Einführung in Linux

Ulrike Nitzsche
IFW Dresden

Stand: 22. Oktober 2008

Inhaltsverzeichnis

Kapitel 1

Einführung

1.1 Betriebssysteme

Ein Betriebssystem (engl. *operating system* OS) ist eine Software, die den Betrieb eines Computers ermöglicht. Betriebssysteme bestehen in der Regel aus einem Kern (engl. *kernel*), der die Hardware des Computers verwaltet, sowie grundlegenden Systemprogrammen, die dem Start des Betriebssystems und dessen Konfiguration dienen. Man kann zwischen

- Einbenutzer- und Mehrbenutzersystemen (*single user* vs. *multi user*)
- Einzelprogramm- und Mehrprogrammsystemen (*single task* vs. *multi task*)
- Stapelverarbeitungs- und Dialogsystemen (*batch* vs. *interactive*)

unterscheiden. Heute wird der Begriff **Betriebssystem** meist auf den Betriebssystemkern beschränkt, d.h. das Betriebssystem in diesem Sinn stellt lediglich eine Schnittstelle für Systemfunktionen (*system calls*) bereit, die von Programmen genutzt werden können. Die Aufgaben des Betriebsystems umfassen

- Steuerung und Überwachung der Programmausführung
- Starten und Stoppen von Programmen
- Betriebsmittel (Ressourcen wie CPU-Zeit, Speicher, Plattenplatz, Bandbreiten) verwalten und optimiert zuordnen
- Kosten der Benutzung erfassen (*accounting*)
- Zugang und Berechtigungen prüfen: Passwortschutz, Zugang zu Ressourcen
- Verbindungen mit anderen Rechnern herstellen

Eine etwas weitere Fassung des Begriffes beinhaltet neben der Verwaltung der Hardwareressourcen die Bereitstellung von Diensten wie Interprozesskommunikation, Datei- und Verzeichnisdiensten, Netzwerkdiensten und eine Kommandosprache. Wir werden uns in dieser Einführung diesem Aufgabenteil widmen.

1.2 Geschichte von Unix und Linux

1963-1969 gab es ein Gemeinschaftsprojekt des MIT, General Electric und der Bell Labs von AT&T, das ein völlig neues Betreibssystem (Multiplexed Information and Computing Service - *multics*) zum Ziel hatte. Aus Kostengründen zog sich Bell Labs aus dem Projekt zurück. Zwei Mitarbeiter der Bell Labs - Ken Thompson und Dennis Ritchie - nutzten Ideen aus diesem Projekt, um eine abgespeckte Version (*unics*) zu schreiben, um auf einer DEC PDP-7 das von Thompson geschriebene Spiel „Space Travel“ zu spielen. 1973 war die Version 4 dieses OS fast vollständig in der Hochsprache C implementiert, nicht mehr in Assembler. Damit war die Voraussetzung gegeben, dass UNIX auf praktische alle Rechnerarchitekturen portiert werden kann. UNIX verbreitete sich bei AT&T sehr schnell, allerdings konnte es auf Grund einer Entscheidung der Kartellgerichte nicht kommerziell verwertet werden. Deshalb wurde UNIX gegen eine symbolische Gebühr an Universitäten lizensiert, wo es breit eingesetzt wurde. Eine besondere Rolle spielte die University of California Berkeley (UCB), an der die Berkeley Software Distribution (*BSD*) entwickelt wurde. Später entstanden daraus nach Entfernung des AT&T-Codes die freien BSD-Varianten FreeBSD, NetBSD und OpenBSD.
Seit 1975 verkaufte AT&T auch kommerzielle Quellcodelizenzen, aus denen auf spezielle Hardware optimierte UNIX-Versionen (DEC: *ultrix, OSF1*; Hewlett-Packard: *HP-UX*; SUN: *SunOS, Solaris*; IBM: *AIX*; Siemens: *sinix*; Microsoft: *Xenix*; ...) entstanden. Preis dieser Entwicklung war eine starke Zersplitterung von UNIX, die auch durch Standardisierungsversuche nicht aufzuheben war. Ab Mitte der 80er Jahre wurde der Quellcode von AT&T den Universitäten nicht mehr zur Verfügung gestellt.
Deshalb entschloss sich Andrew Tanenbaum, Informatik-Professor in Amsterdam für Lehrzwecke ein OS (*Minix*) für PC zu schreiben, das die Funktionalität von UNIX hat und im Quelltext für wenig Geld zur Verfügung gestellt wurde. Es sollte aber im wesentlichen Lehrzwecken dienen, deshalb war Tanenbaum sehr restriktiv beim Einbau neuer Funktionalitäten.
Der Minix-Anwender Linus Torvalds entwickelte schließlich ein kleines, auf einem 386er lauffähiges OS (*Linux*), das keine Verbindung zu Minix im Quellcode hatte. Die Version 0.12 war im Januar 1992 fertig, wurde über das usenet und FTP verbreitet und war der Beginn der Weiterentwicklung dieses OS durch eine weltweite Nutzergemeinde. Auf dem Linuxkernel waren alle Programme des ebenfalls Anfang der 80er Jahre am MIT entstandenen GNU-(*„GNU's Not UNIX“*) Projektes lauffähig. Das ist eine Sammlung von UNIX-Werkzeugen wie Compiler, Programmen zur Dateisystem- und Softwareverwaltung, Kommandosprachen etc., die ebenfalls aus einer freien Entwicklergemeinde entstanden sind und mittlerweile einen systemübergreifenden Quasistandard für Aufgaben dieser Art unter UNIX darstellen.

1.3 Linux: Kernel und Distributionen

Ursprünglich wurde von Linus Torvalds der Name „Linux“ nur für den eigentlichen Systemkern benutzt. Der Linuxkernel wird zusammen mit anderen Programmen - Anwendungsprogrammen, Installationsroutinen, Dokumentation usw. - weitergegeben. Eine solche Auswahl bezeichnet man als „Linux-Distribution“.

1.3.1 Der Linux kernel

Der Kernel erfüllt die Aufgaben des OS im engeren Sinne (s. Abschnitt 1.1). Der Kernel selbst ist ein Programm, das beim Bootvorgang geladen wird und im Hauptspeicher liegt. Der Kernel erhält in Abständen CPU-Zeit und kontrolliert in diesen Intervallen die Programm- und Ressourcensteuerung. Um mit der Hardware kommunizieren zu können, benötigt ein Kernel spezielle Treiber. Außerdem sind die Systemanforderungen häufig sehr unterschiedlich: Welche Netzwerkfunktionalität muss erfüllt sein (Fileserver, Router, unterstützte Protokolle)? Soll das System für den Desktop- oder den Serverbetrieb optimiert sein? Welche Hardware soll unterstützt werden? Es ist unsinnig einen Kernel zu bauen, der alles kann - er würde zuviel Speicherplatz benötigen. Eine Auswahl der Treiber und Funktionen ist also notwendig. Diese Auswahl kann auf 2 Arten erfolgen: Zum einen kann vor dem Kompilieren des Kernels angegeben werden, welche Bestandteile dieser enthalten soll („Kernelkonfiguration“). Auf der anderen Seite können einzelne Komponenten als Module kompiliert werden, die dann bei Bedarf in den Speicher geladen und wieder entfernt werden können.

1.3.2 Linux Distributionen

Eine übliche Linux-Distribution für den PC enthält z.B.

- einen möglichst vielseitigen Linux-Kernel
- die GNU-Software, welche das grundlegende Basissystem bereitstellt (Kommandozeilenwerkzeuge)
- einen Kommandozeileninterpreter (engl. *Shell*)
- das X-Window-System, auf dem grafische Anwendungen laufen können
- einen Desktop bzw. Windowmanager (KDE, Gnome, fvwm, fluxbox, ...)
- verschiedene Anwendungsprogramme wie Office-Pakete, Editoren, E-Mailprogramme, Browser, Multimediaprogramme, ...
- Konfigurationsprogramme, um das System eigenen Bedürfnissen anzupassen, Software zu installieren und Hardware einzurichten (YaST, linuxconf, ...)

- Entwickler-Werkzeuge wie Compiler oder Interpreter für Programmiersprachen
- Softwarebibliotheken
- ein Boot-Manager zum Starten des installierten Systems und eventueller anderer installierter Systeme (lilo, grub)

Häufig benutzte Desktop-Distributionen sind z.B. SuSE, RedHat, Debian, Gentoo und Kubuntu.
Daneben gibt es Livesysteme, d.h. Systeme, die von CD laufen und nicht auf der Festplatte installiert werden. Das bekannteste ist KNOPPIX.
Spezialdistributionen werden für spezielle Einsatzszenarien erstellt, z.B. fli4l (Router), OpenWRT (WLan-Router), Beowulf (Linux cluster) ...
Eine gute Übersicht über verschiedene Linuxdistributionen findet sich in [1]

1.4 Vergleich von Unix/Linux mit Windows

Eine detailierte (nicht ganz unparteiische) Zusammenstellung der Unterschiede und Gemeinsamkeiten zwischen Linux und Windows kann unter [2] nachgelesen werden. Dabei wird auf

- Philosphie
- Technische Merkmale
- Benutzbarkeit
- Graphische Oberfläche
- Dateisystem
- Kommandozeile
- Sicherheit

ausführlich eingegangen.

1.5 Parallelbetrieb von Linux und Window

Es gibt mehrere Möglichkeiten, Linux und Windows parallel zu nutzen. Je nach Ziel des Parallelbetriebs (Lehre, „Reinschnuppern“, Bereitstellen von Applikationen) und dem finanziellen Rahmen kann man zwischen folgenden Varianten wählen:

1.5.1 2 verschiedene Rechner

Vorteile

- Keine Kenntnisse über Bootvorgang, Plattenpartitionierung etc. nötig
- Beide Betriebssysteme stehen gleichzeitig zur Verfügung
- Keine Wechselwirkungen (außer evtl. über's Netzwerk) zwischen den Betriebssystemen

Nachteile

- Ressourcenverschwendung (Hardware, Strom, Platz,...)
- parallele Datenhaltung

1.5.2 Parallelinstallationen mit boot-Auswahl

Vorteile

- Nur Hardware eines Rechners nötig
- Daten können z.B. über eine VFAT-Partition sowohl von Windows als auch von Linux aus gelesen und bearbeitet werden

Nachteile

- Know-How für Einrichtung der Bootumschaltung nötig, z.B. über Umschalten der aktiven Partition oder Benutzen eines Bootmanagers (unter Windows Konfiguration über `C:\boot.ini`, unter Linux mittels **lilo**, konfiguriert in /etc/lilo.conf oder **grub**, konfiguriert über Dateien in `/boot/grub`).
- Gleichzeitiges Nutzen von Windows- und Linux-Applikationen nicht möglich, dazu ist rebooten in das jeweiligen OS nötig.

