Unix

Grundlagen des Betriebssystems

Ralf Berhorst

Düsseldorf im Dezember 1994

Vorwort

Wegen der ständig steigenden Zahl an UNIX - Systemen im Bereich der zentralen Gruppen des DATEX - P - Dienstes beim FA1 Düsseldorf, ist dieses Skript entstanden. Es soll die Arbeitsgrundlage für einen gleichnamigen Lehrgang sein.

Dieses Skript macht nicht die zahlreichen und ausführlichen Fachbücher überflüssig, sondern es soll als erster Schritt in Richtung UNIX dienen. Es ist für Benutzer geschrieben, die nicht als Systemprogrammierer oder Systemadministrator arbeitet. Vielmehr wird hier derjenige angesprochen, der einen groben Überblick über das System erhalten und einige grundlegende Befehle erlernen möchte.

Alle im Skript aufgeführten Befehle wurden auf einer Sun Sparcclassic mit SunOS 4.1.3 erstellt. Es wird keine Verantwortung für die einwandfreie Funktion der aufgeführten Beschreibungen und Kommandos übernommen.

An dieser Stelle möchte ich mich bei allen Kollegen bedanken, die mir bei der Erstellung dieser Unterlagen, mit Rat und Tat zur Seite gestanden haben.

Ralf Berhorst

1. Auflage Dezember 1994, Version 1.0

Rechte

Alle Rechte liegen beim Autor.

Die in diesem Skript verwendeten Soft- und Hardwarebezeichnungen oder Markennamen der jeweiligen Firmen, sind eingetragene Warenzeichen unterliegen im allgemeinen einem marken bzw. patentrechtlichem Schutz.

0 Inhaltsangabe

1 Einführung

1.1 UNIX - Geschichte

1.1.1 Enstehung

Mitte der 60er Jahre begaben sich die AT&T Bell Laboratories und das Massachusettes Institut of Technologie (MIT) an die Entwicklung eines neuen Betriebsystems für den Mainframe - Rechner GE645 von Gerneral Electric. Bis dato waren die meisten vorhandenen Betriebssysteme sogenannte *Closed Shop Batch Systeme*, bei denen der Programmierer sein Programm in Form von Lochkarten oder -streifen beim Operator ablieferte. Hintereinander wurden so alle Programme exklusiv abgearbeitet. Danach konnte der Programmierer, bei fehlerfreiem Lauf des Programmes, seine Ergebnisse abholen.

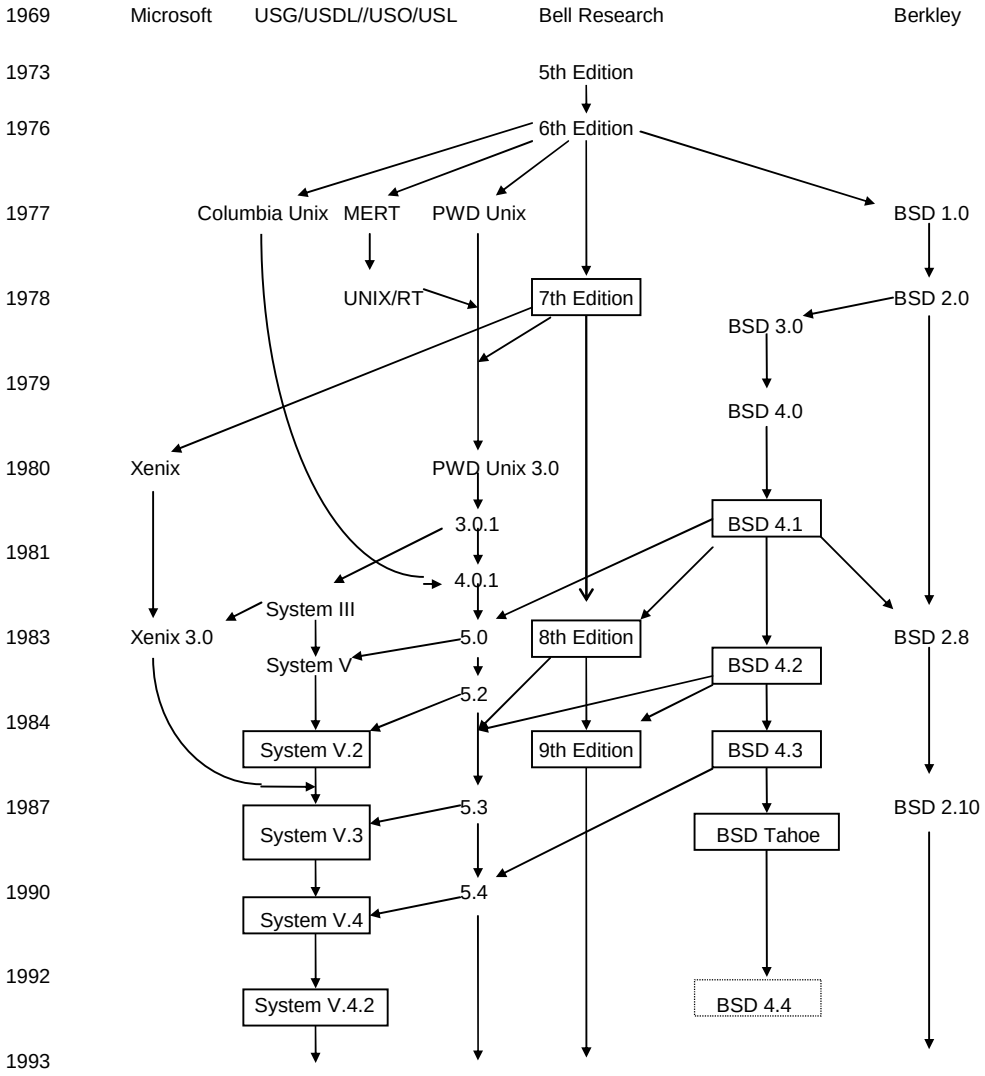


Abbildung 1 Der UNIX -Stammbaum [RICHT]

Ziel der Entwicklung war es, ein Betriebsystem zu erstellen, das interaktive Sitzungen mehrerer Benutzer am Rechner und den Zugriff auf gemeinsame Daten erlauben sollte. Die Entwickler gaben dem System den Namen MULTICS (Multiplexed Information an Computering System). Als man 1969 die erste lauffähige Version präsentierte, stellten die Bell - Laboratories jedoch ihre Mitarbeit an dem Projekt ein, da es nicht ihren Erwartungen entsprach.

1969	Bei den Bell - Laboratories setzten sich daraufhin Ken Thompson und Dennis Ritchie zusammen. Diese hatten bis dahin an dem Projekt MULTICS mitgearbeitet. Sie übernahmen einige Ansätze aus dem Projekt und versuchten, im Gegensatz zu dem äußerst komplex aufgebauten MULTICS, ein möglichst einfaches und trotzdem leistungsfähiges System aufzubauen.
1970	Mit der Portierung der Entwicklungsumgebung von einem Honeywell - 635 - Rechner auf eine PDP-7 von DEC, durch Ken Thompson, enstant somit die erste Implementation des neuen Betriebssystems auf einer PDP-7. Brian Kernigham, ein weiter Mitarbeiter, nannte das neue System scherzhaft UNICS (Uniplexed Information and Computing System), woraus sich später der Name UNIX entwickelte.
1971	Portierte Thompson das System auf eine PDP-11/20. Er schrieb dazu das Textverarbeitungssystem ***runoff***, aus dem sich später der ***troff***, das erste Typesetting- Programm (Textverarbeitung), das belichtungsfähige Vorlagen produzieren konnte, entwickelte.
1973	Die ersten Versionen waren noch in Maschinensprache auf der PDP-7, bzw. PDP-11 geschrieben worden. Damit folgende Versionen von UNIX auf andere Maschinen portiert werden konnten, setzten Dennis Ritschie und Ken Thompson den Kern von UNIX in die von ihnen zu diesem Zweck entwickelte Sprache C um. Zu diesem Zeitpunkt bestand der Kernel aus 20000 Zeilen Programmcode, von denen rund 2000 Zeilen in Assembler geschrieben waren. Für die Implementierung auf andere Systeme, mußte eigentlich nur der spezielle Assemblerteil des UNIX angepaßt werden.
1974	Western Electric Corporation, ein Tochterunternehmen von AT&T, begann Lizenzen hausintern und an verschiedene Universitäten zu vergeben (Version 5 und später Version 6).
1979	Die „7th Edition" von UNIX (kurz „V7"), wurde über die Firma Western Electric Corporation vertrieben und war der Ausgangspunkt der ersten kommerziellen und vieler nicht - kommerzieller UNIX - Portierungen.
1982	Nachdem sich AT&T von den Bell Operating Companies trennen mußte, wurde die UNIX Support Group (USG) mit dem Vertrieb beauftragt. Zuvor hatten sie das PWB UNIX vertrieben und brachten nun UNIX SYSTEM III auf den Markt. Die USG änderte im Lauf der Zeit ein paarmal ihren Namen, zuerst in UNIX System Development Laboratory (USDL), dann in UNIX Software Operation (USO) und schließlich in UNIX System Laboratories (USL). USL ist seit 1993 ein Tochterunternehmen von Novell.
1983	USDL stellt mit dem UNIX SYSTEM V den Nachfolger vom UNIX SYSTEM III vor. UNIX SYSTEM IV war eine Zwischenversion, welche jedoch nie freigegeben wurde.
1985	USO führt die UNIX SYSTEM V Release 2 ein. Damit die UNIX - Portierungen anderer Hersteller auf Konformität zu SYSTEM V überprüft werden können, veröffentlichte AT&T die System V Interface Definition (SVID), in welcher der Leistungsumfang und die Benutzeroberfläche von SYSTEM V verbindlich beschrieben ist.
1987	USO führt die UNIX SYSTEM V Release 3 ein.
1990	SYSTEM V Release 4 wird eingeführt.
1992	SYSTEM V Release 4.2 wird eingeführt.

1.1.2 XENIX

Das von der Firma Microsoft aus dem Jahr 1987 stammende XENIX, war ursprünglich eine UNIX - Portierung für die PDP-11 und 8086 - PC´s aus dem UNIX „7th Edition“. Microsoft wollte hiermit auch den Support für ihr UNIX liefern, den AT&T zu dieser Zeit nicht anbieten konnte. Microsoft verpflichtete sich in einem Abkommen mit AT&T, welches die Kompatibilität zu UNIX System V sicherstellte. Dieses, speziell für kleine Maschinen adaptierte System, erreichte eine große Anzahl von Installationen.

1.1.3 BSD UNIX

Auf der Basis einer „7th Edition“, welche auf einer VAX von Digital Equipment (DEC) installiert war, entstand an der *University of Berkley at California* (auch unter UBC bekannt) eine UNIX-Version, die hauptsächlich im technisch - wissenschaftlichen Bereich vertreten ist. BSD UNIX (**B**erkeley **S**ystem **D**istribution) diente als Ausgangsbasis für die Portierung wie z.B. ULTRIX von DEC und SunOS von Sun Microsystems. Ende 1992 wurde die letzte Version BSD 4.4 und die Auflösung der BSD- Entwicklungsgruppe angekündigt.

1.1.4 UNIX SYSTEM V

SYSTEM V ist der de facto - Standard im kommerziellen Bereich, der auf der SVID basiert und somit eine einheitliche Funktionalität innerhalb der SYSTEM V- Familie bietet. Bekannte Portierungen von SYSTEM V.3 sind MUNIX von PCS, SCO UNIX von der Santa Cruz Operation und UNIX/386 von Interactive. Portierungen von SYSTEM V.4 gibt es derzeit von Intel, Dell Esix, Consensys, Microport, Generics, Siemens Nixdorf und PCS.

1.2 Standards

Der Splitting- Prozeß, den UNIX während seiner Entwicklung durchlief, und die verschiedenen Portierungen, die daraus resultierten, erschwerten die breite Einführung des Systems auf kommerziellem Gebiet. Vorteile der Kompatibilität des Systems gegenüber anderen Betriebssystemen wird der Anwender nur dann nutzen, wenn er auf der einen Maschine die gleiche Umgebung wieder findet, wie auf der anderen Maschine. Genauso muß bei einem Update des Systems, die gewohnte Funktionalität erhalten bleiben. Deshalb setzten sich in den frühen 80er Jahren Herstellervereinigungen und Standardisierungskomitees zusammen, um gemeinsame Schnittstellen und Entwicklungsumgebungen für UNIX-Systemen zu definieren. Als da wären:

POSIX	*Portable Operating System for Computer Environments* (von der amerikanischen Benutzervereinigung /usr/group) welche den Leistungsumfang und die Schnittstelle eines Betriebssystems definiert. Sie schloß sich der IEEE (*Institute of Electrical and Electronics Engineers*) an und bekam den Namen **P1003**.
SVID	Definiert die Kommandos und die Funktionen für die Implementierung von UNIX System V. AT&T war ebenfalls in der Benutzervereinigung /usr/group aktiv, und somit ist es nicht verwunderlich, daß deutliche Einflüsse dieser Gruppe auch in der SVID sichtbar sind (z.B. Tastaturtreiber).
X/OPEN	Eine Vereinigung von einigen europäischen DV-Herstellern, deren ursprüngliches Ziel es war, die Probleme der nicht - englisch -

	sprachigen Länder zu berücksichtigen. Ende 85 erschien die erste Ausgabe der *X/OPEN Portability Guide* (XPG), dessen *X/OPEN System Interface* (XSI) auf der SVID von AT&T basiert und diese um den POSIX- Standard der IEEE ergänzt.
FIPS	Das amerikanische *Nation Institute of Standards and Technology* (NIST) übernahm den POSIX- Standard als Teil des *Federal Information- Processing Standard* (FIPS). Alle Rechnersysteme, die von den amerikanischen Behörden angeschafft werden, müssen den FIPS - Standard angehören.
OSF	Die *Open Software Foundation* ist eine Herstellervereinigung, welche die Vormachtstellung von AT&T in Bezug auf die weitere Entwicklung von UNIX vermeiden soll. Einziges von der OSF am Markt etablierte Produkt ist *Motif*, eine graphische Benutzeroberfläche und Entwicklungsumgebung für das *X Window System*. Motif hat die aktuelle Version 1.2 und ist in „C" geschrieben. Motif 2.0 wird in „C++" geschrieben.
ANSI	Das Komitee X3J11 des *American National Standards Institutes* (ANSI) begann Mitte der 80er Jahre, die Programmiersprache „C" zu standardisieren. Zu dieser Zeit gab es für „C" nur den K&R-Standard, der durch das Buch *„The C Programming Language"* von Kernighan und Ritchie, als de - facto - Standard somit festgelegt war.

SYSTEM V.4 war das erste System, das sowohl den POSIX - Standard als auch dem X/OPEN Portability Guide 3 entspricht und ist auch ANSI- konform.

1.3 UNIX - Merkmale

Grundlegende Eigenschaften von UNIX, die in praktisch allen Derivaten und „Look - Alikes" vorhanden sind:

- Kommunikation über „*pipeline*"
- Leistungsfähiger Kommandointerpreter **sh** (Bourne - Shell), der Kommandoprozeduren abarbeiten kann und Programmiersprachen ähnliche Konstruktionen kennt.
- Auf den meisten UNIX- Implementationen gibt es zusätzlich einen weiteren Kommandointerpreter **csh** (C-Shell), der über den Alias- und den History- Mechanismus verfügt, und Konstruktionen erlaubt, die der Sprache C entsprechen.
- Auf manchen Implementationen existiert noch die menuegesteuerte **vsh** (Visual- Shell), die **rsh** (Restricted- Bourne- Shell) und die **ksh** (KORN- Shell).
- Anschulß unterschiedlicher Bildschirmterminals standardisiert über die Datei „*termcap*" oder „*terminfo*"- Datenbank.
- Bildschirmorientierter Editor **vi**
- Programmiersprache C
- Compiler für C++, COBOL, PASCAL, FORTRAN, APL, MODULA-2, ADA...

