und führte bei mir zu folgendem Satz aktiver (nicht auskommentierter) Zeilen:

```
ACTIVE_CONSOLES="/dev/tty[1-6]"
CHARMAP="ISO-8859-15"
CODESET="Lat15"
FONTFACE="Terminus"
FONTSIZE="8x14"
VIDEOMODE=
```

Die Dateien der Konsolenfonts befinden sich im Verzeichnis `/usr/share/consolefonts`.

Unter dem X-Window-System (man sagt auch einfach "unter X"), das zur Zeit noch nahezu allen grafischen Oberflächen zu Grunde liegt, werden andere Zeichensätze verwendet. Im allgemeinen braucht man sich darum nicht weiter zu kümmern, da die meisten Programme, soweit sie eine grafische Ausgabe benutzen, bereits in geeigner Weise konfiguriert sind.
Uns interessiert die Wahl eines geeigneten Zeichensatzes für das Terminal. Wir setzen hier voraus, dass das Terminalprogramm `xterm` aus einem bereits existierenden, aber als unzulänglich empfundenen Terminal heraus aufgerufen wird:

`xterm -fn` *Font*`&`

Für die gängigsten Zeichensätze gibt es kurze Aliasnamen, die man unter `/etc/X11/fonts` findet, z. B. `/etc/X11/fonts/misc/xfonts-base.alias`. Die Fontdateien selbst findet man unter `/usr/share/fonts/X11`, z. B. `/usr/share/fonts/X11/misc/6x13-ISO8859-1.pcf.gz`.
Statt des Parameters `-fn` kann man für die Auswahl des Zeichensatzes auch `-fs` benutzen, wobei hier eine Größenangabe in Punkteinheiten (printer points, pt) erwartet wird (ein Punkt misst ca. 0.353 mm, das vorliegende Tutorial benutzt die Größe 11 pt).
Für das Funktionieren dieser Art der Fontspezifikation spielt anscheinend die Datei `~/.Xresources` eine wichtige Rolle, die aber nicht bei allen Distributionen mitgeliefert wird.
`xterm` kennt viele Paramter (die Manpage ist ein Hammer). Einige beschreibt folgende Tabelle:

`-fn`	Fontname
`-fs`	Fontgröße
`-fg`	Vordergrundfarbe
`-bg`	Hintergrundfarbe
`-cr`	Cursorfarbe
`+u8`	setzt die `utf8`-Resource zurück
`+wc`	setzt die `wideChars`-Resource zurück
`-n`	Icon-Name

Ein kompletter Aufruf sieht bei mir folgendermaßen aus:

```
xterm -fs 9 -fg white -bg black -cr red +u8 -n vitus&    oder
xterm -fs 9 -fg white -bg black -cr red +wc -n vitus&
```

Der Parameter `+wc` ermöglicht die richtige Darstellung von Umlauten in der `xterm`-Konsole, nicht aber im Emacs (-> S. 36), der Parameter `+u8` leistet gerade das Umgekehrte.
Je nach Window-Manager (ich verwende Fluxbox), erscheint der Icon-Name z. B. in der Icon-Leiste am unteren Rand des Desktops. Mehrere gleichzeitig geöffnete Terminals (z. B. für LAN-Verbindungen), lassen sich so leichter unterscheiden. Voraussetzung ist dass eine geeignete `~/.Xresources`-Datei im HOME-Verzeichnis existiert und mit dem Kommando

```
xrdb -merge ~/.Xresources
```

aktiviert worden ist.

Üblicherweise wird bei der Linuxinstallation die Tastaturbelegung Deutsch festgelegt. Es gibt aber gute Gründe, gelegentlich auf Englisch zu wechseln. Auch hier muss man wieder zwischen Konsole und X-Umgebung unterscheiden.
Auf der Konsole kann man mit `loadkeys` eine der Tastaturtabellen laden. Diese befinden sich im Verzeichnis `/usr/share/keymaps/i386`. Mit dem Aufruf `loadkeys -d` kann man jederzeit wieder zur Default-Einstellung zurückkehren. Will man einzelne Tasten umkodieren, so kann man selbst eine kleine Tastaturtabelle erstellen.

Eine solche mit dem Inhalt `keycode 58 = Control` setzt z. B. die CapsLock-Taste auf Control um. Dazu muss man freilich den Keycode von CapsLock kennen. Diesen ermittelt man mit dem Programm `showkey`, mit dem man zum Drücken der gewünschten Taste aufgefordert wird.

Unter X kann man die Tastaturbelegung mit dem Programm `xmodmap` verändern. Man erstellt eine Datei, die zeilenweise `xmodmap`-Kommandos enthält, wie z. B.:

```
xmodmap -e "keycode  32 = o O odiaeresis Odiaeresis"
xmodmap -e "keycode  34 = bracketleft braceleft"
```

Rechts vom Gleichheitszeichen können bis zu vier Argumente stehen. Das erste gibt das unmodifizierte Zeichen an, das zweite das Zeichen mit Shift, das dritte das Zeichen mit `Mode_switch` (z. B. die AltGr-Taste) und das vierte mit `Mode_switch` und `Shift`.
Auch hier gibt es wieder ein Programm, das die Keycodes ausgibt, nämlich `xev`. Anders als das Konsolenprogramm `showkey` erhält man auch die Bezeichnung der Taste (z. B. `braceleft`).

Mit `xmodmap` kann man die Tastatur bis ins letzte Detail konfigurieren. Trotzdem wird heute mehr zur Konfiguration mit `setxkbmap` geraten. Wie es in der Manualseite heißt, wird hier die Tastaturbelegung aus einer Reihe von Komponenten zusammengestellt (kompiliert). Diese Komponenten befinden sich alle im Verzeichnis

```
/usr/share/X11/xkb
```

Es handelt sich durchwegs um lesbare Dateien, die auf sechs Unterverzeichnisse verteilt sind. Die Zusammenhänge sind sehr komplex, da es sehr viele Kombinationsmöglichkeiten gibt.
Wenn man mit den Voreinstellungen zufrieden ist, kann im günstigsten Fall alles mit dem einfachen Aufruf `setxkbmap -layout de` für die deutsche Tastaturbelegung erledigt sein. Meistens wird man aber doch die eine oder andere Zusatzspezifikation vornehmen wollen. Bei mir sieht der Aufruf folgendermaßen aus:

```
setxkbmap -v 10  -model pc105 -layout "de,us" -rules evdev \
          -variant ",altgr-intl" -option "" -option "grp:alt_shift_toggle" \
          -option "lv3:lwin_switch" -option "compose:menu"
```

Ein paar Erläuterungen sind hier wohl am Platze.
Zunächst wird die maximale Verbosität 10 eingestellt, was natürlich überflüssig ist, wenn alles klappt. Auch die Parameter `-model` und `-rules` sind hier nicht nötig, weil bereits voreingestellt. Die Angabe `-layout "de,us"` brauchen wir hingegen, da wir ja alternativ auch die amerikanische Tastaturbelegung zur Verfügung haben wollen. `-variant ",altgr-intl"` spezifiert eine Erweiterung der amerikanishen Belegung, die auch Umlaute und akzentuierte Zeichen zugänglich macht. Die nächste Option ermöglicht es uns, durch gleichzeitiges Drücken von `Shift` und `Alt_L` von Deutsch auf Englisch (und umgekehrt) umzuschalten. Die beiden letzten Optionen legen den `Mode_switch` (-> S. 26) auf die linke Windows-Taste und `Compose` auf die `Menu`-Taste.
Letzteres ermöglicht es uns z. B., durch Drücken der `menu`-Taste und anschließendes Eintippen von Apostroph (singlequote ') und `e` ein akzentuiertes é einzugeben.
Wer sich für Details interessiert, findet viel Interessantes z. B. in den Dateien

```
/usr/share/X11/locale/iso8859-1/Compose
/usr/share/X11/xkb/symbols/de
/usr/share/X11/xkb/symbols/us
/usr/share/X11/xkb/rules/evdev.lst
```

Neben `setxkbmap` gibt es noch das Programm `xkbcomp`. Es hat vor allem Bedeutung, wenn X-Server und X-Client auf verschiedenen Maschinen laufen, dh. der Server seine Dienste über das lokale Netz anbietet.
Eine sehr lesenswerte Einführung zur Tastaturkonfiguration unter X findet man übrigens unter

```
www.x.org/releases/X11R7.6/doc/
```

unter dem Titel "The XKB Configuration Guide". Auf nur vier Seiten erfährt man hier, worauf es ankommt, darüberhinaus auch etwas über die mögliche Konfiguration durch Dateien im Verzeichnis `/etc/X11/xorg.conf.d/`.

8. Kurzer Abriss zur Installation von Software-Paketen

Bevor man ein Linux-System auf seinem Computer installiert, muss man sich zunächst darüber klar werden, welche Distribution sich am besten für die persönlichen Wünsche eignet. Bei `distrowatch.com` kann man derzeit (17. Nov. 2015) unter 273 Möglichkeiten auswählen. Die meisten davon verfolgen spezielle Zielsetzungen, sodass ernstlich nur einige wenige im oberen Bereich der Rangliste infrage kommen. Ich werde mich im wesentlichen an Mint orientieren. Sowohl Mint als auch Ubuntu sind ursprünglich aus Debian hervorgegangen, so dass hoffentlich die meisten Erörterungen auch für die beiden letzteren Distributionen gelten sollten. Insbesondere ist ihnen das Paketmanagementsystem gemeinsam, und das ist es, worauf es in diesem Kapitel ankommt.

In der Regel bezieht man die gewünschte Software, soweit sie nicht schon in der Basisinstallation enthalten ist, aus dem zu der Distribution gehörigen Repository. Wo dieses im Internet beheimatet ist, kann man in der Datei `/etc/apt/sources.list` oder in dem Ordner `/etc/apt/sources.list.d` nachlesen. In meiner Mint-Installation heißt die relevante Datei

```
/etc/apt/sources.list.d/official-package-repositories.list
```

und die zentrale Zeile darin (die den Hauptteil der Software abdeckt) lautet

```
deb http://archive.ubuntu.com/ubuntu trusty main restricted universe \
  multiverse
```

Um zu verstehen, was die einzelnen Felder in dieser Zeile bedeuten, ruft man besten im Browser die Seite

`http://www.archive.ubuntu.com/ubuntu/dists/trusty/`

auf. In den verschiedenen Unterordnern findet man die Dateien, die dem Paket-Management die für die Installation notwendige Information über die einzelnen Softwarepakete liefern. Für den interessierten Linuxer bieten diese Dateien einen kurzgefassten Überblick über die in seiner Distribution verfügbare Software.
Die umfangreichste dieser Dateien hat in unserem Falle die Adresse

`http://www.archive.ubuntu.com/ubuntu/dists/trusty/universe/binary-amd64/Packages.gz`

Sie trägt derzeit (17. Nov. 2015) das Datum des 8. Mai 2014, ist also relativ alt. Offenbar sind die Repositories zeitlich aufeinander für (in meinem Falle) die Distribution Linux Mint 17.2 (Rafaela) abgestimmt. Man könnte hier ansetzen, aktuellere Quellen wählen und in `/etc/sources.list.d` eintragen, nähme hiermit aber das Risiko von Inkompatibilitäten oder zumindest Instabilitäten in Kauf.
Risikofreudige Linuxer können auf "Rolling Release"-Distributionen, wie z. B. Siduction oder Linux Mint Debian (`-> http://www.linuxmint.com/download_lmde.php`) umsteigen, die auf höchste Aktualität hin entwickelt worden sind.

Will man sein System auf den neuesten Stand bringen oder neue Software installieren, so muss man zunächst die Information über die Paketquellen aktualisieren. Dies geschieht durch das Kommando

```
apt-get update
```

Anschließend kann man mit

`apt-get install` *Softwarepakete*

ein oder mehrere neue Pakete (Paketnamen durch Leerzeichen getrennt) installieren. Die Aktualisierung bereits installierter Pakete erfolgt mit dem gleichen Kommando.
Will man das ganze System aktualisieren, so geschieht dies mit

```
apt-get dist-upgrade
```

Man sollte dies in nicht allzu großen Abständen (etwa alle ein bis zwei Monate) durchführen, um allzu große Aktualitätssprünge zu vermeiden.
Das ist besonders wichtig für Rolling-Release-Distributionsn, die, wie im Falle Debian, auf einem instabilen (unstable) Zweig des Repositories beruhen. Das Siduction-Manual gibt detaillierte

Anweisungen zur Vermeidung von eventuell irreparablen Schäden an der Installation (arbeiten auf der Konsole unter Abschaltung von X, zurücksetzen des Runlevels auf 3 etc.).
Treten Inkompatibilitäten oder Inkonsistenzen auf, so hilft manchmal das Kommando

```
apt-get -f install
```

wenn nicht, so kommt man um manuelles Eingreifen (z. B. Deinstallieren von Programmpaketen) nicht herum.

Man sollte noch wissen, dass das Kommando `apt-get` auf das Kommando `dpkg` aufsetzt, das auch für den Normal-Linuxer von Interesse ist.
Wie immer geben die Manualseiten Auskunft über die vielfältigen Anwendungsarten dieses Programms, daher sollen hier nur wenige Hinweise folgen. Wichtig ist beispielsweise, dass man mit `dpkg` gezielt eine ältere Version eines Pakets installieren kann. Das kann nötig werden, wenn Inkonsistenzen im System aufgetreten sind, die ein "Downgrading" erforderlich machen. Daher verlangt `dpkg` bei der Installation nicht (wie `apt-get`) als Argument nach dem Paketnamen sondern nach dem Paket*datei*namen, also, um ein konkretes Beispiel zu nennen,

```
dpkg -i evince-common_3.10.3-0ubuntu10.2_all.deb
```

Der Paketname lautet hier `evince-common` (es ist der Teil des Dateinamens bis zum ersten Unterstrich), die Versionskennung ist `3.10.3-0ubuntu10.2` (der zweite Teil des Darteinamens), `all` bezeichet die Architektur (in diesem Fall also "alle", spezifische Architekturen sind z. B. `i386` oder `amd64`).
Die Installation mit `dpkg` ist aber nur möglich, wenn die entsprechende Paketdatei auf dem Rechner lokal vorhanden ist, und zwar in dem Ordner, von dem aus man `dpkg -i` aufruft (es sei denn, man hat dem Dateinamen den Pfad vorangestellt). Man muss sie also erst vom Repository herunterladen. Im obigen Fall findet man sie in dem Ordner

```
http://www.archive.ubuntu.com/ubuntu/pool/main/e/evince/
```

Die Paketdateien installierter Pakete findet man übrigens in dem Verzeichnis
`/var/cache/apt/archives` (wenn man Plattenplatz sparen will, kann man sie dort löschen).
`dpkg`-Kommandos, die sich auf die Paketnamen beziehen sind z. B.:

`dpkg -l "*`*string*`*"`	listet Informationen über installierte Pakete auf, in deren Namen die Zeichenkette *string* vorkommt
`dpkg -P` *Paketname*	entfernt ein Paket einschließlich seiner Konfigurationsdateien

Recht nützlich sind auch folgende Kommandos:

`dpkg-deb -I evince-common_3.10.3-0ubuntu10.2_all.deb`	listet Informationen über eine bestimmte Paketdatei auf
`dpkg-deb -c evince-common_3.10.3-0ubuntu10.2_all.deb`	listet den Inhalt einer bestimmte Paketdatei auf
`dpkg-deb -x evince-common_3.10.3-0ubuntu10.2_all.deb .`	entpackt eine Paketdatei im aktuellen Verzeichnis

Manchmal kennt man den Namen eines Kommandos, weiß aber nicht, zu welchem Paket es gehört. Das folgende Protokoll zeigt am Beispiel von `chmod`, wie es geht:

```
vitus@pluto ~ $ dpkg -S chmod | grep "\<chmod\>"
manpages-dev: /usr/share/man/man2/chmod.2.gz
coreutils: /usr/share/man/man1/chmod.1.gz
coreutils: /bin/chmod
mc: /usr/lib/mc/fish/chmod
vitus@pluto ~ $
```

Man sieht, dass das Kommando `dpkg -S` genau genommen nicht nach Dateinamen sucht, sondern nach Zeichenketten, die in Dateinamen enthalten sind. Daher ist es oft nützlich, den Output dieses Kommandos an `grep` (-> S. 15) weiterzuleiten, um allzuviele Suchtreffer zu vermeiden. Man sieht jedenfalls, dass `chmod` zu dem Paket `coreutils` gehört (die Namensgleichheit mit einer Datei im Paket `mc` (midnight commander) ist vermutlich nur zufällig).

9. Einfache Bash-Skripte

Skripte stellen eine große Erleichterung für die Arbeit auf der Kommandozeie dar.
Im einfachsten Fall erstellt man eine Textdatei, die aus zeilenweise angeordneten Befehlen besteht, die in der gewählten Zusammenstellung bei der Arbeit häufig vorkommen. Durch den Aufruf des Dateinamens werden dann die darin enthaltenen Befehle nacheinander abgearbeitet.

Damit das funktioniert, muss der Anwender Lese- und Ausführrecht für diese Datei besitzen. Das stellt man am besten sicher durch den Befehl (-> S. 8)

```
chmod 755 Skriptname
```

Damit kann jeder das Skript ausführen. Die Ziffer 7 gibt dem Eigentümer zusätzlich noch Schreibrecht.

Damit ein Skript an jedem beliebigen Ort im Dateisystem ausgeführt werden kann, muss es in einem Ordner angelegt sein, der, wie man sagt, im Pfad liegt. Es existiert eine Variable `PATH`, die eine Liste dieser Ordner zum Inhalt hat, z. B.:

```
root@pluto etc # echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
root@pluto etc #
```

Der hier gezeigte Pfad bezieht sich auf den ROOT-User, für den Normal-User wird er im Allgemeinen anders lauten.
Man sieht, dass die einzelnen Verzeichnisse im Pfad durch einen Doppelpunkt (`:`) voneinander getrennt aufgelistet werden. Das aktuelle Verzeichnis ist aus Sicherheitsgründen normalerweise nicht im Pfad enthalten. Um es trotzdem mit eizubeziehen, kann man den Pfad erweitern durch

```
PATH=$PATH":."
export PATH
```

Das Kommando `export` macht die Erweiterung über die aktuelle Sitzung hinaus permanent. Ein Skript, das nicht im Pfad liegt, kann man trotzdem von überall her aufrufen, wenn man seinem Namen seinen kompletten Pfad voranstellt. Vom Verzeichnis des Skripts aus lautet der Aufruf dann einfach *./skript*.

Eine andere Art des Aufrufs ist (wenn das Skript im Pfad liegt) `source` *skript*. Anders als im zuvor geschilderten Fall werden hier die im Skript enthaltenen Kommandos abgearbeitet, als hätte man sie direkt eingetippt, d. h. es wird keine eigene Shell (subshell) aufgemacht, die nach Beendigung des Skripts wieder geschlossen wird. Man kann das Kommando `source` auch durch einen Punkt ersetzen, also einfach *. skript* schreiben. Übrigens muss das Skript bei dieser Art des Aufrufs nicht als ausführbar gekennzeichnet sein (-> z. B. die Datei `/etc/profile`).

