

IF MYTH 05: Linux ist weiblich

Ein Kurs im Sommerstudium der Informatica Feminale 2001 in Bremen

Sibylle Nägle und Patricia Jung

07.–09. September 2001

Inhaltsverzeichnis

Intro	3
1 1. Tag	4
1.1 Geschichte	4
1.2 Komponenten	4
1.2.1 Kernel	5
1.2.2 Die Shell	7
1.2.3 Die Benutzerinnen	8
1.2.4 X-Server	10
1.2.5 Windowmanager	11
1.2.6 Desktop-Umgebungen	12
1.2.7 Clients, Server und andere Programme	13
1.3 Bootvorgang	16
1.4 Der Ausschaltknopf	19
1.5 Hilfe zur Selbsthilfe	20
1.5.1 Selbstauskunft	20
2 2. Tag	22
2.1 Dateisystem	22
2.1.1 Unterstützte Vielfalt	22
2.1.2 Mounten	24
2.1.3 Dateien	26
2.1.4 Verzeichnisbaum	27
2.1.5 Bewegen im Verzeichnisbaum	27
2.1.6 Benutzerinnen	28
2.1.7 Dateirechte	29
2.1.8 Dateirechte ändern	30

2.1.9	Weitere Befehle	32
2.1.10	Extrawurst: die Drucker	33
2.2	Editoren	34
2.2.1	Vi	34
2.2.2	emacs	36
2.2.3	Weitere Editoren	37
2.3	Crashkurs für Hobby-Sysadminen	38
2.3.1	Benutzerinnenverwaltung	38
2.3.2	Installieren von Software	40
2.3.3	Prozesse im Griff	43
2.3.4	Die Zeit unter Kontrolle	45
3	3. Tag	46
3.1	Die Bash	46
3.1.1	Eingebaute Befehle und Hilfsmittel	46
3.1.2	Shellskripte	50
3.2	Nützliches Kleinzeug	53
3.3	Netzwerk	57
3.3.1	Im Netzwerk bewegen	57
3.3.2	Wer ist wo online?	58
4	Anhang: Weiterführende Ressourcen	62

Intro

Dieser Kurs richtet sich an Anfängerinnen. Es setzt keine tieferen Betriebssystemkenntnisse von anderen Rechnerplattformen voraus, wohl aber Erfahrung im allgemeinen Umgang mit Computern. Ziel des Kurses ist, prinzipielle Konzepte von Linux/Unix zu vermitteln und den ersten Einstieg zu erleichtern. Es ist nicht das Ziel, möglichst viele Unixbefehle kennenzulernen.

Das vorliegende Skript ist nicht als Lehrbuch gedacht*, sondern soll als Gedankenstütze für die Kursnachbereitung dienen. Die behandelten Aspekte werden oft nicht erschöpfend dargelegt, sodass zusätzliche Notizen von Nutzen sein können. Ohne die Ausführungen im Kurs und den entsprechenden praktischen Übungen am Rechner mögen Teile des Skripts unverständlich erscheinen oder Vorkenntnisse voraussetzen, die die Nur-Leserin u. U. nicht hat.

Allerdings sollte es nach Abschluss des Kurses möglich sein, anhand des Skriptes die Abschnitte im Eigenstudium nachzuarbeiten, für die keine Zeit mehr war. Wir haben bewusst mehr Stoff ausgearbeitet, als wir vermutlich schaffen werden, um Raum für individuelle Präferenzen zu geben.

Da es keine benoteten Scheine gibt, legen wir Wert auf Teamarbeit auch bei der Lösung der Aufgaben. Die Mailingliste if-myth-05@lists.answergirl.de bietet eine Plattform, auf der Aufgaben diskutiert und gemeinsam Lösungen gefunden werden können. Allerdings merkt frau sich unserer Erfahrung nach Dinge besser, die sie selbst ausprobiert hat. Wer eine Lösung nicht im Alleingang findet, sollte eine gemeinsam gefundene Lösung daher auf jeden Fall selbst praktisch ausprobieren.

Bei Fragen stehen die Dozentinnen, Sibylle und Patricia, soweit es die Zeit erlaubt, persönlich, aber auch auf der Mailingliste zur Verfügung. Bitte denkt beim Fragen daran, dass auch andere Kurs-Teilnehmerinnen am selben Problem interessiert sein können, und gebt Eure Erkenntnisse (sofern Ihr sie nicht ohnehin aus der Mailingliste bezogen habt) weiter!

Die Vorkenntnisse der Kursteilnehmerinnen sind sehr unterschiedlich und verlangen daher von den Fortgeschritteneren eine gewisse Rücksicht. Sollte sich die eine oder andere jedoch über mehrere Abschnitte unterfordert fühlen, tretet bitte an uns heran: Wir versuchen eine Lösung zu finden. Wenn Ihr alles schon könnt, was dieses Skript zu vermitteln sucht, seid Ihr allerdings im falschen Kurs...

Wir wünschen uns und Euch eine gute Zusammenarbeit, viel Spaß und den einen oder anderen Geistesblitz

Sibylle und Patricia

*In einigen Abschnitten enthält es Material aus einem von Patricia Jung und Detlev Degenhardt verfassten Skript für Unix-Kompaktkurse am Rechenzentrum der Uni Freiburg.

1 1. Tag

1.1 Geschichte

Linux ist gerade mal 10 Jahre alt!

- Am 25. August 1991 schickt der 21-jährige finnische Student Linus Torvalds eine Mail an die Minix-Newsgroup, in der er MitstreiterInnen für sein Kernel-Projekt *FREAX* sucht. (Die ersten Versionen heißen noch nicht Linux.)
- Schon am 17. September wird die Version 0.01 auf einem ftp-Server zum Download bereitgestellt – 10.000 Zeilen Quelltexte. (Der aktuelle Kernel 2.4.9 umfasst 3,7 Millionen Zeilen Quellcode!)
- Die erste Distribution kommt 1992 auf den Markt, im selben Jahr folgen die erste Slackware-Distribution und die erste SuSE. 1993 entstehen Red Hat und Debian. Zur Zeit gibt es schätzungsweise rund 140 Linux-Distributionen.
- Seit April 1993 läuft das X-Window-System auf Linux, und am 16. April 1994 wird Versionsnummer 1.0 vergeben.
- Im Oktober 1996 wird die Idee zu KDE geboren, die Version 1.0 erscheint 1998. 1999 folgt GNOME mit der Version 1.0.

1.2 Komponenten

Eine der wichtigsten Eigenschaften von Unix ist seine große Flexibilität. Statt großer Anwendungspakete und komplizierter Dienstprogramme werden einfache Werkzeuge zur Verfügung gestellt, die lediglich eine Aufgabe (perfekt) erfüllen können, aber zu mächtigen Werkzeugen zusammensetzbar sind. Dieses Baukastenprinzip kann frau als Grundphilosophie von Unix bezeichnen.

Es gilt auch im Großen: Wenn frau eine Linux-Distribution in den Händen hält, so ist darin wesentlich mehr enthalten, als nur das Betriebssystem, und was zum Betriebssystem gehört, ist weniger, als frau auf den ersten Blick annehmen möchte.

Frau kann sich im Prinzip ihr eigenes Linuxsystem von Grund auf selbst zusammensetzen. Das kostet lediglich Online-Gebühren und eine Menge Arbeitsstunden. Einfacher wird es, diese Arbeit Leuten und Firmen zu überlassen, die das ohnehin tun, den Distributoren, und es ihnen u. U. in Form eines Kaufpreises zu vergüten.

Ein Distributor ist also lediglich jemand, der verschiedene Komponenten so zusammensetzt, dass ein möglichst unproblematisches und zu wartendes System zusammen kommt. Darin enthalten sind Eigenentwicklungen des Distributors (in der Hauptsache das Installationsprogramm und diverse Wartungsprogramme) sowie Software von Dritten, die so angepasst wird, dass sie mit den anderen Komponenten der Distribution zusammenspielt.

Das hat Vor- und Nachteile: Der Vorteil besteht darin, dass frau sich unter der Vielzahl der Distributionen genau die herausuchen kann, die ihren Bedürfnissen am ehesten

entspricht. Auch wenn im Zusammenhang mit Linux meist nur die großen Distributionen SuSE Linux, Red Hat Linux, Linux Mandrake, Calderas OpenLinux und die einzige nicht-kommerzielle Distribution in dieser Aufzählung, Debian, genannt werden: Es gibt viel, viel mehr.

Ein Nachteil besteht darin, dass frau u. U. eine Menge Frösche küssen muss, ehe sie die Prinzessin findet. Ein wahrscheinlich noch schwerwiegenderer: Was bei der einen Distribution so funktioniert, kann bei einer anderen komplett anders aussehen. Daher ist es gut, darüber Bescheid zu wissen, wie ein Linuxsystem an sich aufgebaut ist, statt das eigene Wissen an Tools einer bestimmten Distribution festzumachen. Allerdings gibt es auch hier in Konfigurationsdetails Unterschiede; besonders SuSE ist ein gutes Beispiel dafür, wie man doch wieder alles anders als alle anderen machen kann...

Doch damit wollen wir uns jetzt nicht aufhalten – lasst uns zunächst untersuchen, aus welchen Bestandteilen ein Linuxsystem aufgebaut ist.

1.2.1 Kernel

Wenn wir heute von Linux, Windows etc. sprechen, meinen wir meist weitaus mehr als das pure Betriebssystem selbst. Wir beziehen oft „alles, was an Software dabei war“ mit in die Definition ein. Das ist verständlich, denn mit dem Betriebssystem selbst kann die normale Anwenderin überhaupt nichts anfangen ist, sie kann noch nicht einmal auflisten, welche Dateien es auf dem Rechner gibt.

Nichtsdestotrotz sollten Linuxerinnen um den Unterschied zwischen dem Betriebssystem selbst (dem *Kernel* oder *Betriebssystemkern*), der Linux heißt, und „allem, was an Software dabei war“ (der *Distribution*) wissen, denn das hilft beim Fragenstellen im Fehlerfall.

Der Betriebssystemkern selbst sorgt nicht nur für das Ausführen und Verwalten von Prozessen, sondern stellt auch die Treiber für Dateisysteme, Hardware etc. bereit. Er ist dafür verantwortlich, dass wir es bei Linux mit einem *Multiuser/Multitasking*-Betriebssystem zu tun haben.

Was bedeutet Multiuser/Multitasking? Das altgediente (und immer noch nicht völlig ausgestorbene) DOS ist ein *Singleuser/Singletasking*-System. Der Rechner kann nur von einer Benutzerin (*user*) gleichzeitig bedient werden, diese kann auch nur eine Aufgabe (*task*) bzw. ein Programm zur gleichen Zeit ausführen. (Eine Ausnahme macht *print*, das auch im Hintergrund laufen kann.)

Windows 3.x selig war ebenfalls ein Singleuser-System. Hier können zwar schon mehrere Programme gleichzeitig arbeiten, aber nur in Form des nichtpräemptiven Multitaskings (to pre-empt – engl.: vorwegnehmen): Auf keinem Rechner mit nur einem Prozessor (egal, welches Betriebssystem) können mehrere Programme wirklich gleichzeitig laufen. Der Trick besteht darin, unmerklich für die Benutzerin jedem Programm kurzfristig Rechenzeit zur Verfügung zu stellen und zwischen den Programmen sehr schnell hin- und herzuschalten.

Fortsetzung folgt ...

... Fortsetzung

Bei Windows 3.x ist nun nicht das Betriebssystem für dieses Umschalten (*scheduling*) verantwortlich, sondern die einzelnen Applikationen müssen ihre Rechenzeit von sich aus wieder freigeben („nicht präemptiv“). Dies tun sie häufig nicht, was dann die bekannten „Hänger“ verursacht.

Windows 9x geht hier schon einen Schritt weiter: Das Betriebssystem selbst sorgt für das Scheduling (also: präemptives Multitasking). Aber auch Win 9x sind immer noch Singleuser-Betriebssysteme. Zudem nehmen sie es mit dem *Speicherschutz* nicht so eng: Wo es nur eine Benutzerin gibt, die alles darf (sogar am Betriebssystem vorbei direkt auf die Hardware zugreifen), ist es dem Betriebssystem selbst unmöglich, User-Prozessen weniger Rechte zu geben als Systemprozessen. Wenn Anwendungsprogramme nicht ganz sauber programmiert sind, können sie sich so beim Belegen des Hauptspeichers in die Quere kommen können.

Hier kommt jetzt die Stärke von Unixsystemen zum Vorschein: Auf ihnen können bereits seit vielen Jahrzehnten mehrere Benutzerinnen mehrere Programme gleichzeitig ausführen. Dass Prozesse nicht einfach Speicherseiten anderer Prozesse überschreiben dürfen, ist ebenfalls seit Langem selbstverständlich. Gleichzeitig ist das Benutzerinnenkonzept (verglichen z. B. mit Windows NT) recht einfach (wenngleich auch weniger feingliedrig), sodass die Trennung von administrativer Nutzung des Systems und Arbeiten als Anwenderin leicht in die Praxis umgesetzt werden kann.

Allerdings umfasst Linux selbst inzwischen unheimlich viel Funktionalität (kein Wunder für ein Betriebssystem, das vom Großrechner bis hin zur Armbanduhr auf aller möglichen Hardware läuft). Wer keine SCSI-Geräte hat, braucht z. B. auch keinen Treiber dafür im Betriebssystem. Da passt es gut, dass Linux im Quelltext erhältlich ist und frau sich ihr ganz persönliches Betriebssystem konfigurieren und kompilieren kann.

Dazu benötigt sie das Quelltextpaket, das normalerweise nach `/usr/src/linux` installiert wird. In diesem Verzeichnis findet sich eine Datei namens `README`, die (auf Englisch) erklärt, wie sich frau ihren eigenen Kernel bäckt.

Grundsätzlich ist es jedoch für einen Kernel heutigen Ausmaßes keine gute Idee, ständig und immer sämtliche Funktionalität im Speicher zu halten, auch wenn sie gar nicht gebraucht wird (weil das CD-ROM-Laufwerk augenblicklich gar nicht benutzt wird). Daher kann frau Treiber, die beim Booten nicht gebraucht werden, gut und gern in „ladbare Module“ (*loadable modules*) auslagern, anstatt sie alle fest in einen *monolithischen Kernel* einzukompilieren. Diese werden erst dann zum Kernel in den Speicher geladen, wenn sie benötigt werden. Ein solcher modularer Kernel wird von allen aktuellen Distributionen eingespielt, sodass das Selbstkompilieren eines ans eigene System angepassten Kernels heute in der Regel nur noch aus zwei Gründen nötig ist:

1. Frau will eine aktuellere Kernel-Version einspielen, als sie der Distributor zur Verfügung stellt.
2. Frau braucht Funktionalität, die im offiziellen Kernel nicht drin ist und vom Distributor nicht eingespielt wurde. Dann ist Kernelflicken mit Hilfe sogenannter *Patch*

Dateien angesagt.

Kernelbäckerei Ein `make clean` säubert die Kernel-Quellen von den Objektdateien etc. voriger Kernel-Kompilierungen. Mit `make menuconfig` kann frau den Kernel jetzt nach Gutdünken konfigurieren, wenn sie auf einer grafischen Oberfläche arbeitet, gibt es noch die Variante `make xconfig`. Sollte es bei beiden Fehlermeldungen geben, weil Bibliotheken für die grafische oder pseudografische Ausgabe nicht installiert wurden, geht auch `make config`. Hier wird frau der Reihe nach alle Einstellungen abgefragt und kann auch nicht wieder zurück gehen – also nur im Notfall zu empfehlen.

Zu den meisten Punkten kann frau sich Hilfe und eine Empfehlung abholen, ansonsten gilt es, Y, N oder M (für Module) zu sagen.

Die Konfiguration wird (so von der Konfiguriererin nicht anders festgelegt) in der Datei `/usr/src/linux/.config` abgelegt. Ein `make dep` schaut nach, ob alle Abhängigkeiten erfüllt sind; ein `make bzImage` kompiliert den neuen Linux-Kernel, der dann unter `/usr/src/linux/arch/i386/boot/bzImage` liegt und installiert werden kann.

Wird der Bootloader LILO benutzt, so sorgt `make bzlilo` statt `make bzImage` gleich mit für die Neukonfiguration des Bootloaders.

Dann gilt es noch, die Kernel-Module mit `make modules` zu kompilieren und mit `make modules_install` zu installieren.

Aufgabe:

Welche Linux-Version ist momentan aktuell?

Aufgabe:

Finde einen Kernel-Patch! Für welchen Kernel ist er gedacht?

Aufgabe:

Sind auf dem Arbeitsrechner die Kernel-Quellen installiert? Wenn ja: welche Version? Finde einen Punkt heraus, der als Modul konfiguriert ist! Woran hast Du das erkannt?

1.2.2 Die Shell

Die meisten Betriebssystembefehle sind nichts anderes als C-Programme, die über die sogenannten *system calls* auf den Kernel zugreifen. Doch da frau nicht direkt mit dem Kernel „sprechen“ kann, weil der ja dafür sorgen muss, dass sich die Applikationen beim Zugriff auf die Hardware nicht ins Gehege kommen, stellt sich die Frage, wie sie mit dem Kernel kommuniziert.

Die Kommunikation zwischen Benutzerin und dem Betriebssystem läuft auf der Kommandozeile immer über einen Kommandointerpreter (*shell*):

Benutzerin \iff Shell \iff Betriebssystemkern

Das ist bei DOS genauso (hier heißt die Shell `command.com`), nur fällt es da nicht so auf. Die Shell hat die Aufgabe, Eingaben anzunehmen, eventuell ein wenig zu verarbeiten und an die entsprechenden Stellen weiterzuleiten. Es gibt verschiedene Shells; allerdings sind nicht immer alle auf jedem Unix-Rechner vorhanden.

Der Name (Muschel, Schale) kommt daher, dass die Shell den Kern des Betriebssystems umschließt und die Kommunikation zwischen Kernel und Benutzer ermöglicht.

Die Benutzerin „sieht“ also nur die Shell und nicht das Betriebssystem. Das ist wichtig zu wissen, weil es auf ein und demselben System verschiedene Shells gibt, die unterschiedliche Funktionen bereitstellen. Man kann sie sich im allgemeinen aussuchen, je nachdem, welche Shell dem eigenen Arbeitsstil am meisten entgegenkommt.

Die Shell meldet mit dem *Prompt*, dass sie für Benutzereingaben bereit ist. Typischerweise ist das Prompt-Zeichen ein `$`, ein `>` oder ein `%`. Auch Rechnernamen und Verzeichnisnamen können angezeigt werden (Frau kann das einstellen.).

```
[trish@lillegroenn skript]$
```

Üblicherweise tippt frau einen Befehl ein und bestätigt die Eingabe mit der Return-Taste. Die Shell bereitet die Eingabe auf und sucht dann nach einem entsprechenden Programm, ruft es auf und wartet auf seine Beendigung. Während dieser Zeit übernimmt das aufgerufene Programm die Kontrolle und kommuniziert gegebenenfalls mit der Benutzerin (z. B. durch eine Datenausgabe). Nach der Beendigung des Programms wird dies der Shell gemeldet, die nun wieder Eingaben annehmen kann.

Unter Linux wird meist die `bash` („Bourne Again Shell“), eine Neu- und Weiterentwicklung der „Bourne Shell“ `sh`, verwendet, die wir auch im Kurs benutzen werden. Viele Konzepte lassen sich auch auf andere Shells übertragen, aber eigentlich gibt es nur eine Eigenschaft, bei der sich alle Shells auf der Grundlage der Linux-Dateisysteme einig sind: Auf Unix-Betriebssystemen muss frau auf Groß- und Kleinschreibung achten!

Aufgabe:

In der Datei `/etc/passwd` steht auf jeder Zeile hinter dem letzten Doppelpunkt, welche Shell beim Login für die jeweilige Benutzerin gestartet wird. Was außer `/bin/bash` ist da zu finden?

Zum Anzeigen der Datei kannst Du das Programm `more` verwenden.

1.2.3 Die Benutzerinnen

Bevor eine Benutzerin aber überhaupt mit einer Shell arbeiten kann, muss sie einen Zugang dazu bekommen. Zu diesem Zweck bekommt sie von der Systemadministratorin einen Benutzernamen verpasst, der auf dem System eingetragen wird. Diesem Name ist eindeutig eine UserID (*UID*) zugeordnet. Da die Benutzernamen nur bedingt ein Geheimnis sind, wird der Zugang zudem mit einem Passwort gesichert.

Beiden Angaben macht frau am sogenannten *Login-Prompt* des Rechners:

Linux Mandrake release 7.0 (Air)
Kernel 2.2.18 on an i686
lillegroenn login:

Nach dem Einloggen (UserID und Passwort eingeben; dabei wird das Passwort nicht angezeigt) öffnet das System die *Login-Shell*.

Aufgabe:

Finde Deine UserID mit dem Kommando `id` heraus!

Aufgabe:

Wo steht die UserID in der `/etc/passwd`?

Jede neue Benutzerin wird beim Anlegen zugleich in eine oder mehrere *Gruppen* eingliedert. Jeder Gruppe ist eine *GroupID* zugeordnet.

Aufgabe:

Finde mit dem Kommando `groups` heraus, welchen Gruppen Du angehörst!

Damit in einem Multiuser/Multitasking-Betriebssystem nicht das große Chaos ausbricht, muss genau geregelt sein, welche Benutzerin was darf. Daher bekommt jede Datei und jedes Verzeichnis einen Satz von Zugriffsrechten (*permissions*) zugeteilt. Diese regeln, welche Benutzerin welche Aktionen mit der jeweiligen Datei ausführen darf. Mögliche Aktionen sind: Lesen (read), Schreiben (write) und Ausführen (execute).