- Programmiertools **lint** (Syntax-/Semantik- Checker für „C“), **dbx** (Debugger für „C“), **adb** (Maschinensprache- Debugger), **sdb** (Symbolischer Debugger)
- Künstliche Intelligenz: PROLOG, LISP
- Vernetzungsmöglichkeiten(**uucp**, **cu**, TCP/IP, e-mail)

Positive Eigenschaften:

- Mehrfachbenutzersystem
- maschinenunabhängige Dateistruktur
- Vorhandensein auf vielen Rechnern und leichte Portierbarkeit
- sehr viele Werkzeuge, die sich vielseitig kombinieren lassen
- relativ einheitliche Oberfläche (X- Windows, Motif)
- viele kommerzielle Anwendungen vorhanden
- Schutzmechanismen auf Dateiebene, Vorhandensein von UNIX-Derivaten der Sicherheitsklasse C2 und höher (orange book, red book vom Department of Defence)

Nachteile:

- sehr hoher Lernaufwand
- Syntax manchmal sehr kryptisch
- zu wenige und manchmal unklare Fehlermeldungen (z.B. permission deny beim kopieren)
- Befehlsbeschreibung in den Manuals manchmal unklar
- Stabilität nicht immer voll gewährleistet

Datenbanken:

- Oracle
- Informix
- Uniplus
- db++
- Empress/32
- C-Isam
- Unify

– Ingres

1.4 Oberflächen (X- Window)

X ist ein portables, netzwerkfähiges Window- System für Graphikbildschirme, das auf vielen Rechnerarchitekturen läuft und die verschiedensten Graphikbildschirme bedient. Mit SYSTEM V.4 haben die graphischen Benutzeroberflächen (*graphical user interface* GUI) offiziell Einzug in UNIX gehalten. Das *X- Windows System*, kurz X oder X11 genannt, wurde am *Massachusetts Institute of Technology* (MIT) entwickelt. Durch eine eigene Portierung von X auf Basis der Releases 4 oder 5 vom MIT, setzen einige Hersteller eine eigene Oberfläche ein (z.B. OPEN LOOK bei Sun, Motif der OSF und Athena vom MIT).

Im September 1987 erschien *X Version 11 Release 1* (X11R1). Im gleichen Jahr wurde das X-Consortium gegründet, das die Entwicklung von X unterstützen und den Standard sichern sollte. Die aktuelle Version ist seit Anfang 1993 X11R5, jedoch ist X11R6 bereits veröffentlicht.

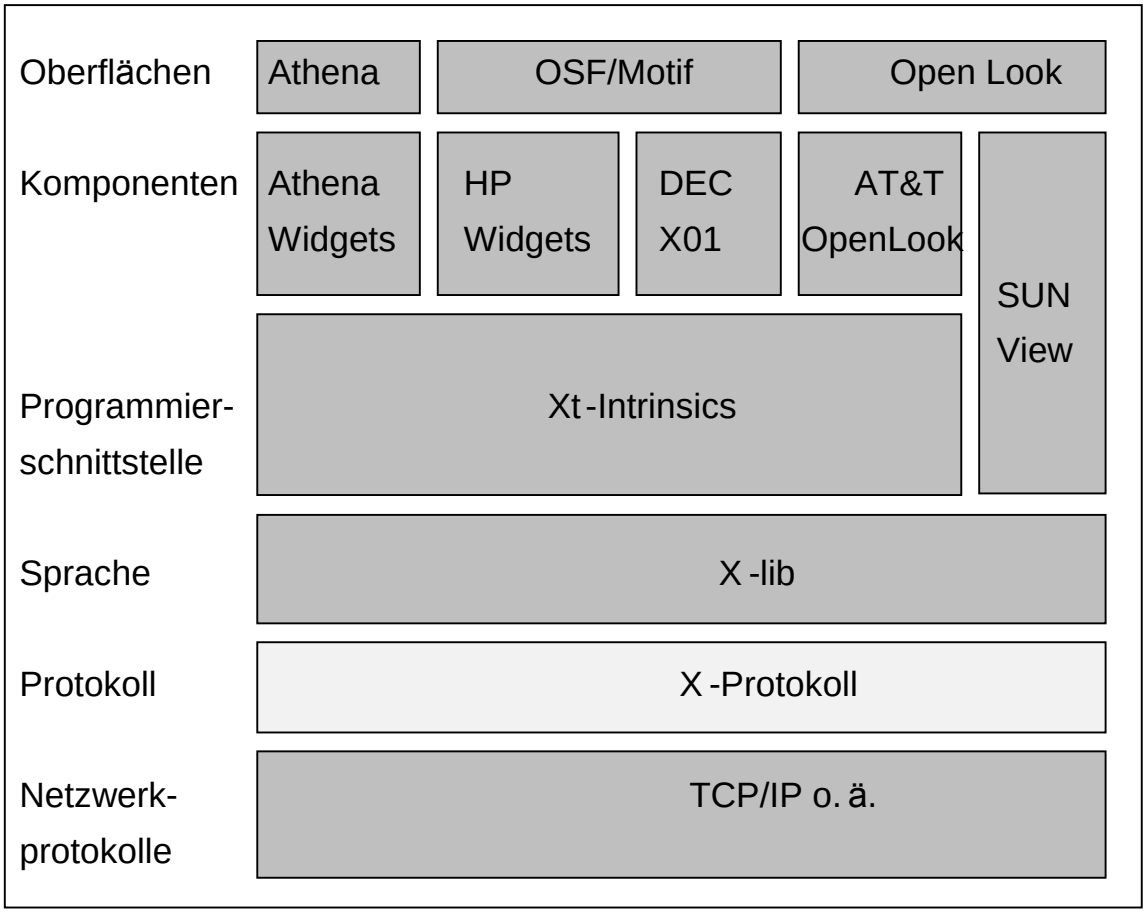


Abbildung 2 Schichtenmodell der X - Window Oberfläche

Ziele bei der Entwicklung waren:

- Hardwareunabhängigkeit
  - geräteunabhängige Anwendungen durch Auslagern der Hardwareansteuerung in andere Schichten
  - Unterstützung verschiedener Terminals und Hardwareplattformen
- Netzwerktransparenz
  - transparenter Zugriff auf remote Ressourcen
  - Kommunikation unabhängig von physikalischer Schicht
- Multitasking / Multiuser Fähigkeit
  - mehrere Anwendungen können gleichzeitig ablaufen
  - ein Display kann von mehreren Benutzern gleichzeitig im Gebrauch sein
- Multiwindow /Multiscreen Fähigkeit
  - mehrere Fenster im Text / Graphik, die sich überlappen können
  - mehrere Oberflächen auf mehreren Terminals

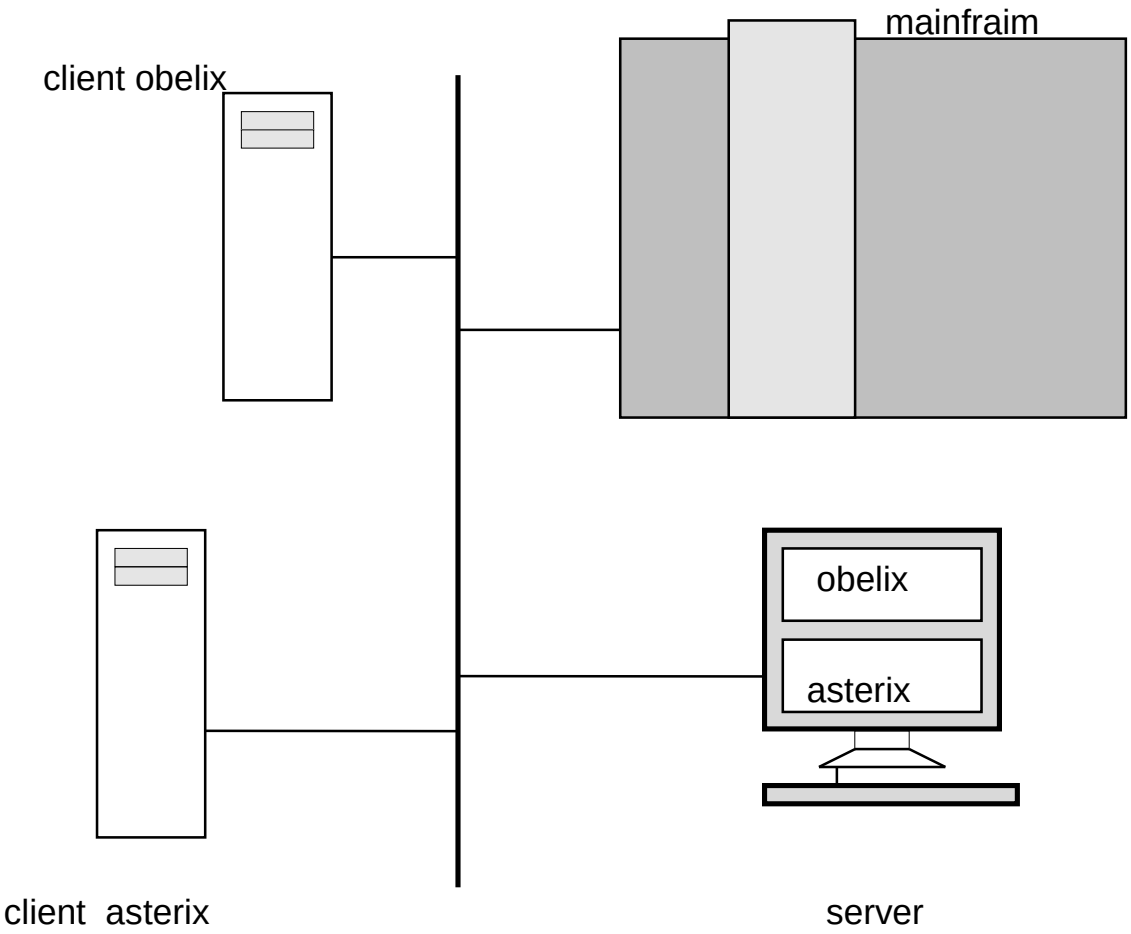


Abbildung 3 client - server - Modell

X ist ein Window- System, mit dem auf einem Graphikbildschirm Windows erzeugt und verwaltet werden können. Folgende Eigenschaften zeichnen das System aus:

– Bei X werden die hardwarenahen Teile für die Ansteuerung des Graphikbildschirms in einem Server- Prozeß isoliert und damit vor den Anwendungsprogrammen verborgen. Der s**erver** steuert den Bildschirm an und verwaltet die Eingabegeräte wie z.B. Tastatur, Maus, Tablett usw. Über ein maschinenunabhäniges Kommunikationsprotokoll, das X- Protokoll, stellt der **server** graphische Funktionen bereit, die von den Anwendungen, den **clients**, genutzt werden.

– X ist netztransparent, d.h. Programme verhalten sich bei der Ausführung auf einem beliebigen Rechner im Netz genau so, wie wenn Sie auf dem lokalen System gestartet worden währen. Da die Programme mit dem **server** über das X- Protokoll kommunizieren, kann jede Anwendung Ausgaben auf allen Graphikbildschirmen erzeugen, die von einem X-Server angesteuert werden. Dadurch lassen sich - bei gleicher Oberfläche - vorhandene Recherkapazitäten sinnvoll aufteilen und größere Systeme z.B. auch von kleineren Rechnern aus mitbenutzen.

Der X-Server ist nicht Bestandteil des Betriebsystemkerns, sondern läuft als gewöhnliches Anwenderprogramm unter dessen Kontrolle.

2 Systemarchitektur

2.1 Komponenten

Ein UNIX-System besteht im wesentlichen aus 3 Komponenten:

- dem Kern (= *kernel*)
- den Systemprogrammen und der Shell (Kommandointerpreter)
- den Utilities (Systemprogramme und Anwenderprogramme)

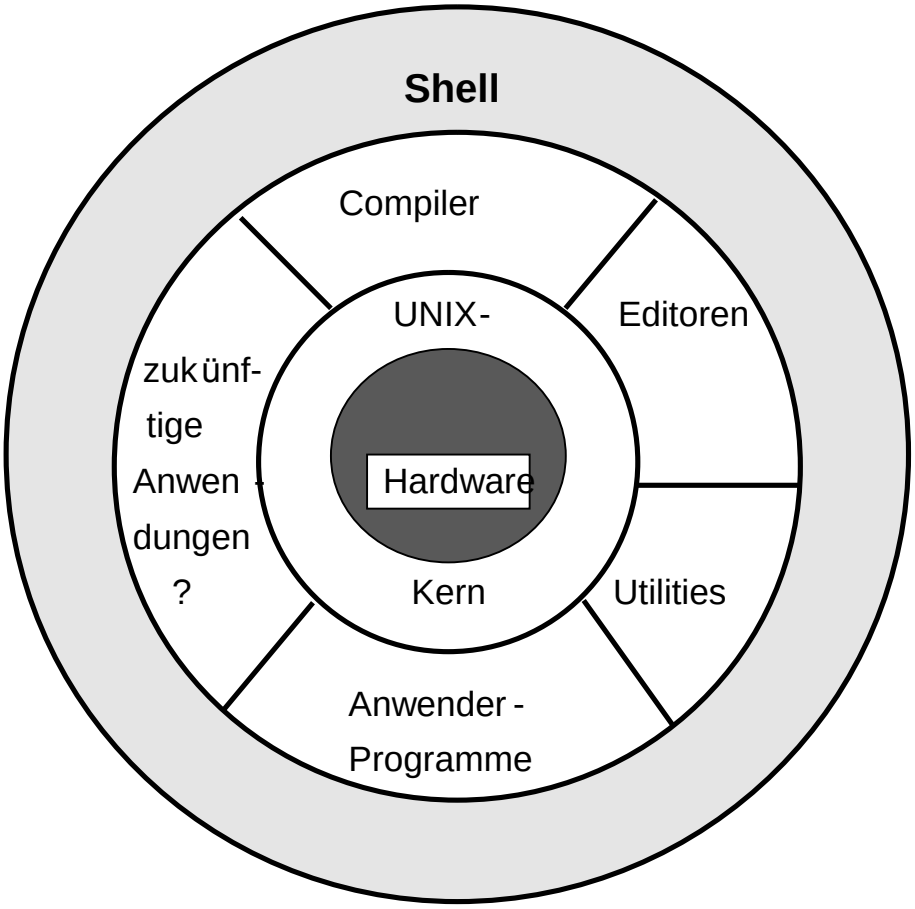


Abbildung 4 Systemaufbau

Der Kern hat die Aufgabe, alle dem System zur Verfügung stehenden Ressourcen zu verwalten. Dazu zählen alle Peripheriegeräte, das UNIX- Filesystem, die Verteilung der vom Anwender gestellten Aufgaben und das Verwalten des Speichers und der Prozesse.

Die Systemprogramme realisieren die Multi- User- Umgebung, die das Anmelden über ein Terminal oder Netzwerk ermöglichen. Die Shell ermöglicht den Dialog mit dem Benutzer, startet Programme und stellt weitere Funktionen des Systems zur Verfügung.

Die Utilities sind Werkzeuge wie **vi**, **cc**, **dbx**, **awk** oder **sed**, welche dem Anwender zur Verfügung stehen.

2.2 Dateisystem

2.2.1 Allgemein

Das Dateisystem unter UNIX hat eine hierarchische Baumstruktur, dessen Wurzel als **root** bezeichnet und durch ein "/“ dargestellt wird. Der Pfadname definiert den Weg, dem man folgen muß, um eine bestimmte Datei zu finden.