Wenn man viel auf der Kommandozeile arbeitet, wird man oft auf schon vorher verwendete Kommandos zurückgreifen wollen. Diesem Wunsch kommt die HISTORY entgegen, in der die vorangegangenen Kommandos gespeichert werden (die eintsprechende Datei lautet beispielsweise `~/.bash_history`). Man kann sich mit ↑ die zurückliegenden Kommandos sukzessive auf der Kommandozeile ausgeben lassen (mit ↓ geht man wieder in die entgegengesetzte Richtung). Gegebenenfalls kann man dann das soeben gelistete Kommando mit RETURN wieder aufrufen.
Man kann es aber auch "editieren", d. h. Veränderungen vornehmen und dann erneut abschicken. Dabei spielt eine Rolle, welcher Editiermodus eingeschaltet ist, `emacs` oder `vi`. Das Editieren orientiert sich dann an dem jeweiligen Editor, was unter Umständen irritierend sein kann. Einfache Kommandos wie ←, → oder BACKSPACE sollten aber wie erwartet funktionieren.
Welcher Editiermodus eingeschaltet ist, sieht man mit `set -o`, neu setzen kann man ihn mit `set -o emacs` oder mit `set -o vi`.

Damit kommen wir zu der Frage, wie man Skripte erstellt. Solange sie nur wenige Zeilen lang sein sollen, kann man die Methode des Here-Documents anwenden (-> S. 17).
Im allgemeinen braucht man aber einen Texteditor. Als Standard-Editor ist auf Unix-Systemen der `vi` stets vorhanden (heute meist in der erweiterten Version `vim`). Er wird zumeist als kontraintuitiv empfunden und ist deshhalb nicht leicht zu erlernen. Bei Programmierern ist er

trotzdem oft erste Wahl, da er Könnern außerordentlich schnelles Arbeiten ermöglicht. Auf vielen Systemen ist ein Tutorial installiert, das man mit `vimtutor` aufrufen kann.
Die wichtigste Alternative ist der `emacs`. Er ist ähnlich leistungsfähig wie der `vim`, und er enthält interessante Features, die beim `vim` fehen, wie das Directory-Listing oder eine asynchrone Shell. Der `emacs` ist ein wahrer Elefant, sein Binary (`\usr\bin\emacs24-x`) ist sage und schreibe 14 Megabyte groß.
Oft wird auch der `nano` (eine Linux-Modifikation von `pico`) empfohlen. Er ist nicht sehr leistungsfähig, dafür aber leichter zu erlernen.
Es versteht sich fast von selbst, dass es für Linux geradezu unzählige weitere Alternativen gibt (elvis, gedit, jed, jedit, joe, mcedit (mc), SciTE (scintilla), vile, ...).
Alle diese Texteditoren sind (mehr oder weniger) gleich gut geeignet für die Erstellung von Skripten, editorspezifische Fragen sollen daher auf später verschoben werden.

Damit kommen wir endlich zu Skript-Beispielen.
Der einfachste Fall liegt vor, wenn wir eine bestimmte Aufgabe wiederholt, eventuell auf verschiedenen Computern auszuführen haben, z. B. das Mounten eines externen Datenträgers mit mehreren Partitionen. Ich möchte beispielsweise nicht, dass dies automatisch geschieht (man muss vielleicht `mountall` deinstallieren), sei es aus Sicherheitsgründen, sei es weil ich den jeweiligen Verzeichnissen eigene charakteristische Namen geben möchte. Andererseits muss ich, wenn ich die Automatik ausschalte, selbst dafür sorgen, dass auch die richtige Block-Device gefunden wird. Dabei hilft mir, dass die Partitionen solcher Datenträger eine weltweit (?) eindeutige Kennung (GUID = Globally Unique Identifier) haben (bzw. bei der Einrichtung der Dateisysteme zugewiesen bekommen). Diese lässt sich z. B. mit `blkid \dev\sdb1` ermitteln.
Ein Skript zum Mounten einer externen Festplatte mit zwei Partitionen könnte dann z. B. so aussehen (die Leerzeichen sind wichtig!):

```
#!/bin/bash
if [ ! -d /mnt/cnmemnt1 ]; then
  mkdir -p /mnt/cnmemnt1
fi
mount -v -t ntfs -o umask=000 -U 4C5CBB035CBAE73C /mnt/cnmemnt1
if [ ! -d /mnt/cnmemex2 ]; then
  mkdir -p /mnt/cnmemex2
fi
mount -v -t ext2 -o noatime,nodiratime -U fe3f8438-5b37-4973-9472-3962299fad9e \
  /mnt/cnmemex2
```

Ein paar Kommentare dazu könnten noch nützlich sein. Die erste Zeile des Skripts hat eine spezielle Struktur. Sie beginnt stets (manchmal kann man sie auch weglassen) mit den Zeichen `#!`, an das sich dann der Pfad der Shell anschließt (dort könnte statt `bash` auch `sh`, `perl` oder `python` stehen, wenn man eine andere Skriptsprache verwenden möchte.
Die erwähnte universelle Kennung wird mit `-U` eingeleitet, die Verzeichnisnamen sollen mich an die Plattenmarke (CN Memory) und das Dateisystem der jeweiligen Partition erinnern, und der Parameter `umask` legt fest, dass von den vollen Zugriffsrechten (777) des Verzeichnisses nichts abgezogen wird (oft ist vom System `umask=022` voreingestellt).
Mit `[ ! -d /mnt/cnmemnt1 ]` und der Folgezeile wird abgefragt, ob das Verzeichnis (directory) `cnmemnt1` *nicht* existiert (das Rufzeichen steht für *nicht*) und neu erzeugt werden muss, gegebenenfalls (dafür steht der Parameter `-p`) einschließlich des übergeordneten (parent) Verzeichnisses `/mnt`.

Was Skripte freilich erst interessant macht, ist die Verwendung von Variablen. Sie ermöglichen es, ein Skript für gleichartige aber im Detail verschiedene Aufgaben zu verwenden. Ein einfaches Beispiel ist die Konvertierung einer Textdatei nach Postscript:

```
enscript -X latin1 -B -f Courier9 -i $1 -L $2 -M A4 -N n --output=$3.ps $3
```

Aufgerufen wird das Skript mit drei Parametern (man nennt sie *Positions*-Parameter), die nacheinander den Variablen `$1`, `$2` und `$3` zugeordnet werden, nämlich die Einrücktiefe, die Zeilenzahl

und der Name der Textdatei, hier im folgenden Beispiel eine Manualseite:

```
enscrn9 10 66 enscript.man
```

Als Ergebnis erhält man dann die Datei `enscript.man.ps`.
Das Beispiel wurde gewählt, weil `enscript` einige einfache Textformatierungen unterstützt, wie sie z. B. in Manualseiten vorkommen. Man findet diese normalerweise in `/usr/share/man`. Allerdings liegen sie im `nroff`-Format vor, sodass man sie zunächst konvertieren muss:

```
nroff -man enscript.1 >enscript.man
```

Wie die Manualseite ausweist, ist `enscript` ein recht mächtiges Programm mit vielen Parametern. Bei etlichen wird man es normalerweise beim Default-Wert belassen, einige andere (wie z. B. die Codierung Latin1, die Schrift Courier9 und das Ausgabemedium A4) wird man im Skript festlegen, sodass sie beim Aufruf nicht angegeben werden müssen.

Neben den eben behandelten Aufrufparametern und den frei definierbaren Variablen spielen in Skripten die fest eingebauten Variablen eine wichtige Rolle (die Variablennamen selbst beginnen nicht mit einem `$`-Zeichen, nur wenn man auf sie Bezug nimmt, muss man ihnen ein solches voranstellen, z. B. gibt das Kommando `echo $$` die Prozess-ID der laufenden Shell aus). Es sind dies

`0`	Der Skriptname selbst
`@`	Die transferierten Aufrufparameter (`$1`, `$2`, `$3`, ...) in getrennten Strings
`*`	Die transferierten Aufrufparameter in einem String zusammengefasst
`#`	Die Zahl der Aufrufparameter
`?`	Der Exit-Status des zuletzt beendeten Kommandos (0 für "ohne Fehler")
`$`	Die Prozess-ID der laufenden Shell
`!`	Die Prozess-ID des zuletzt gestarteten Hintergrund-Jobs

Andere Variable, die in Skripten von Bedeutung sein können, sind `LANG`, `USER`, `HOME`, `SHELL`, `TERM`, `PATH` und vor allem `IFS`.Letztere definiert das Trennzeichen für Felder (`IFS` steht für *Internal Field Separator*) in Textdateien.

Dies gibt uns die Gelegenheit, ein wichtiges Strukturelement in Skripen kennenzulernen, nämlich die `for`-Schleife. Sie ermöglicht uns, eine Reihe von auf die Elemente (Felder) einer Liste bezogenen Anweisungen abzuarbeiten, ohne sie einzeln ausschreiben zu müssen.
Eine Rolle spielt dabei die Möglichkeit, Fehlermeldungen abzufangen, die z. B. ausgegeben werden, wenn ein Objekt unserer Anfrage nicht existiert oder von falscher Struktur ist (wir haben ein ähnliches Beispiel bereits auf S. 30 kennengelernt).
Dazu dienen die Wenn-dann-sonst-Abfragen (`if`, `then`, `elif`, `else` sind die entsprechenden Schlüsselwörter).
Als Beispiel wollen wir uns die Verzeichnisse auflisten lassen, die im Pfad (-> S. 29) liegen. Das folgende Skript erfüllt diese Aufgabe.

```
#!\bin\bash
IFS=:
for dir in $PATH; do
  if [ -z "$dir" ]; then dir=.; fi
  if [ ! -e "$dir" ]; then
    echo "$dir doesn't exist"
  elif ! [ -d "$dir" ]; then
    echo "$dir isn't a directory"
  else
    ls -ld $dir
  fi
done
```

Als erstes sehen wir, dass das Trennzeichen `IFS` auf Doppelpunkt (:) gesetzt wird. Die einzelnen Felder in der Variablen `PATH` werden nämlich durch einen Doppelpunkt voneinander getrennt.

Zwischenbemerkung: Standardmäßig ist `IFS` auf `<space><tab><newline>` gesetzt. Im Englischen bezeichnet man diese Zeichen als *whitespace characters*, um anzudeuten, dass sie bei Terminal- oder Druckerausgabe unsichtbar sind. Diese Zeichen sind auch die natürlichen Trennzeichen für Wörter in einer Textdatei. Mit `echo $IFS` wird auf dem Terminal nur eine Leerzeile ausgegeben. Mehr sieht man mit `echo "'$IFS'"`, am besten in der Emacs-Shell (aufzurufen mit `M-x shell`). Da kann man sich auch die Länge von Zeilen anzeigen lassen, die nur aus Leerzeichen und Tabulatoren bestehen, indem man den Cursor mit `C-e` ans Zeilenende setzt. Dass `IFS` aus drei Zeichen besteht bedeutet, dass jedes der drei als Trennzeichen erkannt wird.

Die mit *for* beginnende Zeile des Skripts weist nun sukzessive der Variablen *dir* die durch einen Doppelpunkt voneinander gatrennten Felder zu, die dann in den folgenden Zeilen verschiedenen Abfragen unterworfen werden. Dies geschieht mit Hilfe der Abfrageoperatoren `-z`, `-e` und `-d`.

Operatoren dieser Art gibt es sowohl für Zeichenketten (Strings) als auch für Dateien und Zahlen. Die Stringabfragen können sich auf zwei Zeichenketten (z. B. `if [ "`*string1*`" = "`*string2*`" ]`) beziehen oder nur auf eine (z. B. `if [ -n "`*string*`" ]`). Die wichtigsten zeigt die folgende Tabelle. Dabei bedeutet Gleichheit soviel wie Identität und Größe bemisst sich nach der Position in der lexikalischen Ascii-Zeichenfolge:

`=`	Abfrage auf Gleichheit
`!=`	Abfrage auf Ungleichheit
`<`	Abfrage auf kleiner
`>`	Abfrage auf größer
`-n`	Abfrage auf Verschiedenheit von Null
`-z`	Abfrage auf Gleichheit mit Null

Die folgenden Operatoren beziehen sich auf Dateien bzw. Verzeichnisse (wie oben wieder auf zwei oder nur eine), wobei die Abfrage *wahr* (*true*) ergibt, wenn die zugehörige Erläuterung zutrifft:

`-e`	Die Datei existiert (unabhängig von Typ und Größe; identisch mit `-a`)
`-d`	Die Datei existiert und ist ein Verzeichnis
`-f`	Die Datei existiert und ist eine reguläre Datei
`-r`	Der Aufrufer hat Leserecht bezüglich der Datei
`-s`	Die Datei existiert und ist nicht leer
`-w`	Der Aufrufer hat Schreibrecht bezüglich der Datei
`-x`	Der Aufrufer hat Ausführungsrecht bezüglich der Datei, bzw. das Recht, in ihr zu suchen, wenn es sich um ein Verzeichnis handelt
`-O`	Der Aufrufer ist der Eigentümer der Datei
`-G`	Der Aufrufer gehört derselben Gruppe wie die Datei
`-nt`	Die erste Datei ist neuer als die zweite
`-ot`	Die erste Datei ist älter als die zweite
`-ot`	Die erste Datei ist älter als die zweite

Einer besonderen Betrachtung wert sind arithmetische Operationen, und zwar sowohl Abfragen als auch die Auswertung arithmetischer Ausdrücke.

Die Angelegenheit wird dadurch kompliziert, dass es für solche Abfragen zwei verschiedene Formulierungen gibt. Die eine arbeitet mit dem `test`-Operator, den wir bereits in seiner Schreibweise in Form eines Paares eckiger Klammern (`[...]`) kennengelernt haben. Innerhalb dieser und von diesen durch Leerzeichen getrennt sind die folgenden Abfrageoperatoren zwischen ganzen Zahlen gültig:

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

Mit diesen gebildete relationale Ausdrücke können noch durch die folgenden Operatoren miteinander verknüpft werden:

`-a`	und
`-o`	oder

Dabei können mit Hilfe von runden Klammern (die backslashgeschützt und durch Leerzeichen abgetrennt sein müssen) Gruppen dieser relationalen Ausdrücke gebildet werden. Zur Illustration empfiehlt es sich, den folgenden Ausdruck auf der Kommandozeile einzugeben:

```
if [ \( 1 -gt 2 \) -o \( 2 -gt 1 \) ]; then echo true; else echo false; fi
```

Für Rechenopertionen stehen in der Bash im Wesentlichen die gleichen Operatoren zur Verfügung, die man auch von anderen Programmiersprachen kennt, also

`+`	Addition
`-`	Subtraktion
`*`	Multiplikation
`/`	Division
`%`	Modulo (Rest einer ganzzahligen Division)

sowie noch einige mehr.

Empfehlenswerter als die Methode mit den eckigen Klammern ist für arithmetische Operationen allerdings die Dollarexpandierung mit den doppelten runden Klammern.

Zur Erinnerung (-> S. 21): Dollarexpandierung mit einfachen runden Klammern ermöglicht es, das Resultat eines Kommandos wie den Inhalt einer Variablen zu behandeln, z. B. `echo $(ls -l)` (gleichbedeutend mit `echo ‘ls -l‘`)
Die relationalen Operationen lauten jetzt aber anders:

`<`	kleiner als
`<=`	kleiner als oder gleich
`==`	gleich
`>=`	größer als oder gleich
`>`	größer als
`!=`	ungleich
`&&`	und
`||`	oder

Um den Wert einer logischen Abfrage im Terminal auszugeben schreibt man jetzt in Anlehnung an obige Kommandozeile (die inneren Klammerpaare kann man weglassen):

```
echo $(((1 > 2) || (2 > 1)))
```

Man erhält den Rückgabewert `1` , der wie z. B. in der Spache `C` für `wahr` steht. Das exakte Gegenstück zur Kommandozeile weiter oben erhält man unter Weglassung des `$`-Zeichens mit

```
if (((1 > 2) || (2 > 1))); then echo true; else echo false; fi
```

wobei auch hier wieder die inneren Klammerpaare überflüssig sind.

Befindet sich innerhalb des Doppeklammernpares statt einer Abfrage ein arithmetischer Ausdruck, so wird er augewertet un das Ergenis ausgegeben, z. B.:

```
echo $((9*7))     ergibt     63
```

Ein hübsches Skript aus dem Buch von Newham und Rosenblatt [3] illustriert einige der eben aufgeführten Möglichkeiten:

```
#!/bin/bash
for dir in ${*:-.}; do
  if [ -e $dir ]; then
    result=$(du -s $dir | cut -f 1)
    let total=$result*1024
    echo -n "Insgesamt fuer $dir = $total bytes"
    if [ $total -ge 1048576 ]; then
```

```
      echo " ($((total/1048576)) Mb)"
    elif [ $total -ge 1024 ]; then
      echo " ($((total/1024)) Kb)"
    fi
  fi
done
```

Zur Funktionsweise dieses Skripts einige wenige Anmerkungen.
Der Ausdruck `${*:-.}` ist ein Beispiel für einen String-Operator (hier nicht behandelt). Eine Variable (hier `$*`, der String der Aufrufparameter) wird auf Existenz abgefragt und im Falle der Nichtexitenz auf einen vorgegebenen Wert (hier `.` fuer das laufende Verzeichnis) gesetzt.
Das Kommando `du` (disk usage) berechnet die Gesamtdatenmenge in einem Verzeichnis. Das Ergebnis wird hier weitergeleitet an `cut -f 1`, wodurch nur das erste Feld der Resultatszeile von `du` im Ergebnis erscheint.
Das Kommando `let` in der fünften Zeile des Skripts weist die Shell an, den folgenden Ausdruck rechts vom Gleichheitszeichen als einen arithmetischen Ausdruck und nicht als einen String zu interpretieren.

Das Beispiel weiter oben mit Abfragen auf `1 > 2` usw. ist natürlich zu trivial, um sinnvoll zu sein. Man wird im Allgemeinen statt dessen mit Variablen arbeiten wollen. Möglich gemacht wird dies durch das Kommando `declare -i`, wodurch Variablen als vom Typ ganzzahlig deklariert werden. Illustriert wird dies am besten durch folgendes Protokoll:

```
$ val1=12 val2=5
$ result1=val1*val2
$ echo $result1
val1*val2
$
$ declare -i val1=12 val2=5
$ declare -i result2
$ result2=val1*val2
$ echo $result2
60
```

Mit dem bisher Gezeigten ist es nicht möglich, Inhalte von externen Dateien zu verarbeiten. Dazu braucht man das Kommando `read`, das von Standard-Input einliest.
Wollen wir z. B. bei sämtlichen Dateien in einem Verzeichnis und all seinen Unterverzeichnissen die Dateiendung `.lis` in `.listing` abändern, so wird dies durch das Skript

```
#!/bin/bash
while read var; do
  mv $var ${var%$1}$2
done
```

bewirkt. Zuerst wird mit `while read var; do` eine Schleife eingeleitet (sie funktioniert ähnlich wie eine solche mit `for`), die zeilenweise von Standard-Input die Variable `var` einliest. In der folgenden Zeile wird mittels eines String-Operators die Endung `$1` von der Variablen `var` abgeschnitten (aber nur im Falle der Übereinstimmung) und durch die Endung `$2` ersetzt. Das externe Kommando `mv` (move) interpretiert die Variable `var` als Dateinamen und ändert diesen entsprechend ab.
Den benötigten Standard-Input erzeugen wir mit dem Kommando `find` (-> S. 14), dessen Resultat wir mittels einer Pipe an unser Skript, das wir `alter` nennen wollen, weiterleiten. Die gewünschten Dateiendungen übergeben wir als Parameter. Der komplette Aufruf sieht dann folgendermaßen aus:

```
find . -name "*.lis" -print | ./alter .lis .listing
```

Im Rahmen dieses Einführungstextes kann natürlich nur eine kleine Auswahl dessen beschrieben werden, was die Bash bietet. Wer Gefallen am Skripting findet, sollte sich allerdings bewusst sein,

dass die Skriptsprachen Perl und Python noch weitaus leistungsfähiger sind als die Bash, und Python in seinen Grundzügen relativ leicht zu erlernen sein soll. Perl ist die ältere der beiden, Python ist konzeptionell moderner, beide sind auf allen Computerplattformen vertreten und haben riesige User-Gemeinden. Die Verlage Rheinwerk (`www.rheinwerk-verlag.de`, vormals Galileocomputing) und O'Reilly (`www.oreilly.com` bzw. `www.oreilly.de`) bieten eine große Auswahl an Büchern, auch in deutscher Sprache.

