Inhalt

[Einführung: Das Betriebssystem Linux 1](#_Toc474744999)

[Aufbau des Betriebssystems 1](#_Toc474745000)

[Die Benutzer 2](#_Toc474745001)

[Die Dateien 2](#_Toc474745002)

[Das Dateisystem 3](#_Toc474745003)

[Die Verzeichnisstruktur 3](#_Toc474745004)

[Das Wurzelverzeichnis / 4](#_Toc474745005)

[Das Verzeichnis /bin 4](#_Toc474745006)

[Das Verzeichnis /boot 4](#_Toc474745007)

[Das Verzeichnis /dev 4](#_Toc474745008)

[Das Verzeichnis /etc 4](#_Toc474745009)

[Das Verzeichnis /home 4](#_Toc474745010)

[Das Verzeichnis /lib 5](#_Toc474745011)

[Die Verzeichnisse /mnt und /media 5](#_Toc474745012)

[Das Verzeichnis /proc 5](#_Toc474745013)

[Das Verzeichnis /root 5](#_Toc474745014)

[Das Verzeichnis /sbin 5](#_Toc474745015)

[Das Verzeichnis /tmp 5](#_Toc474745016)

[Das Verzeichnis /usr 5](#_Toc474745017)

[Das Verzeichnis /var 6](#_Toc474745018)

[Das Verzeichnis /opt 6](#_Toc474745019)

[Die Linux-Distributionen 6](#_Toc474745020)

[Debian 7](#_Toc474745021)

[Fedora 7](#_Toc474745022)

[Gentoo Linux 8](#_Toc474745023)

[OpenSUSE 8](#_Toc474745024)

[Slackware 8](#_Toc474745025)

[Ubuntu 8](#_Toc474745026)

[Raspbian 8](#_Toc474745027)

[Linux Mint 9](#_Toc474745028)

[Linux Desktops 9](#_Toc474745029)

[Gnome vs. KDE 9](#_Toc474745030)

[XFCE 9](#_Toc474745031)

[MATE und Cinnamon 9](#_Toc474745032)

[LXDE 9](#_Toc474745033)

[PIXEL 10](#_Toc474745034)

[BASH-Grundkurs 10](#_Toc474745035)

[Bash? Wozu brauche ich das?! 10](#_Toc474745036)

[Die Bedeutung verschiedener Schriften: 10](#_Toc474745037)

[Noch einige Tipps: 11](#_Toc474745038)

[Kapitel 1: Das System erkunden 11](#_Toc474745039)

[Betrachten, Navigieren, Suchen - pwd, ls, cd und find 11](#_Toc474745040)

[Textdateien lesen – less & tail 14](#_Toc474745041)

[Kapitel 2: Dateimanipulationen 14](#_Toc474745042)

[Dateien anlegen und kopieren - touch, mkdir und cp 14](#_Toc474745043)

[Verschieben und umbenennen - mv 16](#_Toc474745044)

[Dateien löschen - rm und rmdir 16](#_Toc474745045)

[Links erzeugen - ln 17](#_Toc474745046)

[Kapitel 3: Dateirechte 18](#_Toc474745047)

[Dateirechte ändern - chmod 18](#_Toc474745048)

[1. ugoa-Methode 19](#_Toc474745049)

[2. numerische Methode 20](#_Toc474745050)

[Besitzer ändern - chown und chgrp 21](#_Toc474745051)

[Kapitel 4: Das System konfigurieren 21](#_Toc474745052)

[Mit alias einen neuen Befehl erzeugen 21](#_Toc474745053)

[Mit nano die Konfigurationsdatei .bashrc editieren 22](#_Toc474745054)

[Umgebungsvariablen 23](#_Toc474745055)

[Aufruf von Programmen - der Suchpfad $PATH 24](#_Toc474745056)

[Kapitel 5: Umleitungen 25](#_Toc474745057)

[Ausgabe in Datei umleiten - > und >> 25](#_Toc474745058)

[Fehlerausgabe von normaler Ausgabe trennen 26](#_Toc474745059)

[Inhalt einer Datei für die Eingabe nutzen - < 26](#_Toc474745060)

[Rohrleitungen zu einem anderen Programm - | 27](#_Toc474745061)

[Auslesen und filtern - cat und grep 27](#_Toc474745062)

[Kapitel 6: Systemadministration 28](#_Toc474745063)

[Der Superuser - su und sudo 28](#_Toc474745064)

[Kapitel 7: Hilfe! 30](#_Toc474745065)

[Programmdokumentationen - man, info und help 30](#_Toc474745066)

# Einführung: Das Betriebssystem Linux

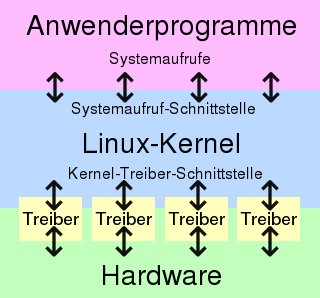
Linux ist ein freies Multiplattform-Mehrbenutzer-Betriebssystem, das den Linux-Kernel enthält. Im praktischen Einsatz werden meist sogenannte Linux-Distributionen genutzt, in denen der Linux-Kernel und verschiedene Software zu einem fertigen Paket zusammengestellt sind.

## Aufbau des Betriebssystems

Das zentrale Kernstück des Betriebssystems, der Linux-Kernel (meist nur Kernel genannt) bildet eine Trennschicht zwischen Hardware und Anwenderprogrammen. Das heißt, wenn ein Programm auf ein Stück Hardware zugreifen will, so kann es niemals direkt darauf zugreifen, sondern nur über das Betriebssystem.

Dazu bedient sich das Programm der Systemaufrufe. Über den Systemaufruf teilt das Anwenderprogramm dem Betriebssystem mit, dass es etwas zu tun gibt. Will etwa ein Programm eine Zeile Text auf dem Bildschirm ausgeben, so wird ein Systemaufruf gestartet, dem der Text übergeben wird. Das Betriebssystem erst schreibt ihn auf den Bildschirm.

Auf der anderen Seite muss das Betriebssystem die Möglichkeit haben, mit den einzelnen Hardware-Komponenten zu sprechen. Mittels seiner Treiberschnittstelle spricht es spezielle Geräte-Treiber an. Erst die Treiber kommunizieren dann direkt mit den Geräten.



Aufbau des Betriebssystems

Zu den Anwenderprogrammen zählen alle von uns gestarteten Programme (Videoplayer, Webbrowser ...), wie auch die grafische Oberfläche des Betriebssystems, das Desktop-Environment. Letzteres ist nicht ein Programm, sondern eine Sammlung von Programmen, die zusammen die gewohnten Funktionalitäten beisteuern.

Ein ganz spezielles Anwenderprogramm ist die Shell - die "Benutzeroberfläche". Es existieren viele verschiedene Shells - wir werden hier mit der Bash (Bourne again shell) arbeiten. Diese ist die Standardshell auf Linux Systemen. Alle Shells stellen dem Benutzer eine Kommandozeile zur Verfügung, mit der Befehle eingegeben werden können, die direkt als Systemaufrufe an das Betriebssystem weitergeleitet werden.

Linux ist ein Multitasking-Betriebssystem: das heißt, es können mehrere Prozesse - so nennt man Programme, sobald sie in den Speicher geladen sind und laufen - gleichzeitig laufen. Das bedingt, dass das System die verfügbare Rechenzeit des Prozessors in kleine Zeitscheiben aufteilt (im Millisekunden Bereich), die dann den jeweiligen Prozessen zur Verfügung stehen. Diese Aufgabe übernimmt eine übergeordnete Instanz - der Scheduler. Dieser verwaltet die Zuteilung der Zeitscheiben an die verschiedenen Prozesse.

Daher kann kein Prozess die ganze Rechenleistung für sich beanspruchen und auch ein "hängender" Prozess kann nicht das ganze System lahmlegen.

## Die Benutzer

Linux ist auch ein Multiuser-System, das heißt, es können mehrere Benutzer an verschieden Terminals auf demselben Rechner arbeiten. Dazu ist es natürlich notwendig, dass jeder Benutzer eindeutig identifiziert ist. Die User (engl., Benutzer) werden zwar mit ihren Namen verwaltet, intern arbeitet ein Unix-System aber mit Usernummern. Jeder Benutzer hat also eine Nummer welche UserID oder kurz UID genannt wird.

Jeder Benutzer ist auch Mitglied mindestens einer Gruppe. Es kann beliebig viele Gruppen in einem System geben und auch sie haben intern Nummern (GroupID oder GID). Im Prinzip sind Gruppen nur eine Möglichkeit, noch detailliertere Einstellungsmöglichkeiten zu haben, wer was darf.

Eine spezielle Rolle hat der Benutzer mit der UserID 0 - er ist Root (engl., Wurzel). Root steht außerhalb aller Sicherheitseinrichtungen des Systems - kurz - er darf alles. Er kann mit einem Befehl das ganze System zerstören, er kann die Arbeit von Wochen und Monaten löschen usw. Aus diesem Grund meldet sich auch der Systemverwalter im Normalfall als normaler Benutzer an - zum Root-Benutzer wird er nur dann, wenn er Systemverwaltungsarbeiten abwickelt, die diese Identität benötigen.

## Die Dateien

Datei- und Verzeichnisnamen können bis zu 256 Zeichen lang sein. Dabei wird in jedem Fall zwischen Groß- und Kleinschreibung unterschieden. Die Dateinamen

DATEI   
datei   
Datei

bezeichnen drei unterschiedliche Dateien. Ein Dateiname darf beliebig viele Punkte enthalten, also zum Beispiel auch Datei.Teil.1.txt. Ein Punkt gilt als normales Zeichen in einem Dateinamen. Dateien, die mit Punkt beginnen, gelten als versteckt und werden normalerweise nicht angezeigt - zum Beispiel .datei. Das Zeichen zum Trennen von Verzeichnis- und Dateinamen ist der Slash ("/") statt dem Backslash ("\") bei Windows.

Es gibt verschiedene Dateiarten:

(in Klammer die offizielle Darstellung, wie sie symbolisiert werden)

Normale Dateien (-)  
Verzeichnisse (d)  
Symbolische Links (l)  
Blockorientierte Geräte (b)  
Zeichenorientierte Geräte (c)  
Named Pipes (p)

Wir sehen hier schon, dass auch Verzeichnisse bloß eine bestimmte Dateiart sind. Eine spezielle nämlich, in der andere Dateien aufgelistet sind. Mit einem Dateibrowser (von Windows kennen wir "Explorer", bei Apple den "Finder") sehen wir uns immer nur genau diese Verzeichnisse an, sofern wir nicht mittels verschiedener Plugins die Dateien selbst auswerten und Textdokumente, Bilder anzeigen oder Videos und Musik wiedergeben.