Bei den Datei- und Verzeichnissnamen gibt es keine Vorschriften wie sie geschrieben werden. Ob klein oder groß geschrieben, mit Zahlen und Sonderzeichen, UNIX nimmt alles. Alles ist jedoch nicht sinnvoll. Auch Dateiendungen, wie .exe, .bat oder .txt unter MS - DOS, sind nicht notwendig.

Beginnt ein Pfadname mit einem "/", so handelt es sich um einen Pfad mit der **root** als Ausgangspunkt. Solche Pfade nennt man **absoluter** Pfad (z.B. /usr/include/stdio.h). Beginnt ein Pfad nicht mit einem "/", handelt es sich um einen **relativen** Pfad (z.B. X11/bin/dbxtool), ausgehend von dem aktuellen Arbeitsverzeichnis. Die einzelnen Verzeichnisse werden bei der Angabe eines Pfades immer durch ein "/" getrennt.

Ein normaler Benutzer kann üblicherweise nur in seinem Verzeichnis Manipulationen vornehmen. Darunter liegende Verzeichnisse sind für gewöhnliche Benutzer allenfalls lesbar!

Kommandos sind in Dateien gespeichert und werden durch Angabe des Dateinamens ausgeführt. Die Peripheriegeräte des Systems, also Festplatte, Magnetband, CD-ROM Laufwerke, Terminals, Drucker usw., selbst der Hauptspeicher des Systems werden im Dateisystem abgebildet und können so von den Programmen über Dateinamen angesprochen werden. Dieser Aufbau stellt somit eine einheitliche Schnittstelle zu den Ressourcen des Systems und vereinfacht somit dem Benutzer den Zugriff auf diese Ressourcen, egal ob Daten, Peripherie, Prozesse oder Terminals.

Dateien sind Datenströme beliebigen Inhaltes - sie können ASCII-Texte, binäre Dateien, ausführbare Programme oder Graphiken enthalten. Das Betriebssystem unterstützt keine weitere Strukturierung der Daten in einer Datei (z.B. Datenbanken mit Index- Sequentiellem Zugriffsverfahren).

Verzeichnisse sind ebenfalls als Dateien implementiert, aber bei ihnen setzt der Kern eine bestimmte Strukturierung voraus. Verzeichnisse enthalten Einträge, in denen die Namen der in ihnen enthaltenen Dateien und eine Dateinummer, die ***inode***- Liste (index- node), stehen (ähnlich der FAT unter MSDOS). Eine ***inode*** ist eine Dateistruktur auf der Festplatte, in der die administrativen Informationen über eine Datei oder ein Verzeichnisse abgelegt sind. Die *inode*-Liste enthält u.a. den Dateityp, den Dateibesitzer, die Zugriffsrechte der Datei und die Adressen der Dateiblöcke auf der Festplatte. Bei dem Zugriff auf eine Datei geben Sie deren Namen an. Der Betriebssystemkern ermittelt, mit dem entsprechenden Verzeichnisse, die diesen Namen

zugeordnete *inode*- Liste. Mittels der *inode*- Liste werden dann auf die Datenblöcke der Datei, die durch inodes gekennzeichnet sind, zugegriffen.

Spezialdateien (*special files*) enthalten keine Daten. Wenn ein Programm darauf zugreift, werden vom Betriebssystem spezielle Funktionen ausgeführt. Zu den Spezialdateien gehören u.a. die Gerätedateien (im Verzeichnis /dev), mit denen die Peripheriegeräte im Dateisystem abgebildet werden. Diese Dateien und die Betriebssystemaufrufe bilden sozusagen den „Gerätetreiber“.

Links bieten die Möglichkeit, mit einem weiteren Dateiname auf die gleiche Datei zuzugreifen. Dabei wird einer vorhandenen *inode* ein neuer Name zugeordnet. Wenn ein Name entfernt wird, stehen die Daten noch unter dem anderem Namen zur Verfügung.

Bei symbolischen link (auch *soft- link* genannt) werden ebenfalls zusätzliche Namen für eine Datei eingerichtet. Im Gegensatz zu normalen Links (auch *hard- link* genannt) erlauben symbolische Links auch Verweise auf Directories und Verweise über Dateisystemgrenzen hinweg. Löscht man die Zieldatei, auf die ein soft- link verweist, führt der Zugriff auf die Datei über den soft- link zu einer Fehlermeldung des Programms. Richtet man später wieder eine Datei entsprechenden Namens ein, funktioniert wieder alles wie zuvor.

2.2.2 Verzeichnis - Hierarchie

Um einen Überblick über das historisch gewachsenen Dateisystems zu erhalten, werden hier die wichtigsten Verzeichnisse vorgestellt.

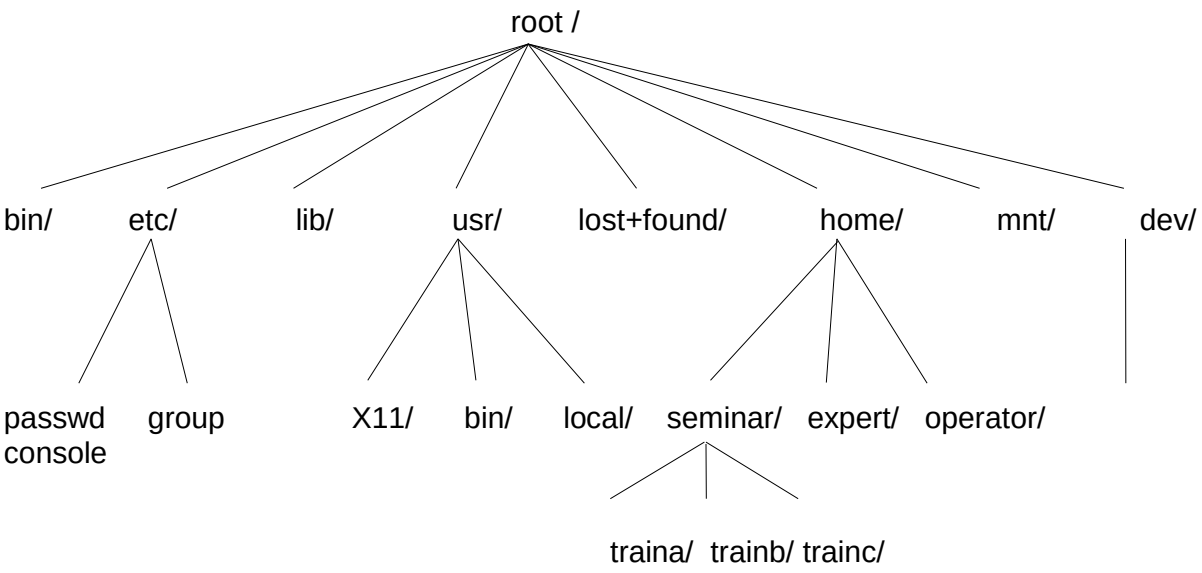


Abbildung 5 Dateibaum

Auszug aus dem Dateisystem:

`/bin` (*binaries*) Dieses Verzeichnis enthält ausführbare Programme wie z.B. **who**, **date** und **ls**.

`/lib` (*libraries*) enthält Bibliotheken für die Programmentwicklung, wie z.B. die C- Standard- Programmbibliothek und andere Dateien.

/etc	(*et cetera*) Hier befinden sich Systemverwalterkommandos, Systemprogramme und einige Konfigurationsdateien wie z.B. die Datei **/etc/passwd**, in der die Benutzerkennungen abgespeichert sind.
/tmp	Hier werden von den Systemkommandos temporäre Dateien abgelegt.
/dev	(*devices*) Diese Verzeichnis enthält alle Gerätedateien.
/usr	Dort werden neben den Anwendungen auch Konfigurationsdateien abgelegt.
/usr/man	enthält die *manpages* des Online- Manuals.
/home	In diesem Verzeichnis wird für jeden Benutzer ein eigens Heimatverzeichnis, in der Regel mit dem Benutzernamen, angelegt. Nach dem Anmelden in das System, befindet sich der Benutzer normalerweise dort.
/mnt	Dient als *mountpoint* für zeitweise anzuschaltende Geräte.
/var	enthält alle veränderlichen Systemdateien, wie z.B. die Protokolldateien in **/var/adm**, die Warteschlangen für das Druckersystem in **/var/spool** und die Mailboxen der Benutzer in **/var /mail**.

2.3 Prozeßsystem

UNIX muß als Multitasking- Betriebssystem für jedes aktive Programm ein Reihe von Daten verwalten: z.B. Ablaufprioritäten, Ladeadressen und Benutzernummern.

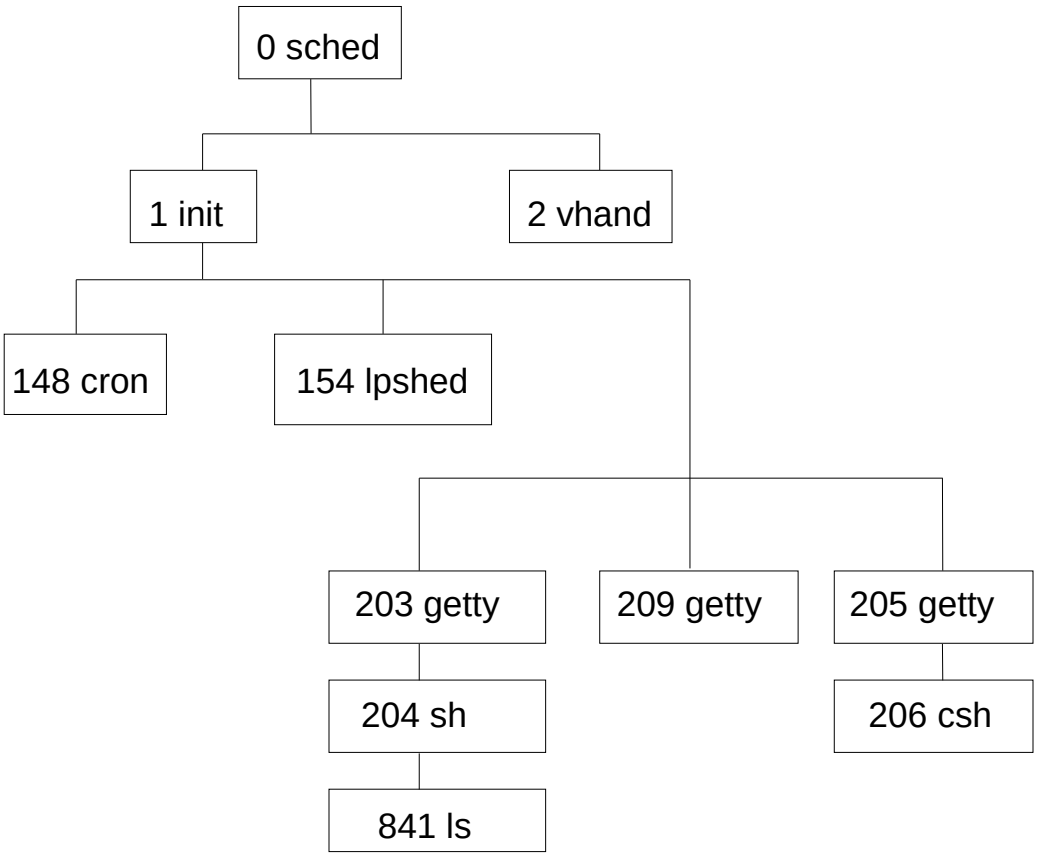


Abbildung 6 Prozeßstruktur

Definitionen:

Ein Programm ist eine zusammengehörige Menge von Befehlen und Daten. Jedes Programm ist in einer Datei gespeichert. Unter UNIX sind Programme nur als Prozeß ablauffähig.

Ein Prozeß ist ein geladenes Programm. Die Befehle des Programms werden im Arbeitsspeicher, im sogenannten Befehlssegment, und die Daten im Datensegment abgelegt. Zu jedem Prozeß gehört eine Prozeßumgebung.

Prozeßumgebung umfaßt alle Attribute, die das Betriebssystem zur Verwaltung der Prozesse benötigt, z.B.:

- Prozeßnummer PID (*process identificationnumber*)
- Prozeßeigentümer UID/GID (*user identification / group identification*)
- aktuelles Dateiverzeichnis
- Priorität (Reihenfolge der CPU- Zuteilung)

Wie die Dateistruktur ist auch die Prozeßstruktur hierarchisch aufgebaut: UNIX unterscheidet zwischen Vater- und Sohnprozessen. Jeder Prozeß hat einen Vater (einzige Ausnahme: der sched-Prozeß). Eventuell ist der Prozeß selbst auch Vater weiterer Prozesse (Sohnprozesse).Beim Booten von UNIX entstehen die folgenden Prozesse:

shed	Memory- Managment, interner Prozeß des Betriebssystems zur Ein- und Auslagerung von Programmen. Er hat die PID 0.
init	Initialisierung (letzter Boot- Schritt). Startet die Unix- System-Prozesse gemäß den Informationen, die in der Datei **/etc/inittab** abgelegt sind. Er hat die PID 1.
cron	Der sogenannte Dämonprozeß, der jede Minute die **crontab**- Dateie untersucht und eventuell dort abgelegte Zeitaufträge ausführt.
lpshed	Verwaltung der Druckerwarteschlange.
vhand	Regelt das *paging* (auslagern von Speicherblöcken auf die Festplatte)
getty	Terminalprozeß, der die Login- Meldung ausgibt, den Login- Namen und das Passwort liest und, wenn beide Eingaben richtig sind, eine Shell startet.

2.4 Festplattenstruktur

Bei der Generierung des UNIX - Systems muß die Platte für die Aufnahme der UNIX - Dateisysteme vorbereitet werden. Dazu wird die Platte zuerst formatiert und danach auf ihr ein File - System eingerichtet. Anschließend ist die Platte wie folgt eingeteilt:

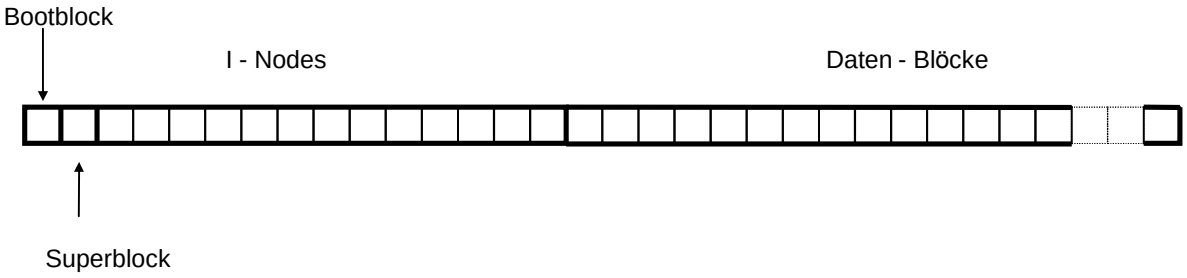


Abbildung 7 Festplattenaufbau

Bootblock

Informationen für den Systemstart

Superblock

Er enthält Informationen über das Dateisystem:

- die Größe und den Namen des Dateisystems,
- die für die Indexknoten (*inode*) reservierte Blockzahl
- Pointer auf das erste Element der Liste freier Datenblöcke
- Pointer auf das erste Element der Liste freier *inodes*

Zu jedem gegebenen Zeitpunkt gibt es mindestens zwei Superblöcke, einen auf der Platte und einen im Hauptspeicher, wobei der Block im Hauptspeicher der „jüngere“ ist.

Das **sync** - Kommando bringt den Superblock auf der Platte auf den neuesten Stand.