10. Der Emacs-Editor

Das wichtigste Werkzeug des Programmierers ist der Editor. Im Unterschied zum Vi ist der Emacs nicht standardmäßig Bestandteil jeder Linux-Distribution. Bei Debian-artigen Systemen wird die derzeit neueste Version installiert durch

```
apt-get install emacs24
```

Anders als manch andere Linux-Software ist der Emacs hervorragend dokumentiert. Auf seiner Homepage findet man das `Emacs`-Manual zum Download [6]. Ein hervorragendes Einführungsbuch, ebenfalls in englischer Sprache, ist "Learning Gnu Emacs" von D. Cameron u. a. [7].
Durch Googlen findet man zwei kurze deutsche Einführungstexte [8][9], die allerdings schon etwas betagt sind.
Lokal auf dem Rechner (jedenfalls in meiner Mint-Installation) findet man Tutorials in verschiedenen Sprachen (darunter auch deutsch, Dank an Hermann für den Hinweis) unter

```
/usr/share/emacs/24.3/etc/tutorials
```

Ich werde mich im folgenden an der Referenz-Karte [10] orientieren, die es, wie auch das `Emacs`-Manual bei `gnu.org` gibt, und zwar, anders als dieses, auch auf deutsch. Überdies bezieht sich die Referenzkarte auf die derzeit neueste `Emacs`-Version 24.

Der Emacs kann mit einer Reihe von Parametern aufgerufen werden, wie die Manualseite ausweist, also

`emacs [`*Parameter*`]` *Datei*

Der Aufruf kann von der Konsole aus erfolgen, wir wollen aber im folgenden den Aufruf aus einem X-Terminal-Fenster annehmen. Ruft man von da aus mit dem Parameter `-nw` (no window) auf, dann lädt der Emacs die Datei *Datei* in einen Puffer (buffer) gleichen Namens und stellt deren Anfang im Fenster (window) des X-Terminals dar. Ohne den Parameter `-nw` öffnet der Emacs statt dessen einen eigenen Rahmen (frame), der sich farblich und grafisch deutlich vom X-Terminal abhebt. Damit dieses für anderweitigen Gebrauch freigegeben wird, muss man das Aufrufkommando mit einem Ampersand (`&`) abschließen (-> Hindergrund-Jobs S. 19).
Man kann beim Aufruf *Datei* auch weglassen, dann wird im Fenster oder Rahmen ein Scratch-Puffer angezeigt.
Am unteren Rand des Fensters sehen wir zwei Zeilen. Die Moduszeile (mode line), die gegenüber dem übrigen Text invertiert dargestellt ist, gibt ganz rechts Information über den Modus (-> später), links davon über Zeilen- und Spaltennummer, weiter links eine Prozentangabe über die Lage des Cursors im Text und weiter links den Puffernamen (buffer name).
Die nächsten beiden Zeichen rechts neben dem Doppelpunkt kennzeichnen den Status des Puffers:

`--`	Puffer unverändert nach dem letzten Speichern	nicht schreibgeschützt
`**`	Puffer verändert nach dem letzten Speichern	nicht schreibgeschützt
`%%`	Puffer unverändert nach dem letzten Speichern	schreibgeschützt
`%*`	Puffer verändert nach dem letzten Speichern	schreibgeschützt

Man kann zwischen *schreibgeschützt* und *nicht schreibgeschützt* umschalten mit `C-x C-q` (`read-only-mode`) oder (veraltend) `M-x toggle-read-only`.
Die unterste Zeile, Minipuffer (minibuffer) genannt, dient für Eingaben auf einen Prompt hin (-> später) und zur Ausgabe von Meldungen.
Wird der Emacs mit eigenem Rahmen aufgerufen (ohne `-nw`), erscheint am oberen Rand des Fensters eine Menüzeile, die auf Mausklicks reagiert. Beim Aufruf mit `-nw` bleibt die Maus (abgesehen vom Clipboard) ohne Funktion.

Während des Hochfahrens lädt der Emacs eine Initialisierungsdatei namens `.emacs` aus dem HOME-Verzeichnis des aktuellen Benutzers. Diese Datei enthält Kommandos in der Sprache Emacs-Lisp und kann an persönliche Bedürfnisse angepasst werden (dazu später mehr).

Von den Aufrufparametern seien (neben `-nw`) einige wenige in folgender Tabelle zusammengefasst:

`emacs -nw`	lädt den Emacs im X-Terminal-Fenster (ohne eigenen Rahmen)

`emacs -q`	lädt den Emacs ohne Initialisierungsdatei
`emacs -l` *Lisp-Code*	lädt die Datei *Lisp-Code* an Stelle von `.emacs`
`emacs -f` *Funktion*	exekutiert beim Start die Lisp-Funktion *Funktion*
`emacs --eval` *Ausdruck*	evaluiert beim Start den Lisp-Ausdruck *Ausdruck*

Im Folgenden werde ich mich an die Referenzkarte halten und nur die Kommandos kommentieren, die einer Erläuterung bedürfen.

Das betrifft zunächst die Schreibweise der Kommandos. `C-z`, Emacs unterbrechen (suspend), bedeutet, die Steuerungstaste (Strg, englisch Control oder Ctrl) und `x` gleichzeitig zu drücken, während `M-f` (im Text ein Wort vorwärts springen) verlangt, zuerst die Escape-Taste (ESC) zu drücken, sie dann loszulassen und anschließend die Taste `f` zu drücken. Die Schreibweise mit dem `M` kommt daher, dass es anscheinend auf manchen Tastaturen eine `Meta`-Taste gibt (es kann z. B.die linke `Alt`-Taste sein), bei deren Fehlen die `ESC`-Taste als Ersatz eingesetzt wird.
Ein grundsätzlicher Unterschied zwischen `<ESC>` und `<Ctrl>` besteht übrigens darin, dass `<ESC>` ein eigenes Zeichen in der Ascii-Tabelle ist (dezimal 27, oktal 033), `<Ctrl>` hingegen nicht.

Eine merkwürdige Schreibweise begegnet uns in Kommandos wie z. B. `C-M-s` (Suche mit regulären Ausdrücken). Hier muss zuerst `<ESC>` gedrückt und losgelassen, dann gleichzeitig `<Ctrl>` und `s` gedrückt werden.
Ungewohnt mag auch sein, dass mit der Bezeichnung `DEL` die `Backspace`-Taste gemeint ist, während `C-d` das Löschen des Zeichens unter dem Cursor (delete-char) beschreibt. Bei den meisten `Emacs`-Installationen (aber nicht bei allen!) hat das Drücken der `Entf`-Taste die gleiche Wirkung.

Zu **"Emacs verlassen"**

`C-x`	`suspend-frame`	Emacs unterbrechen (unter X: ikonisieren)
`C-x C-c`	`save-buffers-kill-terminal`	Emacs verlassen

"`kill-terminal`" ist etwas irreführend, denn beim Emacs-Aufruf mit `-nw` wird ein bereits geöffnetes X-Terminal verwendel, das beim Verlassen von Emacs bestehen bleibt.

Zu **"Dateien"**

`C-x C-f`	`find-file`	Datei öffnen
`C-x C-s`	`save-buffer`	Datei speichern
`C-x s`	`save-some-buffers`	Alle noch nicht gesicherten Dateien speichern
`C-x C-v`	`find-alternate-file`	Puffer von der Festplatte neu einlesen oder durch eine neue Datei ersetzen
`C-x C-w`	`write-file`	Puffer unter neuem Namen speichern
`C-x i`	`insert-file`	Datei einlesen und an der Cursor-Position einsetzen

Zu **"Hilfe"**:
Allen Kommandos, die man per Tastenkombination (key) eingibt, liegen Funktionen zugrunde, die oft recht lange, durch Bindestriche unterteilte Namen haben. Man kann sie direkt (dh. ohne die Tastenkombination zu benutzen) aufrufen durch `M-x` *Funktion*, wobei die Eingabe von *Funktion* auf einen Prompt hin im Minibuffer erfolgt. Für viele Funktionen (es sind Tausende) existiert gar keine Bindung an eine Tastenkombination (key binding).

Man wird nun zum einen Hilfe in Anspruch nehmen wollen, wenn man sich der zugehörigen Tastenkombination nicht sicher erinnert, dann hilft das Kommando `C-h k` (oder `C-h c` für eine einzeilige Auskunft). Zum anderen sucht man vielleicht nach einer zu einem Schlüsselbegriff passenden Funktion. Sucht man z. B. nach Funktionen die sich auf Codierungssysteme beziehen, so hilft das Kommando `C-h a` (apropos), konkret `C-h a coding`. Man erhält dann eine Liste von Funktionsnamen, in denen die Zeichenkette `coding` vorkommt.

Da diese Liste den Umfang von wenigen Zeilen übersteigt, wird sie in einem eigenen Fenster angezeigt, entweder in der unteren oder der rechten Hälfte des ursprünglichen Fensters. Das

ursprüngliche Fenster (das linke oder das obere) bleibt das aktive, das den Cursor enthält. Mit `C-x o` kann man das andere Fenster zum aktiven machen, sodass der Cursor in dieses Fenster wechselt und alle Kommandos dort wirksam werden. Führt man z. B. dort den Cursor zu einem Funktionsnamen und drückt `<Return>`, so erhält man eine ausführliche Beschreibung der betreffenten Funtion.

Das Kommando `C-x 1` schließt das jeweils nicht aktive Fenster (oder die nicht aktiven, wenn es mehrere sind).
Das Kommando `C-h f` fragt mit einem Prompt im Minipuffer nach einem Funktionsnamen. Dabei wird man durch Drücken der `<TAB>`-Taste mit Vorschlägen zur Namensvervollständigung unterstützt.

Mit `C-h m` erhält man Information zu den verschiedenen Moden. Wie bereits erwähnt, wird ganz rechts in der Moduszeile der jeweilige Modus angezeigt. Er wird beim Laden einer Datei automatisch der Dateiendung entsprechend eingestellt. Viele Tastenkommandos variieren mit dem Modus. Das ist einerseits willkommen, weil man z. B. dadurch einen an die Programmiersprache des Quelltextes angepassten Satz von Kommandos zur Verfügung gestellt bekommt. Andererseits kann es auch irritieren. In diesem Fall kann man durch Einstellen des Fundamentalmodus mit `M-x fundamental-mode` allen Problemen aus dem Wege gehen.

Nicht beschrieben auf der Referenzkarte ist das Kommando `C-h v`, das Variablen beschreibt (`->` später).

Zu **"Fehlerbehandlung"**:
Bevor man zum ersten Mal ein bisher unbekanntes Programm aufruft, sollte man wissen, wie man notfalls unbeschadet wieder herauskommt.
Beim Emacs gelingt dies mit `C-x C-c`, spätestens nachdem man einige Fragen nach dem Abspeichern von Puffern und/oder dem Beenden von aktiven Prozessen (`->` S. 44) mit `y` oder `yes` beantwortet hat.
Meist ist es allerdings nicht nötig, den Notausgang zu suchen, da der Emacs noch eine Reihe von anderen Möglichkeiten bietet, Fehler zu korrigieren. Die folgende Tabelle bietet einen Überblick:

`C-x C-c`	`save-buffers-kill-terminal`	Emacs verlassen
`C-g`	`keyboard-quit`	Abbruch eines Tastaturkommandos
`C-c C-c`	`comint-interrupt-subjob`	Abbruch eines Shell-Kommandos
`C-l`	`recenter-top-bottom`	Bildschirmanzeige in Ordnung bringen (zentriert die Cursor-Zeile vertikal)
`C-x C-v`	`find-alternate-file`	Buffer von der Festplatte neu einlesen
`C-_`	`undo`	Änderungen rückgängig machen (Control- und Shift-Taste gleichzeitig drücken)

Die Undo-Funktion erlaubt (durch wiederholte Eingabe von `C-_`) eine Vielzahl von Undo-Schritten. Darüberhinaus bietet sie noch die Möglichkeit der Richtungsumkehr. Bemerkt man, dass man zu weit gegangen ist, so kann man die Folge der Undo-Schritte durch ein beliebiges Kommando (z. B. eine Cursorbewegung) unterbrechen und dann durch erneute Eingaben von `C-_` die überzähligen Schritte wieder rückgängig machen (Redo).

Zu **"Inkrementelle Suche"**:
Inkrementell bedeutet, dass die Suche bereits mit der Eingabe von `C-s` und dem ersten Zeichen des Suchstrings im Minibuffer beginnt, der Cursor dann bei weiteren Zeicheneingaben weiter vorwärts springt, gegebenenfalls bis zum Auffinden der gesuchten Zeichenkette. In diesem Fall kann man, solange man die Suche noch nicht mit `<Return>` beendet hat, durch erneutes Eingeben von `C-s` weitere Fundstellen aufsuchen, durch Eingeben von `C-r` die Suchrichtung umkehren oder aber auch mit Hilfe von `<Backspace>` und Eingabe neuer Zeichen die Suche abändern kann, was wiederum durch Cursor-Sprünge quittiert wird.
Falls die Zeichenkette nicht (oder nicht in weiteres Mal) gefunden wird, erhält man die Fehlermeldung "Failing I-search" und bei erneuter Eingabe von `C-s` beginnt `Emacs` die Suche erneut, ausgehend vom Textanfang. Man muss es einfach ausprobieren.

Die Suche nach regulären Ausdrücken wird (statt durch `C-s`) durch das Kommando `C-M-s` (zur Erinnerung: das bedeutet `<ESC> <Ctrl>-s`) eingeleitet. Anders als bei `C-s` haben jetzt einige der im Suchfeld (Minibuffer) eingegebenen Zeichen eine spezielle Bedeutung, ähnlich wie es uns schon im Zusammenhang mit `grep` (-> S. 12 und -> S. 15) begegnet ist. Es gibt aber gewisse Abweichungen gegenüber `grep`, deshalb folgt hier eine kurze Zusammenfassung:

`.`	Genau ein beliebiges Zeichen (anders als bei der Shell)
`^`	Beginn der Zeile
`$`	Ende der Zeile
`\<`	Beginn eines Wortes (ein Wort besteht aus Buchstaben und Ziffern)
`\>`	Ende eines Wortes (ein Wort besteht aus Buchstaben und Ziffern)
`\b`	Anfang *oder* Ende eines Wortes
`[...]`	Ein Zeichen aus der durch `...` gekennzeichneten Zeichenkette (-> S. 12)
`[^...]`	Ein Zeichen *nicht* aus der durch `...` gekennzeichneten Zeichenkette
`|`	Zeichen für "oder" bei alternativer Suche
`\`*n*	Rückbezug auf *n*-te Gruppe (*n* = 1, 2, ... 9 -> S. '`backref`')

Als Wiederholungsoperatoren kennt `Emacs`:

`?`	Zeichen oder Gruppe vor dem `?` kommt nullmal oder eimal vor
`*`	Zeichen oder Gruppe vor dem `*` kommt nullmal oder beliebig oft vor
`+`	Zeichen oder Gruppe vor dem `+` kommt einmal oder beliebig oft vor

Anders als bei `egrep` (`grep -E`) werden Zeichengruppen durch `\(`*string*`\)` geklammert dargestellt (im `Emacs` haben die runden und die geschweiften Klammern keine spezielle Bedeutung). Sollen Zeichen mit besonderer Bedeutung innerhalb eines regulären Ausdrucks literal gesucht werden, so müssen sie mit vorangestelltem Backslash eingegeben (maskiert, geschützt) werden, also `\$`, `\^`, `\*`, `\+`, `\[`, `\]` und `\\`.
Will man alternativ entweder nach *string1* oder nach *string2* suchen, so erreicht man dies durch eine "Oder"-Verknüpfung. Der entsprechende Suchstring lautet dann `\(`*string1*`\|`*string2*`\)`.

Zu **"Cursor-Bewegung"**: Der `Emacs` ist (wie auch der `Vi` so konstruiert, dass er ohne die Cursor-Tasten und ohne `Pos1`, `Ende`, `Bild↑` und `Bild↓` auskommt. Normalerweise funktionieren sie aber erwartungsgemäß, ebenso wie die `Backspace`-Taste, die bei manchen (exotischen) Implementierungen Probleme bereiten kann.
Auf der Referenzkarte werden die Cursor-Kommandos verständlich beschrieben. Es sollen daher nur einige wenige explizit aufgeführt werden, die man sich unbedingt auswendig merken sollte.

`C-b`, `C-f`	Ein Zeichen rückwärts bzw. vorwärts
`M-b`, `M-f`	Ein Wort rückwärts bzw. vorwärts
`C-p`, `C-n`	Eine Zeile rückwärts bzw. vorwärts
`C-a`, `C-e`	An den Anfang bzw. das Ende der Zeile gehen

Absätze (paragraphs) werden normalerweise durch Leerzeilen voneinander getrennt. Genau beschrieben wird dies durch die Variablen `paragraph-separate` und `paragraph-start`.
Ein Lisp-s-expression (sexp) ist ein geklammerter Emacs-Lisp-Ausdruck (z. B. `(next-buffer)`), ein für Lisp-Programmierer wichtiger Begriff.

Zu **"Löschen und Ausschneiden"**:
Hier begegnet uns erstmals der Unterschied zwischen "Löschen" (delete) und "Ausschneiden" (kill). Im zweiten Fall wird der gelöschte Text in einen Ausschneidepuffer (kill ring) kopiert, aus dem er durch einen Rückholbefehl (yank) erneut (typischerweise an einer anderen Stelle) wieder eingefügt werden kann.
Beim gewöhnlichen Löschen (delete), bei dem pro Tastendruck nur einzelne Zeichen entfernt werden, findet keine Speicherung im Ausschneidepuffer (kill ring) statt.
Die wichtigsten Ausschneidebefehle sind:

`M-d`	`kill-word`	
`M-DEL`	`backward-kill-word`	(`<ESC> <Backspace>`)
`C-k`	`kill-line`	Zeile vom Cursor bis zum Ende löschen

C-w	kill-region	Abschnitt (region) löschen
M-w	kill-ring-save	(wie C-w, aber ohne zu löschen)

Die Rückholbefehle lauten:

C-y	yank	
M-y	yank-pop	ersetzt den zuletzt zurückgeholten Text durch den nächstälteren (nur unmittelbar nach C-y oder M-y)

Zu **"Markieren"**:
Durch das Markieren (C-@ oder C-SPC, SPC = space = Leertaste) wird eine Marke (mark) gesetzt, die bestehen bleibt, solange sie nicht durch andere Befehle gelöscht oder anderweitig gesetzt wird. Bewegt man nach dem Markieren den Cursor an eine andere Stelle, so definieren die Zeichen zwischen Marke (mark) und Cursorposition (point) einen Bereich (region).
Außer durch erneutes Eingeben von C-@ oder C-SPC kann die Marke z. B. durch M-@ (mark-word) verändert werden, wobei dieses Kommando auch ein Argument erlaubt: ESC n C-@ setzt die Marke vom Cursor aus n Worte in Vorwärtsrichtung.