Diese Rechte können getrennt vergeben werden für

- die Besitzerin einer Datei/eines Verzeichnisses,
- die Mitglieder der Gruppe, der sie angehört, und
- alle anderen Benutzerinnen.

Eine Ausnahme bilden lediglich User mit der ID 0. In der Regel ist das nur eine Benutzerin namens *root*, hinter der sich die Systemadministratorin versteckt, und die alles darf.

Aufgabe:

Welchen Gruppen gehört *root* an?

Um Unberechtigten nicht leichtfertig den Zugriff zum Rechner zu ermöglichen, darf frau nicht vergessen, sich beim System abzumelden, wenn sie mit der Arbeit fertig ist. Dazu dient das Kommando

```
[trish@lillegroenn skript]$ exit
```

oder die Tastenkombination *Strg+D*, wenn frau von einer *virtuellen Konsole* aus arbeitet oder von einer Kommandozeile aus auf einem entfernten Rechner eingeloggt ist. Grafische Benutzeroberflächen bieten in der Regel einen entsprechenden Menüeintrag.

Achtung: Wenn frau die grafische Benutzeroberfläche *nach* dem Einloggen selbst gestartet hat, muss sie sich nach dem Beenden derselben extra ausloggen.

Frau ist erst dann abgemeldet, wenn sich das System wieder mit dem Login-Prompt meldet!

1.2.4 X-Server

Im Gegensatz zu anderen Betriebssystemen ist die grafische Benutzeroberfläche bei Linux u. a. Unixbetriebssystemen nicht ins Betriebssystem integriert. Auf diese Art und Weise ist es möglich, Linuxrechner komplett ohne den Overhead eines GUIs zu installieren und zu benutzen – besonders für Rechner sinnvoll, die als Server, Router oder Firewall ihren Dienst tun. Auch ältere i386er können so noch als Text-Terminal betrieben werden.

Um grafisch zu arbeiten, gibt es zwei Alternativen: die (auf speziell auf Nicht-Intel-Architekturen verbreitete, aber auch auf PCs laufende) *Framebuffer-Konsole* oder das *X-Window-System*, ein Client-Server-System, das älter als Linux ist und auf den meisten Unices eingesetzt wird. Unter Linux hat die Open-Source-Implementation des *XFree86*-Teams die größte Verbreitung.

Dahinter steckt folgendes Modell: Ein *X-Server* läuft auf einem Rechner und verwaltet die Grafikkhardware. Er ist quasi der „Treiber“ und muss in der Lage sein, die Grafikkartenchips anzusprechen. Bis XFree86 3.3.6 muss frau daher einen X-Server entsprechend des Chipsatzes (z. B. *XFree86_Mach64* oder (den für die meisten Chipsätze geeigneten, aber recht rudimentären *XFree86_VGA16*)) installiert haben, ab XFree86 4.0 gibt es nur noch ein *Binary* für alle Chipsätze; die hardwareabhängigen Teile sind in Module ausgelagert.

Aufgabe:

Finde die Liste der unterstützten Grafikkartenchipsätze auf der Homepage des XFree-Projekts, <http://www.xfree86.org/>!

Wenn ein grafisches Programm, ein *X-Client* sich auf einem Bildschirm darstellen will, fragt es bei einem beliebigen X-Server (es muss nicht der der lokalen Maschine sein) an, ob es das darf. Wenn die Benutzerin, mit deren Rechten der Client läuft, auf den X-Server zugreifen darf (weil es ohnehin ihr eigener X-Server ist oder weil sie oder ihr Rechner Zugriffsrechte auf einen entfernten X-Server hat), wird das grafische Programm auf dem jeweiligen X-Server dargestellt.

Auf diese Weise ist es sehr einfach, Unix-Maschinen aus der Ferne zu bedienen und zu warten; außerdem wird es möglich, Lasten zu verteilen: Die schmalbrüstige Workstation beschränkt sich darauf, die grafische Oberfläche eines Programms darzustellen, während die Berechnungen auf einem leistungsstärkeren Rechner ausgeführt werden.

Allerdings bringt diese Architektur auch Sicherheitsprobleme mit sich, die in „alten Unix-tagen“ keine Rolle gespielt haben. Frau sollte die Remote-Ausgabe grafischer Programme daher besser auf vertrauenswürdige LANs beschränken und für das Remote-Login auf jeden Fall *SecureShell* verwenden. (Das sogenannte *X-Forwarding* der SecureShell kann jedoch abgeschaltet werden, sodass es nicht mehr möglich ist, die Darstellung grafischer, auf dem entfernten Rechner laufender Programme dem lokalen X-Server zu überlassen.)

Aufgabe:

Logge Dich mit `ssh username@hostname` auf dem Rechner Deiner Nachbarin ein und starte dort das Programm `xeyes`. Auf welchem Bildschirm erscheint es? Warum?

Die Konfiguration des X-Servers erfolgt bei der Installation. Die meisten modernen Distributionen erkennen Grafikkarte und (wenn möglich) auch Monitor automatisch, allerdings schadet es nichts, diese Angaben, speziell die jeweiligen Handbücher zur Hardware dabei liegen zu haben, um nach Fragen zu Horizontal- und Vertikalfrequenz (des Monitors), zum Chipsatz der Grafikkarte und zur Auflösung Auskunft geben zu können.

Die Datei `XF86Config` (der Pfad kann je nach Distribution variieren, sollte laut *Filesystem Hierarchy Standard* aber `/etc/X11/XF86Config` sein) wird dabei geschrieben und kann auch später mit einem Texteditor oder einem distributionsabhängigen Tool wie `yast` (SuSE) bzw. `xConfigurator` (Red Hat, Mandrake) oder aber auch `XF86Setup` vom XFree-Team geändert werden. Auskunft über das vorgesehene Tool gibt in der Regel das Handbuch der Distribution.

Aufgabe:

Finde die X-Server-Konfigurationsdatei auf Deinem Rechner mit dem Kommando `locate`.

Aufgabe:

Schau mit `more` in die Datei hinein. Du solltest darin weitere Aufgabengebiete des X-Servers finden, die wir uns gemeinsam anschauen. Suche nach den Stichworten `FontPath`, `XkbLayout`, der Section `"Pointer"`!

1.2.5 Windowmanager

Um eines kümmert sich der X-Server jedoch herzlich wenig: Ob die verschiedenen X-Clients einheitliche Fensterrahmen und Buttons bekommen, an denen sie über den Desktop bewegt, vergrößert oder verkleinert werden können, ist ihm ebenso egal wie die Frage, ob die Anwenderin alle Fenster schlecht zugänglich übereinander oder übersichtlich verteilt vorfindet.

Sich darum zu kümmern, ist die Aufgabe eines ganz speziellen X-Clients, des Windowmanagers. Erst dessen Einsatz beim Dirigieren der Client-Fenster macht sinnvolles

Arbeiten auf der grafischen Oberfläche möglich. Durch Wahl des Windowmanagers entsprechend den eigenen Vorlieben und Bedürfnissen wird ein Unix-Desktop zur individuell angepassten Arbeitsoberfläche.

Die Auswahl reicht vom rudimentären *evilwm*, der für minimale Fensterdekorationen und Tastaturbedienbarkeit steht, über den Klassiker *fvwm2* (funktional, konfigurierbar, aber für heutige „Sehgewohnheiten“ eher mit altbackenem Outfit) bis hin zum schrillen, durch *Themes* anpassbaren *Enlightenment*.

Außer der beschriebenen Grundfunktionalität sorgen viele Windowmanager für Programmauswahl- und Aktionsmenüs, die gern auf Mausclicks auf den Desktop-Hintergrund (Unix-Mäuse haben drei Maustasten!) hin erscheinen, *virtuelle Desktops* sorgen bei Vielfenster-Benutzerinnen für Übersicht. Manche Windowmanager bieten sogar ein gewisses Session-Management. Natürlich unterscheiden sie sich auch nach der Art und dem Umfang ihrer Konfigurierbarkeit.

Aufgabe:

Stell Deinen Kommilitoninnen ein bis zwei Windowmanager kurz (ggf. mit Bild) vor!

1.2.6 Desktop-Umgebungen

So individuell sich die Arbeitsoberfläche durch Auswahl und Konfiguration des Windowmanagers gestalten lässt, er behebt ein altes Problem des Unix-Desktops nicht: die Individualität der Anwendungen. Während Windows-Benutzerinnen in der Regel davon ausgehen können, dass sie links oben in der Menüleiste das *Datei*-, rechts das *Hilfe*-Menü erwischen und dass gängige Aktionen anwendungsübergreifend über ein und dasselbe Tastenkürzel erreichbar sind, hatten es Unix-Benutzerinnen hier weniger gut. Die Freiheit, sich nicht dem Style-Guide eines Herstellers beugen zu müssen, führte hier dazu, dass jede/r Programmierer/in seine/ihre GUIs nach eigenem Gutdünken schrieb.

So fehlt klassischen X-Clients nicht nur ein einheitliches Aussehen, sondern auch die Fähigkeit des *Drag&Drop* über Applikationsgrenzen. (*Copy&Paste* geht in der Regel so: Markieren mit der linken Maustaste und Einfügen mit der mittleren.)

Diesem Missstand begegnen *Desktop-Umgebungen* (auch *Desktop-Environments* genannt) wie KDE oder GNOME.

Das sind Software-Suites, die zum einen einen (z. T. austauschbaren) Windowmanager, zum anderen konsistent gestaltete GUI-Applikationen mitbringen. (Das wird u. a. dadurch erreicht, dass Software, die für eine Desktop-Umgebung geschrieben wird, ein bestimmtes GUI-Toolkit benutzt: Qt für KDE und GTK für GNOME.)

Dazu kommen „Hilfsprogramme“ wie Soundserver, Protokolle und Bibliotheken für die Kommunikation zwischen den Applikationen, Themes u. a.

Auch wenn es zwischen Anhängern der einen und der anderen Umgebung oft fanatisch anmutende Grabenkämpfe gibt – die Entscheidung für oder gegen das eine oder das andere Environment sollte frau allein davon abhängig machen, was ihr persönlich besser zusagt.

Desktop-Umgebungen haben natürlich auch einen Nachteil: Die Ansprüche an die Hardware sind nicht zu vernachlässigen. Auf älteren Rechnern lässt sich mit einem schlanken Windowmanager oft produktiv grafisch arbeiten – die vielleicht moderner wirkende Oberfläche von KDE und GNOME zwingt ihn dagegen in die Knie.

Sofern die entsprechenden Bibliotheken installiert sind, können natürlich auch KDE-Applikationen unter GNOME oder einem Standalone-Windowmanager gestartet werden und umgekehrt. Allerdings lernt ein guter alter X-Client nicht allein dadurch Drag&Drop, dass er plötzlich unter KDE laufen darf.

Wie wähle ich meinen Windowmanager/meine Desktop-Umgebung? Loggt frau sich grafisch ein, wird anschließend der Windowmanager/die Desktop-Umgebung gestartet, der/die in der Datei `~.xsession` steht. Startet frau den X-Server mit dem Befehl `startx` „von Hand“, wird die Datei `~.xinitrc` zu Rate gezogen. Steht in dieser Datei beispielsweise der Befehl `startkde`, startet KDE.

In den Menüs des *Display-Managers*, der bei grafischem Login für den Benutzerinnendialog sorgt, aber auch in den „Startmenüs“ von Windowmanagern finden sich oft Einträge zum Wechseln in eine andere grafische Umgebung. Manche Distributionen wie Red Hat bringen extra Tools (`switchdesk`) mit, die bei der Festlegung behilflich sein können.

Aufgabe:

Überprüfe, ob der Poolrechner via Menü Alternativen bietet, und probiere sie durch!

Aufgabe:

Für Mutige etwas später im Kurs (bitte nicht „zwischenrein“): Sichere ggf. die entsprechende Konfigurationsdatei und versuche, einen alternativen Windowmanager zu starten!

1.2.7 Clients, Server und andere Programme

Wie wir schon beim X-Server und den X-Clients gesehen haben, ist die Client-Server-Architektur auf Unixbetriebssystemen nicht wegzudenken. Auch heutige Linux-Distributionen folgen dieser Tradition weiterhin, und so gehört schon eine Menge Arbeit hinzu, um ein Linuxsystem so hinzubiegen, dass es weitgehend serverlos ist.

Die meisten Server werden beim Booten des Rechners gestartet und laufen dann von den Usern weitgehend unbeachtet im Hintergrund vor sich hin. Selbst wenn ein Client ihre Dienste anfordert, geht das in vielen Fällen von den Usern unbemerkt vonstatten (auch wenn frau sich natürlich gezielt vergewissern kann, ob ein Dienst aktiv ist).

Solche Programme nennt frau *Daemonen* (von „Disk and Execution Monitor“); ihre Namen enden meist auf ein d. Sie verrichten die unterschiedlichsten Aufgaben: So sorgt der `kerneld` dafür, dass Kernelmodule geladen werden. (Er tanzt insofern aus der Reihe,

als dass es sich nicht um einen eigenständigen Prozess, sondern eine Kernelfunktionalität handelt.) Der `syslogd` protokolliert (loggt) die Ausgaben von Dienstprogrammen, sodass sie den Usern nicht auf die Nerven gehen, die Systemadministratorin aber bei Bedarf nachlesen kann, was so passiert ist, der `crond` führt von den Usern hinterlegte Aufgaben zu den angegebenen Zeiten aus...

Neben diesen „systemerhaltenden“ Daemonen lässt sich kaum ein Linuxsystem ohne Internetserver denken. Welche installiert sind und welche laufen, hängt jedoch von der Distribution und vom Installationsumfang ab.

Will frau dafür sorgen, dass sie sich remote auf einer Maschine einloggen kann, muss der SecureShell-Server `sshd` laufen. (Heutzutage, wo frau im Netz niemandem mehr trauen kann, sollte der `telnetd` zum unverschlüsselten Remote-Einloggen hingegen nicht mehr laufen, es sei denn, es besteht ein besonderer Grund dazu.)

Ein Mailserver (alias „`smtpd`“) ist eine gute Idee (und sei es nur deshalb, damit der Cron-Daemon seine Ausgaben an die lokalen User senden kann). Hier gibt es vom sehr kryptischen, aber weit verbreiteten Sendmail bis hin zum exotischen Qmail eine breite Auswahl (Postfix, Zmailer, smail, Exim, um die bekanntesten zu nennen). Als Webserver ist meist Apache dabei, oft läuft er auch bereits, um Zugriff auf ein lokales Hilfesystem zu gewähren.

Aufgabe:

Läuft auf Deinem Rechner ein Mailserver? Für Mail (`smtp`) ist ein sogenannter *wellknown port* reserviert, hinter dem der Mailserver lauschen müsste, sofern einer läuft.

Suche in der Datei `/etc/services` heraus, welche Portnummer das ist und verbinde Dich mit dem `telnet`-Client damit: `telnet localhost port-nummer`

Welcher Mailserver meldet sich?

Auch Nameserver (Bind), Newsserver, IRC-Server, Listserver, FTP-Server ... – Server für jeden erdenklichen Internetdienst gehören zur Standardausstattung vieler Distributionen, mit der zunehmenden Ausrichtung an Endusern allerdings nicht mehr bei allen. Wichtig zu erwähnen wäre noch der „Superserver“ `inetd`. Sofern er läuft, müssen nicht alle anderen Netzdienste ständig laufen – er startet sie, wenn ein Client ihn entsprechend auffordert und die Konfiguration es ihm gestattet.

All diese Server müssen natürlich konfiguriert werden, nur für sehr gängige Szenarien bringen die Distributionen bereits eine entsprechende Konfiguration mit. Meist passiert das über im Verzeichnis `/etc` liegende ASCII-Konfigurationsdateien. Eine gewisse Vereinfachung bieten distributionsabhängige Konfigurationstools (wie `yast2` bei SuSE) oder distributionsunabhängige Werkzeuge wie Webmin oder `linuxconf`. Allerdings setzen auch diese Tools ein gewisses Verständnis des Dienstes voraus, den es zu konfigurieren gilt, und decken oft nur triviale Szenarien ab.

Wo es Server gibt, braucht frau auch Clients – sie gibt es mittlerweile für fast alle Dienste sowohl als GUI- als auch als Kommandozeilenprogramme.

Aufgabe:

Finde drei Mailprogramme (Mail User Agents, MUAs) für Linux und prüfe, ob sie auf Deinem Rechner installiert sind!

Doch nicht alles auf einem Linuxsystem sind Clients und Server. Wer die Kommandozeile als mächtiges und schnelles Interaktionsmittel mit dem Betriebssystem kennengelernt hat, kann nicht auf eine Reihe von Werkzeugen verzichten, die vom GNU-Projekt (<http://www.gnu.org/>) stammen.

Warum GNU? Unix-Hacker und -Häcksen sind Spielkinder, die einem Sport besonders fröhnen: dem Spaß an *rekursiven Akronymen*, also selbstbezüglichen Abkürzungen. So steht GNU für „GNU is not UNIX“ und weist auf das ursprüngliche Ziel hin, ein freies unixoides Betriebssystem zu entwickeln, das nichts mit dem originalen AT&T Unix zu tun hat.

Die GNU-Leute (darunter der Begründer der Free Software Foundation (FSF), Richard Stallman) begannen mit den Tools, während der Betriebssystemkern des Projekts, GNU Hurd, bis heute noch nicht endusertauglich ist. Als Linus Torvalds Anfang der Neunziger an einem Betriebssystemkern schrieb, dem die Tools fehlten, lag es nahe, beides zu vereinen. Aus diesem Grund sehen sich die FSF-Leute bei der Bezeichnung Linux für ein Linuxsystem unterrepräsentiert und bestehen auf der Bezeichnung GNU/Linux, wie sie im Namen der Debian-Distribution (Debian GNU/Linux) zu sehen ist.

Alle GNU-Utilities unterstehen (wie auch der Linux-Kernel und ein Großteil sonstiger Open-Source-Software) der GNU General Public Licence GPL, die im Wesentlichen besagt, dass die Software im Quellcode veröffentlicht werden muss, es erlaubt ist, daraus zu kopieren, aber wenn dies geschieht, verlangt sie, dass der daraus abgeleitete Code ebenfalls der GPL unterstehen muss.

Sie implementieren die klassischen Unix-Kommandozeilentools zum Suchen, Manipulieren und Anzeigen von Dateien und Prozessen und gehen in ihrer Funktionalität meist über die der klassischen Unixtools hinaus. Eine der Stärken von Unix ist die, dass es eine große Anzahl einfacher, hochspezialisierter Werkzeuge gibt, die bei Bedarf baukastenartig zu komplexen, ein bestimmtes Problem lösenden Kommandozeilen zusammengebaut werden.

Dabei ist *einfach* wohl eher ein Understatement, da die meisten Werkzeuge durch eine oftmals sehr große Reihe von Optionen in ihrer Wirkung gesteuert werden können. Diese Optionen, auch *Flags* genannt, werden in der Regel mit einem Minuszeichen „-“ eingeleitet, bei GNU- oder *long* Optionen auch mit zwei Minuszeichen.

Dabei muss frau bei der Angabe von Optionen aufpassen, weil die Wirkung sich von der erwarteten unterscheiden kann, wenn sie mehrere Optionen miteinander kombiniert. Es sei zudem darauf hingewiesen, dass sich die möglichen Optionen zu einem Kommando von Unix-Derivat zu Unix-Derivat unterscheiden können. Gerade die GNU-Utilities kennen zudem eine Reihe von Optionen, die ihre Pendants auf anderen Unixsystemen nicht beherrschen.

Während Flags meist optional sind, benötigen die meisten dieser Kommandos Argumente (Ausnahmen sind Befehle wie `pwd` („print working directory“), `exit`...). So kann frau z. B. `mkdir` zum Erstellen eines Verzeichnisses nicht ohne Angabe eines Verzeichnisnamens aufrufen.

Aufgabe:

Was tun `pwd` und `exit`?

1.3 Bootvorgang

Nach so viel Theorie wenden wir uns jetzt den praktischeren Seiten zu. Das erste, was frau mit einem Rechner machen muss, um ihn nutzen zu können, ist ganz klar einschalten. Doch was passiert dann?

Wenn das BIOS seine Überprüfungen beendet hat, lädt es den Startcode des Betriebssystems. Auf Linuxrechnern – egal, ob nur Linux oder auch andere Betriebssysteme installiert sind – ist beim Booten von Festplatte an dieser Stelle in der Regel ein im Master-Boot-Record der Festplatte installierter *Bootmanager* gefragt, der es erlaubt, zwischen mehreren Operativsystemen zu wählen (das können auch verschiedene Linux-installationen sein). Er startet dann das passende OS, d. h., er lädt den Kernel.

Der unter Linux meist eingesetzte Bootmanager heißt *Lilo* („Linux Loader“). Es gibt aber mit *Grub* eine – zunehmend verwendete – Alternative.

Woher einen Kernel nehmen? Das Bootmanager-Dilemma: Wenn der Bootmanager den Kernel laden will, befindet sich noch kein Betriebssystem und damit auch keine Unterstützung für Dateisysteme im Hauptspeicher. Der Linuxkernel ist zwar, wie wir gesehen haben, nichts weiter als eine Binärdatei, aber das Wissen um `/boot/vmlinuz` o. ä. nutzt uns mangels Dateisystem zu diesem Zeitpunkt herzlich wenig. Der Bootloader muss also wissen, an welcher physikalischen Stelle auf der Festplatte sich der Kernel befindet.