Jede Datei eines UNIX - Systems wird durch einen eindeutigen *inode* (ganze positive Zahl) auf der Platte repräsentiert. Die Nummer des *inode* wird vom System vergeben und befindet sich im Verzeichniseintrag der zugehörigen Datei. Inodes sind Dateibeschreibungen des Systems, die in der Regel aus einem 64 - Byte Eintrag bestehen. Ein *inode* enthält die folgenden Informationen:

Dateigröße in Bytes
Dateityp (reguläre Datei, Gerätedatei, Verzeichnis)
Anzahl der links
Dateieigentümer
Gruppe
Zugriffsrechte
Datum der Erstellung
Datum der letzten Änderung
Datum des letzten Gebrauchs
Pointer auf die ersten 13 Dateiblöcke

Kennt man von einer Datei nur die *inode* - Nummer, dann läßt sich mit der Anweisung

```
% ncheck -i inode
```

der Pfadname des inodes ermitteln (manche Systemfehlermeldungen oder Backup - Kommandos verweisen auf inode - Nummern und nicht auf Dateinamen, das Kommando ls -i listet die inodes und Dateinamen des laufenden Verzeichnisses auf).

2.5 Das Anmelden am System (Login - Prozeß)

Damit Sie sich am UNIX-System anmelden können, müssen Sie einen Benutzerkennung und ein Paßwort haben. Dies wird vom Systemadministrator eingerichtet.

```
% login:
```

Geben Sie nun hinter der Login- Aufforderung Ihren Benutzernamen ein. Bitte beachten Sie, daß das UNIX-System zwischen Groß- und Kleinbuchstaben unterscheidet. In der Regel wird die Benutzerkennung in Kleinbuchstaben geschrieben.

```
% passwd:
```

Für das Paßwort, welches normalerweise auch vom Systemadministrator vergeben wird, gibt es, je nachdem wie der Systemadministrator „sein“ System eingerichtet hat, die Möglichkeiten der Groß- und Kleinschreibung und auch mit oder ohne Ziffern. Das Paßwort sollte nach dem ersten Login vom Benutzer mit dem Befehl **passwd** geändert werden.

```
% passwd <user>
```

Nach dem korrekten Login können verschiedene Meldungen erscheinen (z.B. die sogen. motd = „*message of the day*“, oder eine Nachricht über den Erhalt einer *mail*).

Sofern nicht weitere Ausgaben durch die individuelle **.cshrc** (bei der csh, und .profile bei der sh) Datei (s.u.) veranlaßt werden, ist damit der Anmeldevorgang abgeschlossen und das System meldet seine Bereitschaft Kommandos entgegenzunehmen durch Ausgabe des Prompt - Zeichens. Dies ist für den normalen Benutzer durch ein "%" bei der **csh** ("$"bei der **sh**) dargestellt. Der **superuser** hat in der Regel das Zeichen "#" als Prompt. In der Datei **.cshrc** kann mit der Shellvariablen (s.u.) **$prompt** (oder $PS1 bei der ch), das Prompt - Zeichen oder die Prompt - Zeichenfolge geändert werden.

2.5.1 Allgemeiner Kommandoaufbau

Kommando [<Optionen>] [<Parameter>]

```
% ls -l /usr/otto
```

Die Klammern [ ] dienen als Hinweis, daß das Kommando mit oder ohne Optionen bzw. Parameter eingegeben werden kann. Kommando, Option und Parameter sind jeweils durch ein Leerzeichen getrennt.

2.6 Das Home - Verzeichnis

Das HOME - Verzeichnis eines Benutzers enthält im allgemeinen seine eigenen Dateien und Programme, kann aber auch Unterverzeichnisse enthalten, die der Benutzer selbst einrichtet.

Schon bei der Einrichtung eines neuen Benutzers wird in dessen HOME - Verzeichnis eine Datei mit dem Name **.cshrc** (csh, bei der sh .profile) erzeugt. Dabei handelt es sich um ein Shell - Skript, das automatisch mit dem LOGIN gestartet wird und die Arbeitsumgebung des Benutzers

weitgehend bestimmt. Diese Datei entspricht in ihrer Funktion weitgehend der autoexec.bat unter MSDOS.

Weitere Dateien im Home - Verzeichnis können sein:

.exrc	enthält Einstellungen zur Steuerung des Editor **vi**.
.cshrc	Startskript für die **csh** - Shell
.profile	Startskript für die **sh** - Shell.
.plan	Dort kann der Benutzer Informationen z.B. zu seiner Person und seinen Aufgaben im System ablegen, die jeder Benutzer durch den Befehl **finger** abrufen kann.
.history	enthält die zuletzt vom Benutzer eingegebenen Befehle.

Alle diese Dateien sind durch den vorgesetzten Punkt als Systemdateien gekennzeichnet und sollten nur dann geändert werden, wenn dem Benutzer die Funktionen der Dateien bekannt ist.

2.7 Weitere grundlegende Befehle

2.7.1 Standort feststellen und wechseln

Nach dem Anmelden befindet sich der Benutzer im Home - Verzeichnis. Um den Namen des aktuellen Verzeichnisses auszugeben, gibt es das Kommando

```
% pwd (print working directory)
% /home/otto
```

Die UNIX - Antwort auf dieses Kommando ist der absolute Pfadname des Verzeichnisses, der am Bildschirm ausgegeben wird.

Will ein Benutzer sein Verzeichnis wechseln, so gibt er das Kommando **cd** ein, gefolgt von einem Leerzeichen und dem Pfadnamen (absolut oder relativ) des Zielverzeichnisses.

```
% cd /home/otto/bin (change directory)
```

Will der Anwender jedoch vom Arbeitsverzeichnis eine Stufe höher zur Wurzel dieses Arbeitsverzeichnisses, so genügt die Anweisung

```
% cd ..
```

Mit der Anweisung

```
% cd
```

wechselt man zurück in das Home - Verzeichnis.

Hinweis

Der Punkt alleine, ist ein Synonym für das aktuelle Verzeichnis!

2.7.2 Auflisten von Verzeichnisinhalten

Der Inhalt eines Dateiverzeichnisses kann mit dem sehr mächtigen ls - Kommando ausgegeben werden.

```
% ls [ <Option> ] [ <Pfad> ]
```

Von den zahlreichen möglichen Optionen sind folgende von Bedeutung:

```
-l          Langform in Angabe von Zugriffsrechten, Referenzen , Eigentümer,
            Dateigröße und  Datum der letzten Änderung.
-a          alle Dateien und Unterverzeichnisse, auch „Punkt" - Dateien
-i          Ausgabe der inode - Nummer
-F          Ausgabe mit Kennzeichnung (Verzeichnisse mit "/", ausführbare
            Dateien mit "*", Links mit "@")
-R          Rekursiv, erzeugt eine Liste des gesamten Dateibaumes, angefangen
            mit dem aktuellen Verzeichnis.
```

Alle Optionen können miteinander kombiniert werden.

Mit dem Kommando

```
% du
```

kann sich der Benutzer über die Speicherplatzbelegung des aktuellen Verzeichnisses und aller Unterverzeichnisse informieren. Dabei werden Verzeichnisse, für die keine Zugriffsberechtigung besteht, nicht angezeigt.

2.7.3 Verzeichnis erstellen

```
% mkdir <Name>
```

Bei dem Erzeugen einer Datei sind keine weiteren Regeln zu beachten.

2.7.4 Weitere nützliche Kommandos

Nachrichten am Bildschirm ausgeben.

```
% echo "hallo otto\n!"
% hallo otto!
```

Dieses Kommando wird hauptsächlich dazu benutzt, per Shell - Skripts Nachrichten oder Fragen am Bildschirm auszugeben.

Datumsanzeige

```
% date
% Su Nov 11 11:11:11 MET 1992
```

gibt das aktuelle Datum und die Uhrzeit auf dem Bildschirm aus. Standardmäßig wird das Datum in amerikanischer Form angezeigt:

Online Manual

Das UNIX - System verfügt über ein ausführliches Online - Manual, das Informationen über die einzelnen Kommandos (insbesondere alle möglichen Optionen) beinhaltet. Der Inhalt entspricht in der Regel dem des *Reference Manual.*

```
% man ls
```

erzeugt eine durch den **troff** formatierte Anzeige der *manualpages* für das ls - Kommando.

Wer ist Wer?

Das Kommando

```
$ who
```

erzeugt eine Liste aller Benutzer, die zur Zeit am System eingeloggt sind, den Namen des Terminals, an dem Sie arbeiten und das Datum (incl. Uhrzeit) Ihrer Anmeldung.

Sollten Sie einmal nicht mehr wissen, unter welchem Benutzer Sie arbeiten, gibt Ihnen UNIX mit folgendem Befehl gerne Auskunft:

```
% whoami
( % who am i )
```

3 Dateien erzeugen und bearbeiten

3.1 Der Texteditor vi

Das Betriebssystem UNIX verfügt über mehrere Editoren, die Zeilen- oder Bildschirmorientiert sind. Zu der ersten Kategorie zählen die Programme **ed** und **sed** (Stream - Editor für sehr große Dateien), zu der zweiten Gruppe gehört unter jedem UNIX traditionell der **vi.**

```
% vi [ <Dateiname> ]
```

Ist die angesprochene Datei vorhanden, werden die ersten 23 Zeilen (Terminal mit 80x24 Spalten und Zeilen) auf dem Bildschirm dargestellt. In der untersten Bildschirmzeile wird der Dateiname und die Größe der Datei in Zeilen angegeben. Ist keine Datei dieses Namens vorhanden, wird der Bildschirm gelöscht und in der letzten Bildschirmzeile wird der angegebene Dateiname, sowie der Hinweis „[New file]“ ausgegeben.

Der Editor arbeitet in zwei verschiedene Modi, dem Kommandomode und dem Eingabemode.

Nach dem Aufruf befindet sich der Editor zunächst im Kommandomode, d.h. es werden nur Kommandos angenommen (und ausgeführt), nicht aber Texteingaben.

Der Übergang in den Eingabemode erfolgt über Kommandos, die jeweils aus einem einzelnen Zeichen bestehen und nicht mit der ENTER - Taste abgeschlossen werden. Durch diese Kommandos können verschiedene Varianten des Eingabemodes aufgerufen werden, die sich durch die Positionierung der eingegebenen Zeichen im Text unterscheiden. Diese Kommandos sind :

```
i               Eingabe vor der aktuellen Cursorposition (INSERT-MODE)
a               Eingabe hinter der aktuellen Cursorposition (APPEND-MODE)
R               Überschreiben des vorhandenen Textes (REPLACE-MODE)
o               Eingabe in eine neue Zeile unterhalb der aktuellen Zeile (OPEN-MODE)
O               Eingabe in eine neue Zeile oberhalb der aktuellen Zeile (OPEN-MODE)
```

Die Art des Eingabemodes wird im unteren rechten Bildschirmbereich angegeben, z.B. „INSERT-MODE“. Während der Eingabe bewirkt die Enter - Taste einen Zeilenwechsel. Dadurch können ohne Änderung des Modes beliebig viele Textzeilen nacheinander eingegeben bzw. in den Text eingefügt werden.

Alle Eingaben werden durch die Taste <ESC> beendet, danach verschwindet die Modeanzeige, und der Editor befindet sich wieder im Kommandomode.

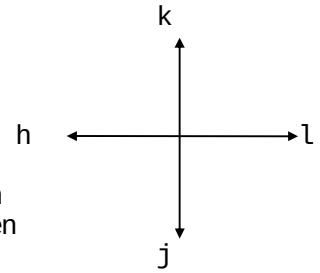
Der Editor kann nur im Kommandomode beendet werden, mit :

```
ZZ              Abspeichern und beenden
:wq             Abspeichern und beenden
:w              Abspeichern und weiter
:w!             Abspeichern einer durch Schreibschutz gesicherten Datei (nur wenn
                der Benutzer auch Eigentümer der Datei ist)
:w <name>       Text in einer Datei unter den angegebenen Namen speichern
:q!             Beenden ohne Abspeichern
:!<Kommando>    Ausführen eines shell- Kommandos
```

Im Kommandomode sind außerdem folgende Operationen möglich.

Cursorbewegung:

Der Text - Cursor kann entweder mit den Pfeiltasten oder mit den Buchstaben h, k, j und l gemäß dem nebenstehenden Schema um eine Position nach links oder rechts, bzw. um eine Zeile nach oben oder unten bewegt werden.



Cursor positionieren:

+	Cursor auf den Anfang der nächsten Ziele
-	Cursor auf den Anfang der vorigen Zeile
$	Cursor auf das Ende der aktuellen Zeile
H	Cursor auf die oberste Bildschirmzeile
M	Cursor auf die mittlere Bildschirmzeile
L	Cursor auf die unterste Bildschirmzeile
<n>H	Cursor auf die n. Bildschirmzeile von oben
<n>L	Cursor auf die n. Bildschirmzeile von unten
<n>G	Cursor auf die n. Ziele der Datei
G	Cursor auf die letzte Zeile der Datei

Blättern :

^f	eine Bildschirmseite vorwärts
^b	eine Bildschirmseite rückwärts
^d	eine halbe Bildschirmseite vorwärts
^u	eine halbe Bildschirmseite rückwärts

Textzeile löschen :

x	Zeichen unter dem Cursor
dw	bis Ende Wort oder nächstes Wort
dd	gesamte aktuelle Zeile
<n>dd	n Zeilen ab der aktuellen Zeile

Zeichenfolge suchen :

/<string>	Zeichenfolge suchen (vorwärts)
?<string>	Zeichenfolge suchen (rückwärts)
/^<string>	Zeile, die mit <string> beginnt
/<string>%	Zeile, die mit <string> endet
n	letzten Suchauftrag wiederholen.
N	rückwärts suchen

Zeichenfolge oder Textteile ändern:

r<char>	Zeichen an der Cursorposition ändern in <char>
cw	Wort ab Cursorposition ändern. Das Wortende wird mit % markiert, das Wort wird bis dahin überschrieben. Weitere Zeichen werden am Ende des Wortes eingefügt. Anzeige: CHANGE MODE, dieser Mode muß mit <ESC> beendet werden.
cc	aktuelle Zeile wird gelöscht, kann sofort neu eingegeben werden (INPUT MODE).
C	Rest der Zeile ab Cursor ändern (CHANCE MODE).
u	letzte Änderung rückgängig machen.
U	alle Änderungen in der aktuellen Zeile rückgängig machen (nur, solange diese nicht verlassen wurde!).

Textteile verschieben :

a) Texteile in den Puffer laden

```
yw              ab Cursor bis Ende Wort
Y               aktuelle Zeile
<n>Y            aktuelle + n weitere Zeilen
```

b) Pufferinhalt in den Text einfügen

```
P               vor dem Cursor bzw. oberhalb des Cursors
p               hinter den Cursor bzw. unterhalb des Cursors
```

Die Steuerdatei .exrc

In der „Rohform“ hat der Editor noch einige Nachteile:

– Die Zeilen werden nicht automatisch durchnumeriert, das ist aber insbesondere bei der Bearbeitung von Programmen wünschenswert.

– Es erfolgt kein automatisches Einrücken, d.h. beim Zeilenwechsel springt der Cursor immer auf die erste Spalte. Bei Programmen (insbesondere bei C - Programmen) ist es aber erwünscht, daß eine Zeile genauso weit eingerückt wird, wie die vorhergehende Zeile.

Diese Nachteile können durch sogenannte *options* behoben werden, die im Kommandomode des vi durch Anweisungen der Form **:set <attr>** aktiviert werden können, wobei <attr> bestimmte Attribute sind, durch die die Arbeitsweise des vi gesteuert wirden.

Zweckmäßiger ist es allerdings, solche Anweisungen in eine Steuerdatei zu schreiben, die als Kommandodatei bei jedem Start des vi durchlaufen wird (ähnlich wie die Datei .profile bei der Anmeldung). Diese Datei hat den Namen **.exrc**. Im Anhang befindet sich eine Beispieldatei.