In einem Unix-Dateisystem hat jede einzelne Datei jeweils einen Eigentümer und eine Gruppenzugehörigkeit. Neben diesen beiden Angaben besitzt jede Datei noch einen Satz Attribute, die bestimmen, wer die Datei wie benutzen darf. Diese Attribute werden dargestellt als "rwx". Dabei steht r für lesen (read), w für schreiben (write) und x für ausführen (execute).

## Das Dateisystem

Das Dateisystem ist die Ablageorganisation auf einem Datenträger eines Computers. Um die Funktionsweise zu verstehen, betrachten wir einen Datenträger, die Festplatte, näher:

Die Festplatte besteht aus mehreren Scheiben mit einer magnetisierbaren Oberfläche, auf die die Schreibköpfe unsere Daten als Einsen (ein) und Nullen (aus) abspeichern. Um diese aber vernünftig adressieren zu können, benutzen wir Dateisysteme. Ein solches teilt die Festplatte (eigentlich die "Partition", denn die Festplatte wird häufig in mehrere Partitionen aufgeteilt, die dann unabhängig formatiert werden können) in kleine Einheiten, die "Blöcke", welche aus Performancegründen häufig noch zu "Clustern" zusammengefasst werden.

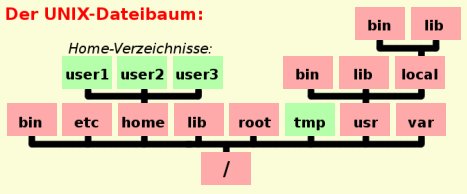
Der Block (oder Cluster) ist dann die kleinste Einheit, in die eine Datei geschrieben wird, jede Datei benötigt dadurch immer diesen Speicherplatz (oder ein Vielfaches) auf der Festplatte.

Von Windows kennen wir NTFS und FAT32, bzw. Apple-Benutzer werden schon von HFS+ gehört haben. Unter Linux werden meist ext2, ext3 oder ext4 (Second, Third bzw. Fourth Extented File System) verwendet. "ext3" unterscheidet sich von "ext2" nur dadurch, dass zusätzlich ein "Journal" geschrieben wird, welches bei Systemabstürzen eine zuverlässige Wiederherstellung möglich macht. "ext4" ist eine performantere Weiterentwicklung von "ext3" und heute Standard. Daneben gibt es gelegentlich noch ReiserFS, XFS oder JFS, aber die Wahl des Dateisystems bestimmt tatsächlich immer das Abwägen zwischen höherer Sicherheit und schnellerer Schreibgeschwindigkeit - mit oder ohne Journal.

## Die Verzeichnisstruktur

Während unter Windows jedes logische oder physikalische Laufwerk (Festplattenpartitionen oder Geräte wie CD-Laufwerk) seinen eigenen Dateibaum, ausgehend vom Laufwerksbuchstaben hat, gibt es unter UNIX-Systemen nur einen Dateibaum, ausgehend vom Wurzelverzeichnis (am Anfang einer bestimmten Festplattenpartition), in den die anderen Laufwerke an beliebiger Stelle eingehängt werden können. Sie sind sozusagen nur Äste des einen Dateibaums.

Nun wollen wir diesen Dateibaum einmal näher betrachten. Linux hat, wie jedes Unix-System, einen sehr genau vorgeschriebenen Dateibaum. Die wichtigsten Verzeichnisse heißen immer gleich, so dass es beinahe egal ist, mit welchem Unix wir arbeiten. Wir werden uns überall zurechtfinden.



Die Verzeichnisstruktur

### Das Wurzelverzeichnis /

Wie bereits gesagt hat Linux nur einen Dateibaum. Eine Partition (bzw. ein Laufwerk) ist also die sogenannte root-partition (Wurzelpartition). Alle anderen Partitionen und Laufwerke (auch CD-Laufwerk, USB-Stick, externe Festplatte) werden an beliebigen Stellen in den Wurzeldateibaum "montiert" (mounted).

In diesem Wurzelverzeichnis darf neben den Verzeichnissen nur eine einzige normale Datei liegen, nämlich der Kernel. Aber auch dies ist auf modernen Systemen nicht mehr der Fall, da der Kernel in das Verzeichnis /boot ausgelagert ist. Im Wurzelverzeichnis befindet sich stattdessen ein symbolischer Link auf den Kernel.

### Das Verzeichnis /bin

Hier liegen Binaries, also binäre Programme, jedoch nur jene, die für die wichtigsten Arbeiten am System benötigt werden. Alle User haben in diesem Verzeichnis Lesezugriff.

### Das Verzeichnis /boot

Hier liegen der Linux-Kernel und alle Dateien, die er zum Starten benötigt. Nur der root-Benutzer hat auf dieses Verzeichnis Zugriff.

### Das Verzeichnis /dev

Hier liegen die Gerätedateien. Gerätedateien sind Schnittstellen zum Kernel, die ein bestimmtes Gerät bezeichnen. Unter Unix ist fast jedes Stück Hardware mit einer solchen Gerätedatei ansprechbar. Wichtig ist, dass diese Dateien keinen physikalischen Platz auf der Platte brauchen.

### Das Verzeichnis /etc

Hier sind alle systemweit gültigen Konfigurationsdateien (Netzwerk, Bootmanager, Sytemstartscripts, ...) abgelegt, sowie die Dateien für die Benutzerverwaltung (z.B. "passwd" und "shadow" mit den Benutzern und ihren Passwörtern).

Viele Prozesse müssen diese Dateien lesen können, um bestimmte Informationen zu bekommen. Daher muss das Verzeichnis für alle User lesbar sein.

### Das Verzeichnis /home

Jeder User, den das System kennt, hat ein eigenes Verzeichnis, das in der Regel den gleichen Namen trägt, wie der User selbst. Wir nennen solche Verzeichnisse die "Home-Verzeichnisse" der Benutzer. Sie liegen alle (außer dem vom Systemverwalter "root") im Verzeichnis /home. Nach dem Einloggen (Anmelden beim System mit Username und Passwort) befinden sich alle User immer in ihrem Home-Verzeichnis.

Innerhalb des eigenen Home-Verzeichnisses darf ein User in der Regel alles machen - auch Dateien anlegen, löschen und verändern. Außerhalb seines Verzeichnisses hat ein normaler Benutzer meist nur Leserechte - verändern, anlegen oder löschen darf er dort nicht.

### Das Verzeichnis /lib

Hier liegen die Libraries, die Systembibliotheken von Unix. Dabei handelt es sich sowohl um statische Bibliotheken für das System, als auch um dynamische, die in etwa funktionieren wie die DLLs von Windows.

### Die Verzeichnisse /mnt und /media

In diesen Verzeichnissen finden sich Unterverzeichnisse, die zunächst eigentlich leer sind. Sie dienen dazu, gemountete Laufwerke und Geräte einzubinden.

Ein Beispiel: in /mnt/datenarchiv könnte eine zweite, im System verbaute Festplatte eingehängt sein und unter /media/cdrom0 finden wir sehr wahrscheinlich unser (erstes) CD/DVD-Laufwerk.

Beide Verzeichnisse erfüllen den gleichen Zweck und die Verwendung ist uneinheitlich, aber meist werden in /media Wechchseldatenträger und alle erst nach Systemstart vom Benutzer eingehängten Dateisysteme (wie externe Festplatten und USB-Sticks) verwaltet.

### Das Verzeichnis /proc

Auch dieses Verzeichnis benötigt keinen Platz auf der Platte, es enthält Informationen, die das laufende System ständig auffrischt. Es sind eigentlich keine Dateien und Verzeichnisse, die hier liegen, sondern Informationen des Kernels.

### Das Verzeichnis /root

Dies ist das Home-Verzeichnis des Systemverwalters "root". Warum ist dieses Verzeichnis aber nicht im Verzeichnis /home? Ganz einfach: /home ist häufig als eigene Partition implementiert. Diese Partition wird aber erst während des Systemstarts eingehängt (mounted). Falls es zu einem Fehler kommen sollte, muss der Systemverwalter das System starten können ohne alle Partitionen einzuhängen. Es muss also auf dem Wurzelverzeichnis bereits ein arbeitsfähiges System bereitstehen. Zu einem arbeitsfähigen System gehört aber unbedingt das Home-Verzeichnis. Daher liegt das Home-Verzeichnis von "root" direkt auf der Wurzel und in derselben Partition.

In frühen Unix-Systemen war das Home-Verzeichnis von "root" tatsächlich das Wurzelverzeichnis (engl. root directory), aber da sich in einem Home-Verzeichnis immer eine Menge Dateien ansammeln, wurde das zu unübersichtlich und sein Home-Verzeichnis ins Verzeichnis /root ausgelagert.

### Das Verzeichnis /sbin

Wie schon in /bin liegen auch hier Binaries, allerdings solche, die hauptsächlich der Systemverwalter benutzt - "System-Binaries", sozusagen.

### Das Verzeichnis /tmp

In diesem Verzeichnis werden, wie schon der Name vermuten lässt, temporäre Dateien abgelegt, die nach Verwendung nicht mehr benötigt werden. Alle Benutzer müssen hier schreiben können, aber es ist kein guter Ort, um seine Daten sicher zu verwahren. Der gesamte Inhalt wird nämlich normalerweise beim Neustart des Systems gelöscht.

### Das Verzeichnis /usr

Dieses Verzeichnis enthält alle wichtigen Programme, die das System anbietet. "usr" steht dabei nicht, wie irrtümlich häufig angenommen für "User", sondern für "Unix System Resources".

Als Unterverzeichnisse finden wir hier analog zur Wurzel /usr/bin, /usr/lib und /usr/sbin und ihre Bedeutung entspricht jeweils denen des Wurzelverzeichnisses.

Einige weitere wichtige Unterverzeichnisse wollen wir noch näher betrachten:

/usr/local dient der Trennung von distributionseigenen und fremden, zusätzlich installierten Programmen. Unter /usr/local finden wir nochmals die gleichen Verzeichnisse wie im Verzeichnis /usr selbst. Selbst kompilierte Programme sollten immer in /usr/local installiert werden und nie direkt in /usr und die Konfigurationsscripts der Programme sehen dies meist auch als Default vor.