Deshalb reicht es beim Lilo nicht, die Konfigurationsdatei `/etc/lilo.conf` zu editieren (per Hand oder per Installationstool), denn er kann beim Booten nicht in diese Datei schauen. Daher schreibt frau diese Daten mit dem Kommando `/sbin/lilo` in den Bootsektor, anders ausgedrückt wird der Bootmanager damit überhaupt erst installiert. Es gilt also zum einen zwischen dem Bootloader Lilo und dem Kommando `lilo` zu unterscheiden, dass den Bootmanager erst schreibt. Zum anderen reicht es nicht, einen Kernel zu kompilieren. Damit er gestartet werden kann, muss er in die Lilo-Konfigurationsdatei als neue Bootmöglichkeit eingetragen und anschließend `/sbin/lilo` aufgerufen werden. Solange frau noch nicht sicher ist, dass der neue Kernel auch ordentlich bootet und funktioniert, sollte sie den Eintrag für den bislang geladenen Kernel noch nicht entfernen, sondern ihn als alternative Bootmöglichkeit anbieten.

Aufgabe:

Schau in die `/etc/lilo.conf` und versuche herauszufinden, welche Bootmöglichkeiten Dein Rechner bietet.

Hat es der Bootloader geschafft, den Kernel zu finden und damit begonnen, ihn in den Hauptspeicher zu laden, ist seine Aufgabe erledigt. Das Betriebssystem ist nun soweit im Speicher, dass es sich und seine Treiber „an den Haaren herbeiziehen“ und die Prozesstabelle aufbauen kann.

Allerdings kann frau mit dem Rechner zu diesem Zeitpunkt immer noch nichts anfangen. Zunächst muss ein Prozess Nummer Eins namens `init` noch eine Menge Arbeit verrichten. Er hat momentan nur die Dateien und Programme zur Verfügung, die auf der *Root-Partition* zu finden sind. Das sollten mindestens die Verzeichnisse `/sbin` (Programme, die i. A. nur von der Superuserin benötigt werden), `/lib` (Bibliotheken, die von den Programmen in `/sbin` und `/bin` geladen werden), `/etc` (Konfigurationsdateien) und `/bin` (Programme, die auch für normale User von Interesse sein können) sein.

Welche Aufgaben anstehen, findet `init` in der `/etc/inittab` beschrieben. Zunächst muss er wissen, in welchen Betriebszustand, in welchen *Runlevel*, er schalten muss. Wenn der Kernel beim Laden keine andere Option per „Kommandozeilenparameter“ mitgeteilt bekam, ist das der *Default-Runlevel*, der in der `inittab` niedergelegt ist.

Runlevel: Aus einem Unixrechner kann ohne Weiteres zur Laufzeit aus einer unverbundenen Workstation ein Internetserver werden. Einzige Voraussetzung ist, dass entsprechende Betriebszustände definiert sind. Wie diese Runlevel aussehen und welche Nummern sie tragen, ist von Distribution zu Distribution verschieden. Gemeinsam sind ihnen lediglich die Runlevel 0 (*halt*), 6 (*reboot*) und 1 (*Single-User-Mode*: Wartungsmodus, in den die Superuserin zum Reparieren gehen kann. Hier gibt es (meist) kein Netzwerk, keine Dienste und nur das, was auf dem Root-Filesystem zu finden ist.). Der Rechner lässt sich also zum Rebooten bringen, indem `root` einfach den Runlevel wechselt: `telinit 6`. Unter den Runleveln 2–5 gibt es meist einen, der den X-Server startet; ansonsten ist ihnen gemeinsam, dass sie Multiuser-Betrieb ermöglichen. Immerhin dokumentieren alle Distributoren in der `/etc/inittab`, welche groben Eigenschaften die verschiedenen Betriebszustände haben.

Traditionell gibt es unter Unix zwei *Init-Systeme*: *Simple Init* und *System-V-Init*. Da außer Exoten wie Slackware oder EasyLinux kaum eine Distribution auf ersteres setzt, beschäftigen wir uns hier nur mit letzterem.

In einem Verzeichnis namens `init.d` liegen verschiedene *Start-Stopp-Skripte*, die die Konfiguration des Netzwerks und verschiedener Dienste vornehmen: pro Dienst ein Skript. Ruft frau sie mit dem Parameter `start` auf, wird der entsprechende Service gestartet, mit dem Parameter `stop` gestoppt. Auch andere Parameter wie `restart` sind möglich, spielen beim Booten und Herunterfahren allerdings keine Rolle.

Fortsetzung folgt ...

... Fortsetzung

Beim Sys-V-Init gibt es zudem Verzeichnisse namens *rcRunlevel.d*. Dort liegen Verweise (*Links*) auf besagte Init-Skripte. Beginnt der Name des Links mit einem S, wird das Skript beim Wechsel in dieses Runlevel (z. B. beim Hochfahren des Rechners) mit dem Parameter **start** aufgerufen. Ist der erste Buchstabe ein K (wie „kill“), ruft **init** es beim Wechsel in einen anderen Runlevel (beispielsweise 1 oder 6 zum Herunterfahren) mit dem Parameter **stop** auf.

Die verschiedenen Skripte werden jeweils in der Reihenfolge der auf den ersten Buchstaben folgenden Zahlen aufgerufen.

Allerdings gibt es Distributoren wie SuSE, die dieses einfache Konzept gern etwas verkomplizieren. Hier reicht es nicht, dass Startlinks vorhanden sind – um einen Dienst zu starten, muss zusätzlich auch noch eine Variable in der Datei */etc/rc.config* gesetzt werden.

Und das ist noch nicht alles an Distributorenschikane: Selbst wenn SuSE im Zuge der Umsetzung der *Linux Standard Base* inzwischen darauf verzichtet, die Initskripte nach */sbin* statt */etc* zu verlegen, ist an dieser Stelle noch keine Einheitlichkeit eingekehrt: Manche Distributoren legen sie nach */etc*, manche nach */etc/rc.d* etc.

Aufgabe:

Finde in der */etc/inittab* heraus, in welchen Runlevel Dein Rechner von sich aus startet. Welche Zeile könnte das sein?

Aufgabe:

Wo liegen die Initskripte und die jeweiligen Links auf Deinem Rechner?

Je nach Runlevel startet **init** nun verschiedene Prozesse: Er sieht in der Datei */etc/fstab*, welche Partitionen er außer der Root-Partition noch einbinden, sprich: *mounten*, muss. Bevor er das tut, überprüft er das darauf enthaltene Dateisystem (so möglich) auf Konsistenz und repariert es ggf.

Da das beim *ext2fs*-Dateisystem sehr lange dauern kann, wird ein solcher *Filesystemcheck* mit dem Programm **fsck** nur dann durchgeführt, wenn eine bestimmte (mit dem Befehl **tune2fs** einstellbare) Anzahl Bootvorgänge überschritten wurde oder wenn der Rechner nicht ordnungsgemäß heruntergefahren wurde (vgl. Abschnitt 1.4 ab Seite 19).

Auch das Root-Filesystem wird geprüft. Damit bis zu diesem Zeitpunkt nicht versehentlich etwas auf ein möglicherweise inkonsistentes Filesystem geschrieben wird, ist es bis nach der Überprüfung nur lesbar (*read-only*) gemountet und wird anschließend *read-write* remountet.

Bei den meisten Linux-Distributionen sorgt das (distributionsabhängige) Boot-Skript *rc.boot* dafür, dass ein benutzbares System hochfährt, auf dem allerdings noch kaum ein Dienst läuft.

Dann werden nacheinander die Dienste gestartet, die für das jeweilige Runlevel vorgesehen sind; zum Schluss kommt oft noch ein `rc.local`-Skript zum Tragen, in dem die lokale Systemadministratorin spezielle Dinge konfiguriert, die beim Booten der Maschine eingestellt werden sollen.

Wenn all das erledigt (die entsprechenden Meldungen können beim Booten in der Konsole normalerweise beobachtet werden), werden auf den *virtuellen Konsolen* `gettys` gestartet. Diese starten das `login`-Programm, auf das frau einen *Login-Prompt* bekommt, an dem sie Username und Passwort eingibt. Auf einer der (mit der Tastenkombination `Strg+Fx` bzw. von der grafischen Oberfläche `Alt+Strg+Fx` zu erreichenden virtuellen Konsolen wird in einem Runlevel oft der *Display-Manager* einer grafischen Oberfläche gestartet (`xdm` bzw. sein KDE-Pendant `kdm` oder das entsprechende GNOME-Programm `gdm`), sodass frau sich gleich grafisch einloggen kann. In Runleveln, in denen das nicht vorgesehen ist, startet sie die grafische Oberfläche mit `startx`.

Aufgabe:

Finde heraus, welche Init-Skripte im Default-Runlevel Deines Rechners beim Booten *nicht* gestartet werden.

Aufgabe:

Sieh Dir das Boot-Protokoll nachträglich mit dem Befehl `dmesg` an! Was ist dabei passiert?

1.4 Der Ausschaltknopf

Bei einem Multiuser-Multitasking-Betriebssystem ist der Ausschaltknopf als Allheilmittel für Rechnerprobleme nicht so recht am Platz, denn was kann eine unschuldig remote eingeloggte, Mail-lesende Benutzerin dafür, dass bei mir gerade die Textverarbeitung oder der lokale X-Server klemmt?

Aber nicht nur auf eventuelle andere Nutzerinnen gilt es Rücksicht zu nehmen, sondern vor allem auch auf Prozesse, die im Hintergrund den verschiedensten Aufgaben nachgehen. Einfach ausschalten lässt sie möglicherweise mit gerade zum Schreiben geöffneten Dateien einen gewaltsamen Tod sterben – die Folge sind Inkonsistenzen im Dateisystem und Datenverlust.

Ausschalten sollte daher der Notfall bleiben – im Normalfall fährt frau ihren Rechner geordnet runter. Dabei bekommen alle Prozesse ein Signal von `init` zugesandt, dass sie schleunigst mit den aktuellen Schreiboperationen fertig werden und sich beenden sollen. Herunterfahren heißt dabei nichts anderes, als in den Runlevel 0 oder 6 wechseln. Außer mit `telinit` geht das auch mit den Befehlen `shutdown`, `reboot` und `halt`.

Alle drei sind in der Regel nur der Superuserin zugänglich. `halt` fährt den Rechner ohne Umschweife herunter; wenn der Rechner steht, ist es Zeit, den Ausschaltknopf zu betätigen, wenn das Motherboard nicht selbst dafür sorgt.

`reboot` wirkt wie ein (sauberer) Warmstart, und tatsächlich ist auf vielen Linuxrechnern auch der „Affengriff“ `Strg+Alt+Del` mit diesem Befehl belegt und auch nichtprivilegierten Usern zugänglich. Ob dem so ist, wird in der `/etc/inittab` festgelegt. Sicher ausschalten kann frau den Rechner dann, wenn er beginnt, wieder hochzufahren.

Wer seinen Rechner nicht alleine nutzt, sollte sich des Befehls `shutdown` zum Herunterfahren oder Rebooten bedienen. Er ermöglicht es, eine Zeitspanne bis zum tatsächlichen Herunterfahren anzugeben, sodass den Mitbenutzerinnen Zeit bleibt, ihre Arbeit schnell zu beenden. Sie werden automatisch durch eine *Broadcastmessage* davon informiert, dass der Rechner in `x` Sekunden vom Netz geht und herunterfährt.

So rebootet

```
shutdown -r now "Maschine rebootet jetzt"
```

den Rechner augenblicklich. Alle eingeloggten User erhalten die Nachricht *Maschine rebootet jetzt* angezeigt. `shutdown -h -t 600` fährt den Rechner in 10 Minuten herunter.

Aufgabe:

Wie würdest Du lediglich eine Warnung an alle eingeloggten User schicken, ohne den Countdown tatsächlich einzuleiten? Nimm die Manpage zu `shutdown` zu Hilfe!

1.5 Hilfe zur Selbsthilfe

In der `bash` (vgl. Kapitel 1.2.2, Seite 7) kann sich frau durch doppeltes Drucken der `Tab`-Taste alle Kommandos anzeigen lassen, die im Suchpfad enthalten sind. Das sind bei einer durchschnittlichen Linux-Distribution ohne Weiteres 1500–2500 Stück (je nach Installationsumfang sogar mehr), wovon etwa 400 grundsätzlich auf den meisten Unixsystemen zu finden sind.

Dass sich all die kaum jemand merken kann – erst recht nicht mit allen Optionen und Spitzfindigkeiten in der Bedienung – liegt auf der Hand. Ohne griffbereite Dokumentation geht es da nicht, und die sollte möglichst direkt auf dem Rechner zugänglich sein.

1.5.1 Selbstauskunft

Ganz allgemein kann frau im Verzeichnis `/usr/doc` auf der Suche nach (in der Regel englischsprachiger) Dokumentation zu einzelnen Programmpaketen fündig werden. Was da liegt, variiert von Rechner zu Rechner.

man Eine sichere Bank, was Informationen zu Kommandozeilenbefehlen, C-Funktionen und Konfigurationsdateien anbelangt, sind die sogenannten *Manpages* (Kurzform für „manual pages“). Diese Online-Hilfeseiten erklären in einer leider meist sehr abstrakten Form die Syntax und Funktionalität einer Datei oder Funktion. Zum Anzeigen der in `man`-Unterverzeichnissen (z. B. `/usr/man` oder `/usr/local/man`) sortiert abgelegten ASCII-Dateien im *roff*-Format nutzt frau den Befehl `man`:

```
man fstab
man man
man -k password
```

Zur Anzeige wird ein sogenanntes *Pager-Programm* wie das Kommando `more` benutzt, das wir bereits in Übungen verwendet haben. Unter Linux kommt allerdings meist sein mächtigeres Pendant `less` zum Einsatz, das sich beim Erreichen der letzten anzuzeigenden Zeile nicht selbst beendet, sondern mit einem `q` („quit“) dazu überredet werden will.

Typisch für Manpages ist ihre Aufteilung in Abschnitte wie *NAME*, *SYNOPSIS* oder *DESCRIPTION*, um einige zu nennen. Der zweite enthält eine Kurzfassung aller möglichen Parameter in einer (nicht immer strikt verwendeten) *Backus-Naur*-Schreibweise. In eckigen Klammern stehen dabei Parameter und Optionen, die je nach Bedarf weggelassen werden können.

Aufgabe:

Was tut `man -k`? Wofür könnte das `k` stehen?

Aufgabe:

Was sind `cat pages`? Nutze die Manpage zu `man`!

Oben links und rechts in einer Manpage steht nicht nur das Stichwort, zu dem die Hilfe gehört, sondern auch eine Zahl in runden Klammern. Damit hat es folgende Bewandnis: Wenn es einen Kommandozeilenbefehl und eine gleichnamige Datei gibt, soll es auch zwei getrennte Manpages geben. Dazu wird jede Manpage einer *Sektion* zugeordnet: Nr. 1 enthält Hilfen zu Benutzerkommandos, während Nr. 5 das Format der gleichnamigen Datei beschreibt. Die Sektionsziffer taucht immer wieder auf, z. B. auch im Namen des Verzeichnisses, in dem die unformatierte Manpage liegt.

Aufgabe:

`man man` bzw. `man 1 man` beschreibt nicht, welche Sektionen es gibt. Finde mit `man -k man` oder `apropos man` heraus, in welchen anderen Sektionen `man` auftaucht und lies nach, welche Sektionen es gibt.

Es gibt natürlich noch weitere Hilfsfunktionen, die sich aber von System zu System unterscheiden, während die Manpages auf fast jedem System vorhanden sind.

Die Benutzung der Manpages ist gewöhnungsbedürftig, aber wie so oft geht es mit zunehmender Erfahrung leichter. Auch wenn GUI-Programme selten eine Manpage haben, ist doch fast alles, was frau auf der Kommandozeile benutzt, in Manpages nachzulesen.

info Manpages haben auch Nachteile – sie lassen sich zum Beispiel nicht verlinken. Das GNU-Projekt hat deshalb ein eigenes Onlinehilfe-System namens *info* geschrieben.

Ruft frau das gleichnamige Kommando ohne weitere Parameter auf, bekommt sie eine Übersicht über alle auf dem System installierten Info-Seiten.

Mit einem Sternchen * ein- und zwei Doppelpunkten ausgeleitete Einträge kann frau mit den Pfeiltasten anwählen und mit *Return* auf die darin verlinkte Seite springen. Damit und mit den Buchstabenkommandos p („previous page“), n („next page“), u („page up“), d („page down“) und q („quit“) kommt frau schon sehr weit.

Aufgabe:

Vergleiche Man- und Infopage zum Kommando `groups`!

2 2. Tag

2.1 Dateisystem

Das Dateisystem ist die Basis eines jeden Betriebssystems.

Es strukturiert einen Massenspeicher wie Festplatte, Diskette oder was auch immer so, dass es auf möglichst effiziente Weise seine Daten darauf wieder findet.

Dazu wird der Massenspeicher zunächst partitioniert, also mit einem oder mehreren Bereichen belegt, und dann die Blöcke der Partition in Verwaltungs- und Dateiabschnitte unterteilt.

Der Massenspeicher stellt sich dem Betriebssystem als eine lineare Kette von normalerweise 1 kB, also 1024 Bit, großen Speicherblöcken dar. Die Dateisysteme organisieren diese Blöcke so, dass sie darauf ihre Daten finden können.

Dies geschieht normalerweise mit zwei Arten von Informationen:

- Verwaltungsinformationen zum Auffinden der Daten, den *Inodes* („Information Nodes“)
- den Daten selbst

Die *Inodes* sind die Dateiköpfe, in denen der Dateiname, die Dateiattribute, Größe und, falls die Datei größer ist als ein Datenblock, weitere Inode-Nummern gespeichert sind.

2.1.1 Unterstützte Vielfalt

Linux unterstützt eine ganze Reihe von Dateisystemen.

Hier eine Auswahl:

Auswahl von Linux unterstützter Dateisysteme	
minix	Minix (kaum noch verbreitet)
ext2	Standard-Dateisystem von Linux
ReiserFS	modernes Journaling-Dateisystem
iso9660	CD-ROM
<i>Fortsetzung folgt ...</i>	

... Fortsetzung	
msdos	DOS
vfat	Windows95
nfs	Network File System
ntfs	WindowsNT
swap	Swap-Partitionen oder -Dateien
SystemV	Verschiedene Unixe
proc	Prozessverwaltung

Minix Das erste Linux-Dateisystem war das von *Minix*, einem UNIX®-Version-7-Nachbau von Andrew Tanenbaum, einem Professor an der Freien Universität Amsterdam. Offiziell wird es auch heute noch von allen Distributionen unterstützt. Minix erlaubt allerdings nur eine Partitionsgröße von 64 MB.

Ext2 Das Standard-Dateisystem von Linux ist im Moment noch das *ext2* („second extended filesystem“). Seine wichtigsten Eigenschaften sind:

- Dateinamen können bis zu 255 Zeichen lang sein.
- Pfadnamen können bis zu 4092 Zeichen enthalten.
- Groß- und Kleinschreibung werden unterschieden.
- Alle Zeichen – bis auf den Slash / und die NULL – sind erlaubt (aber auf Sonderzeichen sollte frau besser verzichten). Es ist sogar möglich, einen aus Leerzeichen bestehenden Dateinamen zu bilden.

Das Ext2-Dateisystem bietet noch einen besonderen Service: Beim Mounten wird im Superblock der Partition ein sogenanntes Valid-Flag auf 0, und beim Unmounten wieder auf 1 zurückgesetzt. Beim Booten wird dieses Flag überprüft, und das Dateisystem gegebenenfalls gecheckt. Außerdem wird bei jedem Mounten im Superblock ein Zähler hochgesetzt, der es ermöglicht, nach einer bestimmten, durch `tune2fs` einstellbaren Anzahl von Mounts das Dateisystem zu überprüfen, auch wenn das Valid-Flag korrekt gesetzt ist.

Beide Informationen werden vom Kernel beim Booten ausgewertet und führen gegebenenfalls zu den Meldungen

```
/dev/hda1: was not cleanly unmounted, check forced, oder  
/dev/hda1 has reached maximal mount count, check forced, oder  
/dev/hda1 has gone too long without beeing checked, check forced
```

Reiser-Dateisystem Seit dem Kernel 2.4 wird das *ReiserFS*, das Reiser-Dateisystem, unterstützt. Es ist ein Journaling Filesystem und zeichnet sich darüber hinaus dadurch aus, dass es die Daten in einem binären Baum verwaltet, was vor allem bei vielen kleinen Dateien einen enormen Performance-Gewinn gibt, aber auch den Plattenplatz optimal ausnutzt.

Journaling Filesystem: Stürzt der Rechner während einer Schreiboperation ab, kommt es zu Inkonsistenzen im Dateisystem. Verwaltungs- und Dateneinträge stimmen nicht mehr überein – die Einträge müssen geprüft und bereinigt werden.

Bei großen Platten dauert diese Überprüfung sehr lange, was vor allem bei Server-Systemen äußerst störend ist. Daher gibt es sogenannte *Journaling Filesysteme*. Diese schreiben sämtliche Dateioperationen mit. Stürzt das System ab, müssen nur die Operationen nachgeprüft werden, die noch nicht beendet waren, was natürlich viel schneller geht.

Virtuelles Dateisystem proc *proc* ist ein Dateisystem, in dem Informationen über das System abgerufen oder Kernel-Parameter verändert werden können.

Dieses Dateisystem beinhaltet keine realen Dateien, die auf der Festplatte Speicher wegnehmen, daher ist es auch nicht schlimm, wenn die Datei `/etc/kcore` riesig ist. Ihre Größe zeigt lediglich die Größe des Arbeitsspeichers an, sie selbst beansprucht aber keinen Platz. *root* darf den Inhalt des Arbeitsspeichers auch lesen.