3.2 Dateien kopieren, umbenennen, löschen

Dateien kopieren

```
% cp <Datei1> <Datei2>
% cp <Dateiliste> <Zielverzeichnis>
% cp [ -r ] <Quellverzeichnis> <Zielverzeichnis>
```

```
-r              Rekursives kopieren
```

In der ersten Version wird der Inhalt von <datei1> in <datei2> übernommen. Wenn <datei2> bereits existiert, behält sie ihre Eigenschaften (Eigentümer, Zugriffsrechte etc.), der vorherige Inhalt wird gelöscht. Wenn die Datei nicht existierte, erhält sie die Eigenschaften von <datei1>.

In der zweiten Version können eine oder mehrere Dateien, ggf. unter Verwendung von Metazeichen wie *, ? usw. in eine anderes Verzeichnis kopiert werden.

Sollten sich die Dateien nicht im aktuellen Verzeichnis befinden (oder dorthin kopiert werden), muß jeweils der Pfadname eingesetzt werden.

Bei der dritten Version werden alle Dateien und Unterverzeichnisse aus dem Quellverzeichnis, in das Zielverzeichnis kopiert.

Folgende Zugriffsrechte müssen (für den Aufrufenden) gewährleistet sein :

```
Quelldatei (en)  r
Quellverzeichnis rx
Zieldatei        w
Zielverzeichnis  wx
```

Beispiel

```
% cp /home/otto/*.c .
```

Mit diesem Kommando werden aus dem Verzeichnis /home/otto alle C - Programme in das aktuelle Dateiverzeichnis kopiert.

Dateien verschieben oder umbenennen

```
% mv <Datei1> <Datei2>
% mv <Verzeichnis1> <Verzeichnis2>
% mv -r <Dateiliste> <Verzeichnis>
```

Wirkung wie bei cp, jedoch werden alle Quelldateien gelöscht. Das bedeutet bei der ersten Version, daß die Quelldatei nur einen anderen Namen erhält bzw. umbenannt wird. Ebenso ist es möglich, Dateiverzeichnisse umzubenennen (Version 2). In der dritten Version werden die Quelldateien, auch aus in den Unterverzeichnissen, in ein anderes Verzeichnis verschoben.

Beispiel

```
% mv *.c cprogs
```

Alle C - Programme werden in das Unterverzeichnis cprogs verschoben. Das Verzeichnis muß existieren.

Einer Datei einen zusätzlichen Namen geben (links)

```
% ln [ -s ] <Datei1> <Datei2>
```

```
-s              soft- link
```

Hier wird keine neue Datei erzeugt, sondern <datei1> erhält einen zusätzlichen Namen. Dieser kann sich durchaus in einem anderen Verzeichnisbefinden (Angebe eines Pfadnamens für <datei2>). Die Anzahl der Namen eines Dateiobjektes wird bei der Auflistung mit dem Kommando **ls -l** unmittelbar vor dem Namen des Eigentümers angegeben (Anzahl der Referenzen).

Zu beachten ist, daß jeder Eintrag eines Namens in einem Dateiverzeichnis als Referenz gilt, so daß z.B. jedes Verzeichnis mindestens zwei Referenzen besitzt (Eine Eintragung im Parent - Verzeichnis, eine zweite im Verzeichnis selbst). Für jedes Unterverzeichnis kommt eine weitere Referenz hinzu. Daher kann man aus der Zahl der Referenzen eines Verzeichnisses auf die Zahl der Unterverzeichnisse schließen.

Beispiel

```
% ln egon anton
```

Die Datei **egon** kann nun auch unter den Namen **anton** angesprochen werden.

Dateien löschen

```
% rm [-ifr] <datei(en>
```

```
-i          Für jede Datei wird rückgefragt, ob sie wirklich gelöscht werden
            soll.
-f          Die Datei(en) werden auch dann ohne Rückfrage gelöscht, wenn sie
            schreibgeschützt sind. Dies gilt allerdings nur für den Eigentümer
            und den Superuser.
-r          Mit dieser Option können Dateiverzeichnisse einschließlich aller
            darin enthaltener Dateien und Unterverzeichnisse rekursiv gelöscht
            werden.
```

Ohne Optionen können nur Dateien gelöscht werden, keine Verzeichnisse mit Inhalt. Sind die Dateien schreibgeschützt, erfolgt eine Warnung und die Rückfrage, ob trotzdem gelöscht werden soll. Hat die Datei nur eine Referenz, so wird sie physikalisch gelöscht (**unter UNIX gibt es keine Möglichkeit, gelöschte Dateien wiederherzustellen**), bei mehreren Referenzen wird nur der angegebenen Name gelöscht und die Zahl der Referenzen um 1 verringert.

Beispiel

```
% rm -ir xyz
```

Das Unterverzeichnis xyz soll mit allen darin enthaltenen Dateiobjekten gelöscht werden. Bei jedem Objekt erfolgt eine Rückfrage, gelöscht wird nur, wenn diese mit y beantwortet wird.

Verzeichnis anlegen

```
% mkdir <name>
```

Verzeichnis löschen

```
% rmdir <name>
```

3.3 Ausgabe von Textdateien

Verketten und Anzeigen von Dateien

```
% cat [ -v [ -te ] ] [ <datei(en)> ]
```

Ohne Angabe einer (oder mehrerer) Dateien wartet das Kommando auf Texteingaben über die Tastatur. Nach jedem Zeilenabschluß mit <ENTER> wird die gleiche Zeile auf dem Bildschirm ausgegeben. Der Rücksprung zur Betriebssystem - Ebene (LOGIN - Shell) erfolgt mit "^d". In dieser Form ist das Kommando nur sinnvoll mit einer Ausgabeumleitung (s.u.).

Wird eine Datei angeben, so wird diese auf dem Bildschirm ausgegeben, bei mehreren Dateien werden diese zunächst aneinander gehängt (verkettet) und danach ausgegeben. Auch die Verkettung hat mit der Ausgabeumleitung eine besondere Bedeutung.

-v	Nicht druckbare Zeichen (außer Tabulator) werden dargestellt. Control - Zeichen werden durch ein vorgesetztes ^ gekennzeichnet.
-t	Das Tabulatorzeichen wird als ^I dargestellt (nur in Verbindung mit der Option-v)
-e	Am Ende jeder Zeile wird das Zeichen % ausgegeben (nur in Verbindung mit der Option- v)

Dateien seitenweise ausgeben

% more [<Optionen>] [<Datei(en)>]

Mit diesem Kommando können Dateien; auf den Bildschirm, seitenweise ausgegeben werden. Ohne Optionen werden jeweils 23 Zeilen dargestellt, danach erscheint, wenn die jeweilige Datei noch nicht vollständig ausgegeben wurde, ein Bereitzeichen (Voreinstellung „:“) und das Programm wartet auf ein Kommando. Am Ende jeder Datei erscheint statt des Doppelpunktes die Zeichenfolge „EOF“. Werden mehrere Dateien ausgegeben, wird am Anfang jeder Datei deren Name angegeben.

-<n>	Eine Bildschirmseite wird mit <n> Zeilen ausgegeben.
+<n>	Die Ausgabe beginnt bei der <n>-ten Zeile.
+/<string>/	Die Ausgabe beginnt bei dem ersten Auftreten von <string>
-c	Die Ausgabe wird nicht gerollt, sondern nach jeder Seite gelöscht.
-p<string>	Anstelle des Doppelpunkts wird <string> als Bereitzeichen benutzt. Enthält der Text die Zeichenfolge „%d“, so wird dafür die lfd. Seiten - Nr. eingesetzt.
-s	Bereitzeichen und Meldung werden invertiert ausgegeben.

Nach Ausgabe des Bereitzeichens oder der Zeichenfolge „EOF“ können folgende Kommandos gegeben werden:

<ENTER>	Ausgabe der nächsten Seite.
<n>	Die Ausgabe wird mit der <n>-ten Seite fortgesetzt.
/<string>/	Das nächste Auftreten von <string> in der aktuellen Datei wird aufgesucht (Vorwärts - Suchen).
?<string>?	wie vorher, aber Rückwärts - Suchen.
q	Das Programm pg wird beendet.
h	Help, mit einer Liste der wichtigsten Kommandos.

% pg [<Optionen>] [<Datei(en)>]

Das Kommando **pg** funktioniert ähnlich wie **more**, aber es wird meißtens in einer **pipe** mit anderen Kommandos verwendet.

Formatierte Ausgabe von Dateien

% pr [<optionen>] [<datei(en)>

Das Kommando formatiert die Ausgabe der Datei(en) gemäß den angegebenen Optionen. Ohne Optionen ist die Wirkung wie bei **cat** <datei(en)>, die Ausgabe erfolgt jedoch in „Seiten“ von je 66 Zeilen mit einer Überschrift (Datum, Dateiname und Seiten-Nr.) und 56 Textzeilen. Mit den entsprechenden Optionen wird dieses Kommando vorzugsweise für die Druckerausgabe in Verbindung mit Ausgabeumleitung oder über **pipe** mit dem Kommando **lp** benutzt.

+<n>	Die Ausgabe beginnt auf Seite <n>.
-<n>	Die Ausgabe erfolgt <n> - spaltig.

```
-m              Alle angegeben Dateien werden nebeneinander ausgegeben, je nach
                Anzahl der Dateien wird die Zeilenbreite begrenzt.
-f              Nach jeder Seite wird ein FOMFEED und ein NEWLINE - Zeichen
                eingefügt.
-l<n>           Es werden <n> Zeilen pro Seite ausgegeben.
-h<text>        Anstelle des Dateinamens wird <text> als Überschrift angegeben.
-o<n>           Jede Zeile wird um <n> Zeichen eingerückt.
-p              Nach jeder Seite wird die Ausgabe angehalten, es erfolgt ein
                akustisches Signal und die Ausgabe kann mit <ENTER> wieder
                fortgesetzt werden (wichtig bei der Druckerausgabe mit
                Einzelblatteinzug!).
-n              mit Zeilennummern
```

Beispiel

```
% pr -l72 -o4 -p -h "Hallo Otto!" ostern.c > ralf
```

Die Datei ostern.c wird auf dem Drucker mit 72 Zeilen pro Seite ausgegeben. Alle Zeilen sind numeriert und um 4 Spalten eingerückt, die Seitenübersicht enthält neben Datum und Seiten - Nr. den Text „Hallo Otto!“. Beim Druck wird nach jeder Seite eine Pause zum Papierwechsel eingelegt.

Ausgabe auf Drucker

```
% lp
```

Dieses Kommando kann prinzipiell auch direkt zur Ausgabe von Textdateien auf dem angeschlossenen Drucker benutzt werden. Da jedoch die Ausgabedatei fast immer nach verschiedenen Gesichtspunkten formatiert sein soll, wird das **lp** - Kommando vorwiegend in Verbindung mit dem **pr** - Kommando benutzt.

Anders als bei einer Ausgabeumleitung auf den Drucker, wird der zu druckende Text bei dem **lp** - Kommando nicht direkt zum Drucker geschickt, sondern zunächst in eine Warteschlange eingeführt, die von dem Drucker - Spooler bearbeitet wird. Jeder derartige Druckauftrag erhält eine Auftrags - Nr., die dem Benutzer mitgeteilt wird. Auf diese Wiese können mehrere Benutzer ohne Kollisionen Druckaufträge erteilen, die dann nacheinander abgearbeitet werden. Der jeweilige Zustand der Drucker - Warteschlange kann mit dem Kommando **lpstat** abgefragt werden.

3.4 Klassifizieren und Vergleichen

Datei - Typ bestimmen

```
% file <datei(en)>
```

Das Kommando versucht die angegebene(n) Datei(en) nach bestimmten Kriterien zu klassifizieren. Je nach Implementation werden verschiedene Meldungen ausgegeben:

```
C - Programm text
executable, not stripped
executable
Commands-text
Ascii-text
Data
English text
```

Die Klassifizierung ist nicht sehr präzise, vor allem bei Quelldateien (C oder FORTRAN) werden häufig mit ascii - Texten verwechselt. Trotzdem ist das Kommando gut geeignet, sich einen Überblick über die in einem Dateiverzeichnis vorhandenen Dateien zu verschaffen.

Dateien vergleichen

```
% cmp [ -l ] <Datei1> <Datei2>
% diff <Datei1> <Datei2>
```

Die angegebenen Dateien werden Byte für Byte miteinander verglichen. Ohne der Option - l werden lediglich die lfd. Nr. des Zeichens und die Zeile der ersten Abweichung angegeben. Ohne die Option werden für alle Abweichungen die lfd. Nr. des Bytes und die oktalen Codierungen (ASCII) der beiden Bytes angegeben.

Sortierte Dateien zeilenweise vergleichen

```
% comm [ -123 ] <datei1> <datei2>
```

Diese Kommando ist besonders geeignet zum Vergleich von Tabellen , die aber alphabetisch sortiert sein müssen. Ohne Optionen werden 3 Spalten ausgegeben, die erste Spalte enthält Zeilen, die nur in der ersten Datei vorkommen, die zweite Spalte nur Zeilen, die nur in der zweiten Datei vorkommen und die dritte Spalte enthält Zeilen, die in beiden Dateien vorkommen. Mit den Optionen werden die Spalten angegeben, die nicht ausgegeben werden sollen. So wird mit der Option -l2 nur die Spalte ausgegeben, welche die gemeinsamen Zeilen der Dateien enthält.

3.5 Zugriffsrechte

Alle Kommandos, die sich auf Dateiobjekte beziehen, werden nur ausgeführt, wenn der Benutzer in den betroffenen Verzeichnissen und den für die Dateien entsprechende Rechte besitzt.

UNIX kennt für Dateiobjekte folgende Rechte:

r	Lesen von Dateiobjekten
w	Dateiobjekte erzeugen, ändern oder löschen
x	bei ausführbaren Dateien (Programmen und Shell- Skripts): starten und ausführen, bei Verzeichnissen: Zugriff auf das Verzeichnis
s	während der Ausführung der Datei wird die Benutzerkennung bzw. die Gruppenkennung der Datei auf die des Anwenders gesetzt. Wichtig für Programme, für deren Ausführung superuser - Rechte erforderlich sind.

Diese werden jeweils

u	user, den Eigentümer des Objektes
g	group, eine bestimmte Gruppe
o	others, allen anderen Benutzern
a	all, alle Benutzer

zugeordnet.

Mit dem Kommando **ls -l** wird eine Datei mit allen Attributen ausgegeben:

```
<Art des Objektes><Rechte user><Rechte group><Rechte other>   ......
```

Für die Art des Objektes steht eines der folgenden Zeichen:

```
d               Verzeichnis
-               normale Datei
c               Gerätedatei (zeichenorientiert)
b               Gerätedatei (blockorientiert)
p               Named Pipe (wird bei der Interprozesskommunikation zwischen zwei
                Prozessen genutzt)
l               soft- Links
```

Eine Änderung der Zugriffsrechte eines Objektes ist nur dem Eigentümer und dem *superuser* erlaubt. Das Komando dazu hat die Form

```
% chmod <ugoa> <+-=> <rwxugo> <dateiobjekt>
```

Der 1. Parameter gibt an, für wen Rechte verändert werden sollen, der 2. Parameter gibt an, wie die Rechte verändert werden sollen und der 3. Parameter gibt an, welche Rechte betroffen sind.

Beispiele

```
% chmod o+r otto.c
```

Die others erhalten zusätzlich die Leserechte der Datei otto.c.

```
% chmod a-w *
```

Allen Benutzern wird das Schreibrecht für alle Dateien entzogen (**Vorsicht als superuser**!!).

```
% chmod a=rwx kasper
```

Die Datei kasper bekommt für alle Benutzer alle Rechte.

```
% chmod o=u .profile
```

Für die Datei .profile erhalten die others die gleichen Rechte wie der user.