1.5.3 remote Zugriff

Vorteile

- Gleichzeitiges Nutzen von Applikationen unter den verschiedenen OS
- Keine komplexen Veränderungen am lokalen OS
- Zusatzaufwand bei der Konfiguration des remote OS, d.h. an *einer* zentralen Stelle

Nachteile

- Voraussetzung ist stabile Netzwerkverbindung

- Aktivierung von Windows rdp-Service nötig. Als Alternative mit zusätzlichen Eigenschaften wie Load Balancing bietet sich der kostenpflichtige Einsatz von Citrix' Metaframe Server auf Basis des ICA-Protokoll an. Sichere Mehrbenutzer- Konfiguration von Windows-Applikationen erfordert zusätzliches Know-How bei der Windows-Administration.
- Zugriff über **ssh** oder **xdmcp** auf Linux muss sicher konfiguriert werden

1.5.4 Live-CD

Vorteile

- Images von Live-CD's für aktuelle Hardware (z.B. **KNOPPIX**) im Internet verfügbar.
- „Probebetrieb" von Linux ohne feste Installation auf der Festplatte oder aufwendige Netzwerkinstallationen möglich

Nachteile

- Speichern von Einstellungen umständlich (aber möglich)
- Für paralleles Nutzen von Windows-Applikationen reboot nötig
- Nachladen von Programmen von CD in den RAM zeitaufwendig, d.h. Start von Programmen mit Verzögerung

1.5.5 Virtuelle Maschine

Vorteile

- Auswahl unter verschiedenen Implementierungen möglich (VMware, XEN, qemu)
- Direkter Zugriff auf Hardware
- Hostbetriebssystem kann Windows oder Linux sein, damit ist auch Test verschiedener Gastbetriebssysteme (verschiedene Windows-Varianten, verschiedene Linux-Distributionen) möglich
- Gleichzeitiger Zugriff auf Applikationen beider OS, einschließlich Cut&Paste möglich, gemeinsame Nutzung von Dateien

Nachteile

- Leichte Performanceeinbußen durch VM (je nach CPU und RAM-Ausstattung)
- Zusätzlicher Festplattenbedarf ($\approx$ 2GB pro Gastbetriebssystem)

1.5.6 Emulatoren

Vorteile

- geringerer Ressourcenaufwand als bei kompletter Emulation des OS (WINE, win4lin)

Nachteile

- funktioniert nur für ausgewählte Applikationen

1.6 GPL

Die GPL (GNU General Public License) wurde im Januar 1989 von Richard Stallman, dem Gründer des GNU-Projektes geschrieben. Sie gewährt folgende Freiheiten:

1. Das Programm darf ohne jede Einschränkung für jeden Zweck genutzt werden. Kommerzielle Nutzung ist hierbei ausdrücklich eingeschlossen.

2. Kopien des Programms dürfen kostenlos oder auch gegen Geld verteilt werden, wobei der Quellcode mitverteilt oder dem Empfänger des Programms auf Anfrage zum Selbstkostenpreis zur Verfügung gestellt werden muss. Dem Empfänger müssen dieselben Freiheiten gewährt werden. Wer z.B. eine Kopie gegen Geld empfängt, hat weiterhin das Recht, diese dann kommerziell oder auch kostenlos zu verbreiten. Lizenzgebühren sind nicht erlaubt. Niemand ist verpflichtet, Kopien zu verteilen, weder im Allgemeinen, noch an irgendeine bestimmte Person aber wenn er es tut, dann nur nach diesen Regeln.

3. Die Arbeitsweise eines Programms darf studiert und den eigenen Bedürfnissen angepasst werden.

4. Es dürfen auch die gemäß Freiheit 3 veränderten Versionen des Programms unter den Regeln von Freiheit 2 vertrieben werden, wobei dem Empfänger des Programms der Quellcode der veränderten Version verfügbar gemacht werden muss. Veränderte Versionen müssen nicht veröffentlicht werden; aber wenn sie veröffentlicht werden, dann darf dies nur unter den Regeln von Freiheit 2 geschehen.

Neben dem Linuxkernel und den Werkzeugen des GNU-Projektes stehen auch Anwendungen wie mySQL, GIMP, OpenOffice und KDE unter der GPL. Andere Lizenzen im Bereich freier Software sind z.B. die Apache-Lizenz, die GNU Lesser General Public License (LGPL) und die BSD-Lizenz.

Kapitel 2

Erste Schritte

2.1 Anmelden, Abmelden, Herunterfahren des Systems

2.1.1 Anmelden auf der Textkonsole

Haben Sie physikalischen Zugang zu einem Linux-System, können Sie sich an sogenannten Textkonsolen anmelden. Der `init`-Prozess startet (spezifiziert in der Datei `/etc/inittab`) meist mehrere `getty`-Prozesse auf den Textkonsolen tty*1* ... tty*n*. Dieser Prozess wartet auf die Eingabe eines Nutzernamens. Wird diese Eingabe mit einem `Enter` abgeschlossen, startet ein neuer Prozess, der `login`-Prozess. Dieser fragt nach einem Passwort und vergleicht nach dessen Eingabe die Angaben für Nutzername und Passwort mit den Daten einer Nutzerdatenbank (lokal in der `/etc/passwd`, netzwerkweit über einen Verzeichnisdienst wie z.B. `NIS` oder `LDAP`). Ist die Authorisierung erfolgreich, wird das Programm gestartet, das in der Nutzerdatenbank als *Startprogramm* für einen Nutzer angegeben ist, das wird in den allermeisten Fällen eine *Shell* sein. Die Abmeldung erfolgt über die Beendigung dieses Programms, bei einer Shell mit **exit, quit, logout** oder **Ctrl-D**.
Stehen mehrere Textkonsolen zur Verfügung, kann man mit der Tastenkombination **Alt-F*n*** zwischen ihnen wechseln und die Arbeit durch das Nutzen mehrerer Konsolen strukturieren.

2.1.2 Graphische Anmeldung

Ist auf dem Linux-System eine graphische Nutzeroberfläche (das sogenannte *X Window System*) installiert, wird die Anmeldung durch deren Programme (**xdm, kdm, gdm**) übernommen. Beachten muss man, dass der X-Server, der diese Programme bereitstellt, auch auf einem entfernten Server laufen kann. Meist wird der Name des Rechners, an dem man sich anmeldet, auf dem Anmeldebildschirm angezeigt.

Von der graphischen Oberfläche kann man mit der Tastenkombination **Ctrl-Alt-F*m*** auf die *m*-te Textkonsole wechseln, von der Textkonsole kommt man mit **Alt-F*n*+1**, wobei `n` die Anzahl der Textkonsolen ist, zurück in das X Window System.

2.1.3 Anmeldung übers Netzwerk

Laufen auf dem Zielrechner spezielle Dienste wie z.B. der **ssh** - *secure shell*, **rsh** - *remote shell* oder **telnet**, so können Sie sich über das entsprechende Netzwerkprogramm auf dem Zielrechner anmelden. **rsh** und **telnet** sind unsicher, da die Passwörter im Klartext übertragen werden. Diese Dienste sollten prizipiell deaktiviert werden!

2.1.4 Herunterfahren des Systems

Bevor man ein Unix-System abschalten kann, muss es geordnet heruntergefahren werden. D.h. laufende Dienste werden konsistent beendet, Operationen im Dateisystem werden beendet, angemeldete Nutzer bekommen einen Hinweis auf die bevorstehende Abschaltung, um ggf. laufende Arbeiten zu sichern. Deshalb darf nur ein Nutzer mit Überblick über die Gesamtsystemfunktionalität (also der Nutzer „root“) oder ein Nutzer mit Zugang zur Systemkonsole den Rechner herunterfahren.
Der Nutzer `root` rebootet ein System mit **shutdown -r** <**time**> [**message**] wobei als <time> eine Zeit im Format `hh:mm` oder `+m` (m ist die Anzahl der Minuten, bevor der shutdown startet) anzugeben ist. `now` steht für `+0`.
Ein Systemhalt wird mit **shutdown -h** <**time**> [**message**] erzeugt.
Hat man physikalischen Zugang zur Systemkonsole, können viele Linuxsystem mit der Tastenkombination **Ctrl-Alt-Del** heruntergefahren werden. Dazu muss man nicht eingeloggt sein, Voraussetzung ist die magische Zeile
`ca:12345:ctrlaltdel:/sbin/shutdown -h now`
in der Datei /etc/inittab. Nur bei Rechnern, auf denen keine lokalen Daten gespeichert werden und ausschließlich als persönliche Arbeitsplätze dienen, sollte diese Zeile in der Datei /etc/inittab enthalten sein.

2.1.5 Kommandozeilen editieren

command line history

Die Shell verfügt über eine Liste der zuletzt abgesetzen Kommandos, eine sogenannte *history*. Mit dem Kommando **history** wird diese Liste nummeriert angezeigt.
Die **bash** unterstützt über die readline-Bibliothek einen `emacs`- und einen `vi`-Modus für den Kommandozeileneditor und den Zugriff auf die `history`. Voreingestellt ist meist der `emacs`-Modus (`set -o emacs`).

Ausgewählte Tastenkombinationen im `emacs`-Modus:

Tastenkombination	Bedeutung
Pfeiltasten links/rechs	Cursor links/rechts
Pfeiltasten hoch/runter	History rückwärts/vorwärts durchlaufen
Tabulatortaste	Komplettierung des aktuellen Argumentes links vom Cursor (s. command line completion)
Ctrl-A oder Pos1	Cursor auf Zeilenanfang positionieren
Ctrl-E oder End	Cursor auf Zeilenende positionieren
Ctrl-R	History rückwärts durchsuchen
Ctrl-H oder Backspace	Zeichen unter Cursor löschen
Ctrl-C	Kommandozeile abbrechen
Ctrl-L	Bildschirm löschen

command line completion

Die Tabulatortaste stellt einen Mechanismus zur Verfügung, um bei der Eingabe von Kommandozeilen Namen zu vervollständigen. Bei der Vervollständigung des ersten Wortes einer Kommandozeile sucht die bash nach einem Alias- oder Funktionsnamen, ansonsten nach einem Pfadnamen, der mit den schon eingegebenen Zeichen beginnt. Gibt es mehrere Kommandos/Pfadnamen, zu denen die eingegebenen Zeichen komplettiert werden können, so werden nach einem zweiten Drücken der Tabulatortaste alle Möglichkeiten angezeigt und man kann weitere Zeichen eingeben und ggf. dann mit der Tabulatortaste vervollständigen. Die Art der Komplettierung von Wörtern der Kommandozeile ist über das Kommando `complete` einstellbar.
Beispiele:
Bei `bunzip2` und `bzcat` nur Dateinamen komplettieren, die auf .bz2 enden:
`complete -f -X '!*.bz2' bunzip2 bzcat`
Bei `unalias` nur Alias-Namen komplettieren:
`complete -a unalias`

2.2 Befehlseingabe

2.2.1 Erste Befehle

Innerhalb einer Shell können Kommandos auf der Kommandozeile eingegeben werden. Sie werden mit einem `Enter` abgeschlossen und der Shell zur Interpretation und Ausführung übergeben.
Erste Informationen über das System erhält man durch Eingabe folgender einfacher Kommandos:

- **ls** - *list* - Zeigt die Namen der Dateien, auch der Unterverzeichnisse, an die sich im aktuellen Verzeichnis befinden
- **pwd** - *print working directory* - Zeigt den Namen des Verzeichnisses an, in dem man sich gerade befindet

- **cd** - *change directory* - Wechselt in ein anderes Verzeichnis: **cd /etc** wechselt ins Verzeichnis `/etc`, Kontrolle mit **pwd**
- **ps** - *process show* - Zeigt die Liste der laufenden Prozesse der aktuellen Shell aus, **ps aux** zeigt alle Prozesse an, die auf dem System laufen
- **top** - Zeigt die Systemaktivitäten an, die die meisten Ressourcen benötigen, interaktives Programm, Beenden mit **q**

Die allgemeine Form eines Kommandos ist:

cmd [**-Optionen**] [**Argumente**]

Die eckigen Klammern zeigen an, dass Optionen und Argumenten optional sind. Es gibt sowohl Programme, die keine Argumente benötigen (**pwd**), als auch Programme, die ohne Argumente nicht ausführbar sind (**cp**).