Aufgabe:

Wie groß ist der Arbeitsspeicher Deines Systems?

Aufgabe:

Schau zuhause in Deinen Arbeitsspeicher rein. Aber Vorsicht, *Pager* wie `more` oder `less` bekommen Probleme mit den nicht druckbaren Zeichen, also lieber vorher mit `strings` filtern:

```
cat /proc/kcore | strings | less
```

2.1.2 Mounten

Jede Partition, die Linux verwenden soll, muss gemountet werden. Das kann fest verdrahtet in der Datei `/etc/fstab` festgelegt oder manuell per Kommandozeile nachgeholt werden.

In der Konfigurationsdatei `/etc/fstab` steht, welche permanenten Dateisysteme existieren, und wie sie gemountet werden sollen.

Sie enthält die folgenden Informationen:

- **Device**
die Gerätedatei des Massenspeichers (z. B. `/dev/fd0` für's Diskettenlaufwerk oder `/dev/hda1` für die erste Partition der ersten IDE-Platte)
- **Mountpoint**
der Mountpoint, also das Verzeichnis, in dem nach dem Mounten die Daten zu

finden sein werden

- **Type**
der Dateisystemtyp (z. B. `ext2` oder `reiserfs`)
- **Options**
Mit den Optionen kann der Mount-Vorgang gesteuert werden (Auswahl):
 - `defaults` Voreinstellungen (`rw`, `suid`, `auto`, `nouser...`)
 - `noauto` Kein automatisches Mounten beim Booten
 - `user` Device darf von normalen Nutzern gemountet werden
 - `ro`, `rw` read only, read write
 - `exec` Ausführung von Binaries gestattet
 - `sync` Ungepuffertes Schreiben
- **Dump**
Gilt momentan nur für `ext2` und besagt, ob das Dateisystem durch den Befehl `dump` gesichert werden soll.
- **Check**
Gibt an, ob das Dateisystem vor dem Mounten überprüfen werden soll. Beim Root-Dateisystem sollte hier eine 1 stehen und bei allen anderen entweder eine 0 (keine Prüfung) oder eine 2. Dateisysteme mit gleicher Nummer werden parallel überprüft, das Root-Dateisystem sollte immer allein und als erstes getestet werden.

Wenn in der `/etc/fstab` Folgendes zum Diskettenlaufwerk steht...

```
/dev/fd0 /floppy vfat noauto,user 0 0
```

..., dann kann es jede Benutzerin durch den Befehl `mount /floppy` mounten bzw. mit `umount /floppy` wieder unmounten.

Fehlt diese Zeile, darf nur `root` das Diskettenlaufwerk mounten und muss alle Optionen extra mitgeben.

`mount` kennt u. a. folgende Flags:

- `-t` Dateisystemtyp
- `-o` Optionen (Es sind die gleichen möglich wie in der `/etc/fstab`)

```
root@ratschni:/ # mount /dev/fd0 /floppy -t vfat
```

Welche Dateisysteme mit welchen Parametern aktuell gemountet sind, steht in der Datei `/etc/mtab`, die sich bei ihrer Ausgabe genau an die Syntax der `/etc/fstab` hält. Außerdem liefert der Befehl `mount` Informationen über gemountete Systeme, deren Mountpoints und Mountparameter.

Die wichtige Information, wieviel Platz sich aktuell auf einer Partition befindet, liefert `df` („disk free“).

Aufgabe:

Schau Dir die Datei `/etc/fstab` an und ermittle, welche Arten von Dateisystemen im Rechenzentrum verwendet werden und was damit gemacht werden kann.

Aufgabe:

Wie voll sind die Platten momentan?

2.1.3 Dateien

Linux kennt deutlich mehr Dateiformate als z. B. Windows und behandelt nahezu alles als Datei. Dem liegt der Gedanke zugrunde, dass gleiche Mechanismen wie z. B. Lesen und Schreiben auf ganz unterschiedliche Dinge angewendet werden können.

- *Reguläre Dateien:*
 - ASCII-Dateien*
von Menschen lesbare Dateien: z. B. Texte, Quellcode von Programmen, ...
 - Binär-Dateien*
maschinenlesbare Dateien: z. B. Programme
- *Geräte-Dateien*
In Linux wird auch Hardware als Datei verwaltet. Dabei wird unterschieden zwischen:
 - *blockorientierten Gerätedateien*, dazu zählen alle Massenspeicher wie z. B. das Diskettenlaufwerk (`/dev/fd0`) oder eine IDE-Festplatte (`/dev/hda1-x`), und
 - *zeichenorientierten Gerätedateien* wie der Maus (`/dev/mouse` bzw. `/dev/cua0-x` bei einer seriellen oder `/dev/psaux` bei einer PS/2-Maus) oder dem Drucker (`/dev/lp1`)
- *Links*
Verweise auf andere, schon vorhandene Dateien. Sogenannte *Hard Links* sind ein weiterer Dateiname für die ursprünglichen Datei. Die Datei selbst wird erst physikalisch gelöscht, wenn sowohl die ursprüngliche Datei als auch alle auf sie verweisenden *Hard Links* gelöscht sind. Anders die *Soft Links*. Sie sind lediglich Verweise auf den ursprünglichen Dateieintrag und gehen ins Leere, wenn die Datei, auf die sie verweisen, gelöscht wurde.
- *Verzeichnisse*
Auch Verzeichnisse werden wie Dateien behandelt. Das aktuelle Verzeichnis ist

z. B. die Datei namens `.` (Punkt), das darüberliegende Verzeichnis heißt `..` (Punkt-punkt).

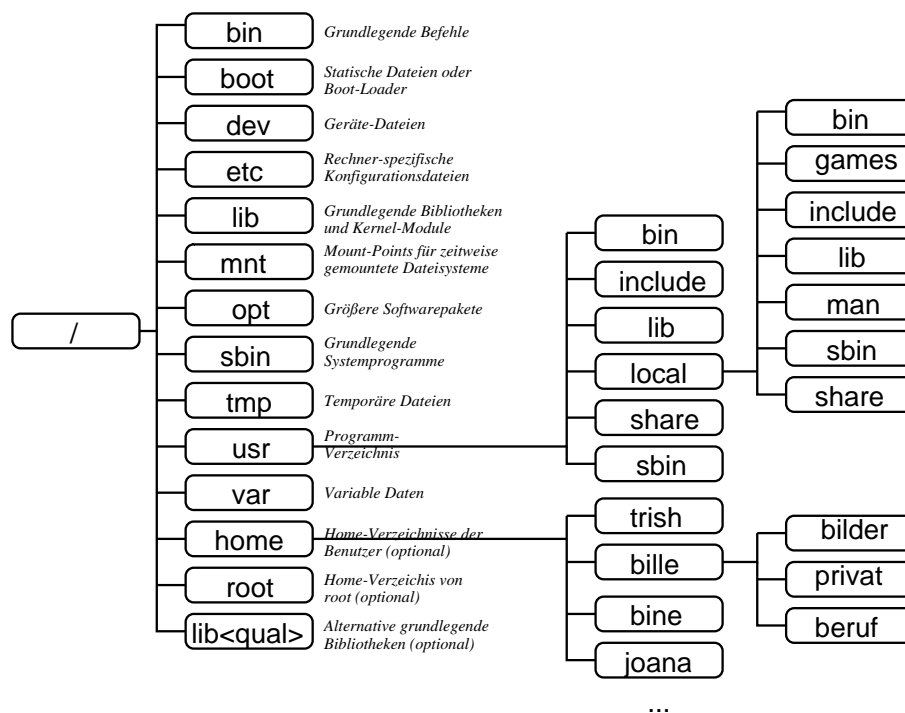
- *FIFOs*
(„First In First Out“) oder *Named Pipes* sind Dateien, die es ermöglichen, dass zwei Prozesse untereinander durch Pipes (vergl. Abschnitt 3.1.1 auf Seite 47) miteinander kommunizieren können.
- *Sockets*
über die Rechner untereinander kommunizieren können.

2.1.4 Verzeichnisbaum

Alle Dateien zusammen ergeben den Verzeichnisbaum mit dem untersten Verzeichnis namens `/`. Die Dateien sind darin nach ihren Aufgaben sortiert.

Um die Systeme untereinander wieder kompatibler zu machen, haben sich die großen Distributoren und andere Software-Hersteller zusammengeschlossen und entwickeln die LSB („Linux Standard Base“).

Dort ist unter anderem im FHS („Filesystem Hierarchy Standard“) festgelegt, wie der Verzeichnisbaum einer Linux-Installation aussehen sollte:



2.1.5 Bewegen im Verzeichnisbaum

Verzeichnis wechseln Um ein Verzeichnis zu wechseln verwendet frau den Befehl `cd`. Der Befehl alleine katapultiert sie in ihr Home-Verzeichnis. Normalerweise wird er aber

benutzt, um in ein bestimmtes Verzeichnis zu wechseln. Die Adresse dieses Verzeichnisses, also die Pfadangabe kann auf die beiden folgenden Weisen angegeben werden:

- *relativer Pfad* Die Angabe des Zielverzeichnisses erfolgt vom momentanen Standpunkt aus. Ein Verzeichnis nach oben wird durch zwei Punkte angegeben (`cd ..`) ein Unterverzeichnis wird einfach durch seinen Namen angegeben (`cd mail`). Die Angaben können durchaus auch verschachtelt werden, z.B. zu: (`cd ../../usr/src`).
- *absoluter Pfad* Dabei wird vom höchsten, also dem root-Verzeichnis ausgegangen: `cd /usr/src`. Den absoluten Pfad erkennt frau in einem führenden Slash (/).

Auflisten des Verzeichnissesinhalts Um sich den Inhalt eines Verzeichnisses anzusehen dient der Befehl `ls` (Liste). Er listet alle im aktuellen Verzeichnis vorhandenen Dateien auf. Wird `ls` mit einem Verzeichnis als Argument aufgerufen, so zeigt er den Inhalt dieses Verzeichnisses an.

Außerdem können noch Optionen mit dazugepackt werden, die die Art der Ausgabe steuern. Eine kleine Auswahl:

Auch versteckte Dateien (Punktdateien) auflisten	<code>ls -a</code> (all)
Dateirechte mit auflisten	<code>ls -l</code> (long)
Dateiattribute anzeigen (/ für Verzeichnis, * für ausführbar, @ für Link)	<code>ls -b</code>

2.1.6 Benutzerinnen

Linux ist ein Multiuserinnensystem, es ist darauf optimiert, in Mehrbenutzerinnen-Umgebungen zu laufen. Daher existieren auf einem Linuxrechner immer unterschiedliche Benutzerinnen, die mit den Dateien unterschiedlich verfahren können.

Das hat vor allem sicherheitstechnische Gründe. Da den Dateien Besitzerinnen zugeordnet sind, kann ein Bedienungsfehler einer Benutzerin nicht dazu führen, dass das ganze System nicht mehr funktioniert, da sie gar keinen Zugriff auf Systemdateien hat.

Die mächtigste Benutzerin im System ist *root*. Sie darf alles – und sollte wirklich nur Verwaltungstätigkeiten machen, weil sonst alle Sicherheitsvorteile flöten gehen.

Dann gibt es natürlich die verschiedenen Benutzerinnen, die am Rechner arbeiten. Darüber hinaus existiert noch eine Reihe virtueller Nutzerinnen, die auch wieder aus sicherheitstechnischen Gründen eingeführt wurden. Die LSB sieht die folgenden Benutzerinnen als Mindestbelegschaft vor:

- *root*
Systemverwalterin mit allen Rechten
- *bin*
Systemverwalterin mit eingeschränkten Rechten
- *daemon*
Unterprozessverwaltung

Darüber hinaus werden noch eine ganze Reihe weiterer Systembenutzerinnen für unterschiedliche Aufgaben vorgeschlagen.

2.1.7 Dateirechte

Bei Linux werden für alle Dateien Rechte vergeben, die regeln, wer was mit den Dateien anstellen darf. Die Rechte sind aufgeteilt nach verschiedenen Benutzerinnengruppen. Der Befehl, um sich die Informationen über die Dateien zu Gemüte zu führen, lautet:

```
bille@ratschni:/ > ls
```

Mit den Optionen `ls -al` werden die üblichen Dateirechte aller Dateien im Verzeichnis angezeigt. (Weitere Optionen siehe `man ls`.)

Ein Satz beispielhafter Dateirechte:

```
-rw-r--r--    1 bille  users  236472 Sep  2 21:42 /home/bille/inf/linux.tex
lrwxrwxrwx    1 bille  users    10 Sep  2 22:50 www -> /home/www/
prw-r--r--    1 bille  users     0 Sep  2 22:53 named_pipe
brw-rw----    1 root   disk    2,  0 Jan 19  2000 /dev/fd0
brw-rw----    1 root   disk    2,  1 Jan 19  2000 /dev/fd1
crw-rw----    1 root   lp      6,  0 Jan 19  2000 /dev/lp0
drwxr-xr-x   29 root   root    4096 Mär 15 21:47 home
drwxr-xr-x   21 root   root    4096 Mai 28 09:24 .
drwxr-xr-x   21 root   root    4096 Mai 28 09:24 ..
```

```
1         2 3         4         5 6 7 8 9
```

Die Dateirechte sind in neun Blöcke geteilt:

1. Dateityp und erlaubte Aktionen. Deren Aufteilung wird im folgenden Abschnitt beschrieben.
2. Anzahl der Dateinamen (ursprünglicher und Hard Links),
3. Besitzerin,
4. Gruppe,
5. Dateigröße bei regulären Dateien bzw. Minor- und Major-Number bei Gerätedateien. Die Minor-Number definiert die Art des Gerätes, (z. B. 2 für das Diskettenlaufwerk und 6 für den Drucker), und die Minor Number besagt, um das wievielte Gerät es sich handelt,
6. Monat der letzten Änderung,
7. Tag der letzten Änderung,
8. Uhrzeit bzw. Jahr (wenn die Datei älter als ein Jahr ist) der letzten Änderung,

9. Dateiname.

Der erste Block der Dateirechte besteht aus 10 Spalten mit je einem Zeichen. Die erste Spalte bezeichnet den Dateityp, die nächsten 3 Spalten zeigen die Rechte der Dateibesitzerin, die 3 weiteren die Rechte der Gruppe, und die 3 letzten die Rechte aller anderen.

- 1 Spalte Dateityp
 - - Reguläre Datei
 - d Verzeichnis
 - l Link
 - c Zeichenorientiertes Gerät
 - b Blockorientiertes Gerät
 - s Socket
 - p Named Pipe
- 3 Spalten Rechte der Besitzerin (user)
 - r lesen
 - w schreiben, löschen
 - x ausführen (bei Programmen), betreten (bei Verzeichnissen)
 - s Programme werden so gestartet, als gehörten sie der Besitzerin
- 3 Spalten Rechte der Gruppe (group)
Die gleichen Rechte wie bei user, außer
 - s
 - * Verzeichnisse: Egal wer eine Datei in ein solches Verzeichnis schreibt, es gehört der Gruppe, die auch das Verzeichnis besitzt.
 - * Dateien: Die Datei wird mit den Rechten der Gruppe gestartet, der das Programm gehört.
- 3 Spalten Rechte aller Anderen (others)
Rechte wie group, aber zusätzlich:
 - t Löschschutz. Datei darf nur von Besitzerin gelöscht werden.

2.1.8 Dateirechte ändern

Zugriffsrechte Wenn wir morgen ein kleines Skript schreiben, ist das zunächst eine „ganz normale Datei“, die die systemüblichen Dateirechte bekommt, wenn wir sie das erste Mal speichern.

Da wir das Skript wie ein Programm aufrufen wollen, muss es natürlich ausführbar sein. Wir müssen also die Dateirechte ändern. Dabei gilt es, genau anzugeben, welcher Benutzerinnengruppe wir welche Rechte zuteilen.

Dazu können wir auf zwei unterschiedliche Weisen vorgehen.

- Durch Angabe
 - der Benutzerinnengruppe (u user g group oder o others)
 - der Änderungsoperation (+ oder -)
 - des Zugriffsrechts (r, w, oder x)

`chmod kleinesSkript ug+x` ändert die Rechte unseres Skripts für uns und unsere Gruppe auf ausführbar.

`chmod kleinesSkript o-r` entzieht allen Anderen die Leserechte.

- Durch Angabe einer dreistelligen Oktalzahl
Die Oktalzahlen ergeben sich folgendermaßen:

- 1 für executable
- 2 für writeable
- 4 für readable

Die Rechte werden für jede Gruppe einzeln berechnet und hintereinander geschrieben.

```
r w x  r - x  - - -
4 2 1  4 0 1  0 0 0
```

```
= 7    = 5    = 0
```

ergibt: `chmod 750 kleinesSkript`

Die Grundeinstellung, mit der zunächst alle neuen Dateien gespeichert werden, wird durch `umask` festgelegt. Die `umask` wird von einer Maximalzahl abgezogen, die bei Verzeichnissen 777, bei Dateien 666 beträgt. Eine `umask` von 022 (bei den meisten Systemen in der `/etc/profile` voreingestellt) ergibt somit die Rechte 755 für Verzeichnisse und 644 für Dateien.

Aufgabe:

Ändere Deine `umask` so, dass Deine neuen Dateien niemand mehr außer Dir lesen kann und dass in neue Verzeichnisse nur Du selbst wechseln darfst.

Besitzerinnen ändern Vor allem `root` kann es sinnvoll finden, einer Datei eine andere Besitzerin zu geben. Dazu gibt es zwei Wege:

- Besitzerin und Gruppe gleichzeitig ändern:
`chown neueUserin.neueGruppe dateiname`
- Besitzerin und Gruppe extra ändern:
`chown neueUserin dateiname`
`chgrp neueGruppe dateiname`

2.1.9 Weitere Befehle

Dateien und Verzeichnisse löschen Dateien werden mit dem Befehl `rm Dateiname` gelöscht. Aber Vorsicht! Bei Linux ist weg auch wirklich weg! Es kann zu ganz bösen Fehlern kommen, wenn frau sich aus Versehen verschreibt. Eine der ärgerlichsten Fehleingaben lautet:

```
rm nichtwichtig. *
```

Das Problem liegt in dem Leerzeichen zwischen Dateinamen und Stern. Damit sind nicht etwa Dateien wie `nichtwichtig.gif` und `nichtwichtig.jpg` weg, sondern alle, da der Stern eben alle Dateinamen im aktuellen Verzeichnis meint.

Wird die Option `-r` (für „rekursiv“) verwendet, können damit auch Verzeichnisse gelöscht werden.

Es kann sich also lohnen, einen *Alias* zu setzen, der `rm` automatisch mit der Option `-i` versieht:

```
alias rm="rm -i"
```

Damit muss jede Löschaktion zusätzlich bestätigt werden. Böse Überraschungen bleiben so aus. Um trotzdem ungenervt einmal ein ganzes Unterverzeichnis löschen zu können, unterbindet die Option `-f` für „force“ die Fragerei, und alles wird auf einmal vernichtet.

Alias Ein Alias ist ein Spitzname. Damit können lange Kommandos abgekürzt oder ungenügende umgeschrieben werden. Ein Alias kann entweder in der Shell gesetzt werden, dann gilt er aber auch nur so lange, wie diese Shell läuft, oder er wird in eine der Konfigurationsdateien `.bashrc` oder `.profile` geschrieben. Dann gilt er überall.

Dateien und Verzeichnisse kopieren oder umbenennen Zum Kopieren dient der Befehl `cp name1 name2`. Er kann sowohl bei Dateien als auch bei Verzeichnissen angewendet werden. Bei Verzeichnissen empfiehlt sich noch die Option `-r` für rekursives Duplizieren, da damit auch alle Unterverzeichnisse kopiert werden.

Zum Umbenennen dient der Befehl `mv name1 name2`. Er funktioniert sowohl bei Dateien als auch bei Verzeichnissen, bei letzteren allerdings nur, wenn sie sich auf der selben Partition befinden.

Verzeichnisse anlegen Ein Verzeichnis wird mit dem Befehl `mkdir verzeichnisname` angelegt. Der Befehl `rmdir` löscht es wieder. Allerdings nur, wenn es leer ist. Da ist es besser, sich des Befehls `rm -r` zu bedienen.

Dateien suchen Manchmal weiß frau einfach nicht mehr, wo sie eine Datei abgelegt hat. Da hilft der Befehl `locate gesuchteDatei`. Das geht rasend schnell, da `locate` auf die Datenbank `locatedb` zugreift.

Eine andere Möglichkeit ist es, den Befehl `find` zu Hilfe zu bitten. Seine Syntax lautet `find Startverzeichnis -name gesuchteDateiOderPattern`

Mit der Option

`-exec Befehl {} \;`

führt er für jede gefundene Datei den gewünschten Befehl aus.

Verschiedene Arten, eine Datei anzuschauen (Pager)

- **cat**
Zeigt den Dateiinhalt „ungebremst“ an.
- **more**
Einfaches Anzeigeprogramm. Beendet sich automatisch, wenn das Ende der Datei erreicht ist.
- **less**
Funktionstüchtigerer Nachfolger von **more**.
Tasten-Kommandos für **less**:
Leertaste – blättert eine Seite vor
Entertaste – blättert eine Zeile vor
b – blättert eine Seite zurück
h – Hilfe (zur Verfügung stehende Kommandos in **less**)
q – beendet **less**
/pattern – sucht im Text nach *pattern*, springt an diese Stelle und markiert es. (Weiter mit n)

2.1.10 Extrawurst: die Drucker

Dein Drucker ist in der Regel an der parallelen Schnittstelle angeschlossen, die unter Linux als `/dev/lp1` im Verzeichnisbaum hängt. Da Linux sie als Datei verwaltet, könnte das Dokument einfach an den Drucker kopiert werden:

```
cp datei /dev/lp1 oder
```

```
cat datei > /dev/lp1
```

Da normalerweise nur *root* Zugriff auf Gerätedateien hat und zudem nicht jede Benutzerin wissen muss, an welchem Device nun der Drucker hängt, ist es komfortabler, die Dienste des Druckerdaemons `lpd` in Anspruch zu nehmen.