/usr/share beinhaltet architekturunabhängige Daten, die verschiedene Programme zum Betrieb benötigen und die nicht verändert werden.

/usr/src enthält die Quelltexte für alle Programme des Standardsystems.

### Das Verzeichnis /var

Dieses Verzeichnis beinhaltet Daten, die von Programmen häufig neu geschrieben werden müssen. Ursprünglich waren diese Daten auch unter /usr zu finden, aber der Wunsch /usr auch schreibgeschützt mounten zu können machte eine Trennung notwendig. Hier werden auf einem Mailserver Mails zwischengelagert (/var/spool) und alle Programme schreiben hierher ihre Log-Dateien (/var/log).

Wir werden auch noch einige andere Verzeichnisse und Unterverzeichnisse finden, die teilweise leer sind oder nur symbolische Links in andere Verzeichnisse enthalten und häufig aus Kompatibilitätsgründen existieren. Ein Verzeichnis sei vielleicht noch erwähnt:

### Das Verzeichnis /opt

Dieses Verzeichnis ist eine Linux-Erfindung. Nachdem immer mehr große Programmpakete installiert wurden, gab es die Überlegung, diese nicht mehr in das Verzeichnis /usr zu packen, sondern ein eigenes Verzeichnis dafür zu schaffen. Tatsächlich nutzen nur wenige Programme von sich aus diese Möglichkeit und auf vielen Systemen bleibt dieses Verzeichnis leer.

## Die Linux-Distributionen

Wie schon besprochen besteht ein Linux-Betriebssystem aus dem Linux-Kernel und einer großen Anzahl verschiedener Anwenderprogramme. Tatsächlich gibt es ein Projekt, das eine Anleitung bietet, wie man aus den Kernelquellen und selbst selektierten Programmen ein komplettes, maßgeschneidertes Betriebssystem bauen kann. Das Projekt nennt sich "Linux From Scratch" oder "LFS" (www.linuxfromscratch.org) und ich kann jedem, der etwas Zeit übrig hat, nur empfehlen, dies selbst einmal zu probieren.

Man erhält ein ultrakompaktes, ultraschnelles Betriebssystem und kann sagen, "das ist mein eigenes reinrassiges Linux-System". Aber was kommt dann?

Man braucht schon einige Zeit bis alle Programme, die so benötigt werden, kompiliert und konfiguriert sind und dann müssen diese auch noch laufend aktualisiert werden. Sicherheitsupdates müssen selbst organisiert und kompiliert werden. Mit all der Administrationsarbeit kommt man zu sonst nichts mehr.

Um das zu vermeiden, gibt es Linux-Distributionen. Diese bieten nicht nur einen fertigen Satz von notwendigen Programmen, sondern auch die regelmäßige Versorgung mit Updates an. Die aus meiner Sicht wichtigsten Distributionen sollen hier aufgeführt werden:

### Debian

"Debian" (www.debian.org) ist eine nicht-kommerzielle Distribution und das "Debian-Projekt" ist nach der "Debian-Verfassung" geregelt, die eine demokratische Organisationsstruktur vorsieht. Darüber hinaus ist das Projekt über den "Debian Social Contract" zu völlig freier Software verpflichtet. Mit einigen 1000 Mitarbeitern ist Debian der Gigant unter den Linux-Systemen.

Da bei Debian eine "stabile Version" immer eine wirklich stabile Version ist, sind die Entwicklungszeiten relativ lang und böse Zungen behaupten auch, dass die stabile Version schon bei Erscheinen veraltet ist.

Allerdings bietet Debian auch immer schon die zukünftigen Versionen an und so gibt es mehrere Zweige, aus denen man sich bedienen kann:

stable - wirklich stabile Version, die auch für den kommerziellen Serverbetrieb geeignet ist!

testing - die zukünftige stable-Version. Ab einem gewissen Entwicklungsstand wird die Distribution "eingefroren" (engl. "frozen") - d.h. es werden keine neueren Versionen von Programmen mehr aufgenommen, sondern nur noch an der Fehlerbeseitigung bei den vorhandenen gearbeitet. Dies entspricht ungefähr dem Zustand, bei dem andere Distributoren ihre "stabilen" Versionen veröffentlichen.

Da ich schon mehrmals eine testing-Version ab dem Anfangsstadium benutzt habe, glaube ich sogar sagen zu können, dass testing nie so instabil ist, wie manche andere Distribution im "ausgereiften" Zustand. Für den Desktopbetrieb kann ich ein eingefrorenes testing jedenfalls empfehlen.

unstable - ist der erste Anlaufpunkt für neue Versionen von Paketen und Programmen, bevor sie in testing integriert werden. Man installiert sich mit unstable das neueste vom neuen, muss aber wissen, dass das nicht immer stabil ist.

experimental - ist kein vollständiger Zweig, denn es dient nur dazu, Programme und deren Funktionen zu testen, die sonst das ganze System gefährden würden. Es enthält immer nur die gerade getesteten, bzw. die von diesen benötigten Programmpakete.

Diese Zweige haben auch immer Codenamen und sind, einer Vorliebe der frühen Entwickler folgend, immer nach Figuren aus dem Film "Toy Story" benannt. So heißt im Moment die stable-Version "Jessie" und "Stretch" ist testing. unstable ist immer "Sid", der Junge von nebenan, der die Spielsachen zerstört, aber es lässt sich auch als Abkürzung für "still in development" (noch in Entwicklung) deuten.

Und wer einen dieser Zweige installiert hat, kann auf eine unüberschaubare Vielzahl an Programmen zurückgreifen, auf Wunsch (und auf eigene Gefahr) auch aus den anderen Zweigen. Für Anfänger ist es wohl nur bedingt zu empfehlen, obwohl sich in den letzten Jahren sehr viel in Sachen Benutzerführung getan hat. Auch steht ein deutschsprachiges Forum (debianforum.de) zur Verfügung, wo man Hilfe bekommt und wo auch dumme Fragen gestellt werden dürfen. Für ambitionierte Linux-EinsteigerInnen, die sich auch mit den Möglichkeiten ihres Betriebssystems auseinandersetzen wollen, könnte es sogar die beste Distribution sein.

### Fedora

Das nicht-kommerzielle "Fedora" (fedoraproject.org) ist der Nachfolger des traditionsreichen, kommerziellen "Red Hat Linux", welches nicht mehr selbständig weiterentwickelt wird. Stattdessen verkauft die Firma Red Hat, das auf Fedora basierende "Red Hat Enterprise Linux".

Fedora ist eine sehr innovative Distribution und vor allem in den USA sehr beliebt. Es werden nur völlig unter freier Lizenz stehende Inhalte akzeptiert, weshalb nach der Installation zum Beispiel keine MP3-unterstützenden Programme zu finden sind. Für Anfänger gibt es bessere Distributionen.

### Gentoo Linux

Das nicht-kommerzielle "Gentoo Linux" (www.gentoo.de) ist eine quellbasierte Linux-Metadistribution - das heißt, alle Programme, inklusive des Kernels, wird selbst kompiliert. Das klingt sehr anstrengend, ist es aber gar nicht so, da die Distribution geeignete Werkzeuge zur Verfügung stellt, mit denen dies einfachst möglich gelingt. Auch sorgt eine große, sehr aktive "Community" bei jedem Problem für Rat und Hilfe. Dennoch ist sie für Linux-Neulinge wohl nicht empfehlenswert.

### OpenSUSE

"openSUSE" (www.opensuse.org) ist die zweitbeliebteste Distribution am Heim-PC. Die nicht-kommerzielle Variante der von Novell aufgekauften kommerziellen SUSE-Distribution (heute "SUSE Linux Enterprise") glänzt mit einem universellen Konfigurationswerkzeug. Sie gilt als anfängerfreundlich. Die neueste Versionsnummer 42.1 bezieht sich übrigens auf die Antwort auf die Frage "nach dem Leben, dem Universum und dem ganzen Rest" aus Douglas Adam's "Per Anhalter durch die Galaxis". Schon 1996 hatte die Version 4.2 diesen Bezug.

### Slackware

"Slackware" (www.slackware.com) ist die älteste noch heute existierende Distribution. Sie verzichtet aus Prinzip auf grafische Einrichtungswerkzeuge und ist daher eher nur für fortgeschrittene BenutzerInnen geeignet.

Bisher nicht vorgekommen sind Distributionen, die nur Abwandlungen anderer Distributionen sind und häufig auch deren Quellen benutzen. Vor allem von Debian gibt es unzählige davon. Sie werden als "Derivate", oder oft auch, etwas abfällig, als "Klone" bezeichnet. Ein solcher "Debian-Klon" hat allerdings Geschichte geschrieben:

### Ubuntu

Das von der Firma des Gründers gesponserte kostenlose Betriebssystem soll nach dem Willen der Entwickler ein einfach zu installierendes und leicht zu bedienendes Betriebssystem mit aufeinander abgestimmter Software sein. Es bedient sich dazu aus den Quellen von Debian unstable und hat das Ziel, nach der enormen Popularität zu schließen, eindeutig geschafft.

Tatsächlich kann Ubuntu von der Live-CD mit wenigen Mausklicks problemlos auf die Festplatte installiert werden und dann erwartet den Benutzer ein weitgehend komplettes Betriebssystem mit vielen Multimedia-Programmen. Releases erscheinen mit schöner halbjährlicher Regelmäßigkeit und der Upgrade auf diese lässt sich ebenfalls auf Mausklick bewerkstelligen. Alle 2 Jahre gibt es eine "Versionen mit verlängerter Unterstützung" (Long Term Support oder kurz LTS), die dann deutlich stabiler als die kürzer unterstützten ist. Für EinsteigerInnen ist Ubuntu (www.ubuntu.com) bestens geeignet.

### Raspbian

Raspbian ist das Debian-basierte Betriebssystem für den Raspberry Pi. Es wird von der Raspberry Pi Foundaition zur Verfügung gestellt und ist das gängigste Betriebssystem von den Single-Board-Computer Familie der Raspberry Pi’s genutzt wird. Raspbian ist optimiert für ARM Prozessoren.

### Linux Mint

Diese Distribution war ursprünglich ein Ubuntu-Klon, aber neuerdings ist Linux Mint auch als LMDE - Linux Mint Debian Edition - erhältlich. Den Entwicklern ist es wichtig, die bestmögliche Integration von Programmen zu bieten, die bei Benutzern beliebt, aber eben nicht quelloffene freie Software sind. Die anderen Distributionen, inklusive Ubuntu, bieten zwar auch die Installation von "non-free"-Paketen an, aber in einem eigenen Zweig und erst nach der Basisinstallation.