2.2.2 Optionen

Optionen beeinflussen das Verhalten eines Kommandos. Optionen werden gewöhnlich durch einzelne Buchstaben bezeichnet und beginnen mit einem Minuszeichen -. Optionen können miteinander kombiniert werden, indem man weitere Zeichen hinzufügt. Eine Übersicht möglicher Optionen eines Kommandos gibt dessen manpage (2.3.2).
Manche Optionen erwarten ihrerseits Argumente (**tar -f Dateiname**, **gcc -o Dateiname hello.c**).
In der GNU-Welt gibt es eine weitere Art von Optionen, die sich durch eine besondere Schreibweise auszeichnen. Sie beginnen mit einem doppelten Minuszeichen --, gefolgt von der eigentlichen Option, die meist ein ausgeschriebenes Wort ist. Lange Optionen sind „sprechender“ als kurze. Allerdings wird die Kommandozeile bei der Benutzung mehrerer langer Optionen auch unübersichtlicher.

2.2.3 Argumente

Argumente dienen nicht zur Steuerung eines Kommandos, sondern liefern diesem Informationen, die es zu bearbeiten hat, z.B. Dateinamen. Die Anzahl der Argumente kann sehr groß sein, darf aber die maximale Größe einer Kommandozeile von 128kB nicht überschreiten.

2.3 Hilfe

2.3.1 Optionen -h, --help

Bei den meisten Programmen (**cmd** steht als Abkürzung für den jeweiligen Kommandonamen) liefert die Eingabe von

cmd -h oder **cmd --help**
eine kurze Beschreibung des Programms. Dabei wird in der Regel auf den Zweck des Programms, Aufrufsyntax und die gebräuchlichsten Optionen bzw. Tastenkombinationen eingegangen.
Sollte die Ausgabe dieser Hilfe nicht auf eine Bildschirmseite passen, so kann man sie sich mit **cmd -h | more** seitenweise anschauen.

2.3.2 Kommando man

Mit dem Kommando `man` ruft man die Handbuchseiten (engl. *manual, manpages*) auf. Es gibt manpages sowohl für Befehle als auch für Systemaufrufe, Funktionen aus Systembibliotheken, spezielle Gerätedateien und Dateiformate. In den manpages kann man mit den Pfeiltasten navigieren und mit `q` zur Shell zurückkehren. Die Manpages enthalten in der Regel folgende Angaben:

- Name
- Syntax
- Beschreibung
- Optionen
- Dateien (welche Dateien nutzt das Programm, z.B. zur Initialisierung)
- Siehe auch (Verweis auf ähnliche Programme)
- Fehler (bekannte Fehler, Inkompatibilitäten)
- Autoren

2.3.3 Kommando info

Zu manchen Programmen, vor allem aus dem GNU-Projekt, gibt es zusätzliche Hilfen über das **info**-Hilfesystem. Dieses System hat ein anderes Bedienkonzept und nutzt die Möglichkeiten von Textverknüpfungen, so dass man in diesen Texten hierarchisch navigieren kann. Für die Navigation sind folgenden Tasten wichtig:

n	springt zum nächsten Dokument in der Hierarchie
p	springt zum vorhergehenden Dokument
u	geht in der Hierarchie eine Ebene hoch
Space/Backspace	Seitenweises vor- und zurückblättern
Pfeiltasten	Bewegen des Cursors
Enter	Wenn sich der Cursor über einem Link befindet, so wird zum entsprechenden Dokument gesprungen
s	Sucht nach dem Begriff, den man eingibt
q	Beenden des Programms

2.4 Weitere Informationen

Weitere Informationen in Form von Tutorials, HOWTOs etc. findet man im Dateisystem unter `/usr/share/doc` oder /usr/doc (abhängig von der Distribution) und im Internet. Einige Links finden sich im Anhang [3],[4],[5].

Kapitel 3

Das Dateisystem

3.1 Dateisysteme unter Linux

Unix speichert seine Dateien in einem einzigen Verzeichnisbaum, der mit dem Wurzelverzeichnis / beginnt. Die einzelnen Verzeichnishierarchien werden durch einen *slash* / voneinander getrennt.
Auch Geräte wie z.B. Festplatten, Bandlaufwerke, Terminals oder Drucker werden unter Unix wie Dateien behandelt.
Lange Zeit war das **ext2** Filesystem (*second extended filesystem*) unter Linux das Standardfilesystem. Sein Hauptnachteil ist, dass es kein journaling filesystem ist, so dass es durch das abwärtskompatible **ext3** Filesystem und andere journaling filesystems wie **xfs**, **jfs** oder **reiserfs** abgelöst wird.

3.1.1 Aufbau

Das ext2 ist schematisch aus Bootblock, Superblock, Inodes und Datenblöcken aufgebaut:

3.1.2 Superblock

Der Superblock enthält folgende Verwaltungsinformationen des Dateisystems:

- Größe des Dateisystems
- Anzahl freier Blöcke
- Liste der freien Blöcke
- Zeiger auf den ersten freien Block in der Liste der freien Blöcke
- Größe der Inode-Liste

- Anzahl freier Inodes
- Liste der freien Inodes
- Zeiger auf den nächsten freien Inode in der Liste der freien Inodes
- Sperr-Felder für Liste der freien Blöcke / Inodes (z.B. für defekte Blöcke)
- Anzeigefeld, ob Superblock verändert wurde
- Information, wann das Filesystem das letzte Mal eingebunden wurde und ob es auch wieder sauber ausgehängt wurde

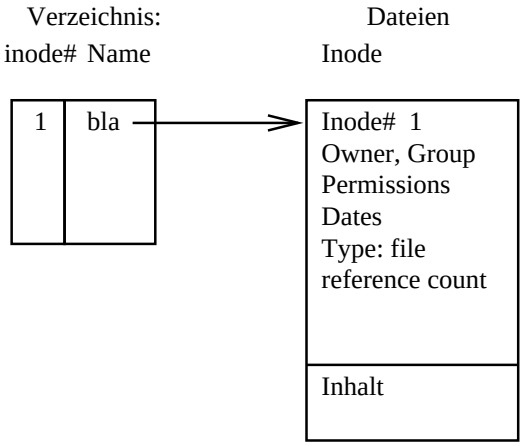
Da bei einer Beschädigung des Superblocks das gesamte Dateisystem unbrauchbar wäre, werden vom Superblock mehrere Kopien, verteilt in mehreren Blockgruppen, gespeichert. Diese Superblock-Kopien erlauben im Fehlerfall eine Reparatur des Original-Superblocks.

3.1.3 Inodes

Die Dateiverwaltung unter Unix wird über sogenannte Inodes (dt. vielleicht Indexeintrag) realisiert. Ein Inode fasst alle Attribute einer Datei zusammen außer dem Inhalt und dem Namen der Datei. Der Inode wird identifiziert über eine eindeutige Nummer für genau die Datei, die er verwaltet (Inode-Nummer). Im Inode sind folgende Informationen gespeichert:

- Besitzer der Datei
- berechtigte Gruppe
- Zugriffsrechte der Datei
- Typ der Datei (einfache Datei, Verzeichnis, Link,)
- Größe der Datei (in Bytes)
- Referenzzähler (Anzahl der Hardlinks = Namen der Datei, Zahl der Verweise aus den Verzeichnissen auf die Datei)
- Datum der letzten Inode-Änderung, des letzten Zugriffs auf die Datei (letzte Dateiöffnung) und der letzten Modifikation der Datei
- Verweis/Verweise auf die eigentlichen Blöcke (Lage auf der Festplatte) der Datei.

Die inode eines Verzeichnisses enthält die Informationen über die inode Nummer und die Namen der Dateien in diesem Verzeichnis. Die inode Nummer einer Datei wird mit dem Kommando **ls -i** angezeigt.



3.1.4 Dateinamen

Groß- und Kleinschreibung werden in einem Unix-Filesystem voneinander unterschieden. Ein Dateiname darf laut Spezifikation aus alle Zeichen außer / und NULL (0x0000) bestehen. Ein bestimmtes Format (DOS: 8.3) ist nicht einzuhalten.
Man unterscheidet zwischen relativen und absoluten Pfadnamen. Absolute Pfadnamen beginnen mit einem /. Alle Dateien, die nicht mit einem / beginnen, sind relative Pfadnamen und werden unterhalb des aktuellen Verzeichnisses gesucht. Befindet man sich im Verzeichnis `/home/student`, so bewirkt ein `cd bla` ein Wechseln in das Verzeichnis `/home/student/bla`, ein **cd /bla** hingegen würde zu einem Wechsel in das Verzeichnis `/bla` führen.
Dateinamen, die mit einem Punkt . beginnen, sind sogenannte versteckte Dateien. Bei einem Aufruf von **ls** ohne Option **-**a werden sie nicht angezeigt. Meist handelt es sich um nutzerspezifische Konfigurationsdateien. Die Datei „..“ ist das hierarchisch übergeordnete Verzeichnis, die Datei „.“ das aktuelle Verzeichnis.
Die Beschränkung auf „a-z,A-Z,0-9,.,_,+,-“ ist nicht notwendig, aber sinnvoll. Dadurch kann man Probleme beim Kopieren der Daten in andere Filesystemtypen (iso9660, vfat, etc.), bei der Shellinterpretation (z.B. haben viele Zeichen eine Sonderbedeutung und müssten geschützt (*quotiert*) werden, Umlaute können Probleme beim alphabetischen Sortieren in Abhängigkeit von der Spracheinstellung bereiten) usw. vermeiden.

3.2 Benutzer- und Berechtigungskonzepte

3.2.1 Benutzertypen

Linux ist ein Multiuserbetriebssystem. Daraus ergeben sich mehrere Anforderungen an das Berechtigungssystem:

- Schutz des Systems vor den Handlungen seiner Benutzer

- Schutz der Benutzer vor den Handlungen anderer Benutzer
- Kooperationsfähigkeit der Nutzer muss erhalten werden

Dafür steht unter Linux ein System abgestufter Berechtigungen zur Verfügung. Ein Nutzer ist mit einem Nutzernamen am System angemeldet. Intern ist dieser Benutzernamen auf eine Identifikationsnummer (*UID*) gekoppelt, die intern ausschließlich benutzt wird. Außerdem ist jedem Benutzer noch eine Gruppenidentifikationsnummer (*GID*). Jeder Prozess ist mit UID und GID des Erzeugers gekoppelt. Zusätzlich kann jeder Nutzer weiteren Gruppen angehören. Mit dem Kommando **id** kann man UID, GID und die Gruppenzugehörigkeit auslesen. Gruppen können übrigens nicht rekursiv gebildet werden, d.h. Gruppenmitglieder können nur Nutzer (*user*), nicht weitere Gruppen sein.
Die Login-Shell startet im Homeverzeichnis des jeweiligen Benutzers. Dieses und alle seine Unterverzeichnisse „gehören“ dem Benutzer.
Neben den Standardbenutzern gibt es den Systemverwalter oder Superuser **root**. Er ist mit allen Rechten ausgestatten, die ihm die Administration (und damit auch das Zerstören) des Systems erlauben. Diesem Nutzer ist immer die UID 0 zugeordnet. Dieser Benutzername sollte nur dann benutzt werden, wenn kein anderer Nutzer die für eine Aufgabe notwendigen Rechte besitzt. Es ist einer der verhängnisvollsten Fehler, sich ausschließlich als root anzumelden. Mit dem Kommanda **su** kann man - als Standardbenutzer angemeldet - vorübergehend die Rechte von root erhalten und sich nach Beendigung der nötigen Aufgaben mit **quit** wieder auf die Rechte als Standardbenutzer beschränken.