Durch das Kommando C-h wird ein Absatz (paragraph, Textbereich zwischen zwei Leerzeilen) markiert, dh. der Cursor (point) wird an den Beginn der den Absatz nach oben begrenzenden Leerzeile gesetzt und die Marke an den Beginn der nach unten begrenzenden. Damit wird der Absatz als Bereich (region) definiert, was nützlich ist, da sich viele Kommandos auf Bereiche beziehen.
Sehr praktisch ist das Kommando C-x C-x (exchange-point-and-mark), wodurch Anfang und Ende des aktuellen Bereichs (region) angezeigt wird.

Zu **"Interaktives Ersetzen"**:
Anders als bei M-% (query-replace) wird bei C-M-% (query-replace-regexp; zur Erinnerung: C-M-% heißt <ESC> <Ctrl>-%) die eingegebene Zeichenkette als regulärer Ausdruck aufgefasst. Allerdings funktioniert das angegebene Kürzel bei mir nicht. Um es permanent zu aktivieren, ergänzt man die Datei ~\.emacs S. 36 um das Lisp-Kommando

```
(global-set-key [?\C-[ ?\C-\]] 'query-replace-regexp),
```

markiert dann (am einfachsten) den .emacs-Puffer mit C-x h und gibt anschließend das Kommando M-x eval-region ein. Damit wird die die Datei .emacs als Emacs-Lisp-Code interpretiert, so wie das auch beim Startaufruf des Emacs der Fall ist.

Beim genaueren Hinsehen bemerkt man, dass in obigem Kommando nichts von einem Prozentzeichen zu sehen ist. Das rührt daher, dass die meisten (nicht alle!) Tastaturzeichen anders interpretiert werden, wenn man sie zusammen mit der Ctrl-Taste eingibt. Bei den Buchstaben erhält man einfach das Zeichen der Ascii-Tabelle, das der Stellung des Buchstaben im Alphabet enstspricht (wobei zwischen Groß- und Kleinbuchstaben nicht unterschieden wird). Das Zeichen C-A (besser bezeichnet durch ^A, nicht zu verwechseln mit dem Kommandokürzel C-A) entspricht also dem Zeichen Nummer 1 der Ascii-Zeichenfolge, C-Z (besser ^Z) dem Zeichen Nummer 26. Die Interpretation der weitern Zeichen, auf die es hier ankommt, zeigt die folgende Tabelle:

Tastatur (deutsch mit Shift)	"	§	$	%	&	/	(
Tastatur (deutsch ohne Shift)	2	3	4	5	6	7	8
Unix-Darstellung	^@	^[	^\	^]	^^	^_	^?
Ascii-Zeichen Nummer dezimal	0	27	28	29	30	31	127
Ascii-Zeichen Nummer hexadezimal	00	1B	1C	1D	1E	1F	7F
Ascii-Zeichen Nummer oktal	000	033	034	035	036	037	177
Abkekürzte Bezeichnung	NUL	ESC	FS	GS	RS	US	DEL

Man sieht also, dass durch die Ctrl-Taste das Prozentzeichen (%) durch ^] ersetzt wird. In dem der Datei .emacs hinzugefügten Kommando muss die eckige Klammer (]) noch durch einen Backslash (\) maskiert, also \] geschrieben werden. Im Emacs ist halt manches ein wenig kompliziert!

Allerdings lassen sich die oben genannten Zeichen nicht einfach direkt als C-A, C-B etc. in eine Textdatei eingeben, was in einigen Fällen durchaus Sinn machen würde (Tabulatorzeichen

^I , Seitenvorschubzeichen ^L). Dies geschieht vielmehr durch die sogenannte quotierte Eingabe (quoted insert), die durch C-Q eingeleitet wird, also z. B. C-Q C-I bzw. C-Q C-L (Q, I und L können durch die entsprechenden Kleinbuchstaben ersetzt werden).
Übrigens lassen sich mitttels quotierter Eingabe alle Zeichen der Latin-1 Zeichentabelle eingeben, falls der Emacs auf Latin-1 (ISO-8859-1) eingestellt ist, und zwar mit C-Q *nnn* <Return> . Dabei ist *nnn* eine dreistellige Oktalzahl, die die Nummer des Zeichens in der ASCII-Tabelle angibt. C-Q 366 <Return> ergibt z. B. den kleinen Umlaut ö, C-Q 243 <Return> das Zeichen für das britische Pfund (£), C-Q 361 <Return> das spanische kleine n mit Tilde (ñ).

Normalerweise wird bei der Suche nach einer Zeichenkette nicht zwischen Groß- und Kleinbuchstaben unterschieden, solange im Suchstring nur letztere vorkommen, andernfalls schon. Das hat seine Ursache darin, daß die Variable (-> später) case-fold-search auf *wahr* (t) eingestellt ist. Man kann dies mit C-h v case-fold-search abfragen und mit M-x set-variable abändern.

Man wird nach Eingabe des Kommandos M-% *string* oder C-M-% *string* natürlich nach der Ersetzungs-Zeichenkette (replacement string) gefragt. Nachdem deren Eingabe durch <Return> abgeschlossen ist, springt der Cursor zum nächsten Suchtreffer und man wird im Minibuffer nach der weiteren Verfahrensweise gefragt. Durch Eingabe eines Fragezeichens (?) bekommt man eine Liste der Möglichkeiten angezeigt, wie die folgende Tabelle veranshaulichen soll:

y oder SPC	Ersetzen und weiterspringen
n	Nicht ersetzen und weiterspringen
,	Ersetzen und *nicht* weiterspringen (aber auch nicht beenden)
^	Zum vorhergehenden Treffer zurückkehren
E	Den Ersetzungs-String editieren
.	Ersetzen und beenden
q oder RET	Nicht ersetzen und beenden
!	Alles weitere ohne Rückfrage ersetzen
C-r	In rekursives Editieren eintreten

Die zuletzt aufgeführte Antwort C-r (statt z.B. SPC) gibt die Möglichkeit, den Such- und Ersetzungsvorgang zwecks korrigierenden Eingriffs zu unterbrechen ohne ihn zu beenden. Das kann insbesondere bei syntaktisch aufwändigen Vorgängen willkommen sein. Man beendet die Unterbrechung durch C-M-c .

Eine Besonderheit bei der Ersetzung regulärer Ausdrücke ist, dass man in der Ersetzungs-Zeichenkette auf geklammerte Zeichengruppen des Suchstrings Bezug nehmen kann. Nehmen wir an, wir wollten zwei Zeichenketten *string1* und *string2*, die durch eine beliebige Zeichenfolge innerhalb einer Zeile getrennt sind, miteinander vertauschen, so erreichen wir dies durch

C-M-% \(*string1*\)\(.*\)\(*string2*\) <Return> \3\2\1 <Return>

Die beliebige Zeichenfolge \(.*\) bleibt zwischen den vertauschten Zeichenketten unverändert bestehen.

Zu **"Mehrere Fenster"**:
Hier muss man zwischen Fenster (window) und Rahmen (frame) unterscheiden.
Startet man den Emacs mit emacs -nw , dann erzeugt er keinen eigenen Rahmen, sondern füllt vielmehr das Terminalfenster aus, von dem aus er gestartet wurde. Er übernimmt dabei auch die für dieses spezifizierten Zeichensätze. Die in der Referenzkarte aufgeführten Rahmenkommandos beziehen sich nur auf den Aufruf emacs & , der für den Emacs einen eigenen Rahmen erzeugt und wegen des Ampersands (&) das Terminal zur anderweitigen Verwendung freigibt.
Die meisten Kommandos dürften selbsterklärend sein, Tags sind nur für Programmierer interessant. Zum ersten mal wird hier der Begriff "Dired" (directory editor) genannt, ohne dass in der Referenzkarte weiter darauf eingegangen. Dem soll der folgende Einschub5 abhelfen.

Einschub Verzeichnis-Editor (dired)
Der Verzeichnis-Editor wird in der Referenzkarte fast gar nicht beschrieben. Es gibt für ihn nämlich eine eigene Referenzkarte, leider jedoch nur auf Englisch.
Aufgerufen wird er durch `C-x d` (dired), worauf im Minipuffer der Pfad zum aktuellen Verzeichnis vorgeschlagen wird. Eingabe von `<Return>` bestätigt diesen Vorschlag, man kann aber auch ein anderes Verzeichnis auswählen. Der Inhalt des gewählten Verzeichnisses wird nun im Fenster angezeigt, und zwar so, wie wir es vom Kommando `ls -la` bereits kennen (-> S. 7). In der Moduszeile (-> S. 36) sehen wir die Angabe `(Dired by name)`, die uns darauf hinweist, dass wir es hier nicht mit einer Textdatei zu tun haben.
Dementsprechend ist nur noch ein Teil der uns geläufigen Tastaturkommandos wirksam, insbesondere die Suchkommandos `C-s`, `C-r`, `C-M-s`, und `C-M-r`, die Cursorbewegungskommandos `C-f`, `C-b`, `M-f`, `M-b`, `C-n`, `C-p`, `C-a`, `C-e`, `M-<`, `M->`, `C-v`, und `M-v`, sowie die Textmakierungskommandos `C-@`, (bzw. `C-<SPC>`), und `C-x C-x`.
Eine Liste der Dired-Kommandos eerhält man mit `C-h a dired`. Wenn man das Kommando aus einem Dired-Fenster heraus aufruft, enthält diese Liste auch die zugehörigen Tastaturkommandos. Sie beziehen sich überwiegend auf die Dateien der Verzeichnisliste. Da diese nicht durch Texteingabe verändert werden kann, finden sehr viele Zeichen als Dired-Kommandos Verwendung, was sich als ungemein praktisch erweist. Viele Kommandos beziehen sich entweder auf die zuvor durch Eingabe von `m` markierten Dateien oder, falls keine Dateien markiert sind (außer eventuell die unter dem Cursor), auf die Datei unter dem Cursor.
Die wichtigsten Dired-Kommandos zeigt die folgende Tabelle.

`n`	`dired-next-line`	Cursor auf die folgende Zeile setzen
`p`	`dired-previous-line`	Cursor auf die vorherige Zeile setzen
`m`	`dired-mark`	Datei unter dem Cursor markieren
`% m`	`dired-mark-files-regexp`	Dateien entsprechend einem regexp-Muster markieren
`u`	`dired-unmark`	Markierung der Datei unter dem Cursor entfernen
`U`	`dired-unmark all marks`	Alle Dateimarkierungen entfernen
`d`	`dired-flag-file-deletion`	Datei zum Löschen markieren
`% d`	`dired-flag-files-regexp`	Datei entsprechend einen regexp-Muster zum Löschen markieren
`!`	`dired-do-shell-command`	Shell-Kommando auf markierte Dateien anwenden
`+`	`dired-create-directory`	Ein Unterverzeichnis erzeugen
`M`	`dired-do-chmod`	Modus (-> S. 8) der markierten Dateien ändern
`G`	`dired-do-chgrp`	Gruppenzugehörigkeit der markierten Dateien ändern
`O`	`dired-do-chown`	Eigentümer der markierten Dateien ändern
`Z`	`dired-do-compress`	Markierte Dateien komprimieren (mit `gzip`)
`C`	`dired-do-copy`	Markierte Dateien kopieren
`D`	`dired-do-delete`	Markierte Dateien löschen
`x`	`dired-do-flagged-delete`	Zuvor mit `d` markierte Dateien löschen
`R`	`dired-do-rename`	Markierte Dateien umbenennen bzw. verschieben
`% R`	`dired-do-rename-regexp`	Markierte Dateien entsprechend einem regexp-Muster umbenennen
`% C`	`dired-do-copy-regexp`	Markierte Dateien entsprechend einem regexp-Muster kopieren
`% l`	`dired-downcase`	Die Namen der markierten Dateien auf Kleinschreibung umstellen

Als sehr nützlich erweist sich der Verzeichnis-Editor für das Backup von markierten Dateien. Diese werden nämlich in auf sie angewandten Kommandos kollektiv durch einen Stern (`*`) vertreten, beispielsweise wie folgt (-> S. 63):

```
! rsync -auv * vitus@192.168.168.216:/home/vitus/backups
```

Voraussetzung für die Übertragung übers Netz ist allerdings, dass das Paket `ssh-askpass` installiert ist (von der Textkonsole aus funktioniert es auch dann nicht).

Zu **"Formatierung"**

Naturgemäß spielt bei einem Editor Textformatierung eine untergeordnete Rolle. Daher sollen hier nur drei Kommandos erwähnt werden, die man zum Umbrechen eines Absatzes (paragraph) braucht

`C-x f`	`set-fill-column`	Umbruchspalte festlegen
`C-x .`	`set-fill-prefix`	Präfix für jede Zeile setzen
`M-q`	`fill-paragraph`	Absatz auffüllen
`C-o`	`open-line`	Zeile am Cursor umbrechen (wie `<Return>`, nur daß hier der Cursor am Ende der oberen Zeile verbleibt)

Mit `C-x .` definiert man einen Präfix durch eine Zeichenkette (sie kann z. B. aus lauter Leerzeichen bestehen), die man am Beginn eines neuen Absatzes eingetippt hat. Mit `M-x auto-fill-mode` schaltet man dann den automatischen Zeilenumbruch ein, ändert eventuell mit `C-x f` n die Umbruchspalte (default 70) auf den Wert n ab und tippt dann den gewünschten Text ein.
Mir ist es allerdings wichtiger, einen bereits bestehenden Absatz umformatieren und dabei neben der Umbruchspalte auch die Einrücktiefe bestimmen zu können.
Meine Lösung besteht darin, dass ich zunächst den folgenden Code-Abschnitt in die Datei `.emacs` im Home-Verzeichnis einfüge.

```
(global-set-key [?\C-[ ?'] 'fill-absatz)
(defun fill-absatz ()
 (interactive)
 (save-excursion
  (setq fooind (current-column))
  (setq filcol fill-column)
  (setq filpre fill-prefix)
  (setq fill-prefix nil)
  (setq fill-column (- filcol fooind))
  (mark-paragraph)
  (indent-rigidly (point) (mark) -80)
  (fill-paragraph nil)
  (setq fill-column filcol)
  (setq fill-prefix filpre)
  (indent-rigidly (point) (mark) fooind)))
```

Dann markiere ich den gesamten Puffer `.emacs` durch `C-x h` und werte seinen gesamten Lisp-Code aus mit `M-x eval-region` (Daumen drücken, dass alles korrekt abläuft).
Nun kann ich einen Absatz (paragraph, ein Textabschnitt, der durch Leerzeilen begrenzt ist) umformatieren. Ich lege zuerst die Umbruchspalte (z. B. 80) fest mit `C-x f 80`, setze dann den Cursor irgendwo innerhalb des Absatzes auf die Spalte, an der der Text beginnen soll und rufe mit `M-'`, wie in der ersten Zeile des obigen Textabschnitts definiert, die Funktion `fill-absatz` auf.
Allerdings habe ich hier stillschweigend vorausgesetzt, dass das Tastaturkommando `M-'` nicht bereits anderweitig belegt ist (man sollte dies gleich zu Anfang mit `C-h c M-'` überprüfen und gegebenenfalls ein anderes Kürzel wählen).

Zu **"Groß- und Kleinschreibung"**

Zu **"Der Minipuffer"**

Die angegebenen Kommandos sollten selbsterklärend sein.
Der Minipuffer (minibuffer) spielt eine doppelte Rolle. Zum einen werden hier Resultate von Kommandos angezeigt, solange sie eine gewisse Zeilenzahl (welche?) nicht überschreiten (andernfalls wird ein weiteres permanentes Fenster geöffnet). Dieses Minipuffer-Fenster (display area) wird bei Eingabe weiterer Kommandos automatisch geschlossen. Im Falle von Shell-Kommandos kann man das jeweils letzte Resultat aber unter dem Puffernamen `*Shell Command Output*` wiederfinden.
Zum anderen dient er zur Eingabe von Kommandos, und solange diese Eingabe nicht abgeschlossen ist, kann er mit `C-x o` zeitweilig verlassen werden, z. B. um anderweitig Text zu kopieren und ihn dann, nach einem weiteren `C-x o`, im Minipuffer einzufügen. Zu **"Puffer"**

Die Kommandos `C-x b` und `C-x k` bieten mit dem Prompt den jeweils zuletzt besuchten Puffer an. Man kann dann mit `M-p` (bzw. umgekehrt mit `M-n`) schrittweise in der History zuirückgehen, um den gewünschten Puffer zu finden (man kann dessen Namen natürlich auch von Hand eintippen). Hat man eine umfangreiche Pufferliste angelegt, ist es meist besser, sich diese mit `C-x C-b` anzeigen zu lassen. Man kann dann (nach Eingabe von `C-x o`) in dieser Liste ähnlich wie in Dired mit `n` und `p` navigieren, mit `<Return>` auswählen und mit `d` löschen, nur dass sich letzteres nur auf den Eintrag und nicht auf die Datei selbst bezieht.
Sehr praktisch ist auch die Funktion `electric-buffer-list`, für die aber kein Tastaturkürzel eingerichtet ist.

Zu **"Vertauschen"**

Zu **"Rechtschreibprüfung"**

Zu **"Tags"**

Zu **"Shells"**

Der Befehl `M-x shell` öffnet ein neues Fenster mit einer aktiven Shell. Hier lassen sich Befehle und Programme ausführen, wie man es von einem Terminalfenster kennt, allerdings mit einigen Einschränkungen. So funktioniert beispielsweise die Terminal-Ausgabe von Kommandos wie `ls -l` wie gewohnt, nur mit dem Vorteil, dass man das Ergebnis, genau wie in einem Textdatei-Fenster kopieren und weiterverarbeiten kann. Einen Pager wie `less` auf diese Weise verwenden zu wollen, ist aber frustrierend, da die Emacs-Shell nicht über die benötigten Terminal-Fähigkeiten verfügt (ausprobieren!).
Praktisch ist, dass man mit `M-p` zuvor abgesetzte Kommandos zurückholen, editieren und wiederverwenden kann (ähnlich wie im normalen Terminal mit `C-p`).
Verlassen kann man die Shell mit `exit`, womit sich das Shell-Fenster in ein normales Textdatei-Fenster verwandelt, wenn man noch mit `M-x fundamental-mode` den Fenster-Modus auf *fundamental* umstellt.

Zu **"Rechtecke"**

Ein Rechteck wird durch den linken oberen (also z. B. das erste Zeichen in der ersten Zeile) und den rechten unteren Eckpunkt eines Textbereiches definiert. Man setzt zuerst eine Marke an einem der Eckpunkte und führt dann den Cursor an den anderen Eckpunkt. Die folgenden Kommandos beziehen sich dann auf dieses Rechteck:

`C-x r r`	Rechteck in ein Register kopieren
`C-x r k`	Rechteck ausschneiden (entfernen)
`C-x r o`	Rechteck (durch Textverschiebung) öffnen
`C-x r c`	Rechteck mit Leerzeichen überschreiben
`C-x r t`	Rechteck durch angegebenen Text ersetzen

Beim Kommando `C-r r t` kann der neu erzeugte Textbereich sowohl breiter als auch schmäler sein als das zuvor definierte Rechteck. Wenn sich Cursor und Markierung in derselben Textspalte

befinden, wird dadurch ein Rechteck der Breite Null definiert, und dementsprechend wird dann lediglich Text (ohne Löschung) eingefügt.
Das folgende Kommando gehört entfernt auch in diese Gruppe:

`C-x r s` Textbereich (region) in ein Register kopieren

Die folgenden Kommandos setzen einen zuvor durch `C-r r r` oder `C-r r y` definierten rechteckigen Textabschnitt an der Cursor-Position ein, wobei diese dann die linke obere Ecke des eingesetzten Textrechtecks bildet.