Für absolute Stabilität setzt Linux Mint immer auf LTS (Ubuntu) oder stable (Debian) Versionen auf, aber die integrierten Programme erhalten auch zwischenzeitig Versionsupgrades. Als Vorbild wird die Benutzerfreundlichkeit und Stabilität von Apple's OS X genannt. Auch Linux Mint (www.linuxmint.com) ist für EinsteigerInnen bestens geeignet.

## Linux Desktops

Anders als bei Microsoft und Apple gibt es bei Linux-Distributionen keinen festgelegten Desktop. Alle Distributionen bieten die Möglichkeit den Desktop zu ändern und zunehmend kann schon bei der Installation der gewünschte Desktop festgelegt werden. Wenngleich für Anfänger im Allgemeinen der Standard-Desktop der jeweiligen Distribution sicher eine gute Wahl ist, möchte ich abschließend auch noch kurz einige Desktops vorstellen.

### Gnome vs. KDE

Gnome und KDE sind die Platzhirsche auf Linux-Bildschirmen. Die Frage, welcher Desktop der bessere ist, ist seit je her umstritten und ungeklärt. Beide kommen mit mächtigen Datei-Managern daher und bieten viel Automatisierung. Jeder Versionsupgrade entfacht die Diskussion erneut und, wie auch Windows BenutzerInnen wissen, wird nicht jede Neuentwicklung als Verbesserung empfunden.

### XFCE

Während erstgenannte Desktops mit Features protzen gehen die EntwicklerInnen bei XFCE einen anderen Weg. XFCE möchte einen schlanken, performanten Desktop bieten, der auf "unnötige Spielereien" verzichtet und auch auf leistungsschwachen und zum Teil für andere Aufgaben (Multimedia) bestimmten Systemen resourcenschonend läuft. Viele heutige Distributionen bieten XFCE als vorkonfigurierte Alternative zum Standarddesktop an.

### MATE und Cinnamon

Der umstrittene Upgrade von Gnome2 auf Gnome3 führte zur Geburt von MATE. Dieser Desktop ist eine direkte Fortentwicklung von Gnome2. Cinnamon dagegen ist aus den Quellen von Gnome3 entstanden, aber deutlich performanter als dieser. Was beide gemeinsam haben - sie sind die Standarddesktops von "Linux Mint" und nur als Ubuntu-Editionen von Linux Mint gibt es auch KDE und XFCE vorkonfiguriert.

### LXDE

LXDE ist eine freie Desktop-Umgebung für Unix und andere POSIX-konforme Plattformen, wie Linux oder BSD. Der Name LXDE steht für „Lightweight X11 Desktop Environment“.

Das Projekt wurde im Jahr 2006 begonnen, um eine Desktop-Umgebung zu schaffen, welche schnell und energiesparend arbeitet. Sie eignet sich daher gut für den Einsatz auf Netbooks oder auf älteren PCs, bei denen die heute üblichen Leistungsanforderungen nicht erbracht werden können. Im Gegensatz zu anderen Desktop-Umgebungen wie KDE oder Gnome funktionieren die einzelnen Komponenten in LXDE mit nur wenigen Abhängigkeiten voneinander und können so einfach nach und nach auf andere Systeme übertragen werden.

### PIXEL

Raspbian verwendet PIXEL, Pi verbessert Xwindows Environment, Lightweight als seine neue Desktop-Umgebung. Es besteht aus einer modifizierten LXDE Desktop-Umgebung und dem Openbox-Stacking-Fenstermanager mit einer neuen Theme und weiteren kleinen Änderungen.

Es können auch mehrere Desktops gleichzeitig installiert werden. Bei der Anmeldung kann dann zwischen den Desktops gewechselt werden. So kann jeder seinen Lieblingsdesktop herausfinden.

# BASH-Grundkurs

Eine Shell (engl. Hülle, Schale, Außenhaut) bezeichnet die traditionelle Benutzerschnittstelle unter Unix-Betriebssystemen. Der Benutzer kann in einer Eingabezeile Kommandos eintippen, die der Computer dann sogleich ausführt. Man spricht darum auch von einem Kommandozeileninterpreter.

Die Bash (Bourne-again-shell - eine Weiterentwicklung der Bourne-Shell) ist die Standard-Shell auf den meisten Linux-Systemen und wurde auf fast alle Unix-Systeme portiert.

## Bash? Wozu brauche ich das?!

Unter Windows spielen Kommandozeileninterpreter für den Durchschnittsbenutzer heute eine geringe Rolle. Für scheinbar alle Zwecke gibt es graphische Tools, die sämtliche Konfigurations- und Bedienungsschritte erledigen können. Scheinbar - denn auch unter Windows kommen professionelle Systemadministratoren nicht ohne Kommandozeile aus.

Unter Linux gibt es eine ähnliche Tendenz: So manche moderne Distribution versteckt die Shell heute ebenso gut wie der große Konkurrent aus Redmont. Dennoch ist sie ein wichtiges Instrument, mit dem sich jede/r vertraut machen sollte.

Es gibt viele nützliche Programme, für die es keine graphische Oberfläche gibt und auch Programme mit graphischer Oberfläche kennen oft Optionen, die nur von der Kommandozeile aus zugänglich sind.

Im Störungsfall ist eine Rettungs-Shell oft das einzige zur Verfügung stehende Mittel, um das Problem zu beheben, aber auch wenn das System nicht streikt, kommt es gelegentlich vor, dass ein Programm nach einem Update nicht mehr funktioniert. Unter Windows bleibt da meist nur warten und hoffen. Unter Linux starte ich das widerspenstige Programm einmal als Shell-Kommando und sehe mir an, was es zu sagen hat. Die Fehlermeldungen sind häufig schon aussagekräftig, aber wenn nicht, gebe ich sie in eine Suchmaschine ein und erfahre so nicht nur, warum das Programm nicht läuft, sondern auch meist, wie ich es zum Laufen bringen kann.

Was mich an Windows immer genervt hat, war, dass es Probleme gab, bei denen ich an einen Punkt kam, wo ich aufgeben musste. Unter Linux gibt es auch Probleme, aber ich weiß, dass ich, eventuell mit einigem Aufwand, die Lösung finden werde.

Geht nicht - gibt's nicht!

## Die Bedeutung verschiedener Schriften:

* Shell-Eingaben werden in Schrift mit fester Breite dargestellt. Sie sind, wenn abgesetzt dargestellt, mit ENTER-Taste (Return) abzuschließen.
* *Kursiv* gesetzte Ausdrücke sind durch einen entsprechenden Inhalt zu ersetzen. Wir werden aber die Testdateien weitgehend selbst erstellen, weshalb zu ersetzende Ausdrücke meist nur bei der Vorstellung des Befehls auftauchen werden.
* Rot werden alle vorgestellten Befehle dargestellt.

## Noch einige Tipps:

Moderne Terminal-Emulationen beherrschen meist copy/paste. Das heißt, ich kann mit der linken Maustaste einen Teil der Ausgabe (zum Beispiel Dateinamen) markieren, mit der rechten Maustaste kopieren und wieder mit rechter Maustaste in der Befehlszeile einfügen. Auch aus anderen Dokumenten kopierter Text kann so eingefügt werden.

Die Bash speichert alle eingegebenen Befehle. Mit den Pfleiltasten ↑ und ↓ können sie zurückgerufen und (eventuell modifiziert) wieder abgesetzt werden. Sollen alle gespeicherten Befehle ausgegeben werden, genügt die Eingabe history, gefolgt von ENTER.

Tippt man den Anfang eines Befehls und drückt dann die Tabulatortaste, versucht die Bash den Befehl zu vervollständigen.

Dies geschieht soweit der Befehl eindeutig ergänzt werden kann. Gibt es mehrere Möglichkeiten, so wird nur bis zu dem Punkt vervollständigt, ab dem sich die Möglichkeiten unterscheiden. Drückt man dann 2x die Tabulatortaste, werden die Möglichkeiten aufgelistet. Tippt man dann weiter und drückt abermals die Tabulatortaste, wird auch der Rest ergänzt.

## Kapitel 1: Das System erkunden

Zunächst öffnen wir ein Terminal-Fenster. Wenn im Menü nicht zu finden, können wir die Suchfunktion aufrufen und "terminal" suchen (die ersten drei Buchstaben einzutippen wird reichen).

### Betrachten, Navigieren, Suchen - pwd, ls, cd und find

Nach dem Aufruf der Shell befinden wir uns in unserem Home-Verzeichnis. Am Anfang der Zeile, in die wir schreiben können, steht etwas wie benutzer@computer:~$ - vor dem $ können noch Pfadangaben zu finden sein, aber wenn dort nur eine Tilde steht, befinden wir uns im Home-Verzeichnis.

Um das zu überprüfen können wir uns mit pwd (print work-directory) den Pfad des aktuellen Verzeichnisses anzeigen lassen:

pwd

Wir sehen uns das "aktuelle" Verzeichnis an, das heißt, das Verzeichnis, in dem wir uns gerade befinden:

ls

Wir bekommen alle Dateien und Verzeichnisse im Home-Verzeichnis aufgelistet.

Die gleiche Ausgabe in detailierter Form erhalten wir mit:

ls -l

Das -l ist eine Option. Es steht hier für eine "lange" Ausgabe - zu jeder Datei bekommen wir Informationen über Größe und Dateirechte (zu diesen kommen wir etwas später).

Allgemein gibt ls (list) den Inhalt des angegebenen Verzeichnisses aus:

ls (-option) verzeichnisname

Wenn wir zuvor nur ls ohne Verzeichnisname ausgeführt haben, so geht das nur, weil dies als ls . interpretiert wird, wobei der Punkt immer für das aktuelle Verzeichnis steht.

Im Home-Verzeichnis gibt es das Verzeichnis Desktop. Darin befinden sich die Dateien, die wir auf unserem Desktop sehen.

Verzeichnis "Desktop" ausgeben:

ls Desktop

Wenn wir ein frisch installiertes System haben, wird da unter Umständen noch nicht viel zu sehen sein. Wenn wir gar nichts auf dem Desktop liegen haben, wird die Ausgabe sogar leer sein.

Wir können dann schauen, ob überhaupt etwas in einem der Verzeichnisse liegt.