3.2.2 Benutzerklassen

Es existieren 3 Benutzerklassen, die entscheidend dafür sind, ob die Berechtigung für einen Dateizugriff existiert oder nicht: **Benutzer, Gruppe** und **Andere** (**user, group, other**). Benutzerklassen sind mit der Eigentümerschaft von Dateien verbunden. Jede Datei ist einem Benutzer (UID) und einer Gruppe (GID) zugeordnet. UID und GID werden in der sogenannten *inode* (3.1.3) gespeichert.

3.2.3 Berechtigungsklassen

Jeder Benutzerklasse ist ein Lese-, Schreib- und Ausführrecht (**read, write, execute**) zugeordnet. Es gibt also 9 verschiedene Rechte:

user-read	Leserecht für Dateieigentümer
user-write	Schreibrecht für Dateieigentümer
user-execute	Ausführrecht für Dateieigentümer
group-read	Leserecht für Mitglieder der Gruppe mit der GID der Datei
group-write	Schreibrecht für Mitglieder der Gruppe mit der GID der Datei
group-execute	Ausführrecht für Mitglieder der Gruppe mit der GID der Datei
other-read	Leserecht für alle anderen Benutzer
other-write	Schreibrecht für alle anderen Benutzer
other-execute	Ausführrecht für alle anderen Benutzer

Beim Zugriff auf eine Datei werden UID und GID des zugreifenden Prozesses mit UID und GID der Datei verglichen. Ist other-read gesetzt, darf jeder Benutzer lesend zugreifen und ein weiterer Vergleich erübrigt sich. Ist lediglich group-read gesetzt, muss der Zugreifende mindestens der Gruppe mit der GID der Datei angehören. Ist ausschließlich user-read gesetzt, so darf nur der Eigentümer selbst die Datei lesen.

3.3 Dateitypen

3.3.1 Typen

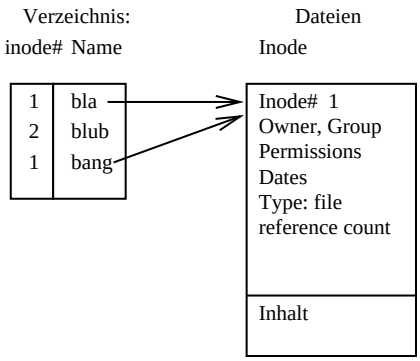
Unter Unix werden alle Ein- und Ausgabeoperationen mit denselben Systemrufen vorgenommen, d.h. es gilt das Prinzip **Alles ist eine Datei**. Den Typ einer Datei kann man mit dem Kommando **ls -l filename** feststellen. Das erste Zeichen der Ausgabe gibt Auskunft über den Dateityp:

- \- reguläre („normale“) Datei
- d Verzeichnis (*directory*)
- l symbolischer Link
- c zeichenorientiertes Gerät (*character device*), z.B. Terminals
- b blockorientiertes Gerät (*block device*), z.B. Festplatten
- s Sockets
- p benamte Pipes

3.3.2 Symbolische und harte Links

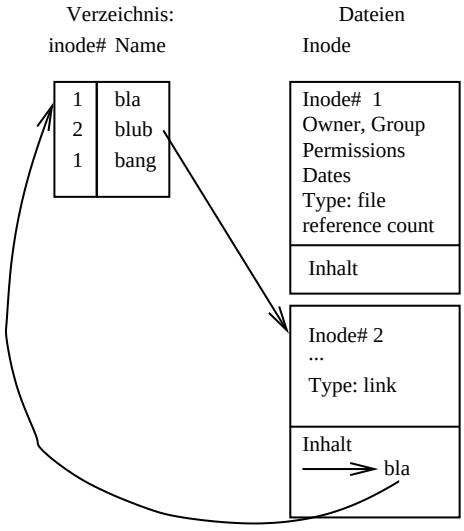
Jeder Verzeichniseintrag besteht aus dem Dateinamen und der zugeordneten Inode-Nummer. Zu jedem Inode kann es beliebig viele Verzeichniseinträge geben. Falls eine Datei mehrere Links hat, ist keine Unterscheidung zwischen dem zuerst vorhandenen und den später hinzugefügten Links möglich, die Links sind alle gleichwertig. Solche Links können mit folgendem Kommando erzeugt werden:

ln Quelle Ziel



Die Auswirkungen dieses Konzepts sind beim Entfernen von Links zu beobachten: Es gibt bei Unixsystemen keine Operation zum Löschen einer Datei, sondern nur eine unlink-Operation zum Löschen von Links. Im Inode einer Datei wird die aktuelle Anzahl der Links gespeichert. Erst wenn der letzte Link zu einer Datei gelöscht wird, der Zähler im Inode also den Wert Null hat, wird die Datei selbst, also der Inode und die Datencluster, zur Wiederverwendung freigegeben und damit logisch gelöscht. Harte Links können nicht über Partitions- bzw. Filesystemgrenzen hinweg erzeugt werden und auch nicht für Verzeichnisse selbst. Diese Grenzen überwindet man mit den sogenannte symbolischen Links, die folgendermaßen erzeugt werden:

ln -s Quelle Ziel



Allgemein muss man beachten, dass ein symbolischer Link anders als ein harter Link auf einen Pfad zeigt. Ein harter Link zeigt auf einen Inode, was bedeutet, dass ein harter Link immer noch funktioniert, wenn die Zieldatei verschoben

wurde. Wenn man aber die Zieldatei verschiebt, und ein Softlink zeigt drauf, kann die Zieldatei nicht wissen, dass ein Link auf sie zeigt, womit der symbolische Link ins Leere zeigt. Gleiches gilt, wenn ein relativer symbolischer Link selbst verschoben wird.

3.3.3 Zugriffsrechte für unterschiedliche Dateitypen

Für jeden Dateityp haben die unterschiedlichen Rechte (read, write, execute) eine unterschiedliche Bedeutung. Im folgenden werden nur Besonderheiten bei den Zugriffsrechten für reguläre Dateien und Verzeichnisse betrachtet:

- **Leserecht für Dateien**: Besonderheit: Um ein Programm auszuführen, ist kein Leserecht für die Programmdatei erforderlich, da das Programm in den Hauptspeicher geladen wird und nicht dafür nicht geöffnet werden muss.

- **Leserecht für Verzeichnisse**: Verzeichnisse enthalten nur Informationen über Dateien und weitere Unterverzeichnisse, deshalb hat das Leserecht eine andere Bedeutung. Hat man nicht die Berechtigung zum Lesen eines Verzeichnisses, so kann man sich den Inhalt des Verzeichnisses nicht anschauen, aber - vorausgesetzt man besitzt das Leserecht auf eine Datei, die in dem Verzeichnis enthalten ist - man kann dennoch den Inhalt einer Datei in diesem Verzeichnis auslesen, wenn man weiss, wie der Name dieser Datei ist.
  Das Leserecht eines Verzeichnisses hat keine Auswirkungen darauf, ob Verzeichnisinhalte gelöscht oder angelegt werden dürfen.

- **Schreibrecht für Dateien**: keine Besonderheiten

- **Schreibrecht für Verzeichnisse**: Schreibrecht bedeutet Recht zum Anlegen und Löschen von Dateien in diesem Verzeichnis

- **Ausführen von Dateien**: Programme und Skripte können ausgeführt werden, wenn des execute bit gesetzt ist. Dabei spielt eine evtl. Endung wie `.exe` oder `.sh` keine Rolle. Zu beachten ist, dass Skripte, im Gegensatz zu Programmen, erst gelesen werden müssen, bevor sie ausgeführt werden können. D.h. für Skripte muss auch das Leserecht gesetzt sein.

- **Ausführen von Verzeichnissen**: Das Ausführrecht für Verzeichnisse ist das Recht, in dieses Verzeichnis hineinzuwechseln. Dieses Recht ist die Voraussetzung für alle weiteren Operationen in diesem Verzeichnis und mit seinen Inhalten. Lesen von Dateien, Anlegen und Löschen von Dateien, Ausführen von Dateien in einem Verzeichnis sind nur möglich, wenn das Ausführrecht gesetzt ist. Das gilt rekursiv auch für alle Unterverzeichnisse. Die einzige Ausnahme ist die Anzeige des Verzeichnisinhaltes, denn dafür braucht man das Leserecht für das Verzeichnis und nicht da Recht zum Hineinwechseln.

Kapitel 4

Befehle zum Arbeiten mit Dateien

Das folgende Kapitel versucht, die wichtigsten Kommandos mit den am häufigsten benutzten Optionen aufzuzählen und kurz zu erläutern. Ausführliche Informationen sind den entsprechenden manpages zu entnehmen.

4.1 Inhalt von Dateien anzeigen

- Ausgabe einer Datei auf Stdout: **cat** *filename*
  Ausgabe der Standardeingabe, z.B. in pipes (s.5.6): **cat -**
- Seitenweise Ausgabe: **more** *filename*
- Erweitertes cmd zur seitenweise Ausgabe: **less** *filename*
- Ausgabe der ersten 5 Zeilen: **head -n 5** *filename*
- Ausgabe der letzten 5 Zeilen: **tail -n 5** *filename*
- Anzeige der Anzahl der Zeichen, Wörter und Zeilen einer Datei:
  wc *filename*

4.2 Dateien verwalten

- Dateinamen und ihre Eigenschaften anzeigen: **ls**
  Häufige Optionen:
  Langformat: **ls -l**
  Versteckte Dateien: **ls -a**
  Datei mit letztem Modifikationsdatum zuerst: **ls -t**
  Sortierreihenfolge umdrehen: **ls -r**
  Inodenummer anzeigen: **ls -i**

- Dateien kopieren: **cp Quelle(n) Ziel**
  Ist mehr als eine Quelldatei angegeben, so muss das Ziel bereits existieren und ein Verzeichnis sein.
  Optionen: Kopieren von Verzeichnissen (*recursive*): **cp -r**
  Nachfrage bei Überschreiben (*interactive*): **cp -i**
  Überschreiben ohne Nachfrage, keine Fehlermeldungen (*force*): **cp -f**

- Dateien umbenennen (*move*): **mv**
  Optionen: **-i, -f**

- Dateien löschen (*remove*): **rm**
  Optionen: **-i, -f, -r**

- Links anlegen (*link*): **ln**
  Optionen: **-s, -i, -f**

4.3 Verzeichnisse

- Anzeige des aktuellen Verzeichnisses (*print working directory*): **pwd**

- Anlegen eines Verzeichnisses (*make directory*): **mkdir**
  Option: **mkdir -p**

- Löschen eines leeren Verzeichnisses (*remove directory*): **rmdir**

- Wechseln des Verzeichnisses (*change directory*): **cd** *Verzeichnisname*
  Besonderheiten:
  cd ohne Argument wechselt ins Homeverzeichnis
  cd - wechselt in das Verzeichnis, in dem man sich vor dem letzten **cd** befand
  cd .. wechselt in das hierarchisch darüberliegende Verzeichnis

Kapitel 5

Die bash

5.1 Aufgaben der Shell

Eine Shell soll Kommandos entgegennehmen und an das Betriebssystem zur Ausführung weiterreichen. Dazu muss die Shell zuerst die Kommandozeile interpretieren. D.h. sie interpretiert einzelne Zeichen oder Wörter der Eingabe und ersetzt sie ggf. durch neue Zeichen oder Worten. So werden Variablen durch ihren Wert ersetzt und Platzhalterzeichen wie * oder ? durch passende Zeichenmuster. Dieser Mechanismus kann benutzt werden, um Kommandos variabler und effektiver zu formulieren.
Kommandos können miteinander kombiniert werden, z.B. kann die Ausgabe eines Kommandos als Eingabe eines zweiten Kommandos benutzt werden oder ein Kommando wird nur ausgeführt, wenn das vorherige erfolgreich war, oder Kommandos werden wiederholt in Form von Schleifen ausgeführt.
Häufig möchte man eine bestimmte wiederkehrende Folge von Kommandos ausführen, die man in eine Datei schreibt und für die spätere Ausführung abspeichert. Diese Dateien werden als Shell-Skripte bezeichnet.
Die Shell bietet einem Benutzer eine konfigurierbare, persönliche Umgebung, um typische Arbeitsabläufe komfortabel und fehlerfrei abwickeln zu können. So kann der Benutzer z.B. mit einer passenden Spracheinstellung arbeiten, den Suchpfad nach ausführbaren Programmen ändern, Kommandos abkürzen, Funktionen definieren, den Eingabeprompt ändern u.v.m. Dies lässt sich in sogenannten Shellvariablen konfigurieren und dauerhaft in einer Konfigurationsdatei sichern, die beim Einloggen abgearbeitet wird.