`C-x r y` zuvor gelöschtes Rechteck einfügen
`C-x r i` zuvor in ein Register kopiertes Rechteck (oder Region) einfügen

Zu **"Abkürzungen"**

Zu **"Reguläre Ausdrücke**

(-> S. 39 "Inkrementelle Suche")

Zu **"Internationale Zeichensätze"**

Zu **"Info"**

Zu **"Register"**

(-> S. 44 "Rechtecke")

Zu **"Tastaturmakros"**

Für häufig wiederkehrende Aufgaben ist es zweckmäßig, die entsprechenden Kommandoabfolgen in Makros abzuspeichern. Die dafür zuständigen Kommandos zeigt die folgende Tabelle (für die letzten beiden gibt es keine Tastaturtkürzel):

`C-x (`	`kmacro-start-macro`	Makrodefinition starten
`C-x )`	`kmacro-end-macro`	Makrodefinition beenden
`C-x e`	`kmacro-end-and-call-macro`	Zuletzt definiertes Makro ausführen
`name-last-kbd-macro`		Dem zuletzt definierten Makro einen Namen geben
`insert-kbd-macro`		Das benannte Makro als Lisp-Code abspeichern (z. B. in `.emacs`)

Ein praktisches Beispiel entnehme ich dem Buch von Cameron et al. [7] Es dient dazu, Vor- und Familiennamen jeweils am Zeilenbeginn zu vertauschen und dabei die Vornamen durch ein Komma von den Familiennamen zu trennen:

`C-x (`	Makrodefinition starten
`C-a`	Zum Zeilenanfang gehen
`M-f`	Mit dem Cursor ein Wort vorwärts gehen
`,`	Ein Komma anfügen
`M-t`	Zwei Worte (links und rechts vom Cursor) vertauschen
`C-n`	Zur nächsten Zeile gehen
`C-x )`	Makrodefinition beenden

Abschließend geben wir dem Makro noch einen Namen mit

```
M-x name-last-kbd-macro name-exch
```

Der Aufruf eines Makros wirkt auf die im Puffer angezeigte Datei, gerade so wie ein Tastaturkommando, ausgehend von der Position des Cursors. In den meisten Fällen werden wir das Kommando vielfach wiederholt anwenden wollen, z. B. um eine bestimmte Änderung in sämtlichen Zeilen einer Datei vorzunehmen. Das erreichen wir, indem wir dem Aufruf ein numerisches Argument voranstellen, in unserem Falle also z. B. durch

```
M-200 M-x name-exch
```

Sollte es gelungen sein, ein besonders nützliches Makro zu schreiben, das wir sehr häufig benutzen wollen, erscheint vielleicht ein Aufruf der obigen Art als etwas zu umständlich. Dann empfiehlt es sich, das Kommando an ein Tastaturkürzel zu binden, allerdings an eines, das noch frei ist. Die Emacs-Entwickler haben für diesen Zweck die meisten mit `C-c` eingeleiteten Kürzel unbelegt gelassen. Vorsichtshalber überprüfen wir, ob das auch für unser Wunschkürzel `C-c m` zutrifft: `C-h c C-c m`.
Wir setzen dann zunächst das Makro in die Datei `.emacs` ein (z. B. am Ende) mit

```
insert-kbd-macro name-exch
```

und fügen dann davor die Zeile

```
(global-set-key [?\C-c ?m] 'name-exch)
```

ein. Als Ergebnis finden wir dann in `.emacs` am Ende die drei zusätzlichen Zeilen

```
(global-set-key [?\C-c ?m] 'name-exch)
(fset 'name-exch
   "\C-a\C-[f,\C-[t\C-n")
```

Nun müssen wir sie nur noch als Textabschnitt (region) definieren (Markierung am Beginn und Cursor am Ende) und als Lisp-Code wirksam machen durch den Aufruf `M-x eval-region`, um sie in Zukunft als Makro einfach mit `M-200 C-c m` aufrufen zu können.

Zum Ende dieses Abschnitts sei noch das Kommando `eval-expression` erwähnt, das es elaubt, einfache Rechnungen direkt im Emacs durchzuführen, indem man auf den Prompt hin einen mathematischen Ausdruck im Echobereich (minibuffer) eingibt. Also beispielweise:

```
M-1 M-x eval-expression (+ 67000.0 (* 6.0 54321.0))
```

Das Ergebnis (392926.0) wird dann an der Position des Cursors eingesetzt. Wenn man das (weitgehend beliebige) Argument `M-1` weglässt, erscheint es stattdessen im Echobereich.

P. s.: Ein sehr viel mächtigeres ebenfalls in Emacs integriertes Werkzeug ist `calc`, (Aufruf: `M-x calc`), eine Beschreibung hier würde jeden Rahmen sprengen.

11. Das Textsatzsystem TEX.

Das Manuskript "Linux auf der Kommandozeile" ist in TEX geschrieben, hier eine Kurzinfo.
Die meisten PC-Benutzer kennen LibreOffice (oder OpenOffice), ein wunderbares Bürosoftwarepaket, das Writer, Calc, Impress, Base, Draw und Math umfasst, und so zumindest für Privatleute das teure Microsoft Office überflüssig machen sollte.
Das Textsatzsystem TEX, das von Donald E. Knuth entwickelt wurde und 1984 erstmalig erschien [11], geht einen anderen Weg und verfolgt eine andere Zielsetzung als die üblichen Textverarbeitungssysteme. Dokumente werden hier als reine ASCII-Texte verfasst und anschließend übersetzt, sodass man während der Texteingabe das Ergebnis noch nicht sieht. Diesem augenscheinlichen Nachteil steht als Vorteil gegenüber, dass man auch komplizierte Konstruktionen ausschließlich über die Tastatur ohne Zuhilfenahme der Maus vornehmen kann.
Dementsprechend spielt TEX seine Stärken besonders beim Satz mathematischer Formeln aus. Nach einer gewissen Einarbeitungszeit kann man diese weniger umständlich und damit flüssiger erstellen als etwa mit OpenOffice Math. Dabei wird eine Qualität erreicht, mit der sich wohl kein anderes System messen kann. Aber auch anderweitig Interessierte werden bei TEX fündig, denn es gibt ausgezeichnete Pakete für Chemie, Musik, Schach, Graphik und vieles anderes mehr.
Bereits wenige Jahre nach dem Erscheinen des TEXBOOKs entwickelte Leslie Lamport auf TEX aufbauend das Textsatzsystem LATEX [12], das sich im Unterschied zu TEX mehr dem Prinzip des "Generic Markup" (der Trennung von Text und Inhalt) verpflichtet fühlt und überdies als leichter erlernbar gilt. Lamports Buch wurde auch ins Deutsche übersetzt, allerdings scheint diese Ausgabe nur noch antiquarisch erhältlich zu sein. Dasselbe gilt auch für das dreibändige Werk von Helmut Kopka, das sich außerordentlich ausführlich mit Zusatzpaketen und speziellen Themen befasst.

Im Laufe der Jahre hat sich um TEX eine riesige Gemeinde gesammelt, die ständig neue Makros und Erweiterungen entwickelt. Bei `ftp.dante.de` kann man genüsslich stöbern und downloaden.
Donald Knuth (* 1938) hingegen hat bereits in den Neunziger-Jahren beschlossen, sein System als endgültig einzufrieren, was bedeutet, dass man heute in TEX geschriebene Dokumente noch in fünfzig Jahren problemlos übersetzten und weiterverarbeiten können sollte.
Von seinem Buch gibt nur eine Teilübersetzung von Fritz Cremer aus dem Jahre 1993. Man findet sie, wie auch den Quelltext des Originals (Download mit Emacs, Installation eines FTP-Clients vorausgesetzt: `C-x d /ftp:ftp@ftp.dante.de:/pub/tex/...`) unter:

`/ftp:ftp@ftp.dante.de:/pub/tex/info/german/texbuch/texbuch.pdf` bzw.
`/ftp:ftp@ftp.dante.de:/pub/tex/systems/knuth/dist/tex/texbook.tex`

Um mit TEX arbeiten zu können muss man zunächst eine Reihe von Paketen installieren. Bei mir sind das die folgenden:

```
tex-common                    common infrastructure for building and installing TeX
texlive                       TeX Live: A decent selection of the TeX Live packages
texlive-base                  TeX Live: Essential programs and files
texlive-binaries              Binaries for TeX Live
texlive-extra-utils           TeX Live: TeX auxiliary programs
texlive-font-utils            TeX Live: Graphics and font utilities
texlive-fonts-extra           TeX Live: Additional fonts
texlive-fonts-recommended     TeX Live: Recommended fonts
texlive-generic-extra         TeX Live: Generic additional packages
texlive-generic-recommended   TeX Live: Generic recommended packages
texlive-lang-english          TeX Live: US and UK English
texlive-lang-german           TeX Live: German
texlive-latex-base            TeX Live: LaTeX fundamental packages
texlive-latex-recommended     TeX Live: LaTeX recommended packages
texlive-plain-extra           TeX Live: Plain TeX packages
texinfo                       Documentation system for on-line information and
                              printed output
```

Der Quelltext eines TEX-Dokuments besteht im einfachsten Fall fast nur aus dem eigentlichen Text, z. B. `vulgar.tex`, hier zunächst exakt wie am Bildschirm eingetippt zu sehen, dann als gesetztes Endprodukt. Dieses erhält man durch Übersetzen des Quelltexts mit `tex vulgar.tex`, wodurch eine geräteunabhängige (dvi = device-independent) Datei `vulgar.dvi` erzeugt wird, die dann wiederum mit dem Viewer `xdvi` betrachtet werden kann (`xdvi vulgar.dvi`).

```
A Vulgar Mechanick can practice what he has been taught or seen done,
but if he is in an error he knows not how to find it out and correct it,
and if you put him out of his road, he is at a stand; Whereas he
that is able to reason nimbly and judiciously about figure, force and
motion, is never at rest till he gets over every rub.

(Isaac Newton, 1642-1727, to Nathaniel Haws, 25 May 1694)
\bye
```

A Vulgar Mechanick can practice what he has been taught or seen done, but if he is in an error he knows not how to find it out and correct it, and if you put him out of his road, he is at a stand; Whereas he that is able to reason nimbly and judiciously about figure, force and motion, is never at rest till he gets over every rub.
(Isaac Newton, 1642-1727, to Nathaniel Haws, 25 May 1694)

Was wir an diesem einfachen Beispiel bereits sehen, ist, dass der Text automatisch rechts- und linksbündig gesetzt wird und dass wir die Eingabe mit der Kontrollsequenz `\bye` abschließen müssen. Man kann auch bereits ahnen, dass die Leerzeile im Quelltext den Beginn einer neuen Zeile zur Folge hat.
Das Layout typischer TEX-Dokumente wird durch zahlreiche Kontrollsequenzen gesteuert. Diese bestehen aus einem einleitenden Backslash und entweder einer Folge von Buchstaben oder einem einzelnen Zeichen, das kein Buchstabe ist.
Dass ich oben einen englischen Text als Beispiel gewählt habe, liegt daran, dass es im Englischen keine Umlaute gibt. Da TEX-Quelltexte ausschließlich aus ASCII-Zeichen (Nr. 32 bis 126 der Zeichentabelle) bestehen, müssen bereits deutsche Umlaute durch Kontrollsequenzen dargestellt werden, also z. B. ä durch `\"a`.
Ich kann an dieser Stelle nur einen leichten Vorgeschmack von der Leistungsfähigkeit von TEX bieten, speziell für Mathematik-Interessierte:

```
Wie l\"ose ich eine quadratische Gleichung?
\vskip -1ex
$$ax^2+bx+c=0$$
$$x={1\over2a}\left(-b\pm\sqrt{b^2-4ac}\,\right)$$
\vskip1ex
Die Gamma-Funktion ist definiert durch
$$\Gamma(x)=\int_0^\infty\!e^{-t}\,t^{x-1}\,\hbox{d}t$$
\vskip -1ex
Man sieht sofort die G\"ultigkeit von
$$\Gamma(x+1)=x\,\Gamma(x)\qquad\hbox{und}\qquad\Gamma(1)=1$$
```

Wie löse ich eine quadratische Gleichung?

$$ax^2 + bx + c = 0$$
$$x = \frac{1}{2a}\left(-b \pm \sqrt{b^2 - 4ac}\right)$$

Die Gamma-Funktion ist definiert durch

$$\Gamma(x) = \int_0^\infty e^{-t}\, t^{x-1}\, \mathrm{d}t$$

Man sieht sofort die Gültigkeit von

$$\Gamma(x+1) = x\,\Gamma(x) \qquad \text{und} \qquad \Gamma(1) = 1$$

12. Netzwerke

Das Thema Netzwerke ist natürlich außerordentlich umfangreich und komplex. Ich beschränke mich hier auf ein einfaches lokales Netz (LAN = Local Area Network), bestehend aus einem oder einigen wenigen PCs und einem Router, der die Verbindung zum Internet herstellt.
Wird ein Rechner in einem solchen Netzwerk hochgefahren, bezieht er im Normalfall von dem im Router aktiven DHCP-Server (DHCP = *Dynamic Host Configuration Protocol*) automatisch eine Netzwerkadresse, sodass er unmittelbar im Netzwerk mit anderen Rechnern und extern auch mit dem Internet verbunden ist. Voraussetzung für letzteres ist natürlich, dass ein Vertrag mit einem Internet-Provider besteht und der Router entsprechend konfiguriert ist. Im günstigsten Fall braucht man sich, soweit es um den Zugang zum Internet geht, um nichts weiter zu kümmern.
Will man sich aber mit einem anderen Rechner im lokalen Netz (oder beispielsweise mit einem Netzwerkdrucker oder einem TV-Receiver) verbinden, muss man sich mit den Netzwerkadressen vertraut machen, durch die die Teilnehmer im Netzwerk identifiziert werden.
Die Basis dafür bildet das *Internet Protocol* (IP), auf das dann das *Transmission Control Protocol* (TCP) aufsetzt. TCP/IP, wie man es auch nennt, ist heute bei allen Betriebssystemen vorherrschend. Jeder Netzteilnehmer wird durch eine Kennung identifiziert, die aus vier durch Punkte getrennten Zahlentripeln besteht. Das gilt auch für das Internet, dessen Teilnehmer man ja durchweg in Form von Klarnamen aufruft. Durch das Kommando `ping www.spiegel.de` kann ich beispielsweise die Netzwerkadresse des SPIEGEL ermitteln. Ich erhalte beispielsweise als Antwort:

```
PING www.spiegel.de (62.138.116.25) 56(84) bytes of data.
64 bytes from 62.138.116.25 (62.138.116.25): icmp_seq=1 ttl=244 time=33.0 ms
```

Jedes in einer solchen Adresse enthaltene Tripel liegt im Bereich der durch acht Bits darstellbaren Zahlen, also zwischen 0 und 255. Ist die letzte Ziffer eine Null, so bezeichnet die Adresse ein Netzwerk, ansonsten einen einzelnen Rechner (bzw. eine in ihm enthaltene Netzwerkkarte). Die Vergabe der Adressen im Internet wird von der ICANN (*Internet Corporation for Assigned Names and Numbers*) koordiniert.
Für unser lokales Netzwerk ist nun entscheidend, dass international festgelegt wurde, dass bestimmte Adressbereiche für lokale Netzwerke reserviert sind, sie also nicht mit Adressen im Internet kollidieren können. Einer dieser Bereiche (und der für private Netzwerke wichtigste) ist der der Adressen zwischen 192.168.0.0 und 192.168.255.255. Rechneradressen (IP-Nummern) können innerhalb dieses Bereiches frei gewählt werden, solange sie eindeutig sind und der Netzwerkmaske entsprechend zum gleichen Netz gehören. Die in privaten Netzwerken geläufigste Netzwerkmaske 255.255.255.0 besagt, dass Rechner, deren Adressen sich nur in den Ziffern des letzten Tripels unterscheiden, dem gleichen Netz zuzurechnen sind.
Etliche Router haben werksseitig 192.168.178.0 oder 192.168.0.0 eingestellt (entsprechend den $3 \times 8 = 24$ Bits, die für die Netzwerkadresse reserviert sind, schreibt man genauer 192.168.178.0/24 bzw. 192.168.0.0/24).
Jeder Unix-Rechner hat zusätzlich die Adresse 127.0.0.1, die sogenannte Loopback-Adresse, fest eingerichtet. Sie wird als `localhost` angesprochen und dient dem internen Netzwerkverkehr.
Wie erwähnt, sorgt bei vielen Installationen der DHCP-Server für die automatische Zuteilung der Adressen, es kann jedoch gute Gründe geben, eine feste Adresse einzustellen.
In diesem Fall muss man die gewünschten Adressen in die Datei `/etc/network/interfaces` eintragen, die dann beispielsweise folgendermaßen aussehen könnte (der zweite Teil ab `auto eth0` sollte wegfallen, wenn man DHCP benutzt):

```
# /etc/network/interfaces -- configuration file for ifup(8), ifdown(8)
# For more sample entries take a look at /etc/network/interfaces.examples

# The loopback interface
# automatically added when upgrading
auto lo
iface lo inet loopback
```

```
auto eth0
iface eth0 inet static
        address 192.168.168.223
        netmask 255.255.255.0
        network 192.168.168.0
        broadcast 192.168.168.255
        gateway 192.168.168.1
```

Die Einstellungen dieser Datei werden wirksam durch den Aufruf

```
/etc/init.d/networking restart
```

Dazu ein paar Bemerkungen.
Das Verzeichnis `/etc` enthält Dateien und Ordner, die für die systemweite Konfiguration des Systems maßgebend sind (Benutzer-spezifische Konfigurationsdateien findet man in den jeweiligen HOME-Verzeichnissen). In dem Unterverzeichnis `init.d` befinden sich Skripte, durch deren Aufruf Dienste gestartet werden, sei es unmittelbar nach dem Systemstart, sei es durch den Administrator im laufenden Betrieb.
Für den Fall, dass das Paket `sysv-rc` installiert ist, wird der Aufruf nach dem Systemstart durch eine Reihe von Symlinks in Unterverzeichnissen `rc0.d` bis `rc6.d` sowie `rcS.d` gesteuert. Ist statt dessen das Paket `file-rc` installiert, das mit `sysv-rc` inkompatibel ist, findet die Steuerung durch Einträge in der Datei `/etc/runlevel.conf` statt.
In beiden Fällen kann der Administrator im laufenden Betrieb einen Dienst durch das Programm `invoke-rc.d` starten, die Secure Shell (`ssh`) (-> S. 54) z. B. durch

```
invoke-rc.d ssh start
```

Der Administrator kann es mit Hilfe von `update-rc.d` einrichten, dass ein Dienst standardmäßig nach dem Systemstart gestartet wird.Für die Secure Shell erfolgt dies durch den Aufruf

```
update-rc.d ssh defaults
```

Durch diesen Aufruf wird (im Falle von `file-rc`) in der Datei `/etc/runlevel.conf` ein entsprechender Eintrag hinzugefügt.
Ausführliche Beschreibungen findet man in den Manualseiten.
Das Thema Systemstart ist allerdings in den letzten Jahren dadurch kompliziert worden, dass in vielen Distributionen (darunter auch Debian und Ubuntu) das herkömmliche SysV-Init-System durch `systemd` ersetzt worden ist. Auch der Start von Diensten ist davon betroffen. Immerhin scheint, zumindest für eine gewisse Übergangszeit, die Funktion der oben erwähnten Utilities erhalten geblieben zu sein.