Dieser stellt alle ankommenden Druckjobs in eine Reihe (Queue), und arbeitet sie hintereinander ab, so dass Konflikte vermieden werden.

Abschicken eines Druckjobs `lpr dateiname` startet einen Druckjob, wenn es nur einen einzigen Drucker im System gibt oder der Standarddrucker in der Umgebungsvariablen `PRINTER` eingetragen ist. Können mehrere Drucker angesprochen werden, so muss der gewünschte mit der Option `-P druckername` ausgewählt werden.

Auflistung aktueller Jobs `lpq` zeigt, welche Jobs gerade abgearbeitet werden. Es liefert außerdem noch die Information, wer den Job abgeschickt hat, welche Datei gedruckt wird und wie groß sie ist.

```
bille@ratschni:/home/bille > lpq
lp1 is ready and printing
Rank   Owner   Job   Files                               Total Size
active bille   395   /home/bille/inf/linux.ps           562130 bytes
1st    bille   396   /home/bille/zug.ps                 86626 bytes
```

Löschen von Druckjobs Jeder Job bekommt eine fortlaufende Nummer (mit `lpq` sichtbar). Mit dieser Nummer kann er per `lprm jobnummer` abgeschossen werden.

```
bille@ratschni:/home/bille > lprm 396
dfA10ratschni dequeued
```

2.2 Editoren

2.2.1 Vi

Textdateien werden mit *Editoren* erstellt bzw. verändert. Ein mächtiger und auf allen Unixsystemen vorhandener Editor ist der `vi`. Seine Bedienung ist durchaus etwas gewöhnungsbedürftig, doch da er bei einigen (System-)Programmen als Standardeditor (den frau mit den Umgebungsvariablen `EDITOR` oder `VISUAL` umstellen kann) dient, kann frau des öfteren in Situationen kommen, in denen sie wenigstens wissen muss, wie sie den Editor ungeschoren wieder verlässt.

Weil der „Original“-Vi selbst Vi-Fans als recht sperrig erscheint, gibt es eine Menge *Klone*, die die Bedienung vereinfachen und zusätzliche Features wie Syntaxhighlighting eingebaut haben. Unter Linux wird es frau immer mit einem Vi-Klon zu tun haben – ob der `vim`, `elvis` oder `nvi` heißt, ist von Distribution zu Distribution verschieden und natürlich Geschmacksfrage. Auch wenn Vi-Klone natürlich auch unter ihrem speziellen Namen ansprechbar sind, lässt sich einer davon immer mit dem Befehl

```
vi filename
```

aufrufen.

Wenn es die Datei *filename* bisher noch nicht gibt, erscheint jetzt ein Bildschirm, in dem an jedem Zeilenanfang eine Tilde steht. Diese markiert eine Zeile und ist nicht wirklich in der Datei enthalten.

Dass Vi so gewöhnungsbedürftig ist, liegt an seiner Geschichte. Der Original-Vi war einer der ersten Editoren, in denen frau eine ganze Datei auf einmal ändern konnte, statt wie bei *Line Editors* wie `ed` zeilenorientiert zu editieren. Wie bei Zeileneditoren kennt der Vi zwei Modi:

Kommandomodus: Positionieren des Cursors, Texte suchen, Texte speichern usw.; erreichbar aus dem Eingabemodus durch Betätigen der *ESC*-Taste

Eingabemodus: Texteingabe; erreichbar aus dem Kommandomodus durch Eintippen von `i`, `a` und anderen `vi`-Kommandos.

Nach dem Aufruf befindet sich der `vi` zunächst im Kommandomodus. Die wichtigsten Befehle in diesem Modus sind:

Vi-Befehle	
h, j, k, l	Positionieren des Cursors (meist auch durch die Cursor-tasten möglich)
<i>Strg+f</i>	ganzer Bildschirm nach unten
<i>Strg+b</i>	ganzer Bildschirm nach oben
G	bewegt den Cursor an den Anfang der letzten Zeile der Datei
x	löscht das Zeichen unter dem Cursor
dw	löscht das Wort, auf dessen erstem Buchstaben der Cursor steht
dd	löscht die Zeile, in der der Cursor steht
o	erzeugt eine neue Zeile hinter der aktuellen und wechselt in den Eingabemodus
O	erzeugt eine neue Zeile vor der aktuellen und wechselt in den Eingabemodus
Y	Kopieren der aktuellen Zeile in den Zwischenpuffer
p	Einfügen des Zwischenpufferinhalts hinter dem Cursor
Yp	Duplizieren der aktuellen Zeile
u	Undo
/Stichwort	Vorwärtssuche nach <i>Stichwort</i>
?Stichwort	Rückwärtssuche nach <i>Stichwort</i>
<i>Strg+r</i>	Rückgängigmachen eines Undo
:.s/Suchbegriff/Ersatz/g	Suchen und Ersetzen in der aktuellen Zeile (ohne Nachfrage)
:%s/Suchbegriff/Ersatz/g	Suchen und Ersetzen in der gesamten Datei (ohne Nachfrage)
:.,\$s/Suchbegriff/Ersatz/gi	Suchen und Ersetzen von der aktuellen Zeile bis zum Dateiende (mit Nachfrage)
:w	Speichern der Datei
:q	Verlassen der Datei, wenn alles gespeichert ist oder nichts verändert wurde
:q!	Verlassen der Datei ohne Speicherung vorangegangener Änderungen
:wq	Speichern und Verlassen der Datei

Wie schon in diesen Beispielen zu sehen, muten Vi-Befehle Nicht-Vi-Benutzerinnen unglaublich komplex an. So kann einem durch Doppelpunkt eingeleiteten Befehl (durch den frau in den sogenannten *ex-Modus*, benannt nach dem Zeileneditor *ex*, gelangt) ein Bereich vorangestellt werden (2,6 – von Zeile 2 bis einschließlich Zeile 6), während „doppelpunktlosen“ Befehlen die Anzahl der Wiederholungen mitgegeben werden kann (2Y kopiert zwei Zeilen, 2dw löscht zwei Wörter).

Das Suchen und Ersetzen funktioniert übrigens ähnlich wie in Perl.

Als Faustregel gilt: Nicht überwältigen lassen, denn es gibt Unmengen weiterer Befehle, die nicht einmal ständige Vi-Benutzerinnen alle kennen. Sie werden natürlich auch in der Manpage zu `vi` erklärt.

Aufgabe:

Schreibe eine etwa fünf- bis zehnzeilige Datei, in der das Wort *Informatica* vorkommt. Kopiere eine Informatica-Zeile ans Ende der Datei. Lösche in einer Informatica-Zeile alle Wörter vor und hinter Informatica. Suche nach Informatica und ersetze alle Vorkommen durch *Informatica Feminale*. Speichere die Datei. Mach die Suchen-und-Ersetzen-Aktion rückgängig. Verlass die Datei, ohne sie zu speichern.

Aufgabe:

Probiere die Kommandos `:!ls` und `:.!ls` aus! Was geschieht und was ist der Unterschied beider Versionen?

2.2.2 emacs

Der Emacs ist der wohl komfortabelste Editor für Tastaturfreaks, die bereit sind, sich seine Tastaturkürzel anzueignen. Er lässt sich zwar auch per Menü bedienen, seine enormen Vorteile erschließen sich allerdings bei der Handhabung per Tastatur.

Die Befehle, mit denen der Emacs gesteuert wird, beginnen immer mit einer der Meta-Tasten *Strg* oder *Esc* bzw. je nach Tastatureinstellung durch die *Alt*-Taste.

In der Beschreibung der Shortcuts wird sowohl in der Literatur als auch im Emacs selbst folgende Konvention beachtet:

Esc bzw. *Alt* → M
Strg → C

Soll die Meta-Taste noch gehalten bleiben, während die nächste Taste gedrückt wird, so verbindet die Tasten ein Minus-Zeichen (-). So wird bei `C-x C-s` jeweils die *Strg*-Taste gehalten, während die `x-` bzw. die `s`-Taste gedrückt werden.

Erfolgt ein Befehl interaktiv (d. h., der Emacs braucht noch eine Bestätigung oder gibt eine Rückmeldung), so erscheint der Dialog am unteren Rand des Emacs-Fensters. Wird zum Beispiel der Befehl zum Speichern der Datei eingegeben, so erscheint unten etwa:
 Wrote /home/sibylle/informatica/linuxkurs.tex

Nach dem Befehl zum Öffnen einer Datei erscheint der aktuelle Pfad, an den ein Dateiname angehängt oder der auf den gewünschten Pfad/Dateinamen geändert wird. In diesem Moment gehen alle Tastatureingaben an den Dialog und nicht mehr in den Text.

Emacs-Tastenkombinationen

Fortsetzung folgt ...

<i>... Fortsetzung</i>	
Emacs beenden	C-x C-c
Datei öffnen	C-x C-f
Datei speichern	C-x C-s
Datei unter anderem Namen speichern	C-x C-w
Andere geöffnete Datei bearbeiten	C-x C-f ↑ oder ↓
Befehl abbrechen	C-g
Undo	C-x _ oder C-u
Zwei Fenster untereinander öffnen	C-x 2
Farbige Syntax-Hervorhebung	C-x font-lock-mode
Datei schließen	C-x k
Nach Zeichenkette vorwärts suchen	C-s
Nach Zeichenkette rückwärts suchen	C-r
Zum Zeilenende	C-e
Zum Zeilenanfang	C-a
Zum nächsten Wort	M-→ oder M-←
Ans Ende der Datei	M->
An den Anfang der Datei	M-<
Markierten Textteil löschen	C-w
Gelöschten Textteil einfügen	C-y
Fragen und Ersetzen	M-% <i>vorher</i> ↔ <i>nachher</i> ↔ Wenn der <i>vorher</i> -Text gefunden wurde <i>y</i> für Ersetzen, <i>n</i> für Nicht-Ersetzen
Eine Shell öffnen	M-x shell
Emacs-Tutorium starten	C-h t
Emacs-Info-System starten	C-h i

Aufgabe:

Lege eine Datei an, in der du dich den anderen Kursteilnehmerinnen vorstellst, und speichere die Datei in deinem Home-Verzeichnis

Aufgabe:

Starte das emacs-Tutorial, und probiere es einmal aus.

2.2.3 Weitere Editoren

Editoren gibt es wie Sand am Meer, sodass frau zumindest an selbstadministrierten Rechnern genau den benutzen kann, der ihr am besten zusagt. Folgende Editoren sind

häufig installiert:

1. `pico`, der (nicht-grafische) Editor, der zum Mailprogramm `pine` gehört
2. `joe`, ein beliebter nicht-grafischer Editor
3. `nedit`, ein Editor für X
4. die mit KDE mitgelieferten Editoren `kwrite` (vor KDE 2.2) bzw. `kate` (ab KDE 2.2)
5. `gedit`, ein Editor aus dem GNOME-Projekt
6. `jove`, ein Emacs-ähnlicher, nicht-grafischer Editor

Aufgabe:

Suche im Web nach weiteren Texteditoren und prüfe, welche dieser und der oben angegebenen Editoren auf Deinem Pool-Rechner installiert sind.

Aufgabe:

Teilt die installierten Editoralternativen unter Euch auf und erstellt eine Kurzbeschreibung und Bedienungsanleitung für die anderen Kursteilnehmerinnen.

2.3 Crashkurs für Hobby-Sysadminen

Im Folgenden werden wir einige häufig anstehende Systemadministrationsaufgaben besprechen. Mangels `root`-Zugang können wir viele allerdings nicht in der Praxis üben.

2.3.1 Benutzerinnenverwaltung

Wenn neue Benutzerinnen auf einem Unixrechner arbeiten sollen, benötigen sie einen *Account*, also einen Usernamen, ein Passwort und ein Homeverzeichnis. Diesen muss die Systemadministratorin anlegen. Sie muss aber auch dafür sorgen, dass unbenutzte Accounts entfernt werden, denn jeder nicht nötige Account ist ein Sicherheitsrisiko.

Die Angaben zu Username, Passwort, Homeverzeichnis, aber auch Login-Shell und Primärgruppe werden in der Datei `/etc/passwd` von Doppelpunkten getrennt in folgender Reihenfolge abgelegt:

```
root:x:0:0:root:/root:/bin/zsh
trish:x:1001:100:Patricia Jung:/home/trish:/bin/bash
Username:Passwort:UserID:GruppenID:Beschreibung:Homeverzeichnis:Login-Shell
```

Welche Gruppe der Primärgruppen-ID entspricht, steht in `/etc/group`. Wenn User Mitglieder in mehr als einer Gruppe sein sollen, wird das dort eingetragen:

```
users::100:testuser
```

Außer `trish` ist also auch `testuser` Mitglied der Gruppe `users`.

Passwörter Zwar werden die Passwörter mit ihrer Festlegung in `/etc/passwd` verschlüsselt abgelegt, doch da die Verschlüsselungsmethode bekannt ist, kann frau die Wörterbücher dieser Welt nehmen, auf dieselbe Art und Weise verschlüsseln und dann mit den Einträgen in der `passwd`-Datei vergleichen. Diese muss für alle User lesbar sein, damit sie ihr Passwort mit `passwd` ändern können.

Tools wie `crack` automatisieren diese Brut-Force-Methode (und dehnen sie auf gängige Modifikationen bei Wörterbuchpasswörtern aus). So kann jede, die einen Account auf dem Rechner hat, theoretisch recht unproblematisch nach unbedacht gewählten Passwörtern fahnden. Systemadministrator(inn)en, die größere Userpools zu verwalten haben, benutzen `crack` o. ä., um zu verhindern, dass durch derart schwach geschützte Accounts das gesamte System gefährdet wird, denn ein Rechner lässt sich viel einfacher von innen als von außen hacken.

Heutzutage ist es eher unverantwortlich, die Passwörter in der `/etc/passwd` abzulegen (auch wenn es sich nicht immer – z. B. bei der Verwendung von NIS – vermeiden lässt). Das Problem wird durch die Einführung von *Shadow-Passwörtern* aus der Welt geschafft. Wenn in der zweiten `passwd`-Spalte ein `x` steht, wurde das Passwort in eine nur für `root` lesbare Datei namens `/etc/shadow` ausgelagert, die (wie man 5 `shadow` zeigt) auch weitere Möglichkeiten der Userverwaltung wie auslaufende Passwörter oder zeitgesteuertes Disablen eines Accounts erlaubt.

Aufgabe:

Werden auf Deinem Rechner Shadow-Passwörter verwendet? Wenn ja, versuch einen Blick in die `/etc/shadow` zu werfen!

Aufgabe:

Welche UserID hat `root`, welche hast Du? Welchen Gruppen bist Du zugeordnet?

User und Gruppen modifizieren Natürlich lassen sich die Einträge in `/etc/passwd`, `/etc/shadow`, `/etc/group` mit einem Text-Editor ändern, das Homeverzeichnis mit `mkdir` anlegen und mit `chown` an die neue Userin übergeben sowie mit `passwd` ein Anfangspasswort setzen.

Dennoch schätzen viele Sysadminen einen gewissen Komfort, den distributionseigene Tools bei der Useradministration bieten. Mit dem im Kurs vermittelten Wissen sollte deren Bedienung in Eigenregie machbar sein. Wir konzentrieren uns daher an dieser Stelle auf Tools, die sich auf der Kommandozeile bedienen lassen und oft (wenn auch nicht immer) vorinstalliert sind.

Zum Einrichten neuer User gibt es (in der Regel in `/usr/sbin` installiert, einem Verzeichnis, das oft nicht im Default-Suchpfad steht) normalerweise das Kommando `useradd`, dem frau auf der Kommandozeile die Parameter für die neue Userin mitgeben muss:

```
# /usr/sbin/useradd -c "Sibylle Naegle" -d /home/bille -g 100 -p passwort  
-s /bin/bash -u 2000 -m bille
```

Die Option `-m` bedeutet dabei, dass das Homeverzeichnis angelegt wird und in `/etc/skel` stehende Konfigurationsdateien hineinkopiert werden. Lässt frau die eine oder andere Option weg, so werden Defaultwerte benutzt.

Aufgabe:

Finde mit Hilfe der Manpage heraus, wofür die übrigen Optionen stehen.

Etwas komfortabler, da interaktiv lassen sich User mit `adduser` anlegen. Allerdings gibt es auch Distributionen wie Caldera, die ihre eigene `adduser`-Version benutzen.

Seltener gebraucht wird das `useradd`-ähnliche Programm `usermod` zum Modifizieren von Accountdaten. Zum Löschen eines Users kann frau `userdel` benutzen:

```
# /usr/sbin/userdel bille
```

Das Homeverzeichnis von `bille` wird dabei nicht angetastet, sodass ihre Daten nicht verloren gehen. Hier muss die Superuserin bei Bedarf selbst Hand anlegen.

Aufgabe:

Wie würdest Du als `root` Billes Homeverzeichnis löschen?

Wie zum Anlegen von Usern gibt es natürlich auch Tools, die neue Gruppen anlegen (`groupadd`) oder eine Gruppe löschen (`groupdel`). Auf jeden Fall lohnt sich immer ein Blick in die passenden Manpages.

2.3.2 Installieren von Software

Das Einspielen neuer Programme auf einem Rechner ist alles andere als eine triviale Angelegenheit: Da sollten die benötigten Bibliotheken in einer passenden Version installiert sein, möglicherweise wird weitere Software zum Funktionieren benötigt, und wenn frau die Software doch wieder loswerden will, soll möglichst alles wieder vom Rechner verschwinden, was sie beim Installieren mitgebracht hat.

Um all diese Dinge kümmert sich bei modernen Distributionen ein Paketmanager. Die weiteste Verbreitung hat der *Red Hat Packet Panager* `rpm` gefunden, der nicht nur bei Red Hat, sondern auch von SuSE, Caldera oder Mandrake eingesetzt wird.

Technisch mit mehr Lorbeeren bedacht wird der Paketmanager der Debian-Distribution und ihrer Abkömmlinge, `dpkg`. Er brilliert vor allem durch sein Frontend `apt-get`, mit dem sich das System übers Netz updaten lässt. Mit dem Tool `alien` lassen sich einfache `rpm`- und `deb`-Pakete zu einem gewissen Grad ineinander umwandeln.

Das Bauen der Binärpakete für einen Paketmanager ist eine Wissenschaft für sich und muss für jede Distribution (und jede Plattform) neu vorgenommen werden. Da die verschiedenen Versionen einer Distribution meist auf unterschiedlichen Bibliotheksversionen aufbauen, deren API oft nicht kompatibel ist, und auch die Distributoren selbst genügend einschneidende Veränderungen vornehmen, ist es oft nur möglich, ein Binärpaket einzuspielen, dass für die vorliegende Version einer Distribution gedacht ist.

Auch wenn mehrere Distributionen auf rpm setzen, heißt das nicht, dass Pakete distributionsübergreifend austauschbar wären. Während Red Hat und Mandrake recht kompatibel zueinander sind, sodass sogar Serverpakete der einen auf der anderen Distribution benutzt werden können, ist bei SuSE höchstens bei Anwendungen an einen Austausch mit anderen Distributionen zu denken – bei Serverpaketen sollte frau wegen der SuSE-Eigenheiten eher zurückhaltend sein.

Im Folgenden widmen wir uns der Bedienung von rpm. Der Aufruf dieses Programms ist auch bei SuSE möglich, die zum Einspielen und Deinstallieren eigentlich Yast(2) propagiert. Letzterer wirkt im Wesentlichen als rpm-Frontend, sorgt aber auch für die Konfiguration SuSE-eigener Besonderheiten, mit denen wir uns an dieser Stelle nicht auseinandersetzen.

Red Hat Packet Manager rpm-Pakete wie `wwwoffle-2.6b-2.i386.rpm` enthalten (mit Ausnahme von SuSE-Paketen) in ihrer Bezeichnung den Namen (`wwwoffle`) und die Version (`2.6b`), das *Patchlevel*, eine Information darüber, wie oft es von Distributorenseite Änderungen am Paket gab (`2`) sowie die Ziel-Plattform (`i386`). Bei Quelltextpaketen fällt letztere natürlich weg.

Widmet sich frau einem bereits installierten Paket, muss sie Namen (und darf Version) angeben (`wwwoffle-2.6b` oder `wwwoffle`). Kümmerst sie sich um ein noch nicht installiertes Paket, ist der gesamte Dateiname gefragt (`wwwoffle-2.6b-2.i386.rpm`).

So ließe sich das besagte `wwwoffle`-Paket mit

```
rpm -iv wwwoffle-2.6b-2.i386.rpm
```

installieren, wobei das Flag `-v` für „verbose“, also geschwätzige Ausgaben sorgt. Will frau zudem einen „Fortschrittsbalken“ aus `#`-Zeichen sehen, fügt sie die Option `-h` („hash“) mit an.

Sofern das Paket bereits installiert ist, bekommt frau eine entsprechende Fehlermeldung. War ein Update auf eine neue Version das ursprüngliche Ansinnen, so benutzt sie statt `-i` das Flag `-U`.

Zum Deinstallieren eines Pakets gibt es die Option `-e` wie „erase“:

```
rpm -e wwwoffle-2.6b
```

Zum Einholen von Informationen über Pakete dient die Option `-q` („query“). Geht es dabei um noch nicht installierte Pakete, so kommt noch ein `-p` („package“) dazu.