5.2 Variablen

Mit Variablen werden Informationen an Prozesse übergeben, um deren Verhalten zu steuern. Sie dienen also der Prozesskommunikation, definieren wichtige Dateien oder Verzeichnisse und legen Optionen für die zugehörigen Kommandos

fest. Dabei werden Variablen nur von Eltern-Prozessen an ihre Kind-Prozesse übergeben.
Variablen und ihre Werte lassen sich mit dem Kommando **set** anzeigen. Variablen müssen mit einem Buchstaben beginnen und dürfen aus alphanumerischen Zeichen und dem _ bestehen.
Jeder Prozess hat eine *Umgebung*, in dem seine Variablen abgelegt sind. Ein Beispiel ist die Variable **$PATH**, die steuert, in welchen Verzeichnissen automatisch nach ausführbaren Dateien gesucht wird. Weitere wichtige Variablen sind **$HOME, $PS1, $LANG**.
Variablen werden mit dem **export** Befehl an die Kind-Prozesse vererbt.
Eine spezielle Bedeutung haben die sog. Pseudovariablen. Die folgende Tabelle enthält eine Auswahl der wichtigsten.

Pseudovariable	wird ersetzt durch ...
$$	die Prozess-ID der Shell
$?	den Rückkehrcode des letzten Vordergrundprozesses
$#	die Anzahl der Aufrufargumente eines Shell-Skriptes
$0	den Kommandonamen des Shell-Skriptes
$n	das $n$te Aufrufargument $0 \leq n \leq 9$
${n}	das $n$te Aufrufargument
$*	alle Aufrufargumente

5.3 alias Mechanismus

- gestatten, das erste Wort eines einfachen Kommandos durch einen String zu ersetzen
- unterstützen im Gegensatz zu Shell-Funktionen keine Argumente

Alias definieren (Achtung! Leerzeichen müssen durch Quotierung (s. Abs. 5.7) geschützt werden):
alias l='ls -l'

Alias nutzen:
l /tmp

Aliasdefinition aufheben:
unalias l

Anzeige aller definierten Aliase:
alias

Durch Voranstellen eine Backslash kann man die Alias-Expansion verhindern:
alias rm='rm -i'
rm file

 # Nutzung des Standard-Kommandos rm:
\rm file

5.4 Startdateien

Startdateien sind Konfigurationsdateien, die beim Starten der bash abgearbeitet werden. Von der Art des Aufrufs der bash hängt ab, welche Konfigurationsdateien in welcher Reihenfolge eingelesen werden. Der Einfachheit halber betrachten wir hier den Fall, dass die bash als „**bash**“ und nicht als „**sh**“ (Abwärtskompatibilität zur bourne-Shell) gerufen wird.

Eine Loginshell wertet standardmäßig folgende Konfigurationsdateien aus:

1. /etc/profile, wenn sie lesbar ist
2. die erste lesbare Datei der folgenden Liste:
   (a) $HOME/.bash_profile
   (b) $HOME/.bash_login
   (c) $HOME/.profile

Eine interaktive Shell, die keine Loginshell ist, wertet standardmäßig die Datei $HOME/.bashrc aus.
Eine nichtinteraktive bash wertet diejenige Konfigurationsdatei aus, deren Name in der Variablen `BASH_ENV` gespeichert ist, ist diese Variable leer, wird keine Konfigurationsdatei eingelesen.
Eine Loginshell, die mit „**sh**“ gerufen wird, wertet die Dateien

1. /etc/profile
2. $HOME/.profile

aus. Ich bevorzuge ein konstantes Verhalten der bash, unabhängig von der Art ihres Aufrufs und nutze deshalb bourne-shell kompatible Startdateien, die durch symbolische links verbunden sind:
ln -s .profile .bashrc
ln -s .profile .bash_profile

5.5 Metazeichen zur Expansion von Dateinamen

Unter Metazeichen fasst man diejenigen Sonderzeichen einer Shell zusammen, die als Platzhalter für Zeichenmuster stehen.

In der Bash stehen folgende Sonderzeichen für:

?	Genau ein beliebiges Zeichen
∗	Beliebig viele (auch 0) beliebige Zeichen
[def]	Genau eines der Zeichen d, e oder f
[ˆczx]	Genau ein Zeichen, aber nicht c, z oder x
[!czx]	Genau ein Zeichen, aber nicht c, z oder x
[a-f]	Genau ein Zeichen aus dem Bereich von a-f

5.6 Ein- und Ausgabeumleitung, Pipes

Prozesse starten in der Regel mit 3 geöffneten *file handles* (Datenkanäle):

- Standardeingabe (stdin, Kanal 0): Programme erwarten von hier ihre Eingaben, bei interaktiven Programmen: Tastatur
- Standardausgabe (stdout, Kanal 1): Ausgabe eines Programms, bei interaktiven Programmen: Monitor
- Standardfehlerausgabe (stderr, Kanal 2): Fehlerausgaben eines Programms, bei interaktiven Programmen: Monitor

Ein- und Ausgabekanäle können in Dateien umgeleitet werden.

Eingabeumleitung: Das Programm liest nun nicht mehr von der Standardeingabe bis zu ihrer Beendigung `STRG-D`, sondern aus einer Datei, bis das Dateiende erreicht ist. Die Umleitung erfolgt durch das Zeichen „<“, gefolgt von einem Dateinamen.

cmd < filename

Ausgabeumleitung: Die Ausgabe des Programms wird nicht mehr auf den Monitor geschrieben, sondern in ein Datei. Die Ausgabeumleitung erfolgt durch das Zeichen „>“, gefolgt von einem Dateinamen.

cmd > filename

Falls die Datei noch nicht vorhanden war, wird sie angelegt. Falls die Datei schon vorhanden ist, wird sie überschrieben, d.h. es wird immer ab dem Dateianfang geschrieben.

Umlenkung der Fehlerausgabe: Die Umleitung der Fehlerausgabe erfolgt genauso wie die Ausgabeumleitung, jedoch wird hier die Zeichenfolge „2>“ verwendet, da stderr das *file handle* 2 ist.

cmd 2> errorfile

Die Ausgabe eines Kommandos kann auch an eine bereits bestehende Datei angehängt werden.

cmd >> filename

Durch das Umleiten von stdin und stderr in „das schwarze Loch“ lassen sich diese auch unterdrücken:

cmd > /dev/null 2> /dev/null

Die Umleitung von stdout und stderr in dieselbe Datei kann man den Datenstrom 2 auf den Datensstrom 1 umleiten:

cmd > outfile 2>&1

Pipes: Eine Pipe verbindet zwei Kommandos über einen temporären Puffer, d.h. die Ausgabe vom ersten Programm wird als Eingabe vom zweiten Programm verwendet. Alles, was das erste Programm in den Puffer schreibt, wird in der gleichen Reihenfolge vom zweiten Programm gelesen. In einer Kommandofolge können mehrere Pipes vorkommen. Der Pipe-Mechanismus wird durch das Zeichen „|“ (senkrechter Strich) aktiviert:

cmd1 | cmd2

Beispiel: Die Anzahl der Einträge (Dateien) im Verzeichnis `/bin` soll gezählt werde. Erste Möeglichkeit:
ls /bin > out
wc -l < out
rm out
Dabei besteht die Gefahr der Fragmentierung des Filesystems durch viele kleine Programme. Außerdem darf das Löschen der (unnötigen) Zwischendatei `out` nicht versäumt werden. Besser ist die zweite Möglichkeit:
ls /bin | wc -l

5.7 Quotierung

Aufhebung der Sonderbedeutung von Zeichen oder Wörtern

Backslash \	Aufhebung der Sonderbedeutung des Folge zeichens
einfache Apostrophe ' ... '	Aufhebung der Sonderbedeutung aller Zeichen zwischen begrenzenden Apostrophen
doppelte Apostrophe " ..."	Aufhebung der Sonderbedeutung der meisten Zeichen zwischen den begrenzenden Doppel-apostrophen; folgende 3 Zeichen behalten ihre Sonderbedeutung: $ ' \ Backslash behält seine Sonderbedeutung vor $ ' " \ Newline

Kapitel 6

Der Editor vi

Der Editor **vi** ist zwar von der Philosophie her etwas gewöhnungsbedürftig, aber er ist einer der schnellsten Editoren überhaupt und meist als Klon (mit Erweiterungen) **vim** auf Linux-Systemen verfügbar.
Der **vi** unterscheidet verschiedene Modi:

- Kommandomodus
- Einfügemodus
- der sogenannten **ex** Modus.

Beim Start befindet sich der **vi** im Kommandomodus, mit entsprechenden Kommandos (s.u.) gelangt man in den Einfügemodus, den man nur mit **ESC** wieder verlassen kann. Durch Eingabe eines **:** gelangt man in den **ex** Modus, der nur jeweils ein Kommando entgegennimmt.
Voraussetzung für die vernünftige Handhabung des **vi** ist das Verständnis der verschiedenen **vi** Modi. Die folgenden Kommandos stellen eine Auswahl der **vi** Befehle da und folgen Ref. [4].

6.1 Editieren

Die nachfolgenden Kommandos schalten den **vi** vom Kommandomodus in den Eingabemodus. Um zurück in den Kommandomodus zu gelangen, ist die Taste **ESC** zu betätigen.

i	Einfügen links vom Cursor
I	Einfügen am Zeilenanfang
a	Einfügen rechts vom Cursor
A	Einfügen am Zeilenende
o	Neue Zeile hinter der aktuellen einfügen
O	Neue Zeile vor der aktuellen einfügen
R	Überschreiben ab Cursorposition
s*Text*	Ersetze ein Zeichen durch *Text*
S*Text*	Ersetze ganze Zeile durch *Text*
ns*Text*	Ersetze `n` Zeichen durch *Text*
cw*Text*	Ersetze Wort durch *Text*
cc*Text*	Wie S*Text*

Eine Ausnahme bildet das Kommand **rc**, es ersetzt das Zeichen unter dem Cursor durch das Zeichen c, ohne in den Einfügemodus zu wechseln.