Die großen Linux-Distributionen stellen durchweg bequeme graphische Werkzeuge zur Verfügung. Trotzdem ist es gut zu wissen, wie man im Falle von Störungen selbst Hand anlegen kann. Dazu eignen sich besonders die Kommandos `ifconfig` und `route`.
Wenn man Probleme mit dem verkabelten **LAN-Netzwerk** hat, kann man versuchen, zunächst die alte Verbindung zu beenden. Mit `ifconfig -a` stellt man zuerst die Interface-Bezeichnung fest (meist ist es `eth0` oder `eth1`), dann gibt man ein (wir brauchen natürlich ROOT-Rechte):

```
route del default
ifconfig ethn down
```

Dann baut man die Verbindung wieder auf (wir wollen die feste Netzwerkadresse 192.168.168.216 einrichten) und richtet einen (oder zwei) Nameserver ein (das Paket `resolvconf` muss installiert sein):

```
ifconfig ethn 192.168.168.216 netmask 255.255.255.0 \
  broadcast 192.168.168.255 up
route add default gateway 192.168.168.1
echo "nameserver 192.168.168.1
nameserver 194.25.2.129" | resolvconf -a ethn
```

Wenn man diese Kommandofolge öfters braucht, empfiehlt es sich, sie in einem Skript, z. B. mit dem Namen `neton`, etwa folgendermaßen (in geringfügig verallgemeinerter Form) zusammenzufassen:

```
#!/bin/bash
int=$(ip -s link | grep ^2: | sed -r -e 's/2: ([^ ]+): (.*)$/\1/')
route del default
sleep 3
ifconfig ${int} down
sleep 3
ifconfig ${int} 192.168.168.$1 netmask 255.255.255.0 broadcast 192.168.168.255 up
sleep 3
route add default gateway 192.168.168.1
sleep 3
echo "nameserver 192.168.168.1
nameserver 194.25.2.129" | resolvconf -a ${int}
```

Schnittstellen für das verkabelte LAN werden in heutigen Linux-Distributionen häufig nicht mehr mit `eth0` oder `eth1` bezeichnet. Wie sie tatsächlich heißen, kann man, wie erwähnt, mit `ifconfig -a` oder mit dem moderneren `ip -s link` herausfinden. Dies wird in der zweiten Zeile des obigen Skripts benutzt, um mittels Kommandoersetzung (command substitution, `->` `echo` auf S. 21) die Schnittstellenbezeichnung der lokalen Variablen `int` zuzuweisen, die im folgenden dann dreimal verwendet wird. Man braucht bei Aufruf des Skripts dann nur noch einen Parameter anzugeben (`$1` im Skript), nämlich das vierte Tripel der gewünschten IP-Adresse (also z. B. `neton 216`).
Herauszufinden, wie die zweite Zeile des Skripts im Detail funktioniert, sei dem interessierten Leser empfohlen (er muss sich dann allerdings ein wenig mit dem Stream-Editor `sed` befassen). Die Sleep-Kommandos (Unterbrechung in Einheiten von Sekunden) sind wahrscheinlich überflüssig.

Eine Funknetz- bzw. WLAN-verbindung (WLAN = Wireless Local Area Network) herzustellen, ist eine komplexe Aufgabe.
Manche Wireless-Karten werden von Linux erkannt, aber das Kommando `ifconfig -a` zeigt keinen Interface-Namen an. In diesem Fall kann man sich mit dem Programm `hwinfo` Information über die WLAN-Karte des Rechners aulisten lassen. Das Ergebnis sieht etwa folgendermaßen aus:

```
33: PCI 300.0: 0282 WLAN controller
  ...
  Hardware Class: network
  Model: "Intel PRO/Wireless 4965 AG or AGN"
  Vendor: pci 0x8086 "Intel Corporation"
  Device: pci 0x4229 "PRO/Wireless 4965 AG or AGN [Kedron] Network Connection"
  ...
  Driver: "iwl4965"
  Driver Modules: "iwl4965"
  Device File: wlp3s0
  ...
  HW Address: 00:13:e8:24:8b:a7
  Link detected: yes
  ...
  WLAN encryption modes: WEP40 WEP104 TKIP CCMP
  WLAN authentication modes: open sharedkey wpa-psk wpa-eap
    ...
    Driver Status: iwl4965 is active
    Driver Activation Cmd: "modprobe iwl4965"
```

```
Config Status: cfg=new, avail=yes, need=no, active=unknown
Attached to: #18 (PCI bridge)
```

Man bekommt z. B. Information über den Treiber. In meinem Fall hat es bereits geholfen, diesen zu entladen und wieder neu zu laden, also

```
rmmod iwl4965        (oder modprobe -r iwl4965 )
modprobe iwl4965
```

Manchmal muss man zuerst noch die erforderliche Firmware nachinstallieren da sie häufig proprietär ist, und daher bei manchen Distributionen nicht standardmäßig enthalten ist. Für Intel-Karten braucht man z. B. bei Debian das Paket `firmware-iwlwifi`, das man im Debian-Repository unter `ftp://ftp.de.debian.org/debian/pool/non-free/f/firmware-nonfree` findet.
Bei Ubuntu scheint die Firmware vor allem in den umfangreichen Paketen `linux-firmware` und `linux-firmware-nonfree` vorzuliegen, die man im Repository unter

```
http://archive.ubuntu.com/ubuntu/pool/main/l/linux-firmware/
http://archive.ubuntu.com/ubuntu/pool/multiverse/l/linux-firmware-nonfree/
```

findet. Welche Repositories MINT standardmäßig verwendet, kann man in den Verzeichnissen `/etc/apt/` bzw. `/etc/apt/sources.list.d/` nachlesen. Man kann dort auch (mit Vorsicht!) Modifikationen und Ergänzungen vornehmen. Die oben genannten Ubuntu-Quellen sind auch darunter.

Anders als die auf S. 51 beschriebenen Aktivierung des verkabelten Netzwerks stellt uns die des Funknetzes vor erhebliche Probleme, wenn die graphischen Konfigurationshilfen nicht auf Anhieb zum Erfolg führen.
Dann braucht man Literaturstudium und Internet-Recherche. Mir hat vor allem Koflers Buch [1] geholfen, sowie die von ihm empfohlene Website `wiki.ubuntuusers.de/WLAN/wpa_supplicant`. Natürlich bieten auch die einschlägigen Manualseiten wichtige Information, sowie die `readme`-Dateien in `/usr/share/doc/wpasupplicant`.
Beim Funknetz liegen fast immer mehrere Zugangspunkte (APs = Access Points) in Reichweite unseres Rechners. Man kann sie sich auflisten lassen (mit Administratorrechten) durch `iwlist scan`. Unter Umständen muss man zuvor die Netwerkkarte aktivieren, z. B. mit `ifconfig wlan0 up` (es soll ab jetzt mit `eth0` die verkabelte Netzwerkkarte und mit `wlan0` die WLAN-Karte gemeint sein). Ausschnitte eines typischen Ergebnisses zeigt das folgende Listing.

```
Cell 07 - Address: 34:81:C4:C6:CC:29
Channel:6
Frequency:2.437 GHz (Channel 6)
Quality=59/70  Signal level=-51 dBm
Encryption key:on
ESSID:"fritzbuchholz"
...
Mode:Master
...
IE: IEEE 802.11i/WPA2 Version 1
    Group Cipher : CCMP
    Pairwise Ciphers (1) : CCMP
    Authentication Suites (1) : PSK
```

Meist kann man leicht herausfinden, welcher AP der eigene Router ist, z. B. durch die ESSID (Extended Service Set Identifier), insbesondere, wenn man sie gegenüber der Werkseinstellung verändert hat (in meinem Fall in "fritzbuchholz", neuerdings einfach 'fritz'). Das Listing liefert auch Auskunft über Verschlüsselung und Authentifizierung (Stichworte TIK, CCMP und PSK). Als Verschlüsselung sollte stets WPA bzw. WPA2 verwendet werden, keinesfalls das ältere WEP, das als ganz unsicher gilt. Im Router sollte man einen möglicht langen und komplizierten Passwortsatz (passphrase) eintragen (in der Fritzbox unter WLAN/Sicherheit).

Das Folgende bezieht sich auf die manuelle Einrichtung des Funknetzes (soll WLAN beim Rechnerstart automatisch aktiviert werden, muss man die Datei `/etc/network/interfaces` editieren).
Zunächst muss man den Modus und die ESSID einstellen, damit die WLAN-Karte mit dem Router kommunizieren kann (dieser läuft, wie oben zu sehen, im Modus Master). Dies geschieht z. B. durch

```
iwconfig wlan0 mode managed
iwconfig wlan0 essid "fritzbuchholz"
```

Das Paket `wpasupplicant` muss installiert werden (oft ist es Teil der Standardinstallation). Das ausführbare Programm ist `/sbin/wpa_supplicant`, die zugehörige Konfigurationsdatei `/etc/wpa_supplicant.conf`. In `/usr/share/doc/wpasupplicant/examples/` findet man Beispiele für letztere. Das Ergebnis sieht bei mir folgendermaßen aus:

```
ctrl_interface=/var/run/wpa_supplicant
network={
  ssid="fritzbuchholz"
  scan_ssid=1
  proto=RSN
  key_mgmt=WPA-PSK
  pairwise=CCMP
  group=CCMP
  #psk="geheim!"
  psk=<64-stellige Hexadezimalzahl>
}
```

Nach `psk=` könnte man die Passphrase vom Router eintragen, aus Sicherheitsgründen sollte man diese aber verschlüsseln. Dies bewerkstelligt man mit

```
wpa_passphrase [ssid] [passphrase]
```

Das auf dem Terminal ausgegebene Ergebnis ist eine 64-stellige Hexadezimalzahl, die man in die `wpa_supplicant.conf` einträgt. Man sollte diese Datei übrigens mit `chmod 0600 wpa_supplicant.conf` nur für `root` lesbar machen (und der Eigentümer muss natürlich `root` lauten).

Jetzt ist es an der Zeit, `wpa_supplicant` aufzurufen, in den meisten Fällen durch

```
wpa_supplicant -i wlan0 -D wext -c /etc/wpa_supplicant.conf
```

Die WLAN-Schnittstelle kann auch anders heißen (bei meiner Debian-Installation z. B. wlp3s0). Der Parameter `wext` bezeichnet einen generischen Treiber, der in den meisten Fällen funktionieren sollte. Weitere Auskünfte gibt die Manualseite.
`wpa-supplicant` muss laufen, solange man die Internetverbindung aufrecherhalten will, dh. man muss ein eigenes Terminal (oder eine eigene Konsole) dafür reservieren (gut eignet sich auch eine asynchrone Emacs-Shell).
Alle hier beschriebenen Programmaufrufe müssen unter Root-Rechten erfolgen.

Zum Schluss muss man die WLAN-Karte mit dem Netzwerk verbinden, eine Route festlegen und den Nameserver einrichen, z. B. durch

```
ifconfig wlan0 up 192.168.168.217/24
route add default gateway 192.168.168.1
echo "nameserver 192.168.168.1
nameserver 194.25.2.129" | resolvconf -a wlan0
```

Noch ein paar wichtige Bemerkungen. Der gerade beschriebene `ifconfig`-Aufruf verbindet (assoziiert) die WLAN-Karte mit einer IP-Nummer. Deaktiviert man die Karte mit `ifconfig wlan0 down`, bleibt diese Assoziierung häufig bestehen, was man mit `ifcofig wlan0` verifizieren kann. Das kann störend sein, wenn man eine neue Verbindung aufbauen will. Disassoziierung erreicht man mit

```
ip address del 192.168.168.217/24 dev wlan0
```

Das Programm `ip` ist sehr leistungsfähig und kann `ifconfig` und `route` weitgehend ersetzen (-> Manualseite).

Bisher haben wir mit Netzwerkadressen immer nur IP-Nummern gemeint. Man möchte Rechner aber lieber mit Klarnamen ansprechen wie `saturn` oder `leopard`. Dies erreichen wir z. B. mit dem Kommando `hostname saturn`. Der Name findet sich dann in der Datei `/etc/hostname` und er wird durch das Kommando `hostname` (ohne Parameter) auf dem Terminal ausgegeben. Allerdings wird man bei der Linux-Installation (z. B. Debian) nicht nur nach dem Rechnernamen (hostname) gefragt sondern auch nach dem Domänennamen (domain name). Für meine Rechner habe ich `buchholz.de` gewählt. Zusammen ergibt das den sogenannten voll qualifizierten Rechnernamen (fully qualified domain name, fqdn), also z. B. `saturn.buchholz.de`. Mit dem Kommando `hostname --fqdn` wird er auf dem Terminal ausgegeben.
Will man andere Rechner im LAN mit Klarnamen erreichen, so muss man dies in die Datei `/etc/hosts` eintragen. Bei mir sieht diese z. B. folgendermaßen aus:

```
127.0.0.1        localhost
127.0.1.1        pluto.buchholz.de        pluto
192.168.168.45   mintvirt.buchholz.de     mintvirt
192.168.168.46   delgrml.buchholz.de      delgrml
192.168.168.15   augias.buchholz.de       augias
...
```

Welche Bedeutung die hier auftauchende zweite Loopback-Adresse hat, wird im Internet diskutiert (ich weiß es nicht).
Mit Hilfe der Einträge in `/etc/hosts` kann man die betreffenden Rechner mit Klarnamen im lokalen Netz ansprechen, z. B. mit `ping augias`.
Aliasnamen für IP-Nummern in `/etc/hosts` einzutragen, ist der erste triviale Schritt in Richtung DNS (Domain Name System). Während die Domänennamen in einfachen lokalen Netzwerken bedeutungslos sind, spielen sie im Inernet eine zentrale Rolle. Für den Zugang zum Internet muss man daher mindestens einen Nameserver angeben, der die Webadressen IP-Nummern zuordnet. Dafür zuständig ist die Datei `/etc/resolv.conf`, und die erste Nameserveradresse ist meist mit der des Routers identisch.

Anwendungen im LAN: `rsync`, `ssh`, `ftp` und `lighttpd`.

Ein lokales Netzwerk (LAN) zu unterhalten macht natürlich in einer Privatwohnung nur Sinn, wenn man (neben dem Router) mehr als ein netzwerkfähiges Gerät betreibt. Vor zehn Jahren hätte daran noch wenig Bedarf bestanden, heute haben wir (insbesondere in Familien) Zweit- und Dritt-PCs, Netzwerkplatten (NAS), TV-Receiver, netzwerkfähige Drucker, Smartphones und anderes mehr.

Das Programm `rsync` ist bereits auf S. 63 ausführlich beschrieben worden.

`ssh` (Secure SHell) dient in erster Linie dazu, sich bei einem entfernten Rechner eizuloggen, z. B.:

```
ssh -X vitus@saturn
```

Dabei ist stillschweigend vorausgesetzt, dass in der Datei `/etc/hosts` der Rechnername `saturn` mit einer IP-Nummer assoziiert ist.
Der Parameter `-X` weist das System des entfernten Rechners (remote host) an, seine X-Window-Ausgabe an den lokalen Rechner (local host) weiterzuleiten (also dessen X-Server zu benutzen). Das bedeutet, dass auch graphischer Output eines Programms auf dem entfernten Rechner auf den Bildschirm des lokalen Rechners ausgegeben wird. Man kann dadurch über ein Terminalfenster auf einem entfernten Rechner weitgehend so arbeiten, als würde man unmittelber vor ihm sitzen und seine Tastatur bedienen (allerdings ohne seinen Desktop zur Verfügung zu haben).
So können selbst Videos übertragen werden, allerdings ohne Ton.
Voraussetzung ist die Installation eines SSH-Servers, z. B. `openssh-server`, auf dem entfernten Rechner, sowie dessen Start, z. B. mit `invoke-rc.d ssh start` (siehe auch S. 50).

Die Secure Shell eignet sich allerdings auch ausgezeichet zur Datenarchivierung über das Netz (bezüglich `tar` siehe S. 12), z. B.:

`tar -cvf -` *Daten* `| ssh vitus@saturn dd of=/mnt/backups/`*Daten*`.tar`

bzw. in umgekehrter Richtung:

`ssh vitus@saturn "dd if=/mnt/backups/`*Daten*`.tar" | tar -xvf - -C` *Datenpfad*

Das Minuszeichen nach dem Parameter `f` bedeutet, dass `tar` (im ersten Fall) Daten auf Standardausgabe ausgibt (um sie nach `ssh` weiterzuleiten), bzw. (im zweiten Fall) Daten von Standardeingabe (weitergeleitet von `ssh`) erwartet.
Der Parameter `-C` bedeutet, dass in das nachfolgend angegebene Vezeichnis entpackt wird und nicht in das laufende.
Wichtig ist zu wissen, dass sich die Standard-Eingabe/Ausgabe stets auf den *lokalen* Rechner bezieht, auch wenn diese von einem auf dem entfernten Rechner aufgerufenen Programm erzeugt wird. Dies kann man sich beispielsweise zu Nutze machen, um eine auf einem entfernten Rechner liegende Video-Datei auf dem lokalen Rechner abzuspielen:

`ssh vitus@saturn cat` *Videodatei* `| mplayer -`

Das funktioniert freilich nur, wenn der Videoplayer (in diesem Fall `mplayer`) die Eingabe über Standard-Input unterstützt. Anders als beim direkten Abspielen auf einem entfernten Rechner, auf dem man mittels `ssh` eingeloggt ist, wird so natürlich auch der Ton übertragen.

Will man mit einem Windows-Rechner Daten austauschen, so eignet sich dafür `rsync`, im Zusammenspiel mit einem SSH-Server, ebenfalls, allerdings sind beide Programme nicht Standard auf Windows-Systemen. Es empfiehlt sich daher, `Cygwin` (`http://cygwin.com`) zu installieren, eine Tool-Sammlung, die Unix-Funktionalität unter Windows zur Verfügung stellt. Das in dieser Sammlung enthaltene `rsync` funktioniert tadellos (abgesehen von einigen im Betriebssystem Windows begründeten Einschränkungen), der SSH-Server soll allerdings etwas unzulänglich sein. Seit einiger Zeit gibt es einen kommerziellen SSH-Server für Windows (`http://www.bitvise.com`), der für private Anwender kostenlos ist, und mit dem ich gute Erfahrungen gemacht habe. Er lässt sich einfach installieren und bedarf normalerweise keiner weiteren Konfiguration. Gegebenenfalls steht ein graphisches Panel zur Steuerung (z. B. für Ein- und Ausschalten) zur Verfügung.
Cygwin bildet allerdings intern die Windows-Laufwerke Unix-entsprechend ab, wobei natürlich auch die deutsche Lokalisierung unberücksichtigt bleibt.
Aus `C:\Benutzer` wird so `/cygdrive/C/users`.

Ein wichtiges Thema wäre hier `Samba`, ein Paket, das Datei- und Druck-Dienste in heterogenen Systemen zur Verfügung stellt. Vielleicht komme ich später darauf noch zurück.