Den Inhalt aller Verzeichnisse ausgeben:

ls \*

Ein Stern steht für alle Datei- oder Verzeichnisnamen und so sehen wir auch den Inhalt aller Verzeichnisse - ls gibt, wie gesagt, den Inhalt des angegeben Verzeichnisses aus und Stern bedeutet "alle"!

Wobei "alle" nicht ganz korrekt ist. Dateien und Verzeichnisse, die mit einem Punkt beginnen, werden unter Linux als "versteckt" betrachtet und normalerweise nicht angezeigt. Wollen wir auch versteckte Dateien anzeigen, müssen wir wieder eine Option benutzen.

Auch versteckte Dateien ausgeben:

ls -a

Jetzt haben wir wohl eine etwas umfangreichere Ausgabe! Um den Anfang zu sehen, müssen wir ein Stück hinaufscrollen. Dort sehen wir zwei seltsame Verzeichnisse: . und .. - diese bezeichnen das aktuelle Verzeichnis und das diesem übergeordnete Verzeichnis. Die beiden gibt es in jedem Verzeichnis! Über deren Bedeutung werden wir in Kürze mehr erfahren. Da es sie aber ohnehin überall gibt, wollen wir sie im Allgemeinen nicht anzeigen. Mit Option -A bekommen wir auch wieder versteckte Dateien ausgegeben, aber ohne . Und ...

Auch versteckte Dateien ohne . und .. ausgeben:

ls -A

Optionen können wir auch kombinieren. Nach dem Minus müssen sie nur aneinandergereiht werden:

ls -lA

liefert versteckte und nicht versteckte Dateien in detaillierter Form (mit großem "A" ohne die Verzeichnisse . und ..).

Nun wollen wir aber das Home-Verzeichnis auch verlassen. Mit cd (change directory) können wir uns zwischen den Verzeichnissen bewegen.

cd verzeichnisname

Wir wechseln ins Verzeichnis "Desktop":

cd Desktop

Mit ls -a können wir die Dateien dieses Verzeichnisses anzeigen lassen. Wenn es sonst keine Dateien gibt, so sehen wir jedenfalls wieder . und ... Jetzt benützen wir .. um wieder zurück ins (übergeordnete) Home-Verzeichnis zu kommen.

Wechsel ins übergeordnete Verzeichnis:

cd ..

Wenn wir cd .. wiederholen erreichen wir das Verzeichnis, in dem als Unterverzeichnis auch unser Home-Verzeichnis liegt. ls zeigt uns das. Wir befinden uns jetzt im Verzeichnis /home. Würden wir cd .. nochmal wiederholen, wären wir im Wurzelverzeichnis, aber dort kommen wir auch anders hin.

Wechsel ins Wurzelverzeichnis:

cd /

Wollen wir ins zuvor benutzte Verzeichnis zurück gibt es eine spezielle Option.

Wechsel ins zuletzt benützte Verzeichnis:

cd -

Um sofort ins Home-Verzeichnis zu gelangen, geben wir cd ohne etwas danach ein.

Wechsel ins Home-Verzeichnis:

cd

(cd ohne Verzeichnis steht für cd ~, wobei ~ das Home-Verzeichnis des jeweiligen Benutzers ist)

Wenn wir wissen, wo's hingehen soll, können wir auch direkt ein Verzeichnis angeben:

cd /usr/share/pixmaps

Nun sind wir in dem Verzeichnis, in dem die Bilddateien für die Desktopicons liegen.

Will ich direkt in das Unterverzeichnis "Desktop" springen, kann ich die Tilde für das Home-Verzeichnis nutzen:

cd ~/Desktop

(Ausgeschrieben wäre das cd /home/benutzername/Desktop)

Unter Linux werden alle Festplatten, Partitionen und andere Geräte wie CD-Laufwerk und USB-Stick in einen Verzeichnisbaum "eingehängt". Das grundlegende Verzeichnis ist die Wurzel /. Nun können wir im ganzen Verzeichnisbaum herumflitzen. Wir können dabei nichts ernsthaft kaputt machen, denn dazu haben wir als normaler Benutzer gar nicht die nötigen Rechte.

Üben:

cd /

ls

cd var

pwd

usw.

Wir können auch nach Dateien suchen:

find verzeichnisname -name dateiname

Wir übergeben find dazu den Verzeichnisnamen, wo wir zu suchen beginnen wollen und den Namen der Datei - wir wählen "syslog", da dieses auf jedem System vorhanden sein sollte:

find /var -name syslog

Wahrscheinlich erhalten wir außer dem Aufenthaltsort der Datei noch einige Fehlermeldungen von Unterverzeichnissen, für die wir keine Leseberechtigung haben. Im Augenblick müssen wir das so hinnehmen.

## Textdateien lesen – less & tail

Jetzt wissen wir, wo "syslog" zu finden ist, aber vielleicht möchten wir auch sehen, was darin geschrieben steht. Dabei hilft uns der Textbetrachter less:

less dateiname

Um syslog zu lesen geben wir ein:

less /var/log/syslog

(um den Pfad nicht abtippen zu müssen, können wir ihn auch aus der vorherigen Ausgabe von find kopieren - siehe auch "Tipps" ganz oben)

In "less" kann man mit den Pfeiltasten ↑, ↓, Bild↑ und Bild↓ navigieren, mit [Ende] ans Ende und [Pos1] an den Anfang springen.

Beenden lässt sich "less" mit der Taste [Q].

Um jedoch neue Einträge in der syslog direkt angezeigt zu bekommen gibt es ein Programm namens tail.  
Das Kommando tail (engl.: "Ende, Rest") gibt die letzten Zeilen einer Datei aus. Auch hier dürfen mehrere Dateien zur Ausgabe übergeben werden. Neben dem Parameter -n nummer (default ist zehn) wird oft die Option -f (Langform: --follow) verwendet. Damit kann man sich das Ende von Dateien anschauen, die immer weiter anwachsen, wie beispielsweise Logfiles. So zeigt tail, auf syslog losgelassen, wer gerade im System passiert und neue Einträge werden sofort angezeigt.

tail -f /var/log/syslog

Den gleichen Effekt erzielen Sie übrigens mit less, wenn Sie einfach [Shift-f] drücken, während er ein File anzeigt. Das Programm wartet dann auf neue Zeilen in der Datei (zu erkennen in der Statuszeile *Waiting for data... (interrupt to abort)*). Den "Follow"-Modus verlassen Sie durch [Strg-c].

## Kapitel 2: Dateimanipulationen

### Dateien anlegen und kopieren - touch, mkdir und cp

Mit touch können wir leere Dateien erzeugen:

touch dateiname

Zunächst erzeugen wir die Testdatei "datei.txt" im tmp-Verzeichnis:

touch /tmp/datei.txt

Jetzt wechseln wir mit cd ins Home-Verzeichnis.

Zum Anlegen eines Verzeichnisses benutzen wir mkdir (make directory):

mkdir verzeichnisname

Wir legen ein Testverzeichnis "mond" an:

mkdir mond

Mit cp (copy) können wir Dateien kopieren:

cp quelle ziel(verzeichnis)

Jetzt wollen wir unsere Testdatei aus /tmp in unser Testverzeichnis kopieren:

cp /tmp/datei.txt mond

Wir können der Datei aber auch einen neuen Namen geben. "datei.txt" wird als "kopie.txt" ins Verzeichnis mond kopiert.

cp /tmp/datei.txt mond/kopie.txt

Ein ls mond überzeugt uns vom Erfolg. Wir haben jetzt schon gesehen, dass es zwei verschiedene Arten von Pfadangaben gibt:

mond/kopie.txt ist ein relativer Pfad. Er enthält die Unterverzeichnisse ausgehend von unserem aktuellen Verzeichnis.

/tmp/datei.txt ist ein absoluter Pfad. Er beginnt mit / für das Wurzelverzeichnis und enthält alle Unterverzeichnisse ab diesem. Er ist von überall im Dateibaum ausgehend eindeutig!

An sich überschreibt cp Dateien ohne Rückfrage. Soll vor dem Überschreiben eine Rückfrage erfolgen, kann man die Option -i benutzen:

cp -i /tmp/datei.txt mond/kopie.txt

Die Frage, ob wir die vorhandene Datei überschreiben wollen, können wir mit j(a) oder n(ein) beantworten.

Will man ein ganzes Verzeichnis kopieren, kann man die Option -R benutzen. Zunächst legen wir mit mkdir noch ein weiteres Verzeichnis an:

mkdir Mond

Wir sehen: unter Linux werden Groß- und Kleinschreibung unterschieden. "Mond" und "mond" sind verschiedene Dateien.

Jetzt kopieren wir das Verzeichnis "mond" mitsamt Inhalt in "Mond":

cp -R mond Mond

Wir erhalten ein Verzeichnis "mond" im Verzeichnis "Mond". Will man nur den Inhalt und nicht auch das Verzeichnis kopieren, kann man als Quelle mond/\* festlegen – die "Wildcard" \* bedeutet wieder "alle Dateien im Quellverzeichnis":

cp -R mond/\* Mond

(eigentlich ist -R hier nicht notwendig, da es im Quellverzeichnis "mond" keine Unterverzeichnisse gibt)

Jetzt haben wir die Dateien "datei.txt" und "kopie.txt" auch direkt in "Mond".

Beim Kopieren können auch die beiden Verzeichnisse . und .. verwendet werden. Wechseln wir zunächst ins Verzeichnis "Mond/mond":

cd Mond/mond

Wir kopieren die Datei "kopie.txt" unter dem Namen "neu.txt" ins übergeordnete Verzeichnis:

cp kopie.txt ../neu.txt

Nun die Datei "neu.txt" zurück ins aktuelle Verzeichnis:

cp ../neu.txt .

Jetzt kopieren wir alle Dateien aus dem aktuellen Verzeichnis ins übergeordnete Verzeichnis. Weil dort schon die gleichnamigen Dateien liegen, wollen wir das Überschreiben mit j bestätigen:

cp -i \* ..

(die Wildcard \* steht wieder für alle Dateien im aktuellen Verzeichnis)

### Verschieben und umbenennen - mv

Zum Verschieben, wie auch zum Umbenennen benutzen wir mv (move):

mv quelle ziel(verzeichnis)

Wir wechseln zunächst mit cd wieder ins Home-Verzeichnis. Anschließend verschieben wir "neu.txt" vom Verzeichnis "Mond" nach "mond":

mv Mond/neu.txt mond

Jetzt finden wir "neu.txt" nicht mehr im Verzeichnis "Mond", sondern nur noch in "mond".