6.2 Positionierung des Cursors

Neben der Positionierung mit den Cursortasten stehen zum Scrollen durch den Text noch folgende Kommandos zur Verfügung:

Ctrl+F oder **l**	Ein Zeichen nach rechts
Ctrl+B oder **h**	Ein Zeichen nach links
Ctrl+P oder **k**	Eine Zeile nach oben
Ctrl+N oder **j**	Eine Zeile nach unten
Ctrl+V	Eine Bildschirmseite nach unten
Alt+V	Eine Bildschirmseite nach oben
w	Beginn des nächsten Wortes
b	Beginn des vorherigen Wortes
n G	Gehe zu Zeile `n`
G	Gehe ans Dateiende
0	Gehe an den Zeilenanfang
$	Gehe ans Zeilenende

6.3 Text kopieren

Das Kopieren folgt immer dem gleichen Schema:

1. Text in einen Puffer kopieren
2. Text aus (einem bestimmten) Puffer einfügen

6.3.1 Text in einen Puffer kopieren

yy	Kopiert aktuelle Zeile in einen Puffer
ny	Kopiert `n+1` Zeilen in einen Puffer

Um einen bestimmen Bereich zu kopieren, kann man wie folgt vorgehen:

1. Setzen des Cursors an den Beginn (oder Ende) des zu markierenden Textes
2. Umschalten in den *Visual*-Modus durch Eingabe von *v*
3. Setzen des Cursors an das Ende (oder Beginn) des zu markierenden Textes
4. Eingabe von *y* (der *Visual*-Modus wird automatisch beendet)

6.3.2 Text aus einem Puffer einfügen

P	Fügt Inhalt des aktiven Puffers *vor* der aktuellen Zeile ein, falls sich eine *Zeile* im Puffer befindet *vor* dem aktuellen Wort ein, falls sich ein *Wort* im Puffer befindet
p	Fügt Inhalt des aktiven Puffers *nach* der aktuellen Zeile ein, falls sich eine *Zeile* im Puffer befindet *nach* dem aktuellen Wort ein, falls sich ein *Wort* im Puffer befindet

6.4 Text löschen

6.4.1 Eingabemodus

Die zuletzt eingegebenen Zeichen können nacheinander mittels der Taste **Backspace** gelöscht werden.

6.4.2 Kommandomodus

x	Zeichen unter Cursor
nx	`n` Zeichen ab Cursorposition
X	Zeichen vor dem Cursor
nX	`n` Zeichen vor der Cursorposition
nJ	Entfernt die Zeilenendezeichen der nächsten `n` Zeilen
dd	Aktuelle Zeile
ndd	`n` Zeilen ab aktueller Zeile
dw	Ein Wort
D	Bis Zeilenende
nD	Alles bis zum Ende der aktuellen Zeile und die `n-1` nächsten Zeilen

6.4.3 Visual Modus

1. Cursor an Beginn des zu löschenden Textes setzen
2. Mit **v** in den *Visual*-Modus schalten
3. Mit den Cursortasten den zu entfernenden Bereich markieren
4. Mit **d** löschen

6.4.4 ex Modus

[`Bereich`]**d**	Löscht die durch [`Bereich`] angegebenen Zeichen.

Beispiel: Die Eingabe von **:1,$d** löscht die ganze Datei (von der ersten bis zur letzten Zeile), die Eingabe von **:3,+10** löscht 10 Zeilen ab der Zeile 3.

6.5 Suchen

/*muster*	Suche nach Muster vorwrts im Text
?*muster*	Suche nach Muster rckwrts im Text
n	Wiederholt letztes Suchkommando
N	Wiederholt die Suche rckwrts

6.6 Ersetzen

ex Modus:

:s/alt/neu/	Sucht und ersetzt `alt` durch `neu` (nur das erste Auftreten in aktueller Zeile)
:s/alt/neu/g	Sucht und ersetzt alle `alt` durch `neu` in aktueller Zeile
:1,$s/alt/neu	Ersetzt erstes Auftreten von `alt` durch `neu` auf allen Zeilen im gesamten Dokument
:1,$s/alt/neu/g	Ersetzt jedes Auftreten von `alt` durch `neu` auf allen Zeilen im gesamten Dokument

6.7 Speichern und Verlassen

6.7.1 ex Modus

:w `f`ilename	Schreibt den aktuellen Puffer in die Datei `f`ilename
:w >> `f`ilename	Hängt den aktuellen Puffer an die Datei `f`ilename
:q!	Verlassen, ohne Änderungen zu speichern
:x	Schreibt die Datei, falls sie geändert wurde, zurück und beendet den **vi** (wie **:wq**)

6.7.2 Kommando Modus

ZZ	Schreibt die Datei, falls sie geändert wurde, zurück und beendet den **vi** (wie **:wq**)

Kapitel 7

Weitere Befehle

In diesem Kapitel werden nützliche Unix-Kommandos mit einigen Einsatzmöglichkeiten vorgestellt. Eine ausführliche Beschreibung kann in den manpages bzw. den Onlinelinks [4][5]nachgelesen werden.

7.1 gzip

Um reguläre Dateien zu komprimieren, benutzt man das Kommando **gzip**.

```
gzip dateiname
```

erzeugt eine komprimierte Datei mit dem Namen dateiname.gz. Mit dem Kommando **gunzip**, das die gleiche Wirkung wie **gzip -d** hat, kann die Datei wieder ausgepackt werden.
Mit dem Kommando **zcat** (identisch mit **gzip -c**) kann der Inhalt einer komprimierten Datei auf der Standardausgabe ausgegeben werden, ohne dass die Datei dazu auf der Festplatte entpackt wird.
Mit der Option **-r** können die Dateien von Dateibäumen rekursiv komprimiert (dekomprimiert) werden. Bsp:

```
gzip -r "$HOME/Eigene Dateien"
```

komprimiert alle regulären Dateien unterhalb des Verzeichnisses `Eigene Dateien` im Homeverzeichnis.

7.2 tar

Mit dem Kommando**tar** (**t**ape **a**rchive) kann man mehrere Dateien zu einem Archive zusammenfassen. Bsp:

```
tar cf archive.tar datei1 ... dateiN
```

werden die Dateien datei1 ... datei*n* zu einem Archiv mit dem Namen archive.tar zusammengefasst. Diese Archiv kann komprimiert werden. Mit **gnutar** erzeugt man ein komprimiertes Archiv mit der Option **z**

```
tar czf archive.tar.gz datei1 ... dateiN
```

Steht nur ein Unix-tar zur Verfügung, hilft die Verwendung einer *pipe*

```
tar cf - datei1 ... datei | gzip > archive.tar.gz
```

(tar schreibt das Archiv auf STDOUT, der von gzip eingelesen, komprimiert und in die Datei archive.tar.gz geschrieben wird).
Den Inhalt eines Archivs kann man sich mit

```
tar tf archive.tar
```

anschauen. Dateien werden mit der Option **x** (e**x**tract) aus dem Archiv entpackt und herausgeschrieben:

```
tar xf archive.tar
```

extrahiert alle Dateien des Archives archive.tar, mit

```
tar xf archive.tar dateiname
```

wird nur die Datei mit dem Namen dateiname entpackt. Weitere Optionen bzgl. Pfadnamen, Dateirechten etc. sind in der manpage nachzulesen.

7.3 grep

grep ist das gebräuchlichste Kommando, um in Dateien nach bestimmten Mustern zu durchsuchen. Diese Muster köennen sogenannte "reguläre Ausdrücke" sein.

7.3.1 Reguläre Ausdrücke

Reguläre Ausdrücke können auch bei der Suche im **vi** oder in **perl** benutzt werden. Eine umfassende Abhandlung finden Sie in [9], eine tabellarische Zusammenfassung in Anhang A von [8]. Zu beachten sind die Unterschiede zwischen regulären Ausdrücken in **grep vi** ... und den Metazeichen der **bash** zur Dateinamenexpansion (s. Abschnitt 5.5). Häufig benutzte reguläre Ausdrücke sind:

.	Genau ein beliebiges Zeichen
$*$	beliebige Wiederholung des letzten Ausdrucks
[def]	Genau eines der Zeichen d, e oder f
[^czx]	Genau ein Zeichen, aber nicht c, z oder x
^	Zeilenanfang
$	Zeilenende
\{n\}	n - malige Wiederholung
\{n,\}	mindestens n - malige Wiederholung
\{n,m\}	n-m - malige Wiederholung

7.3.2 Anwenden von grep

```
grep dresden /etc/hosts
```

gibt alle Zeilen der Datei /etc/hosts aus, die die Zeichenkette "dresden" enthalten.

```
grep -v dresden /etc/hosts
```

gibt alle Zeilen der Datei /etc/hosts aus, die die Zeichenkette "dresden" **nicht** enhalten.

```
grep -v ^$ filename | grep -v \#
```

gibt alle Zeilen der Datei filename aus, die keine Leerzeilen (regulärer Ausdruck ^$, d.h. Zeilenende folgt unmittelbar auf Zeilenanfang) sind und nicht mit dem Kommentarzeichen (#) beginnen. Das Kommentarzeichen # muss vor der Interpretation durch die Shelle geschützt werden.
Ist die Suche erfolgreich, ist der Rückkehrcode von **grep** 0, wurde kein Muster gefunden ist der Rückkehrcode 1, tritt bei der Abarbeitung ein Fehler auf (Datei nicht gefunden oder nicht lesbar etc.), ist der Rückkehrcode 2.

7.4 find

find ist das Kommando, mit dem man Dateien mit bestimmten Eigenschaften im Filesystem findet und bestimmte Operationen mit diesen Dateien ausführt. Es gehört zu den mächtigsten Unix-Kommandos, hat aber auch eine komplexe Syntax. Suchkriterien können u.a. Dateiname, Typ der Datei, letztes Modifikationsdatum usw. sein. Diese Kriterien können miteinander kombiniert werden. Bei der Dateinamensuche können die Metazeichen der Shell zur Dateinamenerweiterung (s. 5.5) benutzt werden.
Operationen, die mit den gefundenen Dateien ausgeführt werden, können das Ausdrucken des Pfadnamens oder das Ausführen beliebiger Kommandos sein.
Syntax:

```
find Pfadname Suchkriterium Operation
```

Der Pfadname gibt an, unterhalb welches Verzeichnisses mit der Suche begonnen wird.
Eine vollständige Beschreibung der Möglichkeiten von **find** findet man in der zugehörigen manpage, hier sollen einige Beispiele Einsatzvarianten von **find** zeigen.

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

Anzeige aller Dateien, die die Endung ".c" haben und sich unterhalb des aktuellen Verzeichnisses befinden.

```
find /tmp -user s0000000 -type d -print
```

Anzeige aller Verzeichnisse unterhalb von `/tmp`, die dem Nutzer s0000000 gehören.

```
find /etc -mtime -30 -a ! -type d -exec grep smith {} \;2>/dev/null
```

Suche im Verzeichnis `/etc` alle Dateien, die in den letzten 30 Tagen modifiziert wurden aber keine Verzeichnisse sind, und suche in ihnen nach der Zeichenkette "smith". Fehlermeldungen (z.B. über fehlende Zugriffsmöglichkeiten) sollen nach /dev/null geschickt werden.

Kapitel 8

Shell-Programmierung

8.1 Schreiben eines Shellskripts, Kommentare

Schreibt man eine Folge von Kommandozeilen in eine Datei und läßt diese Datei von der Shell ausführen, so spricht man von einem Shell-Skript. Skripts erleichtern das Automatisieren von Vorgängen und dienen gleichzeitig der Dokumentation von Kommandos. Um Skripte in der laufenden Shell ausführen zu können, muss das executable-bit gesetzt sein. Befindet sich das Skript im Suchpfad, so kann es mit dem Skriptnamen aufgerufen werden, andernfalls muss der Pfadname angegeben werden.
Das einfachste Beispiel ist folgendes Skript mit dem Namen hello:

```
#! /bin/bash

# Dieses Skript erzeugt eine einfach Ausgabe auf stdout
# Autor: Max Mustermann (m.mustermannmuster.net)
# Changes
# 30.9.2007 18:59 Erstellen des Skripts

echo "Hello world!"
```

Mit
chmod +x hello
wird die Datei ausführbar gemacht. Falls das Verzeichnis, in dem sich hello befindet, im Suchpfad (PATH) enthalten ist, kann es nun mit
hello
aufgerufen werden. Falls nicht, muss der Pfad mitangegeben werden:
./hello

Jede Zeile, die mit einem # beginnt, ist ein Kommentar. Die einzige Ausnahme ist der sogenannte *shebang*. Wenn ein # gefolgt von einem ! in der ersten Spalte

der ersten Zeile einer Datei steht, dann wird der dahinter stehende Befehl mit dem Namen der aufgerufenen Datei als Parameter ausgeführt. Auf diese Weise ist es möglich, dass z.B. auch Nutzer einer **csh** **bash**-Skripte ausführen.