Will frau wissen, welche Pakete auf ihrem Rechner installiert sind, geht das mit `rpm -qa` (`-a` wie „all“). Welche Dateien im (nicht installierten) `wwwoffle`-Paket drin ist, erfährt frau mit

```
rpm -qpl wwwoffle-2.6b-2.i386.rpm
```

(l wie „list“), während

```
rpm -qi wwwoffle
```

„Informationen“ zum installierten `wwwoffle` gibt.

Zu welchem Paket eine Datei gehört, erfährt frau mit

```
rpm -qf /etc/passwd
```

Aufgabe:

Zu welchem Paket gehört das Kommando `ls`? Nenne fünf weitere Programme, die zu diesem Paket gehören! Lies die Paketbeschreibung.

Aufgabe:

Wieviele Pakete sind auf Deinem Rechner installiert? Nutze dazu eine Pipe und das Kommando `wc -l`.

GNU-Make Nicht zu jeder Software gibt es ein für die eigene Distribution geeignetes Binärpaket, viele Tools findet frau grundsätzlich nur im Sourcecode. Die entsprechenden `tar.gz`- bzw. `.tgz`-Files, inzwischen auch des Öfteren mit `bzip2` statt `gzip` komprimierte Quellarchive, gilt es, zunächst auszupacken. Ein aktuelles `tar` vorausgesetzt, geht das mit `tar -xzf` bei mit `gzip` gepackten, mit `tar -xIf` bei mit `bzip2` gepackten `tar`-Archiven. Darin befindet sich in der Regel eine Datei namens `README` und/oder `INSTALL`, die Informationen darüber gibt, wie frau zu einem Binärprogramm kommt. Bei einfachen Projekten liegt eine Datei `Makefile` bei, die diese Informationen enthält, sodass ein einfaches `make` ausreicht, um den Kompilierungsvorgang einzuleiten.

Sobald das Projekt komplexer wird, nach verschiedenen externen Bibliotheken und Hilfsprogrammen verlangt und gar noch portabel für verschiedene Plattformen sein soll, bietet ein festes `Makefile` zu viele Unwägbarkeiten. Dann (und das ist die Regel) liegt den Sourcen ein `configure`-Skript bei. Um es laufen zu lassen, wechselt frau ins Verzeichnis und ruft

```
./configure
```

auf. Läuft das Skript nicht durch, ist meist Nachinstallation von Third-Party-Tools und -Bibliotheken angesagt.

Das `configure`-Skript legt u. a. auch fest, wohin welche Dateien letzten Endes installiert werden sollen. In der Regel ist das `/usr/local`, doch wer Wert darauf legt, die selbst-kompilierte Software später auch wieder rückstandslos zu deinstallieren, wird besser ein eigenes Verzeichnis wie `/usr/local/extra-verzeichnis` wählen (und die installierten

Dateien eventuell von Hand oder mit einem Tool wie `stow` so verlinken, dass sich der Suchpfad nicht ins Unendliche aufbläst). Das geht (meist) mit der `configure`-Option `--prefix`:

```
./configure --prefix=/usr/local/extra-verzeichnis
```

`configure` kennt auch noch weitere Optionen, die z.B. auch die Funktionalität des Binaries anpassen. Welche das sind, erfährt frau mit `./configure --help`.

Gab es mit `configure` keine Probleme, startet `make` den Kompiliervorgang. Für die endgültige Installation braucht frau `root`-Rechte, sofern sie nicht in ein eigenes Verzeichnis installiert: `make install` kopiert dann alle relevanten Daten an Ort und Stelle. Natürlich ist frau nicht vor Kompilierfehlern gefeit: Was sich auf einem System problemlos kompilieren lässt, kann auf einem anderen Probleme machen. Wer sich in C oder C++ auskennt, kann hier sicher die eine oder andere Sache ausbügeln, doch bei komplexer Software ist das oft nicht so einfach. Dann bleibt immer noch die Möglichkeit, eine andere Version auszuprobieren, im Netz nach Leidensgenoss(inn)en und deren Problemlösungen zu fahnden oder aber den Autor(inn)en der Software eine Mail zu schreiben. Viele reagieren darauf ausgesprochen hilfsbereit, sofern frau soviel wie möglich relevante Informationen (welche Distribution, welche Compilerversion, welche Version benötigter Bibliotheken, welche Fehlermeldung etc.) mitliefert. Bei großen Projekten gibt es Bug-Tracking-Systeme, in die diese Informationen eingetragen werden.

Aufgabe:

Lade das Paket `stow`-Paket herunter und installiere es in Deinem Homeverzeichnis!
http://ftp.debian.org/debian/dists/potato/main/source/admin/stow_1.3.2.orig.tar.gz

2.3.3 Prozesse im Griff

In diesem Abschnitt schauen wir uns an, wie wir herausfinden, welche Prozesse laufen, wie wir widerspenstige Programme doch noch zum Aufgeben zwingen oder dazu bringen, die Kommandozeile nicht zu blockieren.

Ein Prozess ist die Folge von Aktivitäten, die zwischen dem Aufruf eines Programms und dessen Ende stattfinden. Von jedem Kommando wird ein neuer Prozess gestartet. Manchmal dauert die Ausführung eines Programms sehr lange. Wenn dieses Programm keine Benutzereingaben benötigt, wäre es gut, wenn frau gleichzeitig noch etwas anderes tun könnte. Hier hilft Unix durch seine Mehrprozessfähigkeit: Bekommt ein Kommando ein `&` anhängt, wird das Programm im Hintergrund gestartet, und die Shell meldet sich sofort wieder mit dem Prompt zurück. Angenommen, wir haben ein Programm mit dem Namen `coffee`, dessen Ausführung viel Zeit in Anspruch nimmt, und ein Programm `tea`, das deutlich schneller geht. Dann würden wir folgendermaßen verfahren:

```
coffee &      Programm coffee wird im Hintergrund abgearbeitet
tea           tea wird normal gestartet
```

Nach der Beendigung des Programms `coffee` erhält die Benutzerin eine Nachricht. Wer nicht mehr so genau weiß, welche Programme in der aktuellen Shell noch im Hintergrund laufen, kann sie sich durch das Kommando `jobs` ansehen. Hat frau vergessen, einen Prozess beim Aufruf in den Hintergrund zu schicken, ist das keine Katastrophe. Mit `Strg+Z` kann sie ihn anhalten und dann mit dem Kommando `bg` („background“) in den Hintergrund schicken. Ebenso lässt sich der letzte Hintergrundprozess mit `fg` („foreground“) wieder in den Vordergrund holen. Laufen mehrere Prozesse im Hintergrund, gibt frau `fg` die Nummer, die `jobs` für den entsprechenden Job ausgibt, als Option mit auf den Weg. Läuft der Prozess im Vordergrund (wurde also ohne das `&` gestartet), so lässt er sich normalerweise (leider nicht immer) mit `Strg+C` abbrechen. Diese Tastenkombination ist das Signal für jeden Prozess, einen *geordneten Abbruch* durchzuführen. Das bedeutet zum Beispiel, daß geöffnete Dateien noch geschlossen werden u. a. Wenn ein Prozess nicht mehr auf `Strg+C` reagiert, ist das dennoch kein Grund zur Panik: Wir arbeiten schließlich mit einem Multitasking-System. Frau geht einfach in ein neues Fenster oder an ein anderes Terminal und sucht die *Prozessidentifikationsnummer PID* des hartnäckigen Prozesses. Dazu verwendet sie das Kommando `ps` („process status“). `ps` listet die Prozesse in der aktuellen Shell tabellarisch auf. Um eine ausführliche Liste aller auf der Maschine laufenden Prozesse zu bekommen, verwendet frau unter Linux die Optionen `-aux`; auf anderen (System-V-)Unixsystemen benötigt sie dagegen die Optionen `-ef`, denn `ps` gehört zu den traditionellen Unixtools, die zwar überall vorhanden sind, sich aber leider überall ein wenig anders verhalten. Hier empfiehlt sich ein Blick auf die Manpage. In der `ps`-Ausgabe sucht frau in der Spalte `COMMAND` den Prozess, den sie *abschießen* möchte. In der Spalte `PID` findet sie die zugehörige Prozessidentifikationsnummer. Jetzt kann sie den hängengebliebenen Prozess durch den Befehl `killpid` abbrechen. Führt das immer noch nicht zum Erfolg, so kann sie `kill` mit der Option `-9` aufrufen – kein Prozess kann dieses Signal ignorieren.

Aufgabe:

Starte `netscape` im Vordergrund einer Shell. Schick es manuell in den Hintergrund. Starte das Programm `xeyes` in derselben Shell im Hintergrund. Hole `netscape` wieder in den Vordergrund. Beende das Programm, ohne auf sein Menü zuzugreifen. Suche nach dem `xeyes`-Prozess in der Prozesstabelle und beende ihn mit dem `kill`-Befehl.

Neben diesen traditionellen Unixtools gibt es auf vielen Linuxsystemen noch zwei Werkzeuge, die das Verwalten von Prozessen leichter machen: `pstree` zeigt in einer symbolischen Baumstruktur an, welcher Prozess von welchem aufgerufen wurde, sprich, welcher Prozess eine *Tochter* von welchem *Elternprozess* ist. In dieser Darstellung sieht frau gut, dass `init` die Mutter aller Prozesse ist.

Aufgabe:

Schau Dir mit `pstree` den Prozessbaum an und probiere die Wirkung einiger Optionen, die Du in der Manpage findest.

Das Heraussuchen von PIDs ist eine recht langwierige Angelegenheit. Schneller ginge es, wenn frau statt der Nummer einfach den Namen des Prozesses angeben könnte. Das geht zwar nicht immer, aber dafür werden von `killall` oder als letzte Rettung auch `killall -9` gleich alle Prozesse gleichen Namens mit in den Abgrund gerissen.

Aufgabe:

Starte drei `xeyes` und beende sie alle gleichzeitig.

2.3.4 Die Zeit unter Kontrolle

Nicht immer will frau am Rechner hocken, wenn der eine bestimmte Aufgabe ausführen soll. Zu diesem Zweck gibt es mit `cron(d)` einen Daemon, der im Hintergrund nichts anderes tut, als nachzuschauen, ob in der aktuellen Minute ein Job für ihn ansteht.

Aufträge für Cron werden in *Crontabellen* eingetragen. Neben der systemweiten, `/etc/crontab`, gibt es auch für jede Userin eine. Um diese anzulegen oder zu ändern, gibt es das Kommando `crontab -e` („edit“). Sofern die Variable `EDITOR` oder `VISUAL` nichts anderes sagt, wird jetzt der `vi` gestartet.

Der Job muss in einer ganz bestimmten Form in die Crontabelle eingetragen werden:

Minute(n) Stunde(n) Tag(e) Monat(e) Wochentag(e) Kommando

Die Spalten werden durch Leerzeichen getrennt; wenn ein Job zweimal am Tag um 12 und 24 Uhr ausgeführt werden soll, kann frau dies mit Komma (in diesem Fall `0,12` in der Spaltenzeile) angeben. Soll ein Job jeden Tag von Montag bis Freitag ausgeführt werden, schreibt frau `1-5` in die Wochentagsspalte. Ein `*` in einer Zeile bedeutet, dass die jeweilige Spalte egal ist. Als Kommando schreibt frau die Kommandozeile, die sie auch in der Shell eingeben würde. Aber Achtung: Manche Programme reagieren nicht wie gewünscht, wenn sie ohne Terminal vom Cron-Daemon aufgerufen werden. Ehe frau sich also vom Rechner entfernt, sollte sie überprüft haben, dass alles glatt geht. Wichtig ist zudem, darauf zu achten, dass in einer Jobzeile keine Zeilenumbrüche vorkommen. Sonst meckert `crontab`, wenn frau versucht, ihren Job abzuspeichern.

X-Clients in Cronjobs aufzurufen ist übrigens ein Problem: Wenn die jeweilige Userin nicht ohnehin grafisch eingeloggt ist, geht das schief.

Grafische Frontends wie `kcron` können dabei helfen, die ersten Hürden bei der gut in der Manpage zu `crontab(5)` erklärten Syntax zu überwinden.

Wenn Cron einen Job ausführt, schickt er (so nicht anders angegeben) die Ausgabe als Mail an die Auftraggeberin. Schon deshalb ist es wichtig, einen funktionierenden lokalen Mailserver aufzusetzen.

Die `/etc/crontab` enthält vor dem Kommando noch eine zusätzliche Spalte: die Userin, in deren Namen der Auftrag ausgeführt wird. Erschwerend kommt hinzu, dass `root` diese Datei von Hand bearbeiten muss – der `crontab`-Befehl übernimmt dies nicht. In dieser Datei wird meist ein `runparts`-Programm aufgerufen, der Skripte abarbeitet, die in den Verzeichnissen `/etc/cron.d/Daily` bis `/etc/cron.d/Monthly` (die Namen variieren von Distribution zu Distribution) abgelegt sind.

Aufgabe:

Wann wird der folgende Eintrag abgearbeitet?

```
0 13 * 9 5-7 echo 'Der Kurs geht weiter!'
```

Aufgabe:

Lass Dir in 10 Minuten von Cron eine Übersicht über die Plattenbelegung (`df`) schicken. Die aktuelle Zeit findest Du mit dem Kommando `date` heraus.

3 3. Tag

3.1 Die Bash

Als wir uns in Kapitel 1.2.2 mit der Shell beschäftigten, war eine der Aussagen die, dass Unix-Shells allgemein, speziell aber die Bash, die unter Linux die größte Rolle spielt, sehr mächtig seien. Einfaches Kommandos-Eintippen und Prozesse-in-den-Vorder-oder-Hintergrund-Schieben kann es da nicht gewesen sein.

In diesem Kapitel stellen wir einige Konzepte vor, die der Benutzerin viele Möglichkeiten an die Hand geben, allerdings auch Übung erfordern.

3.1.1 Eingebaute Befehle und Hilfsmittel

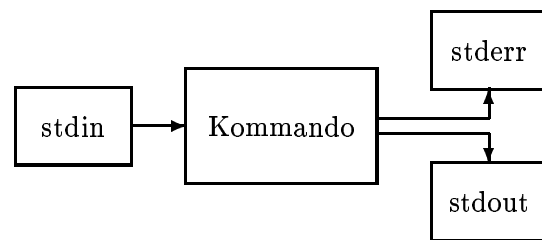
Tipphilfen Wie hieß der Befehl doch gleich? Irgendwas mit `l...` Ob frau nun zu faul zum Tippen oder einfach vergesslich ist, die *Tab-Komplettierung* für Befehle, sofern sie im Suchpfad stehen, ist eines der Features, mit denen die Bash auch gegenüber älteren Unix-Shells punktet. Einfach den Befehl soweit aufschreiben, wie frau sich erinnert, dann einmal die *Tab*-Taste drücken. Reichte die bislang eingegebene Zeichenkette aus, um ein Kommando im Suchpfad eindeutig zu identifizieren, ergänzt die Bash den Rest.

War die bisherige Eingabe noch nicht eindeutig genug, so listet die Bash die Alternativen nach einem Doppeltab auf. Frau vervollständigt soweit nötig und drückt noch einmal *Tab*. Die *Tab*-Komplettierung funktioniert nicht nur bei Kommandos, sondern auch bei Verzeichnissen und Dateiangaben.

Als weitere Hilfe für die geplagte Anwenderin verfügt die Bash über einen sogenannten *History-Mechanismus*, d. h., sie „merkt“ sich alle abgeschickten Kommandozeilen. Mit Hilfe der *Pfeiltasten* `↓` und `↑` kann frau diese Kommandos „wiederverwenden“.

Ein- und Ausgabekanäle Unix-Kommandos besitzen normalerweise drei Kommunikationskanäle:

- die Standardeingabe (stdin),
- die Standardausgabe (stdout) und
- die Standardfehlerausgabe (stderr).



Die *Standardeingabe* ist üblicherweise mit der Tastatur verbunden, damit die Benutzerin Eingaben machen kann. Die *Standardausgabe* und *Standardfehlerausgabe* sind mit dem Bildschirm verknüpft, d. h., Datenoutput und Fehlermeldungen erscheinen auf dem Bildschirm. Wie schon mehrfach erwähnt, handelt es sich bei Tastatur und Bildschirm unter Unix lediglich um Dateien. Normalerweise ist es einem Programm vollkommen egal, aus welcher Datei die Eingaben gelesen werden bzw. in welche Datei die Fehlermeldungen oder die Ausgabe geschrieben werden. Deswegen ist es problemlos möglich, die Kommunikationskanäle in andere Dateien umzulenken:

```
Unixkommando > Ausgabedatei
Unixkommando >> Ausgabedatei
```

lenkt stdout in Ausgabedatei um
lenkt stdout in Ausgabedatei um
und hängt die Information an eine
eventuell bestehende Datei an
lenkt stdin in Eingabedatei um
lenkt die Fehlermeldungen in eine
Datei um

```
Unixkommando < Eingabedatei
Unixkommando 2> Fehlerdatei
```

```
ls -lF > xxx
```

Die erzeugte Liste wird in xxx abgelegt.

```
mkdir test
rm test 2> fehler
```

In der Datei fehler steht jetzt die Fehlermeldung, dass test ein Verzeichnis ist und folglich nicht mit rm gelöscht werden kann.

Zur Abrundung dieses Themas zwei häufige Verwendungen für Umlenkungen:

- 2>/dev/null leitet die Fehlerausgabe des Programms ins Nulldevice, also ins Nichts.
- 2> &1 leitet die Fehlerausgabe auf die Standardausgabe um.

Die Pipe Häufig möchte man die Ausgabe eines Kommandos als Eingabe für das nächste Kommando verwenden, ohne dass die Zwischenergebnisse in Form einer Datei hinterher noch benötigt werden. Dafür bietet Unix die *Pipes* (Röhre, Rohrleitung).

Damit leitet frau die Standardausgabe eines Kommandos direkt auf die Standardeingabe des nächsten Kommandos um. Als Zeichen für die Pipe dient der senkrechte Strich: |

```
ls -lF /dev | more
grep bash /etc/passwd | less
```

Es ist möglich, beliebig viele Kommandos durch Pipes zu verbinden („pipeline“). Voraussetzungen zum Aufbau einer Pipe sind: Das erste Programm schreibt seine Ausgabe nach stdout, das zweite Programm liest seine Eingaben von stdin.

Programme, die ihre Eingabe von stdin bekommen, Operationen mit diesen Eingaben ausführen und das Ergebnis nach stdout schreiben, heißen *Filter* (z. B. `grep`, `cat`, `more`, `sort`).

Startdateien Ähnlich wie unter DOS, wo beim Starten des Rechners die Dateien `autoexec.bat` und `config.sys` abgearbeitet werden, werden auch beim Einloggen in ein Unix-System bestimmte Dateien abgearbeitet und Standardeinstellungen gesetzt.

Das ist zunächst die Datei `/etc/profile`, die von der Systemadministratorin gewartet wird und auf die normale Benutzerinnen keinen Schreibzugriff haben. Beim Einloggen auf der Textkonsole mit der Bash als Login-Shell (oder wenn eine Bash explizit als Login-Shell gestartet wird) werden anschließend die Dateien `.bash_profile`, `.bash_login` und `.profile` im Homeverzeichnis der Benutzerin abgearbeitet. Somit sind die Einstellungen festgelegt, die für die gesamte Sitzung gelten sollen. (Es sei denn, sie werden während der Sitzung von der Benutzerin ausdrücklich geändert.)

Bei interaktiven Nicht-Login-Shells werden die Session-bezogenen Voreinstellungen für die Bash geladen, indem die Datei `.bashrc`, so im Homeverzeichnis der Benutzerin vorhanden, ausgeführt wird. Diese Datei wird jedesmal ausgewertet, wenn eine neue Bash aufgerufen wird – sei es durch das Kommando `bash` oder die Eröffnung eines neuen Fensters.

Dateien wie `.profile` oder `.bashrc`, die mit einem Punkt beginnen, nennt frau auch *versteckte Dateien*, da sie normalerweise vom Kommando `ls` nicht angezeigt werden. Es handelt sich dabei in der Regel um Konfigurationsdateien für verschiedene Unix-Programme, die im Normalbetrieb meistens weniger von Interesse sind und die anzuzeigende Liste nur unnötig vergrößern.

In den o. g. Systemdateien werden insbesondere *Umgebungsvariablen* gesetzt. Frau kann sie sich mit dem Befehl `env` ausgeben lassen – sie werden in der Regel groß geschrieben. Zum Ausgeben eines einzigen Variableninhalts dient das Kommando

```
echo $variable
```

Dabei muss ein `$`-Zeichen direkt vor dem Variablennamen stehen.

```
echo $PATH
```

gibt den Pfad aus, in dem nach ausführbaren Dateien gesucht wird. Wenn frau jetzt ein zusätzliches Verzeichnis (beispielsweise das Verzeichnis `bin` im Homeverzeichnis) in den Pfad mit einbinden will, so kann sie das durch Neudefinition der Umgebungsvariablen tun, zum Beispiel durch das Kommando

```
export PATH=$HOME/bin:$PATH
```

Das abschließende `:$PATH` stellt sicher, dass der alte Pfad zusätzlich erhalten bleibt; die Variable `HOME` enthält den Pfad des eigenen Homeverzeichnisses. Wenn frau diese Zeile eingibt, gilt der neue Pfad natürlich nur für die aktuelle Shell in der aktuellen Sitzung. Beim nächsten Einloggen ist er vergessen. Deswegen kann frau das Kommando als zusätzliche Zeile in die Datei `.bashrc` und/oder `.bash_profile` im eigenen Homeverzeichnis eintragen.