Wir wollen aber nicht, dass sie dort "neu.txt" heißt, sondern "alt.txt":

mv mond/neu.txt mond/alt.txt

So heißt sie nun!

Weil das mit "Mond" und "mond" zu verwirrend ist, wollen wir "mond" überhaupt umbenennen. "mond" soll "anders" heißen:

mv mond anders

Ein ls bestätigt uns - kein "mond" mehr, dafür "anders".

### Dateien löschen - rm und rmdir

Zum Löschen von Dateien benutzen wir rm (remove):

rm (-optionen) dateiname

**Achtung:rm mit Administratorrechten ausgeführt kann im schlimmsten Fall das gesamte Betriebssystem löschen. Und auch als normaler Benutzer kann ich immer noch das Home-Verzeichnis und alle darin befindlichen Dateien vernichten.**

Wir wechseln zunächst ins Verzeichnis "Mond/mond" und erzeugen dort noch einige Dateien:

cd Mond/mond

touch datei1.txt datei2.txt datei3.log

ls um zu sehen, welche Dateien nun vorhanden sind. Eine einzelne Datei löschen wir relativ gefahrlos mit:

rm datei1.txt

Man kann aber auch hier wieder \* benutzen, um mehrere Dateien auf einmal zu eliminieren:

rm \*.txt

löscht alle Datein in "mond", die auf ".txt" enden. Wir überprüfen das mit ls.

Noch gefährlicher ist es, alle Dateien im aktuellen Verzeichnis zu löschen. Wir sollten umbedingt zuvor überprüfen, in welchem Verzeichnis wir uns gerade befinden (pwd).

rm \*

Jetzt befinden sich keine Dateien mehr im aktuellen Verzeichnis. Wir wechseln ins Verzeichnis "Mond":

cd ~/Mond

Am gefährlichsten ist rm, wenn wir die Option -R verwenden. Normalerweise kann rm nur normale Dateien und Links löschen. Der Versuch ein Verzeichnis zu löschen (rm mond) wird mit einer Fehlermeldung quittiert.

rm -R \*

löscht auch alle Unterverzeichnisse mit allen darin enthaltenen Dateien!

Also, immer vor dem ENTER innehalten und überlegen: Will ich das wirklich alles löschen?!

Zum Löschen von leeren Verzeichnissen gibt es auch rmdir (remove directory):

rmdir verzeichnisname

Dieser Befehl ist weniger effektiv als rm -R, aber auch erheblich weniger gefährlich.

Nachdem wir mit cd ins Home-Verzeichnis gewechselt haben, löschen wir das leergeräumte Verzeichnis "Mond":

rmdir Mond

Der Versuch mit rmdir anders auch gleich das Verzeichnis "anders" zu löschen scheitert, da sich in "anders" noch Dateien befinden (diese möchten wir auch noch ein wenig behalten).

### Links erzeugen - ln

Manchmal ist es sinnvoll, eine Datei an mehreren Orten verfügbar zu haben. Oder aber, wir wünschen die Möglichkeit, die Datei auch unter einem Namen vorliegen zu haben. Dazu erzeugen wir mit ln (link) symbolische Links:

ln -s ziel linkname

Das Verzeichnis "anders" soll nun doch auch wieder "mond" heißen:

ln -s ~/anders ~/mond

Wir probieren gleich einmal aus, dem Link zu folgen:

cd ~/mond

Hier legen wir noch rasch ein Unterverzeichnis an:

mkdir irgendwo

In "irgendwo" soll es nun einen Link zur Datei "alt.txt" geben. Der (relative) Pfad des Ziels ist immer vom Ort des angelegten Links aus zu sehen - nicht vom aktuellen Verzeichnis. Da wir den Link in "irgendwo" anlegen wollen führt der Pfad zum Ziel ins übergeordnete Verzeichnis:

ln -s ../alt.txt irgendwo

(wenn der symbolische Link dort genauso heißen soll wie das Ziel, müssen wir nur das Verzeichnis angeben)

Schon gibt es auch dort ein "alt.txt".

Wir wollen "alt.txt" aber auch unter dem Namen "neu.txt" finden können:

ln -s ../alt.txt irgendwo/neu.txt

Ohne -s aufgerufen erzeugt ln sogenannte "harte Links". Ein "harter Link" ist nicht bloß ein zusätzlicher Pfad zu einer anderswo befindlichen Datei, sondern die Datei selbst. Im Prinzip ist jede Datei ein "harter Link", wovon es auch mehrere geben kann. Jede Veränderung eines Links wirkt sich auch unmittelbar auf den anderen aus (zum Beispiel beim Umbenennen). Ob von einer (normalen) Datei mehrere "harte Links" existieren, sehen wir an der Zahl, die bei der Ausgabe von ls -l nach den "Dateirechten" angeführt ist - im Allgemeinen also "1" (z.B. -rw-r--r-- 1).

In der Praxis kommen mehrere "harte Links" nicht so häufig vor und so begnügen wir uns mit ln -s, den "symbolischen Links".

## Kapitel 3: Dateirechte

### Dateirechte ändern - chmod

Unter Linux hat jede Datei (auch Verzeichnisse) einen Besitzer und bestimmte Zugriffsrechte. Wir unterscheiden zwischen Lese-, Schreib- und Ausführungsrechten. Diese können bei jeder Datei für den Besitzer selbst, für Angehörige einer definierten Gruppe und für alle anderen festgelegt werden.

Mit ls -l bekommen wir diese angezeigt. Nach dem ersten Zeichen, das für den Dateityp steht (also -, d oder l), kommen 9 Zeichen, die genau jene Zugriffsrechte angeben:

(1.Zeichen=Typ)(2.-4.Zeichen=Besitzer)(5.-7.Zeichen=Gruppe)(8.-10.Zeichen=Andere)

Es gibt jeweils:

r

für Lesen (read)

w

für Schreiben (write)

x

für Ausführen (execute) - bzw. bei Verzeichnissen, den Inhalt auflisten

-

wenn das entsprechende Recht nicht vorhanden ist

Zum Beispiel:

-rw-r--r--

für eine normale Datei, die von allen gelesen, aber nur vom Besitzer geschrieben werden darf.

drwxr-xr-x

für ein Verzeichnis, das von allen gelesen und aufgelistet, aber wieder nur vom Besitzer geschrieben werden darf.

lrwxrwxrwx

für einen symbolischen Link, den jeder auch wieder löschen darf

Dies sind die häufigsten Fälle. Gelegentlich sieht man aber auch Berechtigungen wie:

-rw-rw----

hier dürfen Besitzer und Gruppe lesen und schreiben. Anderen ist es aber nicht gestattet, die Datei auch nur zu lesen.

Auch seine Berechtigung hat:

-r--------

hier darf nur der Besitzer lesen, aber auch er hat hier keine Schreibrechte, was ihn davor bewahrt, diese Datei eines Tages versehentlich zu löschen. Wenn er die Datei doch einmal löschen will, muss er zuerst die Rechte ändern.

Dazu gibt es chmod (change modus):

chmod modus dateiname

Es gibt 2 Methoden "modus" anzugeben:

### 1. ugoa-Methode

u  
steht für den Besitzer (user)

g  
steht für die Gruppe (group)

o  
steht für Andere (others)

a  
steht für alle (all)

Die Rechte r (lesen), w (schreiben) und x (ausführen) können jeweils hinzugefügt (+) oder entfernt (-) werden. Wir wechseln zunächst ins Verzeichnis "anders" (so wir nicht noch dort sind - "mond" ist ja auch "anders"):

cd ~/anders

Die Rechte der Datei "alt.txt" sind jetzt wahrscheinlich -rw-r--r--. Mit ls -l können wir uns das ansehen.

Wir wollen der Datei die Ausführungsrechte für alle geben:

chmod a+x alt.txt

Ein ls -l offenbart, dass sich die Rechte auf -rwxr-xr-x geändert haben.

Nun wollen wir auch dem Besitzer die Schreibrechte nehmen:

chmod u-w alt.txt

Die Rechte der Datei sind jetzt -r-xr-xr-x.

Nun wollen wir Gruppe und Anderen die Ausführungsrechte wieder nehmen und gleichzeitig dem Besitzer die Schreibrechte einräumen:

chmod go-x,u+w alt.txt

Mit ls -l sehen wir -rwxr--r--. Wir können also u, g und o kombinieren, bzw. durch Komma getrennt, auch mehrere Änderungen beauftragen.

Zuletzt wollen wir allen alle Rechte zugestehen:

chmod a+rwx alt.txt

(es macht nichts, wenn Rechte hinzugefügt werden, die schon vorhanden sind - wir tun dies, wenn wir zuvor nicht einmal nachgesehen haben, welche Rechte die Datei wirklich hatte)

So darf nun jeder alles: -rwxrwxrwx

Häufig findet man aber auch eine andere Darstellung des "modus":

### 2. numerische Methode

Dabei gibt es für jedes Recht eine Zahl:

1  
für Ausführen

2  
für Schreiben

4  
für Lesen

Für die vorhandenen Rechte werden die Zahlen addiert. Für den Besitzer, die Gruppe und Andere werden nun die addierten Zahlen zu einer 3-stelligen Ziffernfolge zusammengestellt:

644 entspricht a+r,u+w

der Besitzer darf Lesen und Schreiben (4+2), Gruppe und Andere dürfen nur Lesen (4)

755 entspricht a+rx,u+w

der Besitzer darf Lesen, Schreiben und Ausführen (4+2+1), Gruppe und Andere dürfen Lesen und Ausführen (4+1), aber nicht Schreiben.

chmod 600 alt.txt

Nur noch der Besitzer darf lesen und schreiben: -rw-------

### Besitzer ändern - chown und chgrp

chown besitzer dateiname

Eine Datei oder ein Verzeichnis wird auf den neuen Besitzer "besitzer" übertragen. Auch dies geht wieder für mehrere Dateien in einem Rutsch:

chown besitzer \* (nicht ausprobieren!) überträgt alle Dateien im aktuellen Verzeichnis an "besitzer".

chown -R besitzer verzeichnis (nicht ausprobieren!)

ändert den Besitzer gleich auch in allen im Verzeichnis befindlichen Dateien und Unterverzeichnissen auf "besitzer".