Eine weitere Form des chmod - Kommandos ist die folgende Variante:

```
% chmod <3 - stellige Oktalzahl> <Dateiobjekt>
```

Jede Oktalziffer definiert die Rechte für eine Benutzerart in der Reihenfolge u g o. Jede Oktalzahl kann bekanntlich durch 3 Bit dargestellt werden. In diesem Sinne stellen diese 3 Bit nacheinander das r- , w- und x- Recht dar, wobei das Recht gesetzt ist, wenn das zugehörige bit eine 1 ist.

Beispiel

```
% chmod 754 otto.c
```

user alle Rechte, group r- und x- Rechte und others nur r- Rechte.

Die Rechte von *user, group* und *others* werden hintereinander in der Reihenfolge rwx angegeben, fehlt eines dieser Rechte, so steht an der betreffende Stellen ein Minuszeichen.

```
% ls -l otto
$-rwxr-xr--  1  root   1403  Fri 11 11:11 otto
  |   |   |   |
  user group other
    7   5   4
```

Wie die Zugriffsrechte können auch der einem Objekt zugeordnete Eigentümer (*user*) oder die Gruppe (*group*) geändert werden.

```
% chown <neuer Eigentümer> <Dateiobjekt(e)>
% chgrop <neue Gruppe> <Dateiobjekt(e)>
```

Somit können also Dateien „verschenkt“ werden!

WICHTIG

Bei dem *chmod* - Kommando ist, vor allen Dingen als superuser, größte Vorsicht geboten.

Der chown - Befehl ist dem superuser vorbehalten.

In diesem Zusammenhang ist noch der folgende Befehl wichtig:

```
% umask <3 - stellige Oktalzahl>
```

Hiermit werden die durch die Oktalzahl angegebenen Bits der Zugriffsrechte auf "minus" gesetzt.

Beispiel

```
% umask 022
```

Das heißt, wird eine Datei neu angelegt, nimmt es die folgenden Zugriffswerte an: **rwxr-xr-x** (XOR).

3.6 Systemdateien

/etc/passwd

Jeder Benutzer der am System arbeitet, ist in dieser Datei vermerkt. Beim Einloggen entnimmt das System (über den getty - Prozess) aus dieser Datei die Information, ob der user überhaupt dem System bekannt ist. Danach, wenn das Passwort korrekt eingegeben wurde, startet der **getty** - Prozess eine neue Shell (oder möglicherweise eine Anwendung). Jede Zeile hat sieben Felder die durch einen Doppelpunkt getrennt sein müssen.

1 2 3 4 5 6 7

otto:3rpa9t3:530:124:Otto Lehmann, Systemprogrammierer, NST - 4256:/home/lehm:/bin/sh

```
1.          Login- Name
2.          das verschlüsselte Passwort
3.          die Benutzerkennung
```

```
4.              die Gruppennummer
5.              Information zu dem Benutzer
6.              Pfad des Login- Verzeichnis
7.              Shell oder Anwendung
```

Das Passwort- Feld kann leer sein. In diesem Fall kann sich der Benutzer ohne Passwort anmelden. Ebenso kann das Informationsfeld leer sein. Ist das letzte Feld leer, wird, je nach „UNIX“ irgendeine Shell angenommen.

Bei sogenannten „C2 - Systemen“ (orange book), wird das Passwort nicht mehr in dieser Datei abgelegt. Das Feld ist dann mit einem Asterix gekennzeichnet. In dem Verzeichnis /etc/shadow existiert dann eine Datei mit dem Passwort.

/etc/group

Alle Benutzer gehören bestimmten Gruppen an, die in dieser Datei definiert sind. Eine Zeile besteht aus 4 Feldern, welche durch einen Doppelpunkt getrennt sind.

```
   1    2 3          4
```

unix_kurs::124:otto,karl,hinz,kunz

```
1.              Name der Gruppe
2.              verschlüsseltes Passwort
3.              die Gruppennummer
4.              user- Namen, durch Kommas getrennt
```

Ein Gruppenpasswort wird nicht unterstützt, das zweite Feld ist also leer oder enthält einen Asterix.

Gehört ein user mehreren Gruppen an, so kann er mit dem Befehl

```
% newgrp <Gruppe>
```

in eine andere Gruppe wechseln.

/etc/termcap

Diese Datei enthält die Informationen die nötig sind, um verschiedene Terminals an das System anzuschließen. Gibt es Probleme mit der Darstellung von Zeichen auf einem Terminal, so muß eventuell ein neuer Eintrag hinzugeführt werden. Mittels der Variablen **term**, wird in der **.cshrc** das entsprechende Terminal eingestellt.

/etc/inittab („nur System V“)

Beim booten eines UNIX - Systems, wird in dieser Datei festgelegt, wie das System hoch gefahren wird (im weitesten Sinne mit der config.sys unter MS-DOS vergleichbar).

Beim **SunOS** unter SOLARIS 1.x durchläuft der **init** - Prozess die nachfolgenden Dateien. Ab SOLARIS 2.0 wird System V, und somit auch die Datei **inittab** unterstützt.

/etc/rc.boot, /etc/rc.local, usw.

Diese Dateien enthalten Befehle die beim Booten des Systems druchlaufen werden.

/etc/hosts

Hier sind die IP- Adressen und die Hostnamen der, über TCP/IP erreichbaren, Rechner aufgeführt.

/etc/motd (message of today)

Dies ist eine Textdatei, welche nach dem Login auf dem Bildschirm angezeigt wird.

3.7 Suchen von Dateien und Mustern in Dateien

UNIX verfügt über zwei Suchkommandos von außerordentlicher Mächtigkeit, **find** zum Suchen von Dateien und **grep** zum Suchen von Zeichenfolgen in Dateien. Beide haben eine Fülle von Möglichkeiten, von denen hier nur die wichtigsten beschrieben werden.

Dateien suchen

```
% find <Verzeichnis(se)> <Argumente>
```

```
-name <Dateobjekt>  Dateiobjekte mit den angegebenen Namen werden gesucht, bei der
                    Verwendung von Metazeichen muß das Dateiobjekt in Anführungszeichen
                    ("Dateiobjekt") gesetzt werden.
-perm <wert>        Dateiobjekte, deren Zugriffsrechte dem oktalen <wert> entsprechen
-mtime <n>          nur Dateien, die vor genau n- Tagen geändert wurden
-mtime -<n>         die innerhalb von n- Tagen geändert werden
-mtime +<n>         die vor n- Tagen geändert wurden (oder früher)
-newer Dateiname    neuer als die angegebene Datei
-size +1024c        nur Dateien, die mind. 1024Byte groß sind, werden gesucht
-type d             nur Directories
-type f             nur gewöhnliche Dateien
```

Mögliche Ausgabearten:

```
-print              Anzeige der gefundenen Dateien auf dem Bildschirm
-exec Befehl {} \;  Ausführung eines UNIX- Kommandos auf alle gefundenen Dateien. {}
                    = einsetzen aller gefundenen Dateien, ; = Endezeichen der
                    Ausführung. ACHTUNG: zwischen schließender geschweifter Klammer und
                    Backslash muß ein Blank stehen!
```

Die Suche beginnt im Startverzeichnis. Es es werden sämtliche Unter- Verzeichnisse nach Dateien mit dem angegebenen Attributen durchsucht und, wenn sie gefunden sind, wird ihr vollständiger Pfadname ausgegeben.

Beispiel

```
% find / -name xy -print
```

Aufsuchen aller Dateinamen mit dem Namen xy im gesamten System.

```
% find /etc /bin -type d -print
```

Aufsuchen aller Verzeichnisse unterhalb von /etc und /bin

```
% find /usr -size +1024c -print
```

Aufsuchen aller mindestens 1024 Bytes großen Dateien unterhalb des Verzeichnisses /usr.

```
% find /tmp -type  f -print -exec rm -i {} \;
```

Aufsuchen aller gewöhnlichen Dateien unterhalb von /tmp. Diese Dateien werden gelöscht und die Namen ausgegeben.

Suchen nach Ausdrücken oder Zeichenfolgen

```
% grep [ <option(en)> ] <Suchmuster> <Datei(en)>
```

grep durchsucht die angegebenen Datei(en) nach dem Suchmuster. Dieses kann aus Zeichenfolgen und Metazeichen wie % * [] ^ | ? 0 \ usw. bestehen. Sollen diese Zeichen nicht von der Shell ersetzt, sonder als reguläre Zeichen aufgefaßt werden, darf das Suchmuster nicht in ´...´oder in "..." eingeschlossen werden.

fgrep verarbeitet nur Zeichenketten ohne Sonderzeichen, ist dafür aber sehr schnell. Einige der Optionen gelten nur für grep, andere nur für fgrep. Ohne Optionen geben beide Versionen alle Zeilen aus, die das Suchmuster bzw. die Zeichenkette enthalten.

Für die Dateinamen können wildcards (* ? [...]) verwendet werden. Ist das Suchmuster in mehreren Dateien enthalten, wird bei der Ausgabe der Zeilen der absolute Pfadname der Datei mit angegeben. Die zu durchsuchenden Dateien können auch durch ein find - Kommando ermittelt werden. Dieses muß dann anstelle des Dateinamens angegeben werden und in `...` (*accent grave*) gesetzt werden.

-c	Es wird nur die Anzahl der Zeilen angegeben, die das Suchmuster enthalten.
-l	Die Namen der Dateien, die das Suchmuster enthalten, werden ausgegeben.
-i	Groß- und Kleinbuchstaben werden nicht unterschieden.
-n	Vor jede Zeile wird die relative Zeilen - Nr. in der Datei gesetzt.
-v	Es werden die Zeilen ausgegeben, die das Suchmuster nicht enthalten.
-s	Keine Fehlermeldung bei nicht existierenden oder nicht lesbaren Dateien (nur grep).
-x	Es werden nur Zeilen ausgegeben, die genau das Suchmuster enthalten (nur fgrep).

Beispiele

```
% fgrep main *.c
```

Von allen C - Programmen des aktuellen Verzeichnisses werden die Zeilen ausgegeben, die das Wort „main“ enthalten.

```
% grep "main()" *.c
```

Hier müssen Anführungszeichen gesetzt werden, da das Suchmuster Klammern enthält!

```
% grep include `find . -name "*.c" -print `
```

Die Anzahl der C- Programme im aktuellen Verzeichnis und in allen Unterverzeichnissen, die das Wort „include“ enthalten, wird angegeben.

```
% grep -cx "  do" *.c
```

Die Anzahl der Zeilen werden angegeben, die genau den Text „ do“ enthalten.

3.7.1 Reguläre Ausdrücke

Zusätzlich zu fest vorgegebenen Mustern "versteht" grep auch eine Mustererkennung-Sprache, genannt reguläre Ausrücke. Reguläre Ausdrücke weisen im Gegensatz zu festen Mustern zwei zusätzliche Erweiterungen auf:

Textmusterposition

und

Einzelzeichenbereich.

Folgende Zeichen haben in regulären Ausrücken eine spezielle Bedeutzng (Metazeichen):

\ ^ $. [ ] | ( ) * + ? { }

Eine regulärer Ausdruck besteht nun aus

```
eine oder mehreren Nicht-Metazeichen (konstante Muster)
eine Escape- Folge wie \t, welches einem TAB entspricht
ein gequoteter Metacharacter, wie \*, welcher einem Asterix
entspricht
^ bedeutet der Beginn einer Zeile
$ bedeutet das Ende einer Zeile
.(Punkt) bedeutet ein Einzelzeichen
[abc] bedeutet die Einzelzeichen a oder b oder c
[a-l] bedoutet die Zeichen a bis l
```

Beispiele: Folgende Datei existiert im aktuellen Verzeichnis

```
erwin   lehmann 30454   bielefeld
fritz   mueller 80100   muenchen
hans    wacker  11111   flensburg
karl    schulz  40200   duesseldorf
otto    maier   20000   hamburg
udo     amann   70345   stuttgart
werner  kohl    55234   kiel
```

Durch den Einsatz von regulären Ausdrücken, können folgende Ausgaben erzielt werden:

```
% grep ´^u´ tabelle
udo     amann   70345   stuttgart
```

```
% cat tabelle | tr -s ´ ´ ´ ´ | cut -d´ ´ -f3-4
| grep ´4.... ´
40200   duesseldorf
% grep ´^[a-f]´ tabelle
erwin   lehmann 30454   bielefeld
fritz   mueller 80100   muenchen
% grep '0\{4\}' tabelle
otto    maier   20000   hamburg
```

3.8 Zählen, Sortieren, Ausschneiden und Zeichen ersetzen

Zeilen, Worte und Zeichen zählen

```
% wc [ -lwc ] <Datei(en)>
```

Ohne Optionen werden für alle angegebenen Dateien nebeneinander die Anzahl der Zeilen, der Worte und der Zeichen (Bytes), sowie der Dateinamen ausgegeben. Dieses Kommando wird besonders oft im Zusammenhang mit Pipes gebraucht.

```
-l          Es wird nur die Anzahl der Zeilen (lines) berechnet.
-w          Es wird nur die Anzahl der Wörter berechnet.
-c          Es wird nur die Anzahl der Zeichen (characters) berechnet.
```

Beispiel

```
% wc otto.c
% 32     64      623      otto.c
```

Zeichen ersetzen

```
% tr [ <Optionen> ] <str1> <str2>
```

```
-s          Komprimiert alle Folgen von gleichen Zeichen aus str1 zu einem
            Zeichen in str2 bei der Ausgabe.
-d          Löscht alle in str1 vorkommenden Zeichen bei der Transfromation.
```

tr ließt von der Standardeingabe und kopiert dies nach einer Zeichentransformation auf die Standardausgabe. Zeichen der Eingabe, welche in der Zeichenkette str1 vorkommen, werden in die entsprechende Zeichen der Zeichenkette str2 transformiert.

Beispiel

```
% cat /etc/passwd | tr -s : ´\011´
```

Die Datei **/etc/passwd** wird auf dem Bildschirm ausgegeben. Zuvor wird durch das **tr**- Kommando, das Zeichen ":" (welches bei dieser Datei das Feldtrennzeichen ist) durch einen Tabulator ersetzt.

Dateien zeilenweise sortieren.

```
% sort [ <opotion> ] [ <sort-field>] <datei>
```

Ohne Angabe von Optionen und Feldern wird die Datei lexigraphisch sortiert auf dem Bildschirm ausgegeben. Die Datei selbst wird nicht verändert. Bei der Angabe von Positionen wird von 0 an gezählt.