Der so gesetzte Suchpfad wird von der Shell übrigens in genau der Reihenfolge durchsucht, in der die Verzeichnisse darin stehen. Sobald eine ausführbare Datei nicht in `PATH` steht, reicht es nicht mehr, ihren Namen zu kennen, frau muss sie unter Angabe des Pfades aufrufen.

Aufgabe:

Sieh Dir Deinen Suchpfad an und erweitere ihn um ein Verzeichnis mit ausführbaren Dateien, das bislang nicht enthalten ist (z. B. `/usr/sbin`). Starte eins der darin enthaltenen Programme (z. B. `/usr/sbin/lsof`) mit und ohne Pfad. Versuche dasselbe in einer anderen Shell.

Wildcards Soll sich ein Kommando auf mehrere Dateien beziehen, so kann frau sich eine Menge Schreibarbeit sparen, wenn sie auf sogenannte *Wildcards* („Platzhalter“) zurückgreift. Die wichtigsten sind im Folgenden aufgeführt:

<code>*</code>	steht für eine beliebige Zeichenkette,
<code>?</code>	steht für ein einzelnes Zeichen und
<code>[Zeichenmenge]</code>	steht für einen bestimmten Zeichensatz.
<code>cat p*</code>	zeigt den Inhalt aller Dateien an, die mit einem kleinen <code>p</code> beginnen
<code>cat p?</code>	zeigt den Inhalt aller zweibuchstabigen Dateien an, die mit einem kleinen <code>p</code> beginnen
<code>cat [pP]*</code>	zeigt den Inhalt aller Dateien an, die mit einem kleinen <code>p</code> oder einem großen <code>P</code> beginnen
<code>cat p*[1-3]</code>	zeigt den Inhalt aller Dateien an, die mit einem kleinen <code>p</code> beginnen und mit einer 1, 2 oder 3 enden

Aufgabe:

Übe den Gebrauch von Wildcards!

Aufgabe:

Suche eine Kombination, die Deinen, und den Namen Deiner rechten Nachbarin matcht, aber den Deiner linken Nachbarin nicht.

3.1.2 Shellskripte

Wildcards, Pipes, Ein- und Ausgabeumlenkung sind aber noch nicht alles, was das Power-Shell zu bieten hat – als „Killerfeature“ haben Unixshells alles eingebaut, was frau von einer Programmiersprache erwartet: Variablen, konditionale Abfragen, Schleifen.

Variablen Eine Variable in der Bash zu setzen ist ganz einfach: Frau denkt sich einen Namen für sie aus und setzt hinter ein Gleichheitszeichen den Wert. Um ihren Inhalt herauszufinden, gilt es, ein Dollarzeichen vor den Variablennamen zu setzen. Will frau den Inhalt ausgeben, benutzt sie das Kommando echo:

```
bash$ variable=wert
bash$ echo $variable
wert
```

Da Leerzeichen von der Shell als Worttrennzeichen benutzt werden, müssen sie geschützt werden, wenn sie Bestandteil eines Strings sind, der einer Variable zugewiesen wird. Dabei gibt es die „sanfte“ Möglichkeit, den gesamten String in doppelte Anführungszeichen (*Doppelquotes*) zu setzen oder die unnachgiebige Variante mit einfachen Anführungszeichen. Bei ersterer dient der Dollar weiterhin zum Herausfischen von Variableninhalten, in letzteren wird er buchstäblich interpretiert:

```
bash$ variable="wert2 $variable"
bash$ echo $variable
wert2 wert
bash$ variable='wert2 $variable'
bash$ echo $variable
wert2 $variable
```

Aufgabe:

Warum bekommt frau auf den Befehl `variable=wert2 $variable` die Antwort `bash: wert2: command not found`?

Das Verhalten der Bash und anderer Programme wird durch *Shell-* und *Umgebungsvariablen* beeinflusst. Erstere werden von der Shell selbst genutzt und können mit dem Kommando `set` angezeigt werden. Darunter fallen zum Beispiel die Variable `IFS`, in der festgelegt ist, welche Zeichen als Worttrennzeichen fungieren, oder `PS1`, die die Eingabeaufforderung (den *Prompt*) festlegt. Die Umgebungsvariablen erfährt frau mit dem Kommando `env`.

Aufgabe:

Wie ist der Prompt bei Dir definiert? Sieh in der Manpage zu `bash` an, welche Platzhalter sich darin verwenden lassen. Ändere Deinen Prompt temporär in `aktuelles_verzeichnis(rechnername)!`

Die vermutlich wichtigste vordefinierte Variable ist `PATH`, die den Suchpfad enthält. Darin sind – wie wir bereits gesehen haben – die nach ausführbaren Dateien zu durchsuchenden Verzeichnisse mit Doppelpunkt voneinander getrennt.

Eine andere Umgebungsvariable, `HOME`, enthält das Homeverzeichnis der Benutzerin. Kindprozesse der Shell erben ihr *Environment*, ihre Umgebung. Variablen, die einfach nur so auf der Kommandozeile gesetzt wurden, gehören nicht da hinein und werden daher mit ihren Inhalten auch nicht an Kindprozesse weitergegeben.

Will frau dafür sorgen, dass eine Variable mit dem aktuellen Inhalt auch auf Kindprozesse übergeht, muss sie sie mit dem Befehl `export` *exportieren*:

```
bash$ variable=wert    #Setzen der Variablen in der aktuellen Shell
bash$ echo $variable
wert
bash$ bash            #Starten einer Kindshell
bash$ echo $variable  #variable wurde nicht an Kindshell exportiert
bash$ exit            #Ausloggen aus der Kindshell
exit
bash$ echo $variable  #zurueck in der Ausgangshell
wert
bash$ export variable #Exportieren von variable
bash$ bash            #Neue Kindshell
bash$ echo $variable  #Kindshell kennt exportierte variable
wert
bash$ exit
exit
```

for-Schleife Es kommt recht häufig vor, dass frau mehrere Dateien gleichartig bearbeiten will. Einfache Kommandos nehmen oft mehrere Dateien als Argumente an, doch wenn die Aufgabe sich nicht mehr mit einem einzigen Kommando lösen lässt, wird es schwierig. Es sei denn, frau besinnt sich auf die `for`-Schleife:

```
for i in *.HTM; do mv $i 'basename $i HTM'html; done
```

Die Shell schaut nach, auf welche Dateien im aktuellen Verzeichnis das Wildcard-Muster *.HTM passt und legt deren Dateinamen einen nach dem anderen (d. h. bei jedem Schleifendurchlauf einen) in der Laufvariablen `i` ab.

Das war der Schleifenkopf und damit das erste Kommando. Will frau mehrere Kommandos hintereinanderweg auf der Kommandozeile schreiben, muss sie die mit Semikola trennen. Das zweite Kommando wird eingeleitet vom Schleifen-Schlüsselwort `do`, das den Beginn des Schleifenrumpfes markiert. In jedem Schleifendurchlauf soll also der Befehl `mv $i `basename $i HTM`html` ausgeführt werden.

Was soll da umbenannt werden? Offensichtlich die Datei, deren Name gerade in der Variablen `i` steht. Der neue Name endet auf `html`, doch was ist das, was zwischen den *Backticks* ' ' steht?

Wenn wir es einzeln anschauen, sieht es gar nicht mehr so fürchterlich aus: `basename $i HTM`. Das Kommando `basename` findet den einfachen Dateinamen seiner Argumentdatei heraus, wobei es sämtliche Pfadangaben wegstreicht. Der Basename von `/etc/passwd` ist zum Beispiel `passwd`.

Gibt frau diesem Kommando ein weiteres Argument mit auf den Weg, wird dies als Dateinamenendung interpretiert, die es ebenfalls wegzustreichen gilt. Im Beispiel hackt `basename` also die Endung `HTM` von der Datei in `i` ab, um anschließend eine neue Endung `html` anhängen zu können. Die Backticks sorgen lediglich dafür, dass `basename` zur Sache kommt, bevor das `mv`-Kommando ausgeführt wird.

Zu guter Letzt gilt es, den Schleifenrumpf mit `done` abzuschließen.

Natürlich muss niemand all diese Kommandos auf eine Zeile schreiben. Tippt frau in der Shell

```
for i in *.HTM
```

ein, erscheint ein sogenannter *Second-Level-Prompt*, der von der Umgebungsvariablen `PS2` bestimmt wird:

```
$ echo $PS2
>
```

Dieser erinnert daran, dass das Kommando noch unvollständig ist. Hinter diesem Prompt können wir weiterschreiben

```
> do mv $i `basename $i HTM`html
> done
$
```

und sparen uns so die Semikola. Schreiben wir

```
#!/bin/bash
```

```
for i in *.HTM
do
mv $i `basename $i HTM`html
done
```

in eine Datei, der wir Ausführbarkeitsrechte verleihen, haben wir ein kleines Programm, ein Shell-Skript, das sich wiederholt ausführen lässt.

Die erste Zeile des Skripts ist einem Kommentar vorbehalten, der besagt, welcher *Interpreter* den Rest ausführen soll – hier also `/bin/bash`. Natürlich kann frau ein Shellskript auch mit „normalen“ Kommentaren versehen, die der Dokumentation dienen. Einfach ein `#` davor, und schon kümmert sich die Shell nicht mehr um den Rest der Zeile.

Aufgabe:

Schreib ein kleines Skript, das mit Hilfe von `wc` herausfindet, wieviele Zeilen jede einzelne Datei in Deinem Homeverzeichnis lang ist.

cut In Shellskripten lassen sich grundsätzlich alle Kommandozeilenbefehle einsetzen, wobei eine Beschränkung auf Standard-Unixtools wie die in diesem Kurs vorgestellten dann angeraten ist, wenn das Skript auf verschiedenen Installationen laufen soll. (Soll es auch auf anderen Unixsystemen oder gar in einer anderen, von der Bourne-Shell abgeleiteten Shell ausgeführt werden, gilt es zudem, die verwendeten Befehlsoptionen zu überprüfen bzw. sich auf Syntaxelemente der Bash zu beschränken, die auch zum Repertoire der anderen Shell gehören.)

Ein in Shellskripten sehr nützlicher Befehl ist `cut`. Mit ihm kann frau aus dem Text einer Datei oder von der Standardeingabe Spalten extrahieren. Ob dies `byte-` bzw. `zeichen-`genau oder anhand von Feldern, die von *Delimitern* (Spaltentrennern) begrenzt werden, geschieht, ist abhängig von der angegebenen Option. Ohne explizite Angabe eines Spaltentrenners dienen Tabulatoren als Feldbegrenzer.

```
cut -d ":" -f 1,5 /etc/passwd
```

beispielsweise sucht aus der Passwort-Datei alle auf diesem Rechner vorhandenen Accounts und den dazugehörigen Kommentar heraus.

Aufgabe:

Schreib ein Shellskript, das für alle Benutzerinnen Deines Rechners eine namentliche Begrüßung ausgibt.

Aufgabe:

Was macht das Kommando `cut -b 6-17`? Was könnte der Unterschied zu `cut -c 6-17` sein?

3.2 Nützliches Kleinzeug

Taschenrechner Natürlich gibt es auch für Linux Unmengen von grafischen Taschenrechnerprogrammen, die sich wie `kcalc` oder `xcalc` auch weitgehend mit der Tastatur bedienen lassen.

Doch es geht natürlich auch vollkommen auf der Kommandozeile. Neben `calc` (das nicht überall installiert ist), heißt das Standardprogramm hier `bc` und ist keineswegs nur ein Taschenrechner, sondern eine Programmiersprache für sich. Wir beschränken uns in dieser Vorstellung allerdings auf den interaktiven Betrieb.

Nach dem Aufruf von `bc` rückt der Cursor auf die erste leere Zeile und wartet auf Eingaben. Die Enter-Taste gibt den Befehl zum Rechnen:

```
6.9+4.5
11.4
```

Dabei ist zu beachten, dass die Division per Default ganzzahlig berechnet wird:

```
6/4
1
6%4
2
```

Das lässt sich verändern, indem frau die Anzahl der auszugebenden Nachkommastellen mit der Spezialvariablen `scale` (der Default ist `scale=0`) festlegt:

```
scale=3
3.11/2
1.555
scale=1
3.11/2
1.5
3.11%2
.11
```

Die Modulo-Operation `%` gibt dabei jeweils den Rest aus, der bei der entsprechenden Genauigkeit übrig bleibt.

Mit den Pfeiltasten kann frau ältere „Befehle“ recyceln. Das jeweils letzte Ergebnis wird in der Variablen `last` gespeichert und kann durch Angabe dieses Variablennamens weiterverwendet werden:

```
8*4
32
last-30
2
```

Die Zahlensysteme für Ein- und Ausgabe legt frau für die aktuelle Session mit den Variablen `ibase` und `obase` fest. So rechnet der folgende Dialog die Zahl `FF` im hexadezimalen Zahlensystem ins oktale um:

```
ibase=16
obase=8
FF
377
```

Für komplexere Berechnungen sei die Manpage ans Herz gelegt.

Aufgabe:

Übe das Rechnen mit `bc`!

ASCII-Dateien schöner ausdrucken Um Textdateien papiersparender auszudrucken, kann frau auf die Filter `a2ps` oder `enscript` (je nachdem, was installiert ist oder was besser gefällt) zurückgreifen:

```
enscript -2 -M A4 textdatei
```

gibt *textdatei* zweiseitig im Hochformat auf A4-Papier (`-M` für „Medium“) auf dem Default-Drucker aus. Zusätzlich wird auf jeder Seite eine Headerzeile mit dem Namen der Datei, dem Ausdrucksdatum und der Seitennummer gedruckt.

```
a2ps -2 -M A4 textdatei
```

hingegen druckt zwei logische Seiten im Querformat A4 nebeneinander auf ein Blatt. Die logischen Seiten werden umrandet und mit hübschen Headern versehen. Zudem sorgt `a2ps` automatisch für Syntaxhighlighting, wenn es sich um Texte in einer gängigen Programmier- oder Auszeichnungssprache handelt.

Das kann `enscript` natürlich ebenfalls:

```
enscript -2 -M A4 -E -o textdatei.ps textdatei
```

Die `-o`-Option („output“) funktioniert auch bei `a2ps` und sorgt dafür, dass das Ergebnis nicht auf dem Defaultdrucker (bzw. dem hinter `-P` angegebenen Drucker) heraus kommt, sondern in einer PostScript-Datei abgelegt wird. Letztere kann frau mit PostScript-Viewern wie `gv`, `kghostview` oder `ghostview` anschauen.

Aufgabe:

Drucke eine Textdatei so in eine Datei, dass vier logische Seiten auf einer Druckseite zu liegen kommen. Schau sie mit einem PostScript-Viewer an!

Papier sparen mit `mpage` Auf vielen Systemen ist `mpage` installiert, das mehrere logische Seiten sowohl aus PostScript- als auch aus Textdateien ähnlich `a2ps` auf einer Druckseite ausgibt, sie bei Bedarf mit Rahmen umgibt und mit Header- und Fußzeilen versieht. Auch `mpage` kennt die Optionen `-o` und `-P`. Am Inhalt der auszudruckenden Seiten nimmt dieses Tool allerdings anders als `a2ps` und `enscript` keine Veränderungen vor.

Packen und Entpacken Beim Installieren von Software haben wir bereits mit dem „Tape Archiver“ `tar` Bekanntschaft geschlossen (vgl. Seite 42). Dieses Tool hat seinen Namen daher, dass es ursprünglich dazu entwickelt wurde, um Backups auf Bändern zu machen. Das geht auch heute noch, doch viel öfter wird `tar` dazu verwendet, um mehrere Dateien in einer (Archiv-)Datei zusammenzufassen. So packt

```
tar -cf archiv.tar .*
```

alle Punktdateien im aktuellen Verzeichnis zum Archiv `archiv.tar` zusammen (`-c` für „create“). Die Option `-f` muss unbedingt direkt vor dem Namen der geplanten Archivdatei stehen, denn sie entscheidet, dass `tar` nicht nach einem Bandlaufwerk sucht und stattdessen in eine Datei schreibt.

Das entstandene Archiv lässt sich mit

```
tar -tf archiv.tar
```

überprüfen (bei unbekanntem *Tarballs* ein wichtiger Schritt, um zu sehen, wo die darin enthaltenen Dateien beim Entpacken geschrieben werden) und mit

```
tar -xf archiv.tar
```

entpacken.

Die unter Linux verwendete GNU-Version von `tar` kann nach dem Ein- oder vor dem Auspacken einen (De-)Kompressionsschritt einfügen, um die Archivdatei klein zu halten. Dazu greift das Programm wahlweise auf die externen Komprimierungstools `gzip` oder (nur bei neueren Versionen von GNU-`tar`) `bzip2` zurück. So gibt

```
tar -tIf archiv.tar.bz2
```

den Inhalt eines mit `bzip2` gepackten *Tarballs* aus, während

```
tar -czvf archiv.tar.gz .*
```

das entstehende Archiv mit `gzip` packt. Die Option `-v` („verbose“) sorgt dafür, dass `tar` während der Arbeit ein paar mehr Informationen rausrückt.

```
tar --help|less
```

gibt eine Kurzfassung aller `tar`-Optionen aus, wobei `less` dafür sorgt, dass sie seitenweise lesbar sind.

Aufgabe:

Pack alle Dateien je zweier Unterverzeichnisse Deines Homeverzeichnis in ein `tgz`- und ein `tar.bz2`-Archiv. Prüfe die Archive, und pack sie in `/tmp` wieder aus. Pack die Archive anschließend in zwei Schritten in einem neu angelegten Verzeichnis aus: Erst dekomprimieren mit `gunzip` bzw. `bunzip2`, dann auspacken mit `tar`.

3.3 Netzwerk

Die Entwicklung des Internets und der Unixbetriebssystemfamilie hängt eng zusammen, und die Entwicklung von Linux ist ohne Netz ebenfalls nicht denkbar. So verwundert es nicht, dass Linux von vornherein mit allem ausgestattet ist, um als Client und/oder Server Teil des Internets zu sein. In diesem Abschnitt beschäftigen wir uns allerdings nicht mit den Grundlagen wie dem *TCP-IP-Stack*, die im Kernel implementiert sind und konfiguriert werden, sondern nur mit einigen wenigen Internet-Client-Programmen und Diagnosetools für die Kommandozeile.

Das Aufsetzen von Mail-, Web-, News- u. a. Internetservern könnte uns ebenso wie Netzwerkgrundlagen und das Konfigurieren des Internetanschlusses in mehreren Zusatzkursen beschäftigen, sodass wir es hier ebenfalls nicht ansprechen können.

3.3.1 Im Netzwerk bewegen

ssh Ähnlich wie das Unixurgestein **telnet** erlaubt es das Kommando

```
ssh rechneradresse
```

sich auf einem entfernten Rechner einzuloggen und dort so zu arbeiten, als säße frau lokal daran.

Wenn sie auf dem Remoterechner eine andere UserID hat, gibt sie diese mit der Option **-l** oder wie in einer E-Mailadresse vor dem **@**-Zeichen an:

```
ssh -l UserID rechneradresse  
ssh UserID@rechneradresse
```

ssh steht dabei für *SecureShell*, und der Name sagt bereits, warum sie **telnet** zum Einloggen auf anderen Rechner vorzuziehen ist: Beim Anmelden auf Remoterechnern via **telnet** o. ä. werden sämtliche Daten (einschließlich des Passworts!) im Klartext übertragen, d. h., sie können von Unbefugten abgehört werden. Dem beugt **ssh** vor, indem der gesamte Datenverkehr verschlüsselt wird. Netter Nebeneffekt: Frau kann (sofern sie lokal unter X arbeitet) auf dem Remoterechner Programme mit graphischer Oberfläche starten und bekommt sie auf dem lokalen Bildschirm angezeigt. Allerdings bieten Administrator/innen auch diese Eigenschaft zunehmend seltener an, denn dank Problemen mit X können sich hierbei Sicherheitslücken auftun.

Will frau sich auf ihrem Rechner per SecureShell einloggen, muss sie den SecureShell-Daemon **sshd** installieren und starten. Seine Konfiguration erfolgt über die ASCII-Datei `/etc/ssh/sshd_config`, während in `/etc/ssh/ssh_config` die globale Konfiguration des Clients vorgenommen wird.

Über **ssh** ist es auch möglich, andere Protokolle zu *tunneln*, sodass beispielsweise ein sicherer Zugriff auf einen POP-Server möglich wird.

scp Ebenfalls im SecureShell-(Client-)Paket enthalten ist **scp**, mit dem frau auf sicherem Weg Daten von Rechner zu Rechner kopieren kann:

```
scp lokale_datei UserID@rechneradresse:/tmp/
```

kopiert die Datei *lokale_datei* im aktuellen Verzeichnis ins */tmp*-Verzeichnis auf dem Remoterechner. Die UserID auf dem entfernten Rechner sowie das @ kann frau weglassen, wenn sie auf dem lokalen Rechner unter demselben Namen eingeloggt ist. Mit der Option *-r* („rekursiv“) lassen sich komplette Dateibäume fernkopieren:

```
scp -r UserID@rechneradresse:~/programme .
```

kopiert das Verzeichnis *programme* im Homeverzeichnis von *UserID* auf dem entfernten Rechner ins aktuelle Verzeichnis auf der lokalen Maschine.

Aufgabe:

Bitte Deine Nachbarin, ein Verzeichnis aus ihrem Homeverzeichnis auf ihrem Rechner in Deines zu kopieren.

ftp Bevor es *scp* gab, wurde das „File Transfer Protocol“ *ftp* zum „Fernkopieren“ von Daten benutzt. Dieser Einsatzzweck verliert an Bedeutung, da FTP ebenfalls alles im Klartext überträgt. Weiterhin populär ist das Protokoll allerdings als *Anonymous FTP* zum Herunterladen von Software u. ä. von FTP-Servern.