Aber: Ich kann nicht verschenken, was mir nicht gehört. chown darf ich nur bei meinen eigenen Dateien ausführen (Dateien also, bei denen ich als Besitzer eingetragen bin). Dies gilt auch für das Ändern von Rechten über chmod.

Mit chown kann auch gleich die Gruppe geändert werden:

chown besitzer:gruppe dateiname (nicht ausprobieren!)

ändert den Besitzer auf "besitzer" und die Gruppe auf "gruppe".

Soll nur die Gruppe geändert werden gibt es chgrp:

chgrp gruppe dateiname

## Kapitel 4: Das System konfigurieren

### Mit alias einen neuen Befehl erzeugen

Man kann der bash auch "Fremdsprachen" beibringen, indem man einen Alias erzeugt:

alias aliasname='programm (-optionen ...)'

Manch einer hat vielleicht schon in der Windows-Shell herumgetippt. Dort benutzt man dir (directory) um ein Verzeichnis detailiert aufzulisten. Wir können es einrichten, dass auch die Bash den Befehl dir versteht:

alias dir='ls -l'

(Hochkomma-Umklammerung ist notwendig, wenn der Befehl aus mehr als einem Wort besteht - d.h. Leerzeichen enthält)

Ab jetzt können wir dir eingeben, um das Verzeichnis aufzulisten:

dir

Ein so gesetzter Alias existiert nur bis zum Beenden der Shell, in der er gesetzt wurde.

Wenn man einen Alias ausdrücklich löschen will, gibt man folgendes ein:

unalias dir

Und weg ist er! Der Befehl dir wird ab jetzt mit der Meldung bash: dir: Datei oder Verzeichnis nicht gefunden quittiert.

### Mit nano die Konfigurationsdatei .bashrc editieren

Der Texteditor nano ist auf den meisten Systemen schon von Anfang an an Bord. Wenn nicht, sollte er für diese Übung nachinstalliert werden. Er ist leicht und intuitiv zu bedienen (in der Fußzeile stehen die verfügbaren Kommandos, die mit [Strg]+[Taste] ausgeführt werden) und mein Lieblingseditor. Er wird aufgerufen mit:

nano dateiname

Über einen Eintrag in der Datei .bashrc im Home-Verzeichnis können Aliase auch dauerhaft eingericht werden. Dazu öffnen wir .bashrc mit dem Editor nano:

nano ~/.bashrc

Mit der Pfleiltaste ↓ gehen wir bis ans Ende der Datei.

Dort fügen wir eine neue Zeile mit der Alias-Zuweisung von zuvor ein:

alias dir='ls -l'

Wir speichern mit den Tasten [Strg]+[O], gefolgt von ENTER ab und schließen den Editor mit [Strg]+[X] - fertig! Beim nächsten Aufruf der Bash steht der Befehl zur Verfügung.

Um zu sehen, welche Aliase bereits vorhanden sind, geben wir folgendes ein:

alias

Variablen definieren und mit echo ausgeben

Ein Programm, mit dem sich beliebige Inhalte auf den Bildschirm drucken lassen ist echo:

echo text

Wir testen das mal:

echo "Hier sitze ich nun"

(wir setzen umklammernde Hochkommas, weil Leerzeichen im Text vorkommen)

druckt in die nächte Zeile: Hier sitze ich nun.

Gut, das ist noch nicht so beeindruckend.

Wir können aber auch den Inhalt von Variablen ausgeben. Diese müssen wir aber zuvor noch definieren:

variable=wert

Der Aufruf der Variablen erfolgt dann immer mit einem vorangestellten $:

echo $variable

Das versuchen wir:

MeinText="Hier sitze ich nun"

echo $MeinText

liefert wiederum Hier sitze ich nun.

Ich kann aber auch Zahlen in Variablen speichern und dann mit ihnen rechnen. Wir verwenden dafür let:

let argument

Argument ist hier eine Variablendefinition oder eine Rechenoperation.

Zum Beispiel:

let A=100

let B=150

let C=$A+$B

echo $C

Hier bekomme ich als Ergebnis 250 ausgegeben. Wenn ich let weglasse, lautet das Ergebnis 100+150!

### Umgebungsvariablen

Es gibt auch viele spezielle Variablen. In diesen stehen bestimmte Informationen zur Verfügung, ohne, dass sie definiert werden müssen:

echo $HOME liefert den Pfad des Home-Verzeichnisses

echo $HOSTNAME den Rechnernamen

echo $LOGNAME den Benutzernamen

echo $UID die Benutzer ID

Sie werden vor allem bei Shell-Scripts benötigt.

Eine weitere, fest definierte Variable ist $RANDOM. Dennoch kann sie sehr wohl definiert werden. Sie gibt nämlich Zufallszahlen aus und bei der Definition wird ihr ein Startwert zugewiesen:

RANDOM=123

echo $RANDOM

echo $RANDOM

echo $RANDOM

usw.

### Aufruf von Programmen - der Suchpfad $PATH

Programme (im weitesten Sinne) sind ausführbare Dateien bei deren Aufruf bestimmte Funktionalitäten zur Verfügung gestellt werden. In der Bash erfolgt der Aufruf durch Angabe des Pfads der Datei.

pfad/programm (-optionen parameter)

Es kann sich beim "Pfad" wieder um einen "relativen", ausgehend vom aktuellen Verzeichnis:

verzeichnis/programm oder einen "absoluten", ausgehend vom Wurzelverzeichnis:

/verzeichnis/programm

handeln.

Auch wenn ich direkt im aktuellen Verzeichnis ein Programm aufrufen möchte, muss ich den Pfad angeben - das aktuelle Verzeichnis . wird hier benötigt:

./programm

Um das Programm ls, welches sich im Verzeichnis /bin befindet, auszuführen, müsste ich also /bin/ls aufrufen. Warum genügt es aber, nur ls aufzurufen?

Die Antwort ist in der Variablen $PATH gespeichert:

echo $PATH

gibt /usr/local/bin:/usr/bin:/bin:/usr/games oder so ähnlich aus. Dies ist der aktuelle "Suchpfad" und Programme, die in Verzeichnissen liegen, die in dieser durch ":" getrennten Liste aufgeführt sind, können direkt durch Angabe ihres Dateinamens aufgerufen werden. Die Reihenfolge legt fest, wo die Bash zuerst nachsieht. Wird in /usr/local/bin ein Programm des angegebenen Namens gefunden, wird in /usr/bin nicht mehr nachgesehen und ein vorhandenes gleichnamiges Programm bleibt unberücksichtigt.

Anders als die fest belegten Umgebungsvariablen zuvor lässt sich $PATH jedoch verändern. Wir erzeugen zunächst ein neues Verzeichnis in unserem Home-Verzeichnis und integrieren es in den Suchpfad:

mkdir ~/bin

PATH=~/bin:$PATH

echo $PATH

gibt jetzt an erster Stelle das Verzeichnis "bin" in unserem Homeverzeichnis an. Wir können dort von uns selbst erstellte Scripts ablegen und sie durch einfachen Aufruf ihres Namens ausführen ... etwas später!

Das Verzeichnis muss nicht unbedingt "bin" heißen, aber es ist eine Konvention, dass ausführbare Dateien in "bin" (binaries) liegen.

Um zu sehen, wie sich das System ohne Suchpfad verhält, können wir $PATH einmal löschen. Um sie nachher nicht wieder neu eingeben zu müssen, speichern wir sie vorher in $Backup:

Backup=$PATH

PATH=

ls

liefert uns jetzt bash: ls: Datei oder Verzeichnis nicht gefunden. Für den Aufruf von ls benötigen wir jetzt tatsächlich den vollen Pfad:

/bin/ls

Rückgängig machen wir das mit unserem Backup:

PATH=$Backup

Wie schon bei Alias besprochen, ist auch die angepasste $PATH nur in dieser Shell verfügbar und auf deren Lebensdauer beschränkt.

Wollen wir ~/bin dauerhaft im Suchpfad haben, müssen wir auch dies in .bashrc eintragen:

nano ~/.bashrc

Fügen wir am Ende einfach diese Zeile hinzu:

PATH=~/bin:$PATH

Achtung: In einigen Systemen ist ein ~/bin-Verzeichnis schon vorgesehen und wird $PATH hinzugefügt, sobald es existiert. Nach dem Anlegen des Verzeichnisses ~/bin sollten wir also ein neues Terminal öffnen und mit echo $PATH überprüfen, ob das Verzeichnis nicht schon im Suchpfad vorhanden ist.

## Kapitel 5: Umleitungen

### Ausgabe in Datei umleiten - > und >>

Manchmal wollen wir die normale Ausgabe nicht am Bildschirm sehen, sondern lieber in eine Datei schreiben.

programm (-optionen parameter) > dateiname

Wir wechseln zunächst wieder mit cd ~/anders in unser Testverzeichnis.

Dort leiten wir die Ausgabe von ls -l in eine Datei um:

ls -l > info.txt

Diese Datei können wir später nach Lust und Laune lesen, ausdrucken, oder einem Freund per Mail schicken. Allerdings funktioniert dieses Verfahren nur mit Programmen, welche die Standardausgabe (die normalerweise auf den Bildschirm druckt) für ihre Ausgabe benutzen.

Den Inhalt der Datei können wir uns mit less ansehen:

less info.txt

Wir können der Datei auch noch etwas hinzufügen. > überschreibt eine bestehende Datei, >> fügt ihr etwas hinzu:

programm (-optionen parameter) >> dateiname

Das Programm uname gibt Informationen zum Kernel (Betriebssystemkern) aus. Wir fügen diese unserer Datei "info.txt" hinzu:

uname -a >> info.txt

Jetzt stehen wir mit less am Ende der Datei "info.txt" noch Angaben zu unserem Betriebssystem.

### Fehlerausgabe von normaler Ausgabe trennen

Wir erinnern uns an das Problem mit den angezeigten Fehlermeldungen von find. Wenn wir dem "Umleitungszeichen" eine Ziffer voran stellen, können wir nur normale oder nur Fehlermeldungen umleiten.

Um die **Fehlermeldungen (2)** nicht mehr zu sehen, können wir sie in eine ganz spezielle Datei umleiten. "/dev/null" ist ein "schwarzes Loch" und alles, was wir dort hin schreiben, ist unwiederbringlich verschwunden.

find /var -name syslog 2> /dev/null

Wir können aber auch die normalen Ausgaben (1) in eine Datei schreiben und Fehlermeldungen verwerfen:

find /var -name syslog 2> /dev/null 1> syslogpfad.txt

Ein less syslogpfad.txt überzeugt uns, dass nur der Pfad von "syslog" in die Datei geschrieben wurde.