-b	Führende Blanks (Leerzeichen) werden ignoriert
-f	Großbuchstaben werden wie Kleinbuchstaben sortiert.
-n	Zeilen mit führenden Ziffern werden numerisch sortiert.
-r	Rückwärtssortierung.
-u	Doppelt vorkommende Zeilen in der sortierten Datei werden gelöscht.
-M	Die ersten drei Zeichen werden als Großbuchstaben angenommen und als Monatenamen (JAN, FEB ... DEC) sortiert. Zeichenfolgen, die keinem Monatenamen entsprechen, werden vor JAN eingefügt.
-t<c>	Definition des Trenners

Beispiele

```
% cat tabelle | sort
erwin   lehmann 30454   bielefeld
fritz   mueller 80100   muenchen
hans    wacker  11111   flensburg
karl    schulz  40200   duesseldorf
otto    maier   20000   hamburg
udo     amann   70345   stuttgart
werner  kohl    55234   kiel
```

```
% cat tabelle | sort +2
hans    wacker  11111   flensburg
otto    maier   20000   hamburg
erwin   lehmann 30454   bielefeld
karl    schulz  40200   duesseldorf
werner  kohl    55234   kiel
udo     amann   70345   stuttgart
fritz   mueller 80100   muenchen
```

```
% cat tabelle | sort +3.2 -4
otto    maier   20000   hamburg
werner  kohl    55234   kiel
erwin   lehmann 30454   bielefeld
hans    wacker  11111   flensburg
udo     amann   70345   stuttgart
fritz   mueller 80100   muenchen
karl    schulz  40200   duesseldorf
```

Ausschneiden

```
% cut [ <option> ] <datei>
```

cut schreibt eine Folge von Zeichen auf den Bildschirm, wobei die Zeichen gemäß den bei cut angegebenen Optionen ausgewählt werden. **cut** interpretiert eine Zeile als eine Folge von Einzelzeichen oder Worten, wobei als Standardworttrenner ein Blank festgelegt ist.

```
-c<n>           Auswahl von Zeichen
-f<n>           Auswahl von Worten
-d<c>           Definition des Trenners
```

Wortauswahl:

```
-f1-3           Auswahl der ersten drei Worte
-f-3            dito
-f1,2,3         dito
-f2,4-5,8       Auswahl des zweiten Wortes, der Worte 4 bis 5 und des achten Wortes
-f5-            Ab dem fünften Wort werden alle Worte ausgegeben
```

Beispiel

Die Ausgabe von **date** lautet:

```
% Fri Nov 11 11:11:11 MET 1994
```

Um aus dieser Zeile den Tag, den Monat und das Jahr zu extrahieren, benutzt man **date** und **cut** in einer Pipe wie folgt:

```
% date | cut -d´ ´ -f 1,2,6
% Fri Nov 1994
```

oder

```
% date | cut -c-8,25-
% Fri Nov 1994
```

Um aus der Datei **/etc/passwd** alle Benutzernamen auslesen zu können, wäre folgendes Kommando möglich:

```
% cat /etc/passwd | cut -d´:´ -f6
```

Doppelte Zeilen eliminieren

```
% uniq [ <option> ] <datei>
```

Mit diesem Befehl lassen sich gleiche aufeinanderfolgende Zeilen löschen. Der Befehl wird normalerweise in einer Pipe verwendet, da er nur bei sortierten Dateien einen Erfolg erzielt. Bei der Angabe von Positionen wird von 0 an gezählt.

-c	Jede Zeile wird einmal ausgegeben. Vor jeder Zeile steht eine Zahl, welche angibt, wie oft die Zeile auftritt
-d	(double) von jeder mehrfach vorkommenden Zeile wird nur eine Kopie ausgegeben.
-u	(uniq) nur einfach vorkommende Zeilen werden ausgegeben.
-<n>	die ersten <n> Felder einer Zeile werden ignoriert
+<n>	die ersten <n> Zeichen einer Zeile werden ignoriert

Ohne Angabe von Optionen wird -u und -d angenommen.

Beispiel

```
% cat /etc/passwd | sort +3 -t: | cut -f2 -d: |
uniq | wc -l
```

In diesem Beispiel werden die Anzahl der Gruppen ausgegeben, die in der Datei /etc/passwd aufgeführt sind.

4 Die Shell als Benutzeroberfläche

4.1 Aufgaben einer Shell

Das Betriebssystem UNIX besteht aus zwei Teilen, dem UNIX - Kern (engl. *kernel*) und einem Kommandointerpreter, von dem man sich vorstellen kann, daß er wie eine Schale (engl. *shell*) um dem Kern herumliegt. Diese Bedieneroberfläche wird daher allgemein Shell genannt.

Im Gegensatz zu vielen anderen Betriebssystemen, ist die UNIX - Shell jedoch nicht ein fester Bestandteil des Betriebssystems, sondern leicht austauschbar. Ähnlich wie bei den Editoren, gibt es daher ganz verschiedene Shell - Versionen. Wesentlich sind folgende Shells:

sh	Die Standard - Shell, die nach einem Mitarbeiter von AT&T auch als *Bourne - Shell* bezeichnet wird und auf den meisten UNIX - Anlagen läuft.
csh	Die *C - Shell*, die aus dem Berkeley - UNIX - System stammt und in ihrer Syntax besonders an die Programmiersprache C angelehnt ist.
ksh	Die *Korn - Shell*, die gegenüber der Bourne - Shell einige Erweiterungen (history - und alias - Funktion) enthält.

Mit der Anmeldung wird für jeden Benutzer „seine“ Shell (auf der Sun defaultmäßig die **csh**) als Prozess gestartet, und der Prompt signalisiert dem Benutzer, daß das System für Eingaben bereit ist.

Jede Shell stellt einen eigenen Prozeß dar und besitzt somit alle Eigenschaften, die auch andere Prozesse haben. Jedoch hat nicht jeder Prozeß eine Shell!.

Ihre wichtigste Aufgabe ist die Interpretation und die Verarbeitung von Kommandos. Dazu laufen nacheinander folgende Schritte ab:

- Das Kommando wird analysiert. Dabei wird das erste Wort als Kommando - Name aufgefaßt. Sind weitere Worte vorhanden, welche durch ein Blank getrennt sind, so werden sie entweder als Optionen (wenn sie als erstes ein Minuszeichen haben) oder als Parameter interpretiert.

- Die Shell verfügt über eine Anzahl eigener Kommandos, die sie unmittelbar ausführen kann. Dazu gehören sowohl Kommandos zur Ablaufsteuerung bei Shell - Programmen, als auch Kommandos wie **alias**, **exit, umask** usw. Ist das eingegebene Kommando nicht mit einem dieser Shell - Kommandos identisch, wird nun ein Programm gleichen Namens gesucht. Dazu greift die Shell auf bestimmte „Suchpfade“ zurück, die in einer Shell - Variablen mit dem Namen **path** beschrieben sind. In dieser Variablen sind die Pfadnamen der Verzeichnisse, in denen die Shell nach Kommandos sucht, nacheinander - durch Blank getrennt - eingetragen. Ein Beispiel für die Variable **path** (die in der Dateie **.cshrc** definiert wird) ist:

- path = /bin /usr/bin /usr/local/bin

- Wird das Kommando gefunden, werden die Wege für Ein- und Ausgaben festgelegt. Sind im Kommando keine speziellen Angaben enthalten, wird als Standard - Eingabe die Tastatur und als Standard- Ausgabe der Bildschirm des Benutzerterminals (oder unter X - Windows das entsprechende Fenster) angenommen. Daneben hat die Shell jedoch die Fähigkeit, bestimmt

Steuerzeichen (< > >> | usw.) zu interpretieren. Damit sind Umleitungen von Ein- und Ausgabe sowie Kommandoverknüpfungen möglich.

– Neben den Steuerzeichen für die Kommandoverarbeitung, kennt die Shell auch sogen. Metazeichen, mit denen Datei- und Pfadnamen dargestellt werden können. Diese Metazeichen werden von der Shell durch die jeweiligen Namen ersetzt. Entsprechendes gilt für Sonderzeichen, die für den Inhalt von Shell- Variablen und die Behandlung von Zeichenketten maßgebend sind.

– Wenn alle eingegebenen Daten analysiert sind, startet die Shell das Kommando (bzw. das entsprechende Programm) mit Optionen und Parametern als eigenen Prozeß, ggf. als Hintergrundprozeß (bei „&“ am Zeilenende) oder übergibt das Programm zur späteren Ausführung an die *cron* - Steuerung (bei **at** - Kommandos).

Die **csh** besitzt für die Kommando - Eingabe die sogen. *history substitution*, welche vor allem bei häufigen und längeren Kommandos für den Benutzer sehr bequem ist. Dazu wird nach der ersten Benutzerabeldung eine Datei **.history** angelegt, in der alle vom Benutzer eingegebenen Kommandos gespeichert werden. Mit dem Befehl

```
% histroy
```

können die letzten Befehle ausgeben werden.

4.2 Umlenkung

4.2.1 Ein- und Ausgabeumlenkung

Die **csh** erlaubt eine Umlekung der Standard eingabe und Standardausgabe, bietet aber bei der Ausgabeumsteuerung eine Schutz gegen versehentliches Überschreiben bereits existierender Dateien. Dieser Schutz kann unterdrückt werden. Die Umlenkung erfolgen duch:

`< name`	Öffnet die Datei name und lenkt ihren Inhalt zu Standardeingabe des Kommandos um.
`<< wort`	Es wird die Eingabe der Shell (bei Skripts der nachfolgende Text) gelesen, bis eine Zeile mit wort beginnt.
`>! datei`	
`> datei`	Die Standardausgabe wird in die angegebene Datei geschrieben. Dabei wird die Datei, soweit notwendig, neu angelegt. Existiert sie bereits und ist die Shell- Variable **$noclobber** nicht definert, so wird diese Datei überschrieben. Ist die Variable **$noclobber** gesetzt, so wird von **csh** eine Fehlermeldung ausgegeben, falls die Datei bereits existiert. Das Kommando wird abgebrochen. Die Form „... >! datei“ verhindert diese Prüfung.
`>&! datei`	
`>& datei`	Mit dieser Anweisung wird die Standardfehlerausgabe auf die gleiche Datei wie die Standardausgabe umgelenkt. Existiert datei und ist **$noclobber** definiert, so wird dies als Fehler gewertet. Die Form „... >&! datei“ verhindert diese Prüfung.
`>>! datei`	
`>> datei`	Die Standardausgabe wird am Ende der angegebenen Datei angehängt. Existiert datei noch nicht und ist $noclobber definiert, so wird

	dies als Fehler gewertet. Die Form „... >>! datei" verhindert diese Prüfung.
>>&! datei	
>>& datei	Die Standardfehlerausgabe wird ebenso wie die Standardausgabe am Ende der angegebenen Datei angefügt. Existiert datei noch nicht und ist **$noclobber** definiert, so wird dies als Fehler gewertet. Die Form „... >>&! datei" verhindert diese Prüfung.

Im Gegensatz zur **sh** wird die Eingabe eines unter der **csh** gestarteten Hintergrundprozesses nicht auf **/dev/null** umgeleitet, sondern der Prozeß wird, sobald er von der Dialogstation gelesen wird, blockiert und der Benutzer darüber informiert. Er kann dann den Prozeß mit **fg** in den Vordergrund holen.

Beispiele

```
% cat otto.c > /dev/hplaserjet
```

Die Datei otto.c wird auf dem Drucker hplaserjet ausgegeben. Das ist nur möglich, wenn die entsprechenden Zugriffsrechte bestehen.

```
% ls -l > liste
```

Die Tabelle mit dem Verzeichnisinhalt wird in eine Datei „liste" geschrieben.

```
% date >> tabelle
```

An das Ende der Datei „tabelle", wird das aktuelle Systemdatum gehängt.

```
% cat text1 >> text2
```

Die Datei „text1" wird an die Datei „text2" gehängt.

Der Befehl **cat** (*concatenate* = aneinanderhängen) läßt sich aber noch vielseitiger verwenden, um z.B. eine Datei zu erstellen.

```
% cat > xdat
```

Alle hiernach folgenden Eingaben über die Tastatur werden zeilenweise in die Datei „xdat" geschrieben. Die Eingabe muß (in einer neuen Zeile) mit ^d beendet werden.

Fehlerausgaben, wie sie beim compilieren von Programmen oder bei dem **find**- Kommando auftreten, könnten mit der folgenden Anweisung umgelenkt werden.

```
% cc laufzeit.c >>& fehler
```

Bei der Verarbeitung mit Pipes spielt noch das Kommando **tee** eine wichtige Rolle.

```
% tee [ -a ] Datei
```

Alle Eingaben über die Tastatur werden zeilenweise, sowohl auf dem Bildschirm ausgegeben, als auch in die angegebene Datei geschrieben. Mit der Option -a wird der Text an bestehende Dateien angehängt.

Beispiel

```
% cat /etc/passwd | tee ergebnis
% mail egon otto karl < nachricht
```

In diesem Beispiel wird die Datei nachricht an alle aufgeführten Benutzer, als Mail, gesendet. Der Befehl mail soll, an dieser Stelle, nicht weiter erläutert werden.

4.2.2 Die Inline - Schreibweise

Die bishergenannten Umkenkungen bezogen sich auf Dateien. Die Shell kennt jedoch einen Mechanismus, der es erlaubt, Eingabezeichen als Teil des Kommanods bei einer Umlenkung zu benutzen. Dies ist die sogenannte Inline - Schreibweise (auch als here-document bezeichnet). Die Inline-Schreibweise beteht aus 3 Teilen:

1. das Kommando selbst mit der Umlenkung (<<)
2. dem Zeichenstrom
3. ein vom Benutzer festgelegtes Markierungswort (ein oder mehrere Zeichen)

Beispiel

```
% cat <<+++
Dies ist eine ganz spezielle
Form der Umlenkung
+++
```

Ausgabe

```
Dies ist eine ganz spezielle
Form der Umlenkung
```

Hinweis

Es wird solange etwas eingelesen, bis das festgelegte Markierungswort; zu beginn einer neuen Zeile, auftaucht. In diesem Fall ist das Markierungswort "+++".

4.3 csh - Shell - Variablen

Die **csh** kennt wie die **sh** Variablen vom Typ Text. Variablennamen dürfen bei der **csh** bis zu 20 Zeichen lang sein. Darüberhinaus kann sie solche Variablen jedoch, flexibler als die **sh,** als Zahlen und logische Werte behandeln. Shellvariablen können, soweit sie nicht bereits vordefiniert sind, auf drei Arten deklariert werden und einen Wert zugewiesen bekommen. Dabei ist darauf zu achten, daß im Gegensatz zur **sh,** Zwischenräume zwischen **set, setenv, @, =, name** und stehen müssen!

`set name = text` Hierdurch wird die Shellvariable name, soweit sie noch nicht existiert, deklariert und erhält als Wert die Zeichenkette text.

@ name = n_ausdruck Hierdurch wird die Shellvariable name, soweit sie noch nicht existiert deklariert, der nummerische Ausdruck ausgewertet und das Ergebnis als Wert der Variablen name zugewiesen. Zugleich wird geprüft, ob es wirklich ein numerischer Wert ist. "@" wird vielfach dort eingesetzt, wo bei der **sh expr** steht.

setenv name = text Hiermit wird die Shellvariable name, soweit sie noch nicht existiert, deklariert und erhält als Wert die Zeichenkette text. Die Variable wird dabei zugleich exportiert, d.h. ihr Gültigkeitsbereich ist global.

unset name Die angegebenen Shellvariablen werden wieder freigegeben, d.h. sie gelten als nicht mehr definiert. In name dürfen auch Metazeichen vorkommen. Das Kommando gilt dann für alle Variablen, deren Namen auf das Muster passen.

unsetenv name Der dem **unset**- Kommando entsprechende Befehl zu **setenv**. Die Definition der angegebenen Variablen wird aufgehoben.

Jeder Benutzer kann in seiner Shell Variablen mit der Anweisung set name=wert definieren. Der Wert (Inhalt) einer Variablen ist grundsätzlich eine Zeichenkette.

Beispiele

```
% set v1 = "Dies ist ein Texte\!"
% set pfad = /usr/bin
```

Enthält der Inhalt einer Variablen einen Text mit Leerzeichen, so müssen diese entweder maskiert werden, oder der Text muß durch "..." bzw. ´...´ begrenzt werden.

Der Inhalt einer Variablen wird angesprochen über den Variablen - Namen im einem vorgesetztem Dollarzeichen (%).

```
% echo $pfad
% /usr/bin
```

Soll ein Variabel als Teil einer Zeichenkette ausgegeben werden, muß der Variablen- Name in geschweifte Klammern gestetz werden.

```
% echo "Der Pfad lautet : ${pfad}"
% Der Pfad lautet : /usr/bin
```

Sonderzeichen wie <, >, |, und & müssen in Variablen maskiert werden, die Kommandoersetzung mit (`...`) (*accent grave*) ist wirksam. Metazeichen werden immer ersetzt, es sei denn, sie werden doppelt maskiert.