Als Username gilt für diese Lese-Zugriffe auf die Dateihierarchie des FTP-Servers zumeist *ftp* oder *anonymous*, als Passwort die eigene E-Mailadresse. Beides schicken (grafische) FTP-Clients oder Webbrowser gern von sich aus an den Server.

Will frau hingegen mit dem klassischen Kommandozeilenclienten *ftp* auf FTP-Server zugreifen, muss sie beides von Hand eingeben. Bequemer ist hier der oft installierte FTP-Client *ncftp*.

Im FTP-Clients gibt frau FTP-Befehle ein, die meist auch auf das Kommando *help* hin aufgelistet werden. So wechselt *cd* das Verzeichnis auf dem Server, *bin* schaltet in den binären Übertragungsmodus, der dafür sorgt, dass auch Nicht-Textdateien heil bei der Empfängerin ankommen, *lcd* wechselt das Verzeichnis auf dem lokalen Rechner, *ls* oder *dir* listet den Verzeichnisinhalt des aktuellen Verzeichnisses auf dem Server auf, *get* lädt die als Argument angegebene Datei herunter und *mget* mehrere, auch durch Wildcards spezifizierte.

Aufgabe:

Log Dich per *ftp* oder *ncftp* (am besten zum Vergleich mit beiden) auf dem KDE-FTP-Server *ftp.kde.org* ein und lade zwei Dateien in einem Rutsch herunter.

3.3.2 Wer ist wo online?

Solange das Netz geht, ist alles in Ordnung. Wenn nicht, gibt es jede Menge Diagnose-tools, mit denen frau den Problemen auf die Sprünge kommt. Einige gängige stellen wir kurz vor.

Erreichbarkeit im Netz prüfen Unsicher, ob frau überhaupt online ist oder ob der eigene Netzzugang funktioniert?

`ping rechneradresse`

spielt mit einem entfernten Rechner „Ping-Pong“, indem es ein Paket hinsendet und eins zurück haben will.

Aufgabe:

Versuche, den Rechner Deiner Nachbarin anzupingen. Denke Dir einen Phantasierechnernamen aus und versuche, diesen zu erwischen. Achte auf die Unterschiede bei der von `ping` ausgegebenen Statistik!

Den Weg zum Ziel finden Welchen Weg (welche *Route*) nimmt ein Paket momentan zu einem Zielrechner? Hierauf gibt `traceroute` die Antwort. In der ausgegebenen Statistik werden auch Engpässe sichtbar.

Aufgabe:

Lass Dir die Route zu einem Rechner Deiner heimischen Hochschule oder Deines privaten Internetproviders ausgeben!

DNS-Informationen holen Auch wenn wir uns mit dem „Domain Name System“ in diesem Kurs nicht beschäftigen können: Die Informationen, die ein Internetclient von einem *Nameserver* anfordert, um zu einer textuellen Rechneradresse die passende numerische IP-Adresse zu bekommen (oder umgekehrt, was sich dann *reverse lookup* nennt), oder den Mailserver herauszufinden, bei dem er die abzuschickende E-Mail abliefern soll, stehen auch den Usern zur Verfügung. So finden die Befehle

```
host -t mx informatica-feminale.de
dig mx informatica-feminale.de
```

heraus, welcher Rechner Mail für die Domain `informatica-feminale.de` entgegen nimmt (`mx` steht für „Mail Exchanger“).

```
host www.informatica-feminale.de
```

sucht die numerische IP-Adresse des Rechners `www.informatica-feminale.de`.

```
host -a informatica-feminale.de nameserver
dig @nameserver any informatica-feminale.de
```

finden alle DNS-Informationen über die Domain `informatica-feminale.de` heraus, fragen dazu allerdings nicht den Default-Nameserver, sondern `nameserver`.

Aufgabe:

Finde die numerische IP-Adresse des Servers heraus, den Du normalerweise als Mailserver benutzt.

Aufgabe:

Besitzt Du selbst eine Domain? Finde heraus, welcher Rechner die Mail für diese Domain annimmt. (Wenn Du selbst keine Domain hast, probiere `answergirl.de` aus!)

Interessiert eher, wer eigentlich hinter einer Domain steckt, gibt `whois` die Antwort, indem es in der Datenbank eines *Network Information Center* (NIC) nachschlägt. Da der als Default eingestellte Whois-Rechner `whois.networksolutions.com` keine Informationen zu de-Domains hat, ist es wichtig, jeweils den richtigen Server zu fragen. Leider sind unter Linux zwei Versionen des `whois`-Programms im Umlauf, die sich unterschiedlich bedienen lassen – welche, sollte ein Versuch oder der Blick in die Manpage verraten.

```
whois -h whois.ripe.net informatica-feminale.de
whois informatica-feminale.de@whois.ripe.net
```

fragt `whois.ripe.net` nach den Registrierungsangaben für die Domain `informatica-feminale.de`.

Aufgabe:

Schlage die Registrierungsinformationen zu Deiner Domain oder der Deines Internetproviders nach!

Aufgabe:

Welche Domain würdest Du gern registrieren? Schau nach, ob es sie schon gibt!

Who is who? Wer ist eigentlich sonst noch alles auf meinem Rechner eingeloggt? `who` gibt die Antwort. Und sollte nach einer durchwachten Nacht unklar sein, wer das eigentlich ist, die da vor dem Rechner sitzt, beantwortet der Rechner auch die Frage `who am i`.

Aufgabe:

Was ist der Unterschied zwischen `who am i` und `whoami`?

Informationen über Accounts auf anderen Rechner gibt das `finger`-Kommando, das allerdings aus Datenschutzgründen immer öfter ohne Antwort auskommen muss.

Aufgabe:

Prüfe, ob Du mit `finger UserID` und `finger UserID@remoterechner` Informationen zu Deinem Account lokal und auf einem anderen Poolrechner heraus finden kannst.

Aufgabe:

„Fingere“ Deinen Uni-Account an! Bekommst Du eine Antwort?

4 Anhang: Weiterführende Ressourcen

Informationen und Downloads Eine naturgegebener Maßen viel zu kurz geratene Zusammenstellung von Adressen, die uns in der Vergangenheit weitergeholfen haben, und die wir daher gerne weiter empfehlen.

Mailinglisten

- *Lynn*:
<http://lists.answergirl.de/>

Die unter *lynn@lists.answergirl.de* erreichbare Linuxerinnenmailingliste bietet die Möglichkeit, sich unter Frauen über Linux auszutauschen. Einsteigerinnen sind ebenso gern gesehen wie „Profis“. Die Anmeldung erfolgt über die angegebene Webseite oder per Mail mit dem Subject: `subscribe lynn` an *listar@lists.answergirl.de*.

Bitte sei so nett und stell Dich in einer ersten Mail an die Liste kurz vor. Das Listenarchiv unter <http://lists.answergirl.de/lynn/> ist passwortgeschützt und steht allen Frauen offen, die mit mindestens einem Posting (Vorstellungsmail und/oder fachlicher Beitrag) auf der Liste sichtbar geworden sind.

Portale

- <http://www.linux.de/>
- <http://www.linuxberg.de/>
- <http://www.go-linux.de/>
- <http://www.linux-community.de/>

Infos

- *Das Linux-Anwenderinnen-Handbuch*:
<http://www.linux-ag.de/linux/LHB/>
- *Linux-User*
<http://www.linux-user.de/>
- *Linux-Praxis*
<http://www.linux-praxis.de/>

Software

- <http://www.freshmeat.net/>

Distributionen

- *SuSE*:
<http://www.suse.de/>
- *Debian*:
<http://www.debian.de/>
- *Red Hat*:
<http://www.redhat.de/>
- *Mandrake*
<http://www.linux-mandrake.com/>
- *Linux From Scratch*:
<http://www.linuxfromscratch.org/>

Spaß für Programmiererinnen

- *Gnu-Humor-Seite*
<http://www.gnu.org/fun/humor.html>

Zu den Kursteilen

- **LSB**
 - *Linux Standard Base*:
<http://www.linuxbase.org/>
 - *Filesystem Hierarchy Standard*:
<http://www.pathname.com/fhs/>
- **Geschichte**
 - *Die Software-Rebellen*:
<http://www.heise.de/tp/deutsch/inhalt/buch/7996/1.html>
 - *Geschichten aus grauer Vorzeit*:
Nr. 1: <http://portal.suse.de/de/content.php?SEARCH&content/linuxology/rbhistory.html>
Nr. 2: <http://portal.suse.de/de/content.php?SEARCH&content/linuxology/rbhistory2.html>
 - *Geschichte freier Software*:
<http://www.kritische-informatik.de/fshistl.htm>
- **Drucken**
 - <http://www.linuxprinting.org/>

Freie Software

- *GNU*
<http://www.gnu.org/>
- *Die Kathedrale und der Basar*
<http://www.linux-magazin.de/ausgabe/1997/08/Basar/basar.html>

Index

- |, 47
- ., 26
- .., 26
- .bash_login, 48
- .bash_profile, 48
- .bashrc, 48
- .profile, 48
- .xinitrc, 13
- .xsession, 13
- /bin, 17
- /dev/null, 47
- /etc, 17
- /etc/crontab, 45
- /etc/fstab, 18, 24
- /etc/group, 38
- /etc/mstab, 25
- /etc/passwd, 53
- /etc/profile, 48
- /etc/services, 14
- /etc/shadow, 39
- /etc/skel, 40
- /lib, 17
- /sbin, 17
- /usr/doc, 20
- ;, 52
- \$, 48
- &, 43
- <, 47
- >, 47
- >>, 47
- 2>, 47

- a2ps, 55
- Abmelden, 9
- Abschießen
 - von Prozessen, 44
- Account, 38
- adduser, 40
- Affengriff, 19
- Alias, 32
- alien, 40

- Anführungszeichen
 - doppelte, 50
 - einfache, 50
- Anonymous FTP, 58
- Apache, 14
- apropos, 21
- apt-get, 40
- Arbeitsspeicher, 24
- ASCII-Dateien, 26
- Ausführbarkeitsrecht, 30
- Ausgabeumlenkung, 47
- Ausloggen, *siehe* Abmelden
- Ausschalten, 19
- autoexec.bat, 48

- Backtick, 52
- Backus-Naur-Schreibweise, 21
- Bandlaufwerk, 56
- basename, 52
- bash, 8, 20, 46–50
- bc, 53
- Benutzername, 8
- Betriebssystem, *siehe* Kernel
- Betriebssystemkern, *siehe* Kernel
- bg, 44
- Binär-Dateien, 26
- Bind, 14
- Bootloader, *siehe* Bootmanager
- Bootmanager, 7, 16
- Bourne-Shell, 8, 53
- Bug-Tracking, 43
- bunzip2, 56
- bzip2, 42, 56

- calc, 53
- Caldera, 4
- cat, 33, 48
- cat page, 21
- CDROM
 - Dateisystem, 22
- chgrp, 31
- chown, 31, 39

- config.sys, 48
- configure, 42
- Copy&Paste, 12
- crack, 39
- Cron-Daemon, *siehe* crond
- crond, 13, 14, 45
- cut, 53

- Daemon, 13
- date, 46
- Datei, 26
 - Endung wegstreichen, 52
 - Größe, 29
 - kopieren, 32
 - löschen, 32
 - Punktdateien, *siehe* versteckte Dateien
 - Rechte, 29
 - umbenennen, 32
- Dateisysteme, 22
 - DOS, 5
 - gemountete, 25
- Datum der letzten Änderung, 29
- Debian, 4
 - Paketmanager, 40
- Default
 - Drucker, 55
 - Nameserver, 60
 - Runlevel, 17
 - Suchpfad, 40
- Deinstallieren
 - rpm-Pakete, 41
- Delimiter, *siehe* Spaltentrenner
- Desktop
 - Environments, *siehe* Desktop-Umgebungen
 - Umgebungen, 12–13
- Device, 24
- df, 25, 46
- dig, 59
- Display-Manager, 13, 19
- Distribution, 4, 5
- Division, 54
- dmesg, 19
- DNS, 59

- do-done, 52
- Dokumentation, 20–62
- Doppelquotes, 50
- DOS, 5, 48
 - Dateisystem, 23
- dpkg, 40
- Drag&Drop, 12
- Drucken, 33
 - Textdateien, 55
- Drucker, 26
 - Daemon, *siehe* lpd
- dump, 25

- EasyLinux, 17
- echo, 48, 50
- ed, 34
- EDITOR, 34, 45
- Editor, 34
- Eingabeaufforderung, 50
- Eingabeumlenkung, 47
- Elternprozess, 44
- elvis, 34
- Emacs, 36–37
- enscript, 55
- Entpacken, 56
- env, 48, 50
- Environment, 51
 - Variablen, *siehe* Umgebungsvariable
- Erreichbarkeit
 - im Netz prüfen, 59
- evilwm, 12
- ex, 35
- Exim, 14
- exit, 9
- export, 51
- Export von Variablen, 51
- Ext2, 23
- ext2fs, 18, 22

- Fehlerumlenkung, 47
- Festplatte, 22
- fg, 44
- FHS, 27
- Filesystemcheck, 18

- Filter, 48
- find, 32
- finger, 60
- Flags, 15
- for-Schleife, 51
- Framebufferkonsole, 10
- FREAX, 4
- Free Software Foundation, *siehe* FSF
- FSF, 15
- fstab, *siehe* /etc/fstab
- ftp, 58
- fvwm2, 12

- gdm, 19
- gedit, 38
- Geräte-Datei, 24, 26
 - blockorientiert, 26
 - zeichenorientiert, 26
- getty, 19
- ghostview, 55
- GNOME, 4, 12
- GNU Hurd, 15
- GNU-Projekt, 15
- GPL, 15
- grep, 48
- groupadd, 40
- groupdel, 40
- GroupID, 9
- groups, 9
- Grub, 16
- Gruppe, 9, 29, 38
 - anlegen, 40
 - löschen, 40
- GTK, 12
- gunzip, 56
- gv, 55
- gzip, 42, 56

- halt, 19
- hexadezimal rechnen, 54
- Hilfe
 - zu tar, 56
- Hintergrundprozess, 43
- History, 46

- HOME, 51
- Homeverzeichnis, 38
 - anlegen, 39
- host, 59

- ibase, 54
- id, 9
- IFS, 50
- inetd, 14
- info, 21
- Informationen, 62
 - über rpm-Pakete, 41
- init, 17, 19, 44
- Init-System, 17
- inittab, 17, 19
- Inode, 22
- Installationsprogramm, 4
- Installieren
 - von rpm-Paketten, 41
- Internet, 57
- Interpreter, 53

- jobs, 44
- joe, 38
- Journaling-Filesystem, 23, 24
- jove, 38

- kate, 38
- kcalc, 53
- kcron, 45
- KDE, 4, 12
- kdm, 19
- Kernel, 5–7
 - Kompilieren, 7
 - Konfigurieren, 7
 - modular, 6
 - Module, *siehe* Loadable Modules
 - monolithisch, 6
- kerneld, 13
- kghostview, 55
- kill, 44
- killall, 45
- Kommando
 - Interpreter, *siehe* Shell
 - Optionen, *siehe* Optionen

- Kommentar, 53
- Kompilieren
 - Kernel, 7
- Komprimieren, 56
- Konfigurieren
 - Kernel, 7
- Kopieren
 - auf andere Rechner, 57
- kwrite, 38

- Löschschutz, 30
- last, 54
- Leserecht, 30
- less, 21, 33, 56
- LILO, 7
- Lilo, 16–17
- Line Editor, 34
- Link, 26, 30
 - Hard Link, 26
 - Soft Link, 26
- Linux Standard Base, *siehe* LSB
- linuxconf, 14
- Loadable Modules, 6
- locate, 32
- Login-Shell, 9, 38, 48
- lpd, 33
- lpq, 33
- ls, 42
- LSB, 27

- Mail Transfer Agent, *siehe* Mailserver
- Mailserver, 14, 45
- Major-Nummer, 29
- make, 42
- Makefile, 42
- man, 20
- Mandrake, 4
- Manpage, 20
- Maus, 26
- mehrspaltiger Druck, 55
- Meta-Taste, 36
- Minix, 4, 22, 23
- Minor-Nummer, 29
- Module
 - ladbare, *siehe* Loadable Modules
 - more, 21, 33, 48
 - mounten, 18, 24
 - read-only, 18
 - read-write, 18
 - Mountpoint, 24
 - mpage, 55
 - MTA, *siehe* Mail Transfer Agent
 - Multitasking, 5
 - Multiuser, 5

- Named Pipe, 27, 30
- Nameserver, 59
- nedit, 38
- NFS, 23
- NIC, 60
- NIS, 39
- NTFS, 23
- Nulldevice, 47
- nvi, 34

- obase, 54
- oktal rechnen, 54
- Online-Dokumentation, *siehe* Dokumentation
- Optionen, 15

- Packen, 56
- Pager, 21, 33
- Paketmanager, 40
- Partition, 22
- Passwort, 8, 38
 - ändern, 39
 - knacken, 39
- PATH, 48, 51
- Permissions, *siehe* Rechte
- Pfad, 27, 48
 - absoluter, 28
 - relativer, 28
- pico, 38
- PID, 44
- pine, 38
- ping, 59
- Pipe, 47
- Plattenplatz, 25

- anzeigen, 46
- Port, 14
- Postfix, 14
- PostScript, 55
 - Viewer, 55
- Primärgruppe, 38
- PRINTER, 33
- proc-Dateisystem, 23, 24
- Prompt, 8, 50
- Prozess, 43
 - Identifikationsnummer, *siehe* PID
- ps, 44
- PS1, 50
- pstree, 44
- Punktdateien, *siehe* versteckte Dateien

- Qmail, 14
- Qt, 12
- Queue, 33
- Quotes, *siehe* Anführungszeichen

- rc.boot, 18
- rc.config, 18
- reboot, 19
- Rechte, 9, 29
 - ändern, 30
- Red Hat, 4
- Red Hat Packet Manager, *siehe* rpm
- ReiserFS, 22, 23
- rekursives Akronym, 15
- remote einloggen, 57
- reverse Lookup, 59
- RIPE, 60
- rm, 32
- rmdir, 32
- root, 9, 28
- Root-Partition, 17
- Route, 59
- rpm, 40–42
- Runlevel, 17–19
- Runterfahren, 19

- s-Bit, 30
- scale, 54
- Scheduling, 5

- Schleife, 51
- Schreibrecht, 30
- scp, 57
- SecureShell, 14, 57
 - Sicherheit, 57
 - tunneln, 57
- Sektion, 21
- Sendmail, 14
- serielle Schnittstelle, 26
- set, 50
- sh, 8
- Shell, 7–8
 - Skript, 53
 - Variablen, 50
- shutdown, 19
- Sicherheit
 - im Zusammenhang mit `ssh`, 57
- Simple Init, 17
- Slackware, 4, 17
- smail, 14
- smtpd, *siehe* Mailserver
- Socket, 30
- Sockets, 27
- sort, 48
- Spaltentrenner, 53
- Speicherschutz, 6
- ssh, *siehe* SecureShell
- sshd, 14, 57
- Stallman, Richard, 15
- Standard
 - Ausgabe, *siehe* `stdout`
 - Eingabe, *siehe* `stdin`
 - Fehlerausgabe, *siehe* `stderr`
- Start-Stopp-Skripte, 17
- Startdateien, 48
- startkde, 13
- startx, 13, 19
- stderr, 47
- stdin, 47
- stdout, 47
- stow, 42, 43
- Strg+D, 9
- strings, 24
- Suchpfad, *siehe* Pfad

- SuSE, 4
 - Besonderheiten, 18
- switchdesk, 13
- Syntaxhighlighting, 55
- syslogd, 13
- System calls, 7
- System-V-Init, 17
- Systemadministratorin, 9

- Tab-Kompletierung, 46
- Tab-Taste, 20
- tar, 42, 56
 - tar.gz-Archiv, 42
 - Tarball, 56
- Taschenrechner, 53
- TCP-IP-Stack, 57
- telinit, 17, 19
- Telnet, 14
- telnet, 57
- telnetd, 14
- Texteditor, *siehe* Editor
- tgz-Archiv, 42
- Tochterprozess, 44
- Torvalds, Linus, 4, 15
- traceroute, 59
- tune2fs, 18, 23
- tunneln
 - über ssh, 57

- UID, *siehe* UserID
- umask, 31
- Umbenennen, 32
- Umgebungsvariable, 48, 50
- Updaten
 - rpm-Pakete, 41
- User anlegen, 39
- useradd, 39
- userdel, 40
- UserID, 8
- usermod, 40

- Valid Flag, 23
- Variablen, 50
- versteckte Dateien, 48, 56
- Verzeichnis, 26, 30
 - anlegen, 32
 - Inhalt auflisten, 28
 - löschen, 32
 - Recht zum Betreten, 30
 - wechseln, 27
- Verzeichnisbaum, 27
- vi, 34–36, 45
- vim, 34
- virtuelle Konsole, 19
- virtueller Desktop, 12
- VISUAL, 34, 45
- Vordergrundprozess, 43

- Warmstart, 19
- wc, 42
- Webmin, 14
- Webserver, 14
- who, 60
- who am i, 60
- whoami, 60
- whois, 60
- Wildcards, 49, 52
- Windowmanager, 11, 12
- Windows
 - 3.x, 5
 - 9x, 6
 - Dateisystem, 23
- Worttrennzeichen, 50

- X-Window-System, 4, 10
- xcalc, 53
- xdm, 19
- XF86Config, 11
- XFree86, 10

- Zeileneditor, *siehe* Line Editor
- Zmailer, 14