8.2 Testen von Shellskripten, Fehlersuche

Beim Test von Shellskripten sind zwei Shelloptionen besonders nützlich: **xtrace** und **verbose**. Ist **xtrace** aktiviert, wird jede Kommandozeile eines Skripts nach ihrer Expansion ausgegeben und erst anschließend ausgeführt. Ist **verbose** aktiviert, sind die Fehlermeldungen noch etwas ausführlicher.
Eine Shelloption wird mit **set** gesetzt:

```
set -v # verbose wird gesetzt
set +v # verbose wird deaktiviert
set -x # xtrace wird gesetzt
set +x # xtrace wird deaktiviert
```

Folgendes fehlerhafte Beispiel wurde [4] entnommen:
`cat fakultaet.sh`

```
#!/bin/sh
zahl=$1
while [ $zahl -gt 1 ]; do
   fakultaet=$fakultaet*$zahl
   zahl=$zahl-1
done
echo "Fakultät = "$fakultaet
```

Ein Aufruf **fakultaet.sh 10** liefert eine Fehlermeldung.
Fügt man die Zeile **set -xv** nach der ersten Zeile (der *magic line*) ein, so findet man schnell den Fehler (undefinierte Variabel fakultaet im ersten Schleifendurchlauf).

Um logische Fehler zu vermeiden bzw. möglichst schnell zu finden, sollten die allgemeinen Ratschläge zur Programmierung beachtet werden:

1. Einbau von Ausgaben mittels **echo** an allen kritischen Stellen des Skriptes, Kontrolle der Variablenbelegung
2. Werden Ergebnisse in *pipes* verarbeitet, kann das Kommando **tee** dazu dienen, die Daten der Zwischenschritte zu kontrollieren
3. Schrittweises Entwickeln der Skripte
4. Test von Funktionen mit statischen Eingabedaten
5. Einbau von Plausibilitäts- und Fehlertests an möglichst allen Stellen

8.3 Bedingte Ausführung, Tests

8.3.1 if-Konstrukte

Der **if**-Mechanismus zur Verzweigung des Programmablaufs wird meist zur Unterscheidung einiger weniger Fälle verwendet. Die Syntax lautet:

if Liste von Kommandos; then
  Liste von Kommandos
[elif Liste von Kommandos; then
  Liste von Kommandos]
[else
  Liste von Kommandos]
fi

Von den angegebenen Zweigen werden die Kommandos höchstens eines Zweiges ausgeführt. Entweder die des ersten Zweiges, dessen Bedingung erfüllt ist oder der optionale "else"-Zweig, falls keine Bedingung erfüllt wurde. Die Bedingung selbst ist der Rückgabewert der Liste der Kommandos (meist also der Rückgabewert des letzten Kommandos der Liste). Häufig wird als Verzweigungskriterium das Kommando **test** (s. 8.3.3) eingesetzt.

8.3.2 Bedingte Ausführung mit && und ||

Mit den Shellsonderzeichen && und || kann ebenfalls eine bedingte Programmausführung programmiert werden.

```
cmd1 && cmd2 || cmd3
```

bewirkt, dass **cmd2** dann ausgeführt wird, wenn der Rückkehrcode von **cmd1** 0 ist. Ist der Rückkehrcode $\neq 0$ (d.h. **cmd1** war nicht erfolgreich), wird **cmd3** ausgeführt.

8.3.3 tests

Ein häufig benutztes Kommando zur Steuerung von Shellskripten ist **test**. Meist wird anstelle von **test expression** das gleichbedeutende Kommando [**expression**] benutzt. Wichtig sind die Leerzeichen vor und nach den eckigen Klammern. Das Kommando **test** überprüft, ob eine Bedingung erfüllt ist und gibt in diesem Fall den Rückkehrcode von 0 aus. Die wichtigsten Testmöglichkeiten sind:

`-d FILE`	wahr, wenn FILE ein Verzeichnis ist
`-f FILE`	wahr, wenn FILE existiert und eine reguläre Datei ist
`-w FILE`	wahr, wenn FILE für Prozesseigentümer schreibbar ist
`-x FILE`	wahr, wenn FILE für Prozesseigentümer ausführbar ist
`FILE1 -nt FILE2`	wahr, wenn FILE1 neuer ist als FILE2
`-n STRING`	wahr, wenn STRING nicht leer ist
`STRING1 = STRING2`	wahr, wenn STRING1 gleich STRING2 ist
`STRING1 != STRING2`	wahr, wenn STRING1 sich von STRING2 unterscheidet
`arg1 -eq arg2`	wahr, wenn arg1 arithmetisch gleich arg2 ist
`arg1 -ne arg2`	wahr, wenn arg1 arithmetisch ungleich arg2 ist
`arg1 -lt arg2`	wahr, wenn arg1 arithmetisch kleiner als arg2 ist
`arg1 -le arg2`	wahr, wenn arg1 arithmetisch kleiner oder gleich arg2 ist
`arg1 -gt arg2`	wahr, wenn arg1 arithmetisch größer als arg2 ist
`arg1 -ge arg2`	wahr, wenn arg1 arithmetisch größer oder gleich arg2 ist
`! expr`	wahr, wenn expr falsch ist
`expr1 -a expr2`	wahr, wenn sowohl expr1 als auch expr2 wahr sind
`expr1 -o expr2`	wahr, wenn mindestens einer der Ausdrücke expr1 bzw. expr2 wahr ist

8.4 Schleifen

Schleifen dienen der wiederholten Kommandoausführung.

8.4.1 for Schleifen

Die **for**-Schleife wird verwendet, um eine Kommandosequenz n-mal auszuführen, wobei die Anzahl vorab fest steht.

```
for Variable [ in Wortliste ]; do
   Liste von Kommandos
done
```

Die **for**-Schleife wird genau so oft durchlaufen, wie Einträge in der Wortliste stehen. Im ersten Durchlauf wird der erste Eintrag der Wortliste an Variable zugewiesen, im zweiten der zweite Eintrag usw.:

```
for i in a b c; do echo $i; done
```

ergibt die Ausgabe

```
a
b
c
```

8.4.2 while Schleifen

Die **while**-Schleife bestimmt den Rückkehrcode der ihr unmittelbar folgenden Kommandos (zwischen "while" und "do") und führt die Kommandos des Schleifenkörpers solange aus, wie der Rückgabestatus des letzten Kommandos der Liste gleich Null ist.

```
while Bedingung do
   Liste von Kommandos
done
```

Mit folgender Sequenz wird die Fakultaet einer ganzen Zahl **n** korrekt ausgerechnet:

```
i=1
fak=1

while [ $i -le n ]
do
   fak=$((fak*i))
   i=$((i+1))
done
echo $fak
```

In den vorletzten Zeilen wird die eingebaute Integer-Arithmetik der **bash** genutzt, die es erlaubt, elementare Rechenoperationen (+-*/) mit ganzen Zahlen auszuführen. Schlüsselwort für die Integer-Arithmetik is $((...)). Um mit nichtganzen Zahlen zu operieren, muss das Programm **bc** benutzt werden.

8.4.3 until Schleifen

until-Schleifen sind genau genommen nichts anderes als **while**-Schleifen, wobei die Bedingung negiert wurde, d.h. sie wird solange durchlaufen, bis die Bedingung wahr wird.

```
until Bedingung do
   Liste von Kommandos
done
```

8.5 Beispiele für Shell-Skripts

```
#!/bin/bash
# Erstes Zeichen der ersten Zeile #! -> she-bang gibt programm an, dass
# die folgenden Kommentare interpretieren soll

# Erstellt: 2008/10/21 von max mustermann (m.mustermann@musterstadt.de)\\
# Das Skript gibt die Groesze einer Datei in Bloecken und die\\
# Zeilenanzahl aus.\\

# Test ist wahr, wenn genau ein Argument uebergeben wurde und dieses
# Argument eine regulaere Datei ist.
if [ "$#" -eq 1 -a -f "$1" ]
then
# Variablen file wird als Wert das erste uebergebene arg zugewiesen
  file="$1"
# Variablen fs wird als Wert die Ausgabe des cmd zwichen () zugewiesen
# ls -s $file gibt neben dem Filenamen die Groesze der Datei in Bloecken
# aus. Diese Ausgabe wird in das cmd cut... gepiped, wobei cut als Trennzeichen
# der einzelen Felder das Leerzeichen verwenden soll und nur das erste Feld
# ausgegeben wird. fs wird also mit der Groesze der Datei in Bloecken belegt.
  fs=$(ls -s $file | cut -f1 -d " ")
# Ausgabe, wobei $file durch Wert der Variablen file, $fs durch Wert der
# Variablen fs ersetzt wird
  echo "Die Date $file ist $fs Byte grosz"
# Ausgabe, wobei $() durch die Ausgabe des cmd zwischen () ersetzt wird
# cat $file gibt Datei auf STDOUT aus, diese Ausgabe wird von wc -l als
# Eingabe benutzt, das cmd zaehlt die Zeilen der Datei $file
  echo "Die Datei $file besteht aus $(cat $file | wc -l) Zeilen"
else
# Ausgabe einer Fehlermeldung, wenn entweder 0 oder mehr als 1 Argument
# uebergeben wurde oder die angegebene Datei keine regulaere Datei, sondern
# z.B. ein Verzeichnis oder ein symbolischer Link ist
# $0 wird durch den Programmnamen ersetzt.
  echo "usage: $0 filename"
# Verlasse das Skript
  exit
fi
```

In diesem Skript wird das Konstrukt $(...) benutzt. Die **bash** ersetzt dieses Konstrukt durch die Ausgabe des Kommandos, das zwischen den runden Klammern steht.

Folgende Zeilen sind nur der "Kern" eines Skripts **tolower**, das alle Dateinamen eines Verzeichnisses kleingeschriebene Namen verwandelt. Schreiben Sie die nötigen Kommentare und Fehlerbehandlungen selbst!

```
#! /bin/bash
for i in *
do
   mv $i $(echo $i | tr '[A-Z]' '[a-z]')
   echo $i done
done
```

In diesem Skript wird das Kommando **tr** benutzt, das Zeichen nach bestimmten Vorgaben ersetzt oder löscht (s. manpage).

Kapitel 9

Einführung in Netzwerke unter Linux

9.1 Grundlagen

9.1.1 Hardware

- Netzwerkkartentreiber in kernel fest einkompilieren oder als modul
- Laden des Treibers mit `dmesg | grep eth` überprüfen
- `ethtool -i eth`*n* gibt Treiverversion, Firmware version u.ä. aus
- Wird kein eth*n*|ath*n*|wlan*n* gefunden, mit `lspci` Chipsatz abfragen und mittels Internet Treiber recherchieren

9.1.2 IP Konfigurieren

- Übersicht über vorhandene Netze erstellen / IP-Adress Plan Innerhalb eines physikalischen Netzen IP-Adressen aus dem selben Netzwerk (beachte: Class A, Class B, Class C-Netze) stammen
  Speziell muss ein router interfaces eine IP-Adresse aus dem jeweiligen Netz haben.
  Keine IP-Adressen doppelt vergeben!
- Konfiguration

9.1.3 Namensauflösung

Jeder IP-Adresse kann ein (oder mehrere) Name(n) aus dem Domain Name Service Adressraum zugeordnet werden.