### Inhalt einer Datei für die Eingabe nutzen - <

Auch der umgekehrte Weg ist möglich: die Eingaben eines Programmes aus einer Datei entnehmen:

programm < datei

Natürlich geht das nur, wenn das Programm auf Eingaben wartet. Beispiel gefällig?

Beim Überschreiben einer Datei mit cp -i gibt es eine Rückfrage die ja oder nein erwartet.

Wir legen eine Datei mit dem Inhalt ja an:

echo ja > input.txt

ja wird hier nicht auf den Bildschirm ausgegeben, wie dies bei echo eigentlich üblich ist, sondern in die Datei "input.txt" geschrieben (siehe oben). Nun wollen wir eine bestehende Datei "datei1.txt" mit "datei2.txt" überschreiben. Zunächst benutzen wir nochmals echo um die beiden Test-Dateien zu erzeugen:

echo "Erste Datei" > datei1.txt

echo "Zweite Datei" > datei2.txt

Anführungszeichen können immer verwendet werden, sind aber nicht notwendig, wenn es sich nur um ein Wort handelt. Nun kopieren wir:

cp -i datei2.txt datei1.txt < input.txt

"datei1.txt" wird ohne Rückfrage überschrieben, das heißt, die Rückfrage kommt schon, aber wir haben sie mit dem Inhalt von "input.txt" bereits beantwortet!

### Rohrleitungen zu einem anderen Programm - |

Pipes (engl., Rohrleitungen) leiten den Datenstrom eines Programms an ein anderes weiter. Der Befehl lautet:

programm1 (...) | programm2 (...)

Wir verwenden wieder die Ausgabe von ls -l, leiten aber diesmal an less weiter:

ls -l | less

Das Zeichen | macht nun die Ausgabe von ls -l zur Eingabe von less.

### Auslesen und filtern - cat und grep

Zuletzt noch zu zwei Programmen, die bislang noch gefehlt haben - cat und grep:

cat dateiname

cat liest Dateien bitweise aus und gibt sie auf die Standardausgabe aus. Dies können wir benutzen, um das Ende langer Textdateien anzusehen:

cat /var/log/kern.log

Bei log-Dateien interessieren uns meist vor allem die letzten Einträge und cat spult die Datei bis zum Ende ab. Manchmal ist aber auch das zuviel und ich will nur alle Einträge sehen, welche meine NVIDIA-Grafik betreffen. Da kommt mir grep zu Hilfe:

grep (-optionen) muster dateiname

Ich kann nämlich grep verwenden, um mit der Standardausgabe eines anderen Befehls als Quelle, Zeilen mit einem bestimmten Muster herauszufiltern und auszugeben.

cat /var/log/kern.log | grep usb

Dieser Befehl liefert alle Zeilen, in denen "usb" vorkommt – allerdings nur genau so geschrieben!

Will ich auch Zeilen mit großgeschriebenen Buchstaben, benötige ich die Option -i.

cat /var/log/kern.log | grep -i usb

Jetzt erhalte ich auch Zeilen, in denen "USB" nur großgeschrieben steht. Meist steht am Anfang jedes Eintrags, welches Modul die Meldung veranlasst hat. Dies ist als "Muster" üblicherweise eine gute Wahl.

Wenn ich alle Einträge sehen will, die NICHT "usb" enthalten, kann ich diese auch ausschließen:

cat /var/log/kern.log | grep -v usb

Nun erhalte ich nur den Rest.

cat kann aber auch mehrere Dateien auf einmal verarbeiten:

cat dateiname dateiname ...

Zum Beispiel:

cat datei1.txt datei2.txt > zusammen.txt

Dabei kann auch eine Wildcard für beliebige Zeichen verwendet werden:

cat \*.txt > alle\_textdateien.txt

Und cat funktioniert nicht nur für Text Dateien:

cat musik.mp3 kopie.mp3

wobei "musik.mp3" für eine beliebige mp3-Datei steht.

Allerdings dürfen Multimediadateien so normalerweise nicht miteinander verbunden werden!

## Kapitel 6: Systemadministration

Bisher haben wir uns nur im Wirkungsfeld eines einfachen Benutzers bewegt. Wenn wir an einem System arbeiten, das von einem professionellen Administrator betreut wird, könnten wir es auch dabei belassen. Auf dem eigenen PC müssen wir aber wohl selbst Hand anlegen und uns in den "Superuser" verwandeln.

### Der Superuser - su und sudo

Der Administrator heißt auf UNIX-Systemen eigentlich "root" (engl. Wurzel), wird aber von Programmen oft auch als Superuser bezeichnet. Wir könnten uns beim Login gleich als "root" anmelden. Das wird aber nicht empfohlen, denn der Superuser ist sehr mächtig und ganz leicht kann ein kleiner Fehler zur Zerstörung des gesamten Systems führen. Deshalb melden wir uns immer als einfacher Benutzer an und werden nur im Bedarfsfall zum User "root". Das Programm su (switch user) erlaubt uns, die Identität zu wechseln:

su (-) (benutzername)

Ohne Benutzerangabe wird zu "root" gewechselt, was wahrscheinlich auch der häufigste Fall ist. Nach Absetzen des Befehls wird man zur Passworteingabe aufgefordert. Wir müssen das Administratorpasswort, welches wir bei der Installation vergeben haben, eingeben. Bei Ubuntu wird zunächst gar kein Passwort für "root" vergeben und das ist auch nicht vorgesehen. Ubuntu-User können diesen Absatz überspringen und bei sudo fortfahren:

su -

Das - sorgt dafür, dass wir eine Login-Shell öffnen, während wir andernfalls nur an Ort und Stelle als "root" weiter arbeiten. Wir erkennen den Unterschied daran, dass wir uns mit su - im Home-Verzeichnis des "root"-Users, also /root und nicht mehr im zuletzt aktiven Verzeichnis wiederfinden.

Wollen wir die Administratorrolle wieder ablegen, beenden wir die "root"-Shell (wie übrigens jede Shell) mit exit:

exit

Wenn wir gerade ein Programm kompiliert haben und unser aktuelles Verzeichnis das entsprechende Programmverzeichnis ist, müssen wir hier auch den Installationsschritt make install setzen und wollen nicht ins Home-Verzeichnis von "root". Hier wechseln wir sinnvollerweise ohne Minus:

su

Ehe wir aber mehr Sicherheit im Umgang mit der Shell erlangt haben, wollen wir noch nicht zum Üben an gobalen Systemdateien herumspielen und ansonst besteht überhaupt kein Unterschied zum normalen User. Wir beenden den Ausflug wieder mit exit.

Um etwas mit Administratorrechten ausführen zu können, ohne deshalb wechseln zu müssen, gibt es das Programm sudo. Auf manchen Systemen wird dies allerdings gar nicht installiert sein und ohne ein wenig Erfahrung sollte es vorerst auch nicht nachinstalliert werden. Bei Ubuntu ist dies allerdings das Standardmittel zur Administration:

sudo programm (-optionen ...)

Wir setzen vor den jeweiligen Befehl einfach sudo. Das geforderte Passwort ist diesmal das Benutzerpasswort!

sudo touch ~/rootdatei.txt

erzeugt in unserem Homeverzeichnis eine Datei, die "root" gehört und für die wir als normaler Benutzer keine Schreibberechtigung haben. ls -l überzeugt uns davon. Um sie zu entfernen benötigen wir wieder root-Rechte:

sudo rm ~/rootdatei.txt

Wer mit sudo was machen darf steuert die Datei /etc/sudoers, aber deren Konfiguration verschieben wir auf einen noch folgenden Teil.

## Kapitel 7: Hilfe!

Das wär's nun schon fast, allerdings gibt es noch einige ganz unverzichtbare Programme, die ich hier vorstellen will:

### Programmdokumentationen - man, info und help

Wir haben bislang nur wenige Optionen verwendet und oft ist das auch nicht notwendig. Für gewisse Aufgaben ist es aber unerlässlich, mehr aus den Programmen herauszuholen. Dazu gibt es zu fast jedem Programm eine Dokumentation. Diese kann ich mir mit man (manual) anzeigen lassen kann:

man programmname

Die Dokumentation von ls erhalte ich mit:

man ls

Viele man-Seiten sind auf Deutsch verfügbar und wenigstens bei den grundlegenden Programmen, die wir hier überwiegend besprochen haben, von sehr guter Qualität. Bei spezielleren Programmen aus Fremdquellen kann die man-Seite aber auch unverständlich oder völlig unübersichtlich daherkommen, bzw. gar nicht vorhanden sein. Ein Blick hinein lohnt dennoch fast immer.

Achtung: Um deutschsprachige "manpages" für die hier besprochenen grundlegenden Programme zu bekommen, muss unter Umständen noch ein eigenes Paket (manpages-de) installiert werden.

Alternativ können wir auch info versuchen:

info (programmname)

Beispiel:

info ls

Ohne Programmangabe erhalten wir zunächst eine Übersicht, aus der wir durch auswählen mit dem Cursor und ENTER den gewünschten Eintrag angezeigt bekommen:

info

Etwas einfacher und nur für ganz grundlegenden Programme, die schon anfangs bei jeder Installation verfügbar sind, ist help:

help (programmname)

Es enthält allerdings weniger die bisher überwiegend besprochenen, ständig verwendeten Programme wie cd, ls oder cp, sondern beschränkt sich mehr auf Programme, deren Optionen auch der geübtere Anwender nicht im Kopf behält. Die Informationen sind auch bewusst knapp und sollen mehr erinnern als lehren:

help echo

Ohne Programmname aufgerufen, erhalten wir auch hier wieder eine Auflistung verfügbarer Programme (bzw. hier ganze Befehle):

help

Und help gibt es fast bei jeder Shell - sogar der Windows-Shell! Wenn wir also eimal vor einer fremden Shell sitzen und schon den dritten Versuch absolviert haben, der command not found hervorgebracht hat, lohnt praktisch immer ein help, das uns dann mitteilt, welche commands uns denn zur Verfügung stehen.

Die hier erlernten Befehle werden übrigens beinahe unverändert auf allen UNIX-Shells und den meisten anderen Shells in ähnlicher Form (nicht allerdings in der Windows-Shell!) funktionieren!!!

Nun, das war's für den Anfang. Wir beenden mit

exit