Ausführlicher soll hier auf `ftp` (ftp = file transfer protocol) eingegangen werden, das eines der ältesten Datenübertragungs-Programme darstellt. Etliche Experten würden es am liebsten ausmustern, vor allem aus Sicherheitsgründen. Außerdem leidet es unter der gravierenden Einschränkung, dass nur einzelne Dateien (wenn auch mehrere auf einmal) übertragen werden können, nicht aber ganze Verzeichnisse.
Sicherheitsbedenken sollten in lokalen Netzen allerdings keine Rolle spielen, und für Transfers von Verzeichnissen, kann man eventuell auf `wget` und `wput` ausweichen, die ebenfalls mit FTP-Servern zusammenarbeiten.
Eines der Hauptargumente für `ftp` ist, dass es, anders als etwa `ssh` oder `rsync`, zur Standardausstattung auf NAS-Geräten (NAS = network access storage) und offenbar auch TV-Receivern gehört. Hinzu kommt, dass der Emacs-Editor mit seiner Verzeichnis-Editierfunktion (dired, siehe S. 42) eine besonders bequeme und effiziente Verbindung zum Server ermöglicht, z. B. verbinde ich mich mittels

`C-x d /ftp:vitus@saturn:/home/vitus`

mit dem HOME-Verzeichnis des Users `vitus` auf dem Rechner `saturn`.
Das angesteuerte Verzeichnis auf dem entfernten Rechner wird genau so angezeigt, wie wir es vom lokalen Rechner kennen. Es stehen uns jetzt natürlich keine Shell-Kommandos zur Verfügung,

wohl aber nahezu alle `dired`-Kommandos (siehe S. 42), wie markieren, kopieren, löschen, umbenennen, auch (wenn man die Rechte hat) das Anlegen von neuen Verzeichnissen.
Auch wenn man auf dem entfernten Rechner keine Schreibrechte hat, findet man dort oft wichtige Informationen und Download-Möglichkeiten, z. B. beim Debian-Repository:

```
C-x d /ftp:ftp@ftp.de.debian.org:/debian
```

Voraussetzung für Obiges ist natürlich die Installation von FTP-Clients und -Servern auf den jeweiligen Maschinen. Als Client bietet sich (bei manchen Distributionen ist es standardmäßig installiert) das Paket `netkit-ftp` an, das oft auch unter der generischen Bezeichnung `ftp` geführt wird. Als Server verwende ich `vsftpd` ("Very Secure FTP Server"), bei dem man für die bei uns üblichen Zwecke die Konfigurationsdatei `/etc/vsftpd.conf` anpassen muss. Die wichtigsten Änderungen zeigt die folgende Tabelle.

`/etc/vsftpd.conf` original	`/etc/vsftpd.conf` angepasst
`listen=NO`	`listen=YES`
`#write_enable=YES`	`write_enable=YES`
`#local_umask=022`	`local_umask=022`
`#idle_session_timeout=600`	`idle_session_timeout=3600`
`#nopriv_user=ftpsecure`	`nopriv_user=ftpsecure`

Aus der letzten Zeile folgt noch, dass wir einen Benutzer `ftpsecure` anlegen müssen, z. B. durch `adduser ftpsecure`.
Anschließend können wir den Server starten mit `invoke-rc.d vsftpd start`.

Am Schluss dieses Kapitels soll noch auf den HTTP-Server `lighttpd` eingegangen werden. Manchmal möchte man gerne HTML-Seiten, die man aus dem Internet heruntergeladen hat, auf einem anderen Rechner im LAN in den Browser laden. Dazu braucht man einen HTTP-Server. Gerade unter Linux gibt es hierfür mächtige Programmpakete, am bekanntesten ist wohl der Apache. Für unsere Zwecke brauchen wir nur etwas Leichtes, möglichst einfach zu Konfigurierendes, also kurz den `lighttpd`.
Das Programm wird gesteuert durch die Konfigurationsdatei `/etc/lighttpd/lighttpd.conf`. Der Eintrag

```
server.document-root        = "/var/www"
```

gibt das Verzeichnis an, von dem aus der Zugang zu den Inhalten gesteuert wird, die der Server einem Besucher anbietet. Defaultmäßig befindet sich dort lediglich die Datei `index.lighttpd.html`.
Will man weitere Bereiche des Rechners zugänglich machen, so erreicht man dies durch Anlegen von Links im selben Ordner, z. B.:

```
ln -s /home home
ln -s /usr/local usrlocal
```

Per Default sind diese Zugänge aber noch nicht freigeschaltet. Dies geschieht erst durch Einfügen einer zusätzlichen Zeile in `lighttpd.conf`:

```
dir-listing.activate = "enable"
```

Änderungen in der Konfigurationsdatei werden erst durch einen Neustart des Servers wirksam:

```
invoke-rc.d lighttpd restart
```

Damit der Server bei jedem Neustart des Rechners automatisch gestartet wird, ruft man einmalig `update-rc.d` auf:

```
update-rc.d lighttpd defaults
```

Jetzt kann man von einem anderen Rechner im lokalen Netz Inhalte des so eingerichteten Rechners anzeigen lassen, indem man im Browser eingibt:

`http://saturn/home` bzw. `http://saturn/usrlocal`

13. Einrichten von Druckern

Einen Drucker unter Linux zum Laufen zu bringen, kann ganz einfach sein, wenn die betreffende Distribution die nötigen Treiber und entsprechende Funktionalität auf dem Desktop bereitstellt. Wenn das nicht der Fall ist, kann man eventuell auf der Kommandozeile weiterkommen. Die Standardsoftware zum Drucken unter Linux ist CUPS (Common Unix Printing System). Die wichtigsten Pakete heißen `cups` und `cups-client`. Bei mir sind noch die Pakete `cups-browsed, cups-bsd, cups-core-drivers, cups-daemon, cups-filters, cups-filters-core-drivers, cups-ppdc, cups-common, cups-server-common` und `printer-driver-hpcups` installiert.

Die Konfigurationsdateien befinden sich unter `/etc/cups`. Meiner Erfahrung nach muss man im Allgemeinen nicht manuell eingreifen, wenn man nicht spezielle Zugangsrechte einrichten will.
Im Übrigen bringt CUPS ausführliche (wenn auch nicht immer leichtverständliche) Dokumentation mit, die im Browser durch Eintippen von `localhost:631` in der Eingabezeile zugänglich wird.
Voraussetzung ist allerdings, dass man den CUPS-Server gestartet hat. Dies geschieht durch die Terminaleingabe (unter Root) von

`/etc/init.d/cups start` oder `invoke-rc.d cups start`

Automatischen Start von CUPS bei jedem Systemneustart erreicht man durch einmalige Eingabe von (unter Root)

`update-rc.d cups defaults`

Diese Angaben beziehen sich auf Debian, gelten vermutlich auch für Ubuntu und Mint.

Neben der Dokumentation bietet `localhost:631` auch umfangreiche Konfigurationsmöglichkeiten.
Wir wollen uns hier auf die Einrichtung eines Druckers auf der Kommandozeile beschränken.
Zunächst braucht man vom Hersteller eine für den Drucker spezifische PPD-Datei (PPD = Postscript Printer Definition; falls der Hersteller weitergehende Installationssoftware bereitstellt – umso besser, dann kann man sich vielleicht das Folgende sparen). Im falle meines Laserdruckers heißt sie `Kyocera_ECOSYS_P2135dn.PPD`. Sie ist eine Textdatei, in der man lesen kann und wird in das Verzeichnis `/etc/cups/ppd` kopiert.
Wenn es sich um einen netzwerkfähigen Drucker handelt, braucht man seine IP-Nummer. Ist im Router DHCP aktiviert (was standardmäßig meist der Fall ist), wird dem per LAN-Kabel ans lokale Netz angeschlossenen Drucker beim Einschalten automatisch eine Netzwerkadresse zugewiesen. Bei WLAN-Anschluss sollte das Druckerhandbuch Auskunft geben. Vermutlich läßt er sich über den Browser konfigurieren. Im Erfolgsfall sollte man die IP-Nummer feststellen könnene, wenn man die Adresse des Routers im Browser aufruft (bei mir genügt die Eingabe von `fritz.box`) und dann zu "Heimnetz" (oder so ähnlich) navigiert. Ob die Verbindung funktioniert, kann man dann von der Kommandozeile aus mit `ping` *IP-Nummer* feststellen.

Das wichtigste Kommando für die Druckereinrichtung ist `lpadmin`. Ich verwende es als Teil eines Skripts, das noch einige andere Kommandos enthält, z. B. um den CUPS-Server neu zu starten. Es sieht für meinen Schwarzweiß-Laserdrucker folgendermaßen aus:

```
/usr/sbin/lpadmin -x ecosysbw
t=‘pidof cupsd‘
export t
kill $t
/etc/init.d/cups restart
/usr/sbin/lpadmin -p ecosysbw -v socket://192.168.168.48:9100 \
  -P /etc/cups/ppd/Kyocera_ECOSYS_P2135dn.en.PPD
/usr/sbin/lpadmin -d ecosysbw
/usr/sbin/lpadmin -p ecosysbw -u allow:all
/usr/sbin/accept ecosysbw
/usr/sbin/cupsenable ecosysbw
```

Man sieht, dass `lpadmim` zunächst mit drei Parametern aufgerufen wird:

- `-p` Der Name des Druckers (frei wählbar)
- `-v` Die Geräte-URI (URI = Uniform Resource Identifier)
- `-P` Der Pfad der PPD-Datei

Der Aufruf `lpadmin -d ecosysbw` ein paar Zeilen weiter sorgt dafür, dass `ecosysbw` zum Default-Drucker erklärt wird.

Die Geräte-URI (Device-URI) besteht hier (es kann noch einiges hinzukommen) aus drei Teilen, dem Netzwerk-Protokoll (hier `socket`), der Netzwerkadresse des Druckers und, durch einen Doppelpunkt getrennt, der Port.

Für einen nicht netzwerkfähigen, an einem NAS (Network Access Storage) angeschlossenen Drucker lautete bei mir die Device-URI `ipp://192.168.168.9:631/printers/kyocera`.

Welches Protokoll von Linux unterstützt wird, erfährt man durch das Kommando `lpinfo -v`. Welches Protokoll der Druckerhersteller unterstützt, und unter welchem Port, erfährt man hoffentlich aus dem Druckerhandbuch oder vielleicht mit Hilfe des Kommandos `nmap`. Die wichtigsten Netzwerkprotokolle für Drucker sind `socket, ipp` (Internet Printing Protocol) und `smb` (Server Message Block), letzteres für Drucker unter Windows. Wie die genannten Beispiele illustrieren, läuft das Protokoll `socket` typischerweise über den Port 9100 und das Protokoll `ipp` über den Port 631.

Für lokale Drucker sind andere Protokolle zuständig, z. B. `parallel` (heutige Rechner haben meist keine parallele Schnittstelle mehr) oder `usb`. An die Stelle von Netzwerkadresse und Port tritt dann eine Gerätedatei (Device), wie man sie im Verzeichnis `/dev` findet. Die Device-URI lautet dann z. B. `usb:/dev/usb/lp1`.

Wie man sieht, kann ein wenig Experimentierfreudigkeit (und entsprechende Frustrationstoleranz) gefragt sein.

Nun wollen wir zu guter Letzt auch etwas drucken. Dafür sorgt das Kommando `lp`. Ein typischer Aufruf sieht wie folgt aus:

```
lp -d ecosysbw -o "fit-to-page media=a4 sides=two-sided-long-edge" \
   -P "1,11-28" Datei
```

Damit werden die Seiten 1 und 11 bis 28 doppelseitig ausgedruckt, wobei darauf geachtet wird, dass das Format DIN A4 möglichst (mit einem gewissen Rand) ausgefüllt wird.
Die Abarbeitung der Druckaufträge kann man mit dem Kommando `lpstat -t` verfolgen. Jeder Druck-Job wird da mit einer Nummer und einer den Umfang bezeichnenden Zahl ausgewiesen, Die Nummer kann man benutzen, um den Druckjob nötigenfalls abzubrechen mit `cancel` *Job-Nummer*.

Alle in diesem Kapitel erwähnten Kommandos haben natürlich einen sehr viel größeren Funktionsumfang als hier bahandelt werden konnte. Konsultation der Manualseiten ist deshalb empfohlen, ebenso wie die erwähnte CUPS-Dokumentation unter der Adresse `localhost:631` im Browser. Den Abschnitt über Netzwerkdrucker findet man beispielsweise, indem man nacheinander `Overview of CUPS` und `Using Network Printers` anklickt.

Nützliche Webadressen waren für mich

```
www.willemer.de/informatik/unix/druckadm.htm        und
www.mpipks-dresden.mpg.de/~mueller/docs/suse9.2/suselinux-adminguide_de\
  /html/ch12s04.html
```

Auch Kofler[1] widmet CUPS ein längeres Kapitel.

14. Kleiner Exkurs über Dateisysteme und Partitionierung

Daten, die permanent verfügbar sein sollen (sie können im Arbeitsspeicher erzeugt worden sein oder aus einer Quelle im Netzwerk stammen), müssen auf Datenträgern (volumes) gespeichert werden, also auf internen oder externen Festplatten, auf (heute meist externen) Flashspeichern (zumeist Sticks) oder auf optischen Medien wie CDs oder DVDs.
Damit man die Daten später wieder findet, muss festgehalten werden, an welcher Stelle (Adresse) und unter welchen Dateinamen sie auf dem Datenträger lokalisiert sind. Dazu braucht jeder Datenträger (genauer gesagt jede Partition eines Datenträgers) ein Dateisystem, das die entsprechenden Informationen enthält.
Am bekanntesten ist vielleicht heute noch das unter DOS verwendete FAT (File Allocation Table oder Dateizuordnungstabelle) und seine Erweiterungen, das erst beginnend mit Windows 2000 von NTFS (New Technology File System) abgelöst wurde. Die Hauptnachteile von FAT sind das Fehlen von differenzierten Benutzerrechten, die Begrenzung in der Länge und Struktur der Dateinamen (8.3-System), sowie die aus heutiger Sicht zu niedrigen Obergrenzen von Datei- und Partitionsgrößen. Trotzdem wird es auch heute (2016) noch überwiegend für die Formatierung von externen Festplatten und vor allem Speichersticks verwendet. Ursprünglich waren FAT12 und FAT16 für Disketten bzw. Festplatten vorgesehen. FAT32 ermöglichte größere Dateien und Partitionen, das damit (auch mit FAT16) kombinierte VFAT längere Dateinamen. FAT wird in all seinen Varianten von Linux voll unterstützt (Pakete wie `mtools` oder `dosfstools` dienen eigens diesem Zweck).
Dasselbe gilt für das heute von Windows ausschließlich benutzte NTFS nur beinahe (kein schreibender Zugriff auf verschlüsselte Dateien). Die Pakete `ntfs-3g` und `fuse` ermöglichen aber vollständigen Lese- und Schreibzugriff auf normale Dateien.
Die wichtigsten Dateisysteme für Linux sind `ext2` (Second Extended Filesystem) und seine Fortschreibungen `ext3` und `ext4`. Sie erlauben z. B. differenziertes Vergeben von Dateirechten an Eigentümer, Gruppen und die übrige Welt (-> S. 7) und im Vergleich zu FAT sehr lange Datei- und Pfadnamen. Die Metadaten der Dateien (z. B. Besitzer, Zeitpunkt der Erstellung etc.) werden in den Inodes (eine Zeigerstruktur, die auf die Dateiblöcke verweist) und dem Superblock gespeichert, die sich im Anfangsbereich des Datenträgers befinden.
Optische Datenträger wie CDs und DVDs werden zumeist mit dem Dateisystem ISO 9660 formatiert. In seiner ursprünglichen Form ist es hinsichtlich Dateinamen und Ordnerstruktur stark eingeschränkt. Wichtige Erweiterungen sind mit den Namen *Rock Ridge* (für Linux), *Joliet* (für Windows) und *El Torito* (für bootbare CDs) verbunden. Auf diese Stichworte sollte achten, wer selbstzusammengestelllte Daten auf eine CD brennen möchte. Er muss dann zunächst, z. B. mit `genisoimage` (früher `mkisofs`) auf der Festplatte ein iso-Dateisystem erstellen, um es dann, z. B. mit `wodim` (früher `cdrecord`) zu brennen. Das Programm `genisoimage` ist sehr leistungsfähig und nicht so ganz leicht zu bedienen (die Manualseite umfaßt mehr als 1300 Zeilen!).
Wegen der Unzulänglichkeiten von ISO-9660 ist als Alternative das UDF (Universal Disk Format) entwickelt worden. Es ermöglicht auf wiederbeschreibbaren Medien sogenanntes "packet writing" und damit auch echtes Löschen und Überschreiben von Daten, dh. eine einer Festplatte ähnliche Verwendung.
Allerdings würde ich für diesen Zweck die DVD-RAM vorziehen, ein in Sektoren eingeteiltes Medium, das in kreisförmig-konzentrischen Spuren (tracks) beschrieben wird (anders als normale CDs und DVDs, die spiralförmig beschrieben werden). Sie kann ohne spezielle Software so wie Festplatten oder Sticks beschrieben werden, man kann sie auch partitionieren und verschiedene Dateisysteme darauf einrichten. Sie hat im Vergleich mit herkömmlichen CDs und DVDs eine weitaus längere Haltbarkeit bei wiederholtem Überschreiben. Auf Grund ihres relativ hohen Rohlingpreises und des Preisverfalls bei Speichersticks hat sie, zumindest im Gebrauch auf PCs keine große Bedeutung erlangt.
Im normalen Linux-Alltag hat man es meist mit dem Brennen von aus dem Internet bezogenen ISO-Images zu tun. Dafür braucht man nur das Programm `wodim`, das man beispielsweise folgendermaßen aufruft (für die Datei `xyz.iso` im aktuellen Verzeichnis):

```
wodim -v -dao speed=0 dev=/dev/sr0 -data xyz.iso
```

Der Parameter `-dao` (Disk At Once) empfiehlt sich für normales Schreiben mit modernen (MMC-konformen) Brennern, `-tao` (Track At Once) für Schreiben in mehreren Sitzungen. Der Gerätename (device name) kann verschieden lauten (z. B. `/dev/sg0`, `/dev/sg1` etc.), bei meinen Rechnern ist er seit langem stets `/dev/sr0`. Die Schreibgeschwindigkeit Null sorgt dafür, dass der (MMC-konforme) Brenner die kleinste mögliche Geschwindigkeit wählt. Die umfangreiche Manualseite für `wodim` sollte man konsultieren, wenn etwas nicht funktioniert, man spezielle Treiber braucht, etc.

Ein Linuxinstallation benötigt normalerweise mindestens zwei Dateisysteme (Partitionen), man muss also den dafür vorgesehenen Datenträger partitionieren. Partitionen sind voneinander getrennte Bereiche einer Festplatte oder eines Flashspeichers, auf denen jeweils ein eigenes Dateisystem eingerichtetr werden kann. Damit die Partitionen adressierbar sind, muss eine Partitionstabelle eingerichtet werden. Bei einer jungfräulichen Festplatte muss man zuerst mit einem Partitionierungsprogramm einen "Disk Label" (dh. eine leere Partitionstabelle) einrichten. Bis vor wenigen Jahren war das bei Intel-kompatiblen PCs ein DOS-Label, seitdem ist der GPT-Label hinzugekommen, der in absehbarer Zeit den DOS-Label verdrängen wird.

Die Partitionstabelle herkömmlicher Art (DOS) wird im 512 Bytes umfassenden MBR (Master Boot Record) eingerichtet und hat Platz für die Adressen von vier Partitionen, die primäre Partitionen genannt werden. Um mehr Partitionen einrichten zu können, hat man zu einem Trick gegriffen. Man ersetzt eine der primären Partitionen (am besten die vierte) durch eine sogenannte erweiterte Partition. Diese dient dann als Behälter (container) für die sogenannten logischen Laufwerke. Hier hat man auch, anders als im MBR, Platz für deren Adressen.