- In Datei `/etc/hosts`

- Über Abfrage eines DNS-Servers
- Reihenfolge der Abfrage in `/etc/nsswitch.conf`
- Konfiguration des DNS-Clients in `/etc/resolv.conf`
  Viele DHCP-Server sind so konfiguriert, dass sie dem client außer der IP-Adresse, Netzmaske etc. auch die Einträge für die `resolv.conf` übergeben.
  Der DHCP-Client kann über die Datei `/etc/dhcp3/dhclient.conf` konfiguriert werden.
- Test der Namensauflösung mit `ping <name>`, Test der Namensauflösung über DNS mit `dig`.

9.2 Kommandos zum Einrichten eines Netzwerkes

- IP-Adresse und Netzwerkmaske konfigurieren
  `ifconfig eth0 192.168.99.6 netmask 255.255.255.0 up`
- Netzwerkkartenkonfiguration abfragen:
  `ifconfig eth0`
- Arp-Tabelle abfragen, um z.B. doppelt vergebene IP-Adresse aufzuspüren
  `arp -a`
- Routing konfigurieren
  `route add default gw 192.168.9.9`
- Routing Tabelle abfragen
  `netstat -nr` bzw. `route -n`

9.3 Linux als Router

Unter Routing versteht man die Vermittlung von Datenpaketen zwischen zwei (logisch) voneinander unabhängigen Netzen. Ein Router muss, um Pakete vermitteln zu können, an beide Netze angeschlossen sein - benötigt im Normalfall also mindestens zwei Netzwerk-Interfaces (Netzwerkkarten, WLAN, Modems, etc.).

- Netzwerk planen: Topologie, Firewallregeln, etc.
- Der Router sollte minimal aufgesetzt sein (Sicherheit!), unnötige Dienste müssen deaktiviert sein (mit `netstat -a`, `more /etc/inetd.conf` kontrollieren), ggf. Dienste aus `/etc/init.d` bzw. `/etc/rc2.d` entfernen
- Protokoll für Anbindung an WAN auswählen: SLIP, PPP (modem, isdn), PPPoE (DSL),...
- Dokumentation lesen

- Sicherstellen, dass Routing deaktiviert ist (Sicherheit!)
  `echo 0 >> /proc/sys/net/ipv4/ip_forward`
- Überprüfen, ob die entsprechenden Kernelbestandteile in den kernel fest einkompiliert sind, oder als module geladen werden müssen (z.B. `pppoe.ko`, `ppp.ko`..., ggf. Modul laden `modprobe pppoe`
- Schnittstelle ins WAN konfigurieren (`/etc/ppp`)
- Testen, ob WAN-Zugriff funktioniert: mit `ifconfig -a` IP-Adresse der WAN-Schnittstelle abfragen, Gegenstelle anpingen
- Überprüfen, ob Zugriff ins interne Netz funktioniert (`ping <IP-Adresse Testrechner>`
- Auf Testrechner im internen Netz interne Schnittstelle des routers als default gateway eintragen
- Überprüfen, ob Kernelbestandteile für das Routing aktiviert sind, ggf. Module laden
- Überprüfen, ob Kernelbestandteile für Firewall, NAT etc. geladen sind, bzw. fest im Kernel einkompiliert sind
- Firewall regeln festlegen, NAT und Firewall aktivieren
- Routing aktivieren `echo 1 >> /proc/sys/net/ipv4/ip_forward`
- Überpruefen, dass Testrechner im internen Netz auf WAN-Schnittstelle zugreifen kann (`ping <WAN IP-Adress>`)
- Überpüfen, dass IP-Adresse im Internet erreichbar ist, z.B. anpingen von www.heise.de, aber über die IP-Adresse
  `ping 193.99.144.85`
- Namensauflösung des Testrechners überprüfen (Router als DNS-Server eintragen, falls DNS-Konfiguration vom Provider übertragen wird, ansonsten öffentlichen DNS-Server benutzen)
- Überprüfen, dass Internetserver über DNS-Namen erreichbar sind
  `ping www.heise.de`
- Detaillierte Erklärungen unter [6]

9.4 Netzwerkserverdienste

- Lesen der Dokumentation
- Planen der Konfiguration

- Installation über Paketmanager (hängt von Distribution ab) oder aus Quellen:
  - Download (meist als komprimiertes tar-Archive *name.tgz* oder *name.tar.gz*
  - Entpacken `tar -xzvf name.tgz`
  - Wechseln in source Verzeichnis `cd name`
  - Lesen von `README` und/oder `INSTALL`
  - Installieren, meist eine Folge von
    ```
    ./configure [--options]
    make
    make install
    ```
- Bearbeiten der Konfigurationsdateien
- Start des Dienstes üeber `inetd.conf` oder als daemon über `/etc/init.d/name start`
- Beispiel Konfiguration eine FTP-Servers am Beispiel von ProFTP [7]

Literaturverzeichnis

[1] http://de.wikipedia.org/wiki/Liste_von_Linux-Distributionen

[2] http://www.ostc.de/windows.html

[3] http://www.tldp.org/

[4] http://www.linuxfibel.de/

[5] http://www.selflinux.de/

[6] http://linuxrouter.minots.net/routing.html

[7] http://www.pro-linux.de/work/server/ftp01.html

[8] Stefan Stapelberg, Unix System V.4 für Einsteiger und Fortgeschrittene, Addison-Wesley, ISBN 3-89319-433-9, 1995.

[9] Jeffrey E.F. Friedl, Reguläre Ausdrücke, O'Reilly/International Thompson Verlag, ISBN 978-3-89721-720-1, 2007.

Anhang A

Übungsaufgaben

1. Übung 1: Erste Schritte

   (a) Loggen Sie sich mit dem Programm XMing über die graphische Oberfläche (mittels XDMCP) mit Ihrer LDAP-Kennung und dem zugehörigen Passwort auf dem Rechner `smith` ein! Starten Sie ein Kommandofenster!

   (b) Loggen Sie sich parallel mit dem Programm **putty** über die Textoberfläche (mittels ssh) auf `smith` ein!

   (c) Rufen Sie in verschiedenen Fenstern die Kommandos **id** und **tty** auf. Vergleichen Sie die Ausgabe dieser Kommandos mit den Zeilen, die mit dem Kommando **who** erzeugt werden.

   (d) Informieren Sie sich mit den Kommandos **man id, man tty, man who** über die Möglichkeiten der entsprechenden Kommandos!

   (e) Ermitteln Sie mit dem Kommando **cal** den Wochentag, an dem Dennis Ritchie (geb. 9.11.1941) geboren wurde! Nehmen Sie ggf. das Kommando **man cal** zur Hilfe!

   (f) Enthält das Jahr 2000 einen Schalttag ?

   (g) Rufen Sie das Kommando **date** so auf, daß es das heutige Datum (ohne Uhrzeit) in der Form "tt.mm.jjjj" ausgibt! Lesen Sie dazu die manpage.

2. Übung 2: Arbeiten im Dateisystem

   (a) Erzeuge im Verzeichnis `/tmp` ein Verzeichnis mit Deinem Account-Namen! Erzeuge in diesem Verzeichnis ein Unterverzeichnis "Linux"! Wie lautet der vollständige Pfadname dieses Verzeichnisses?

   (b) Erzeuge in Deinem Home-Verzeichnis ein Verzeichnis "Linux-Home"! Wie lautet der vollständige Pfadname dieses Verzeichnisses?

(c) Erzeuge in dem Verzeichnis, das in (2a) erstellt wurde, eine Datei mit Namen `datei_1`!

(d) Erzeuge einen hard link dieser Datei in `/tmp/<account>` mit dem Namen `datei_2`!

(e) Kann man einen hard link dieser Datei in `/home/<account>` erzeugen? Warum/Warum nicht?

(f) Erzeuge in `/tmp/<account>` einen symbolischen Link mit dem Namen `datei_3`! Kann man einen symbolischen Link dieser Datei in `/home/<account>` erzeugen? Warum/Warum nicht?

(g) Können "hard links" kopiert werden? Wenn ja, kopiere den unter (2d) erzeugten hard link in ein (neuanzulegendes) Verzeichnis `/tmp/<account>/Linux_1`!

(h) Können "symbolic links" kopiert werden? Wenn ja, kopiere den unter (2f) erzeugten "symbolic link" in ein (neuanzulegendes) Verzeichnis `/tmp/<account>/Linux_2`!

(i) Betrachte das Resultat der links und Kopien mit dem Kommando **ls -li** und vergleiche!

(j) Kann ein Hardlink auf ein Verzeichnis angelegt werden? Warum/warum nicht?

(k) Wechsle in das Verzeichnis `/tmp/<account>/Linux`!

(l) Kopiere mit **cp -r** das aktuelle Verzeichnis in ein weiteres Unterverzeichnis zu Deinem Homeverzeichnis! Was passiert mit den vorhandenen "symbolic" bzw. "hard links"? Welche Alternativen zu **cp -r** existieren?

(m) Der "link count" bei Dateien gibt an, wieviele Verzeichniseinträge auf die I-Node zeigen. Was gibt der Link-Count von Verzeichnissen an?

3. Übung 3: Berechtigungen

(a) Ein Benutzer gibt folgendes ein:
$ **chmod 0 $HOME**
$ **cd**
cd: Can't change to home directory.
$
Was ist passiert, wie kann das Problem behoben werden?

(b) Lege in `/tmp/<account>` ein Verzeichnis `public` und ein Verzeichnis `privat` an! Ändere die Zugriffsrechte so, dass nur Du in das Verzeichnis `privat` wechseln darfst und dass alle Nutzer in dem Verzeichnis `public` Dateien anlegen dürfen, aber bestehende Dateien nicht ändern dürfen!

4. Übung 4: vim

(a) Kopiere die Datei /etc/hosts in Dein Homeverzeichnis!

(b) Ändere die Zugriffsrechte von $HOME/hosts so, dass die Datei schreibbar ist!

(c) Ändere jedes Vorkommen von `de` in `linux`!

(d) Füge an das Ende der Datei eine Zeile `''hier sind wir''` hinzu!

(e) Füge nach jedem Kommentarzeichen (`#`) ein `%%%` ein!

(f) Lösche die 4. Zeile.

(g) Speichere die Datei als $HOME/hosts_vi ab!

(h) Verlasse den vi!

5. Übung 5: bash Teil 1

(a) Definiere einen alias **vi** so, dass /opt/csw/bin/vim gerufen wird!

(b) Definiere einen alias **lt** so, dass das Kommando **ls** die Dateien im Langformat und zeitlich geordnet anzeigt!

(c) Füge das Verzeichnis `$HOME/bin` dem Suchpfad hinzu! Was muss man tun, um diese Erweiterung beim nächsten Einloggen automatisch wirksam werden zu lassen? Editiere die entsprechende Datei!

(d) Stelle ohne Abzählen fest, wieviel Zeilen die Datei /etc/passwd enthält! Wieviele Dateien (einschließlich Unterverzeichnissen und symbolischen Links enthält das Verzeichnis `/etc`?

(e) Wieviele Dateien im Verzeichnis `/etc` beginnen mit `host`?

(f) Schreibe den Inhalt des Verzeichnisse `/etc` in eine Datei `inhalt_etc` in Deinem Homeverzeichnis!

(g) Versuche, das Verzeichnis `/etc` zu löschen! Leite Fehlermeldungen zur späteren Fehleranalyse in eine Datei `r`emove_etc_prot in Deinem Homeverzeichnis um!