Unter Linux gibt es eine Reihe von Programmen zum Partitionieren von Datenträgern. Ich verwende fast immer `fdisk`, obwohl manche `parted` oder das mit einem graphischen Frontend ausgestattete `gparted` vorziehen. `fdisk` spricht einen Datentrger (Festplatte oder Stick) an mit

```
fdisk /dev/sdx
```

wobei x ein Buchstabe zweischen `a` und `z` sein kann. Fast immer erhält die erste interne Festplatte beim Bootvorgang den Buchstaben `a`, eine zweite interne den Buchstaben `b`, weitere Datentrger die folgenden Buchstaben des Alphabets. Bei trennbaren Datenträgern wie USB-Sticks oder USB-Platten hängt der Buchstabe von der Anordnung der USB-Steckplätze und gegebenenfalls von der Reihenfolge des Einsteckens ab. Vorsicht ist also geboten.
Man nennt die Einträge im Verzeichnis `/dev` Gerätedateien (device files), obwohl sie keine Dateien im herkömmlichen Sinn sind (man kann sie beispielsweise nicht editieren). Man sagt gerne, dass bei Unix so ungefähr alles unter die Bezeichnung "Datei" fällt (`->` auch S. 7). Mit dem Aufruf von `fdisk` kommt man in eine Umgebung, die einem sämtliche für die Partitionierung nötigen Kommandos zur Verfügung stellt. Die wichtigsten sind:

- `p` Listet Informationen zu Größe und Struktur des Datenträgers sowie die Partitionstabelle auf
- `m` Listet ein Hilfemenü auf
- `d` Löscht eine Partition
- `n` Erzeugt eine neue Partition (es wird nach Typ, Nummer, Lage und Größe gefragt)
- `t` Ändert die Bezeichnung des Dateisystems einer Partition
- `a` Setzt bzw. entfernt die Kennzeichnung für die Bootbarkeit einer Partition
- `w` Schreibt die Partitionstabelle und verlässt `fdisk`
- `q` Verlässt `fdisk` ohne zu schreiben

Das Kommando `q` ermöglicht es, risikolos zu experimentieren, bzw. Eingabefehler ungeschehen zu machen. `fdisk` partitioniert den Datenträger, erzeugt aber kein Dateisystem. Dies geschieht für die Einrichtung eines `ext3`-Dateisystems, z. B. durch das Kommando

```
mke2fs -j -b block-size -i bytes-per-inode /dev/sdxn
```

Dabei bezeichnet, wie soeben beschrieben, x die Kennung des Datenträgers (a, b, c, ...) und n die Nummer der Partition auf demselben (1, 2, 3, ...).

Der Parameter `-j` legt fest, dass ein für das `ext3`-System typisches Journal angelegt wird, das beim Neustart des Rechners nach einem ungeordneten Shutdown eine schnelle Reparatur des Dateisystems ermöglicht. Mit dem Paramter `-b` wird die Blockgröße bestimmt (1024, 2048 oder 4096 Bytes), dh. die Größe der Portionen, in die eine Datei beim Schreiben auf den Datenträger aufgeteilt wird. Eine Datei belegt auf dem Datenträger also unabhängig von ihrer Größe immer ein ganzzahliges Vielfaches dieser Blockgröße. Der Parameter `-i` gibt die Datenmenge in Bytes an, für die jeweils ein Inode (-> S. 59) zur Verfügung gestellt wird. Da jede Datei einen Inode benötigt, kann man sie, um Platz zu sparen, umso größer wählen, je größer die zu erwartende durchschnittliche Dateigröße auf der Partition ist.
Die Parameterwerte für `-b` und `-i` kann man nach der Formatierung (Einrichtung des Dateisystems) nicht mehr ändern (meist wird man ohnehin die voreingestellten Werte übernehmen), wohl aber kann man ein `ext2`-System nachträglich durch Hinzufügen eines Journals in ein `ext3`-System umwandeln. Dazu und noch für vieles mehr dient das Programm `tune2fs`.

Ein weiteres wichtiges Programm ist `e2fsck`, das `ext2-` bzw. `ext3`-Partitionen auf Fehler überprüft und gegebenenfalls repariert. Man sollte unbedingt Gebrauch davon machen, wenn man beispielsweise versehentlich einen Speicherstick abgezogen hat, ohne ihn zuvor auszuhängen. `e2fsck` bezieht sich wie `mke2fs` und `tune2fs` (und im Unterschied zu `fdisk`) stets auf eine Partition, also beispielsweise mit

`e2fsck /dev/sd`*xn* oder `tune2fs -j /dev/sd`*xn* (Einrichten eines Journals)

Man sollte selbstverständlich vor der Anwendung der obengenannten Programme einen Blick in deren Manualseiten werfen.

Wie bereits erwähnt, braucht eine Linuxinstallation mindestens zwei Dateisysteme, neben der gerade beschriebenen `ext2-` oder `ext3`-Partition eine Auslagerungs-(Swap-)Partition. Das zugehörige Dateisystem wird eingerichtet mit

`mkswap /dev/sd`*xn*

Wenn man nicht die Möglichkeit hat, eine Swap-Partition zu erzeugen, kann statt dessen auch eine Datei herangezogen werden. Man erzeugt zuerst eine leere Datei (z. B. eine 500MB-Datei im aktuellen Verzeichnis) und wendet anschließend `mkswap` darauf an:

`dd if=/dev/zero of=`*Datei* `bs=512 count=1000000`
`mkswap` *Datei*

Die entsprechenden Programme für FAT-Partitionen sind `mkdosfs` (oder `mkfs.fat`) und `dosfsck`.

Um auf eine Partition lesend oder schreibend zugreifen zu können, muss sie eigehängt (gemountet) werden. Als Einhängepunkt (mount point) wählt man ein bereits existierendes (bzw. neu generiertes) Verzeichnis, um dann das Kommando `mount` aufzurufen. Für eine `ext3`-Partition sieht das z. B. folgendermaßen aus:

`mount -v -t ext3 -o noatime,nodiratime /dev/sd`*xn* `/mnt/sd`*xn*

Der Parameter `-t` hat den Dateisystemnamen als Argument, der Parameter `-o` kann eine Reihe durch Kommata getrennte Argumente aufnehmen. Das Argument `noatime` soll dazu dienen, den Verschleiß des Datenträgers zu verringern. Bei Linux wird nämlich bei jedem Dateizugriff (auch bei nur lesendem) ein entsprechender Vermerk auf den Datenträger geschrieben. Das kann dazu führen, dass an ganz bestimmten Stellen außerordentlich häufige Schreibzugriffe erfolgen und zu schnellem Verschleiß führen. Bei Festplatten ist das kein Problem, wohl aber bei Flashspeichern und DVD-RAMs.

Eine FAT-Partition wird z. B. gemountet durch:

`mount -v -t vfat -o umask=000 /dev/sd`*xn* `/mnt/fat`

der Parameter `umask` bedeutet, gibt an, um wieviel die Oktaltripel (-> S. 8) für die Dateirechte ihr Maximum von `777` unterschreiten sollen. Durch den Wert `000` wird sichergestellt, dass die Dateien der FAT-Partition für die Allgemeinheit les- und schreibbar sind.

Es ist vielleicht nützlich daraufhinzuweisen, dass es vom Dateisystem abhängt, welche Mountoptionen unterstützt werden und welche nicht. Die Manualseite gibt detaillierte Auskunft. Viele

Dateien des Betriebssystems werden bereits beim Bootvorgang gebraucht, es muss also dafür gesorgt werden, dass die Systempartition bereits in einem sehr frühen Stadium gemountet wird. Konfiguriert wird dies in der Datei `/etc/fstab`. Da sie bereits bei Installation des Linuxsystems angelegt wird, braucht man sich im günstigsten Fall nicht weiter darum zu kümmern. Besteht jedoch ein Interesse daran, einen externen Datenträger unmittelbar nach dem Hochfahren des Rechners zur Verfügung zu haben, muss man selbst Hand anlegen. Die `fstab`-Zeile für das Mounten der Systempartition sieht bei einem meiner Rechner folgendermaßen aus:

```
UUID=16b36c43-a9db-4aad-80d8-2f7b02576c2d  /  ext3  errors=remount-ro  0  1
```

Der Eintrag besteht aus mehreren Feldern, die durch (ein oder mehrere) Leerzeichen voneinander getrennt sind. Das erste Feld enthält die Gerätebezeichnung. Man könnte `/dev/sda1` erwarten, hier wird aber die `UUID` (Universally Unique Identifier) verwendet. Sie ist eine (hoffentlich) weltweit eindeutige Partitionskennung, die es z. B. ermöglicht, von externen Platten oder Sticks zu booten, ohne einen bestimmten Steckplatz dafür vorsehen zu müssen. Man kann die UUID ermitteln mit `blkid /dev/sd`*xn*.
Das zweite Feld enthält den Einhängeort (mount point) des Dateisystems, das dritte den Dateisystemtyp, das vierte die Mount-Optionen (die auf der Kommandozeile dem Paramter `-o` folgen). Das fünfte Feld gibt an, ob gegebenenfalls ein Dump erfolgen soll, das sechste gibt an, ob und in welcher Reihenfolge die in der `fstab` aufgeführten Dateisysteme überprüft werden sollen.
Die UUID kann auch bei einem Mountbefehl auf der Kommandozeile benutzt werden. Man muss dann nur die Gerätebezeichnung `/dev/sd`*xn* durch `-U` *UUID* ersetzen.

Das Einhängen einer CD oder DVD (beispielsweise in das Verzeichnis `/mnt/iso`) erfolgt durch das Kommando

```
mount -v -t iso9660 /dev/sr0 /mnt/iso
```

Sehr nützlich und zeitsparend kann es sein, eine CD oder DVD zu kopieren und in der Folge mit der Abbilddatei zu arbeiten. Sie lässt sich nämlich ganz einfach mit Hilfe einer sogenannten *Loop Device* mounten:

```
mount -v -t iso9660 -o loop dvd.img /mnt/iso
```

Es sei noch erwähnt, dass man eine (typischerweise große) Datei nicht nur als Swap nutzen kann, wie weiter oben beschrieben, sondern in ihr auch jedes beliebige Dateisytem einrichten und anschließend mounten kann, z. B.:

```
dd if=/dev/zero of=Datei bs=512 count=1000000
mke2fs Datei
```

Zum Schluss dieses Kapitels möchte ich noch kurz auf die Besonderheiten der GPT-Partitionierung (-> S. 60) eingehen. GPT steht für *GUID Partition Table*, wobei GUID *Globally Unique IDentifier* bedeutet. Bei dieser neuen Art der Partitionierung wird kein Unterschied mehr zwischen primären, erweiterten und logischen Partitionen gemacht, die Partitionstabelle wird jetzt in den Platz zwischen dem MBR (Master Boot Record, der erste 512 Bytes große Sektor auf der Platte) und dem Beginn der ersten Partition (meist beginnend mit Sektor 2048) geschrieben (eine sekundäre GPT hinter die letzte Partition). Der MBR bleibt bestehen, er bekommt jetzt eine Schutzfunktion (protective MBR) gegenüber irrtümlicher Anwendung von MBR-Partitionswerkzeugen.
Mit Windows 8 wurde GPT zeitgleich mit UEFI (Unified Extensible Firmware Interface) eingeführt, wodurch das traditionelle BIOS (Basic Input Output System) abgelöst wurde. GPT-partionierte Festplatten können aber trotzdem mit herkömmlichen BIOS-Rechnern betrieben werden, eine Kopplung mit UEFI ist also nicht zwingend.
Wikpedia (deutsch) liefert recht gute Informationen unter dem Eintrag *GUID Partition Table*.
Man kann BIOS-Rechner auch von GPT-Platten booten, allerdings booten diese dann nicht vom MBR aus, sondern von einer separaten, typischerweise 1 MByte großen Partition mit der Typenbezeichnung EF02 (BIOS boot partition), die zuvor eingerichtet werden muss.
Zum Partitionieren von GPT-Platten eignet sich das Programm `gdisk`, sozusagen die Fortschreibung von `fdisk`. Es ist ganz ähnlich zu bedienen wie dieses.

15. Vermischtes (`rsync` , `gpg` , Mauszeiger, `grub`)

Zum **Kopieren bzw. Synchronisieren** von Dateien und Verzeichnissen, auch über das Netzwerk eignet sich das Programm `rsync` , das im letzteren Fall die Installation eines Servers, z. B. `openssh-server` voraussetzt. Ein raffinierter Algorithmus vermeidet unnötige Kopiervorgänge oder minimiert den für sie nötigen Aufwand. Der Aufruf des Programms lautet allgemein

`rsync [`*Optionen*`]` *Quelle Ziel*

Quelle kann eine einzelne Datei oder ein Verzeichnis sein, im ersten Fall kann auch das Ziel eine Datei sein, andernfalls muss es ein Verzeichnis sein. Eine der beiden Seiten kann auf einem entfernten Rechner im Netzwerk liegen, aber nicht beide. Wird das Quellverzeichnis durch einen abschließenden Schrägstrich begrenzt, bedeutet das, dass im Ziel kein neuer Ordner erzeugt, sondern nur der Verzeichnisinhalt kopiert wird. Die wichtigsten Optionen sind

`-v`	Wortreiche (verbose) Protokollierung
`-n`	Testlauf (dry run) ohne tatsächliche Veränderungen
`-r`	Kopieren einschließlich der Unterverzeichnisse (rekursiv)
`-l`	Symbolische Links (`->` S. 10) werden als solche kopiert
`-p`	Dateirechte werden beibehalten
`-t`	Modifikations-Zeitstempel wird beibehalten
`-g`	Gruppenzugehörigkeit wird beibehalten
`-o`	Datei-Eigentümer wird beibehalten (nur mit Root-Rechten)
`-D`	Device-Dateien und spezielle Dateien werden beibehalten
`-a`	Abkürzung für `-rlptgoD`
`-u`	*Ziel*-Dateien, die neuer sind als in *Quelle* bleiben unverändert
`-b`	In *Ziel* werden gegebenenfalls Backup-Dateien angelebt
`--suffix=`*suffix*	Die Backup-Dateien erhalten den Suffix *suffix* (Default `~`)
`--del`	Überzählige Dateien in *Ziel* werden gelöscht
`--exclude=`*Muster*	Dem Muster *Muster* entsprechende Dateien werden ausgenommen

Im folgenden Beispiel soll gezeigt werden, wie das laufende Verzeichnis auf einem Rechner im Netzwerk in einem Unterordner des Home-Verzeichnis des Users `vitus` gesichert wird:

```
rsync -auv --del . vitus@192.168.168.216:/home/vitus/backups
```

In umgekehrter Richtung, wenn man vorsichtshalber mit Backup-Dateien arbeiten will, sieht es dann wie folgt aus:

```
rsync -abv --suffix=".b" --del vitus@192.168.168.216:/home/vitus/backups/ .
```

Zum **Verschlüsseln** von einzelnen Dateien eignet sich besonders gut das Paket `gnupg` . Die einfachste Anwendung ist die mit der symmetrischen Verschlüsselung:

`gpg -c` *Datei*

Man wird zweimal nach dem Passwort gefragt und dann erhält man das Resultat in der Form *Datei*`.gpg`.

Die Entschlüsselung erfolgt mit

`gpg -d` *Datei*`.gpg` Ausgabe auf Standard-Output

Einen roten **Mauszeiger** erhält man, indem man das Verzeichnis `~/.icons/default` anlegt und dort einen Link (`->` später) erzeugt mit

```
vitus@pluto ~/.icons/default $ ln -s /etc/X11/cursors/redglass.theme \
                               index.theme
```

Voraussetzung ist allerdings, dass das Paket `xcursor-themes` installiert ist.

Zum Schluss noch etwas zur **Reparatur des Bootloaders** `grub`.
Wenn das System nicht mehr startet, kann das am Masterbootrecord oder an der Konfigurationsdatei `/boot/grub/grub.cfg` liegen. Wir nehmen an, dass `/dev/sda2` die fehlerbehaftete Systempartition ist. Wenn man eine passende Live-CD hat, dann kann man mit ihr booten und dann, Michael Kofler [1] folgend, die folgenden Kommandos eingeben (natürlich mit ROOT-Rechten; statt `-o bind` kann man auch einfach `--bind` schreiben):

```
mkdir /syspart
mount /dev/sda2 /syspart          (Systempartition)
mount -o bind /dev  /syspart/dev
mount -o bind /dev/pts  /syspart/dev/pts
mount -o bind /proc /syspart/proc
mount -o bind /sys  /syspart/sys
mount -o bind /run  /syspart/run
```

Dann wechseln wir in eine `chroot`-Umgebung, dh. hier ist das Wurzelverzeichnes jetzt `/dev/sda2`. Am besten wählen wir dafür ein anderes Terminalfenster

```
chroot /syspart
mount /dev/sda1 /boot             (separate Bootpartition, falls vorhanden)
update-grub                       (erzeugt /boot/grub/grub.cfg )
grub-install /dev/sda             (Installation in Masterbootrecord)
exit
```

Abschließend unmounten wir wieder:

```
umount /syspart/dev/pts
umount /syspart/dev
umount /syspart/proc
umount /syspart/sys
umount /syspart/run
```

"Warum Linux auf der Kommandozeile?" wird im Netz unter `pulp-fiction.org/owncloud` abgelegt, und zwar im Ordner `linuxdoc` unter dem Namen `linuxr.ps.pdf`. Login und Passwort heißen gleichlautend `computerhilfe`.
Da die von `ps2pdf` erzeugte PDF-Datei fehlerhaft ist, findet man dort auch `linuxr.ps`. Das verpackte Verzeichnis mit den Quelldateien ist unter dem Namen `linux.tar` ebenfalls dort abgelegt.

Zwei Bücher von Michael Kofler sind besonders empfehlenswert: [1][2]
(Vom ersten wird gegen Jahresende 2015 eine neue Auflage erwartet)
Über die Bash gibt es umfangreiche Literatur. Stellvertretend seien zwei englische Bücher und eine deutsche Website genannt: [3][4][5]
Einige interessante Websites seien noch erwähnt: [13][14]

Referenz

[1] Michael Kofler, Linux: Das umfassende Handbuch. Galileo Computing (Rheinwerk-Verlag) 2013.

[2] Michael Kofler, Linux-Kommandoreferenz: Shell-Befehle von A bis Z. Galileo Computing (Rheinwerk-Verlag) 2013.

[3] Cameron Newham, Bill Rosenblatt, Learning the bash Shell, 3rd Edition. O'Reilly Media 2005

[4] Carl Albing, JP Vossen, Cameron Newham, bash Cookbook. Solutions and Examples for bash Users. O'Reilly Media 2007

[5] Ubuntuusers. `wiki.ubuntuusers.de/shell/Bash-Skripting-Guide_für_Anfänger`

[6] GNU Emacs Manuals Online. `www.gnu.org/software/emacs/manual/`

[7] Debra Cameron, James Elliott, Marc Loy, Eric S. Raymond, Bill Rosenblatt, Learning GNU Emacs, 3rd Edition, O'Reilly Media 2004

[8] Der Texteditor EMACS. Rechenzentrum der Universität Hamburg-Harburg, 4. Neuauflage August 1994.
`www.tuhh.de/rzt/tuinfo/editors/emacs.pdf`

[9] Gnu Emacs — Eine Einführung und ein bißchen mehr .... Manuela Jürgens, Gesamthochschule in Hagen, 1997
`ftp://ftp.fernuni-hagen.de/pub/pdf/urz-broschueren/broschueren/a0289401.pdf`

[10] GNU Emacs Reference Cards. `www.gnu.org/software/emacs/refcards/`

[11] Donald E. Knuth, Computers & Typesetting, Volume A: The Texbook. Addison-Wesley Pub Co Inc; May 1986

[12] Leslie Lamport, LaTeX: A Document Preparation System, 2nd Edition, Addison Wesley Professional, 1994

[13] The Linux Documentation Project
`http://tldp.org`

[14] Debian Documentation
`http://www.debian.org/doc`

Index