Beispiele: Das aktuelle Verzeichnis enthält die Dateien asc.d, asc.c, bxd.c und a.out.

Variable	Kommando	Ausgabe
x = hallo|egon	echo $x	hallo|egon
x = `ls | wc -l`	echo $x	4
x = *.[bc]	echo $x	asc.c bxd.c
x = "*.[bc]"	echo "$x"	*.[bc]
x = "Hallo *.[bc]"	echo $x	Hallo abc.c bxd.c

Mit dem Kommando **set** wird eine Liste aller Variablen ausgegeben. Mit dem Kommando **unset** können Variablen gelöscht werden.

Standard - Variablen

Eine Reihe von Variablen haben eine feste Bedeutung in der Shell, sie werden zum Teil in der **/etc/cshrc**, zum Teil in der benutzereigenen Datei **.cshrc** definiert, einige erhalten ihren Wert auch erst während des Betriebs.

Wie bei der **sh** wird der Inhalt der Variablen durch **$name** angegeben, bzw. **${name}** falls in name Sonderzeichen vorkommen. Ist der Text "$name" gemeint, so ist das Dollarzeichen zu maskieren (also etwa \$name). Der Ausdruck **$?name** liefert den Wert 1, falls die Variable **$name** deklariertist und 0, falls nicht. Neben einigen Systemvariablen sind die wichtigsten Umgebungs- und Betriebsvariablen:

$argv	In dieser Variablen befinden sich die Argumente des Prozeduraufrufes.
$cwd	Dies ist der volle Pfadname des aktuellen Verzeichnisses.
$filec	Ist die Variable definiert, wird durch betätigen der ESC - Taste ein unvollständiger Dateiname vervollständigt.
$history	Der Wert von $history legt fest, wieviel Kommandos im Historyspeicher festgehalten werden sollen.
$home	Wird **cd** ohne ein Parameter aufgerufen, so wird das in $home stehende Verzeichnis zum aktuellen Verzeichnis. Darüberhinaus ersetzt die **csh** das Metazeichen "~" in Dateinamen durch diesen Eintrag.
$ignoreeof	Dies verhindert, daß versehentlich die csh durch ein <dateiende>-Zeichen terminiert wird. Die Beendigung ist dann nur mit dem **exit**- oder **logout**- Kommando möglich.
$noclobber	Hierdurch wird verhinder, daß man versehentlich durch Ausgabeumleitung eine bereits existierende Datei überschreibt. Das Kommando wird in diesem Fall abgebrochen und die Fehlermeldung "*kommando:**file exists***" ausgegeben.
$path	Dies gibt den Suchpfad für das Starten von Programmen vor.
$prompt	Dies ist das Bereitzeichen der **csh**. Für den Superuser ist dieses Zeichen standardmäßig "#", für den Benutzer "%".
$shell	Hier steht der Pfadname der aktuellen Shell.
$verbose	Zeigt nach dem Benutzen des history- Mechanismus den Befehl an.

4.4 Kommando - Verknüpfung

4.4.1 Kommando - Verkettung

Bei der Eingabe eines Kommandos über die Tastatur, wird diese Eingabe mit der <ENTER> - Taste abgeschlossen und danach ausgeführt. Erst nach der Ausführung erscheint der Shell - Prompt als Zeichen dafür, daß neue Eingaben angenommen werden können. Es ist aber auch möglich, mehrere Kommandos in eine Zeile einzugeben, die dann nacheinander ausgeführt werden. Dazu müssen die einzelnen Kommandos durch ein Semikolon getrennt werden.

```
% cd /etc ; cat passwd
```

Die Kommandoausführung erfolgt auch hier völlig unabhängig voneinander auf der Ebene der LOGIN - Shell. Werden dagegen die Kommandos in runde Klammern gesetzt, wird für ihre Ausführung eine eigene Shell erzeugt.

```
% ( cd /etc ; cat passwd )
```

Das hat zur Folge, daß diese Klammern, insbesondere für nachfolgende Umleitungs - Anweisungen, als Einheit betrachtet, und somit z.B. die Ausgaben aller Kommandos, in den Klammern, umgeleitet werden. Zwischen den Klammern und den Kommandos muß ein "Blank" stehen.

Beispiel

```
% echo "Ich bin \c" ; whoami ; echo "in \c" ;
pwd > mein
```

Mit dieser Kommandofolge wird nur das letzte pwd in die Datei mein umgeleitet, alle anderen Ausgaben werden auf den Bildschirm geleitet. In der folgenden Form dagegen,

```
% ( echo "Ich bin \c" ; whoami ; echo "in \c" ;
pwd ; ) > mein
```

werden die in runde Klammern gesetzten Kommandos als eine Einheit betrachtet und ihre Ausgaben gemeinsam in die Datei "mein" umgeleitet.

```
% ( find /home/expert -name .cshrc -print | tee
ergebnis ; ) >&! fehler ; cat ergebnis
```

In dem letzten Beispiel wird das Kommando **tee** eingesetzt, um die Standardausgebe in die Datei ergebnis zu lenken. Die Fehlermeldungen werden jedoch mit der Standardausgabe in die Datei Fehler ausgegeben. Mit **cat** wird die Datei ergebins ausgegeben.

Hinweis

1. Zwischen den Klammern und den Kommandos muß mindestens ein Blank stehen.
2. Auch das letzte Kommando muß mit einem Semikolon abgeschlossen werden.

4.4.2 Pipeline

Unter einer *pipe* versteht man eine Kommandoverknüpfung, bei der die Ausgaben des ersten an das zweite Kommando weitergeleitet werden. Die *pipeline* darf beliebig lang sein. Jedes Kommandos reicht seine Ausgaben an das Folgende weiter. Die Eingabe für das erste und die Ausgabe für das letzte Kommando dürfen umgeleitet werden. Innerhalb der *pipeline* dürfen dagegen keine weiteren Umleitungen erfolgen. Die allgemeine Form ist :

```
% kommando1 | kommando2 | kommando3 ...
```

Beispiele

```
% who | wc -l
```

Die Anzahl der zur Zeit am System angemeldeten Benutzer wird ausgegeben.

```
% cat /etc/passwd | cut -f1 -d:
```

Alle dem System bekannten Benutzer werden auf dem Bildschirm ausgegeben.

4.5 Meta - und Sonderzeichen

Damit sind Zeichen gemeint, die im Namen eines Dateiobjekts (Datei und Verzeichnis) ein oder mehrere andere Zeichen ersetzen können. Sie werden auch als *wildcards* oder *joker* bezeichnet.

4.5.1 Metazeichen

Die **csh** verwendet den gleichen Mechanismus zur Expandierung von Dateinamen in den Parametern eines aufgerufenen Programms wie die **sh**. Dies gilt für die Metazeichen *, ? und []. Der Mechanismus der Namensexpandierung kann dabei durch die Deklaration der Shellvariablen **$noglob** unterdrückt werden. In diesem Fall werden die Metazeichen der **csh** (" * , ? , [ ] ") als normale Zeichen betrachtet. Darüberhinaus kennt die csh die Metazeichen "~" und "{....}".

*	Eine beliebige Zeichenfolge.
?	Ein einzelnes Zeichen (nicht Leerzeichen).
[...]	Eines der in den Klammern aufgeführten Zeichen (hier symbolisch durch Punkte gekennzeichnet).
~	Das Tildezeichen wird von der csh durch den Namen des home-Verzeichnisses ersetzt.
{x,y,z,...}	Hierfür werden mehrerer Wörter eingesetzt und zwar eines für jedes in den Klammern vorkommende und durch Komma getrennte Textstück. Z.B. wird hierdurch die Sequenz "erg.{c,d,e}" zu "erg.c erg.d erg.e" expandiert.

Beispiele

```
% grep „ich“ [abc]*
```

Alle Dateien, die mit a, b, oder c beginnen, werden nach dem Wort „ich“ durchsucht.

```
% ls [A-Z]* | wc -w
```

Es wird die Anzahl der Dateien im aktuellen Verzeichnis angegeben, deren Namen mit einem Großbuchstaben anfangen.

```
% cat ?a*.c > sum
```

Die Inhalte aller C - Programme, deren Namen an zweiter Stelle den Buchstaben "a" haben, werden zusammen in die Datei sum geschrieben.

Hinweis

– In den Klammern [...] dürfen keine Leerzeichen (blanks) stehen.

– Nach Ein- und Ausgabeumleitungen (<, >, >> usw.) werden Metazeichen in Dateinamen unmittelbar nach der Umleitung nicht ersetzt.

Metazeichen werden fast ausnahmslos nur in Namen von Dateiobjekten, nicht in Zeichenketten oder Variablen ersetzt!

Zu den Metazeichen gehören auch die Kurzzeichen für bestimmte Dateiverzeichnisse:

```
.           aktuelles Verzeichnis
..          das über dem aktuellen liegenden Verzeichnis (parent directory)
/           das root - Verzeichnis
```

Diese Metazeichen können allerdings nur einzeln oder in Pfadnamen benutzt werden.

Beispiel

```
% cd ../egon
```

Wechsel des aktuellen Verzeichnisses zu dem Verzeichnis „egon“, welches das gleiche *parent directory* hat wie das jetzige.

4.5.2 Sonderzeichen

Von den Metazeichen sind die Sonderzeichen zu unterscheiden. Das sind Zeichen, durch welche die Shell, nach besonderen Regeln, Kommandos interpretiert.

Dazu gehören die Zeichen : **< > | " ´ ` ; & $**

Neben den schon behandelten Sonderzeichen für die Ein-/ Ausgabeumleitung (< , > , >>) für die *pipe* (|), für Kommandoverkettung (;) und für Hintergrundprozesse (&), spielen die übrigen Sonderzeichen vor allem eine Rolle in Zeichenketten.

```
`Kommando`
```

Die Shell interpretiert den in *accent grave* gesetzten Text als Kommando und führt diesen aus.

```
% more `ls *.c`
```

Alle Dateien die auf .c enden, werden durch den Befehl **more** ausgegeben.

Sowohl Meta- als auch Sonderzeichen können mit dem *backslash* - Zeichen "\" maskiert (ungültig gemacht) und dadurch als Text interpretiert werden.

Beispiel

```
% ls a*
```

Listet alle Dateiobjekte auf, die mit "a" beginnen.

```
% ls a\*
```

Listet nur die Datei "a*" auf, wenn sie existiert.

Unter bestimmten Voraussetzungen ist die Verwendung von Steuerzeichen in Zeichenketten erlaubt. Sie werden ebenfalls durch das Maskierungszeichen "\" von anderen Textzeichen unterschieden. Solche Steuerzeichen sind:

\b	backspace (letztes Zeichen löschen)
\c	kein Zeilenvorschub (cr / lf) (Nur am Textende zulässig!!)
\f	lf ohne cr
\n	Neue Zeile (cr / lf) innerhalb eines Textes.
\r	Nur cr (Folgender Text überschreibt die gleiche Zeile.)
\t	Tabulator (Voreinstellung 8 Spalten)
\0ooo	Die 1-3 - stellige Oktalzahl "ooo" mit führender 0 wird als ASCII-Code aufgefaßt und das zugehörige Zeichen ausgegeben.

Wo immer Zeichenketten auftreten (in Kommandos, in Ausgabeanweisungen usw.), können sie entweder ohne Begrenzer oder eingeschlossen in Hochkomma (´´) oder in Anführungszeichen ("") dargestellt werden, wobei die Shell diese Darstellungsarten unterschiedlich interpretiert.

text	alle Meta - und Sonderzeichen werden ersetzt, jedoch keine Steuerzeichen.
´text´	(*accent aigeu*) Nur die Sonderzeichen *accent grave* (`) und $, sowie die Steuerzeichen werden ersetzt.
"text"	Steuerzeichen werden ersetzt, jedoch keine Meta- und Sonderzeichen.

Die Maskierung von Sonderzeichen mit dem Backslash "\" gilt auch für das Leerzeichen (*blank*) und den durch die ENTER - Taste bewirkten Zeilenvorschub. Natürlich wirkt eine solche Maskierung auch bei der Eingabe von Kommandos über die Tastatur.

Das kann z.B. genutzt werden, wenn längere Kommandos über mehrere Zeilen geschrieben werden sollen. Dazu wird vor Betätigung der ENTER - Taste der Backslash eingegeben. Dann erscheint in der folgenden Bildschirmzeile nicht das übliche Bereitzeichen der Shell, sondern das Fortsetzungszeichen (voreingestellt als ">"). Die weiteren Eingaben werden von der Shell als Teil der vorhergehenden Zeile aufgefaßt.

Zeichenfolgen können auf dem Bildschirm ausgegeben werden mit dem Kommando

```
% echo text
```

Beispiele: Das aktuelle Verzeichnis enthält die Dateien asc.d, asc.c, bxd.c und a.out.

Kommando	Ausgabe
echo Dateien: \n a* echo "Dateien: \n a*" echo hallo | wc -c echo "hallo | wc -c" echo *.[cd] um `date ´+%T´` echo "*.? um `date ´+%T´`" echo ´*.? um `date ´+%T´`´	Dateien n asc.d asc.c a.out Dateien: a* 6 hallo | wc -c asc.d asc.c bxd.c um 11:11:11 *.? um 11:11:11 *.? um `date +%T`

4.6 Kommandos zur Benutzerumgebung

Hier sollen einige Kommandos vorgestellt werden, mit denen einem Benutzer Informationen über sich selbst, seiner Arbeitsumgebung und teilweise über andere Benutzer erhalten kann.

% id

Ausgabe von Benutzernummer, Benutzernamen und Gruppennummer sowie Gruppenname.

% tty

Ausgabe des Pfadnamens des Arbeitsterminals

% logname

Ausgabe des Login - Namen.

% finger [<user>]

Ausgabe von Informationen über alle doer bestimmte Benutzer. Der Inhalt der Textdatei **.plan** im Heimatverzeichnis des angegebenen Benutzers wird angezeigt. Hier kann ein Benutzer Informationen über sich selbst eintragen (Kenntnisse, Telefonnummern, warum er am System arbeitet, etc.), die von anderen Benutzern abgefragt werden können. Außerdem erfolgt die Ausgabe des Info - Feldes der Datei **/etc/passwd**.

% date

Ausgabe des aktuellen Datums und der Uhrzeit.

% hostname

Ausgabe des Knotennamens (siehle **/etc/hosts**, IP-Adressen und Hostnamen).

% env

Gibt die als global definierten Variablen aus. Beispiel:

```
HOME=/home/sb43
SHELL=/bin/csh
TERM=vt100
USER=sb43
PATH=.:/usr/5bin:/usr/ucb:/bin:/usr/bin:/etc:/usr/etc:/usr/bin/X11:/iw
```

```
s/ipc:/iws/usm:/iws/dpn/macros/user:/iws/dpn/macros/bnr:/iws/sur
LOGNAME=sb43
PWD=/home/sb43/seminar
CWD=/home/sb43
HOST=LAB2
```

5 Shellprozeduren

Wie bei Batch - Dateien unter MS- DOS, ist es auch unter UNIX möglich, Kommandofolgen in einer Datei zu sichern (der Dateiname ist hier beliebig!). Zur Ausführung dieser Kommandodatei wird aber unter UNIX eine neue Shell erzeugt, die die einzelnen Zeichen der Datei interpretiert und ausführt. Das erfolgt bei der Bourne - Shell mit dem Kommando

```
% sh <Dateiname>
```

oder bei der C - Shell mit dem Kommando

```
% csh <Dateiname>
```

Alternativ (und meist zweckmäßiger) können Ausführungsrechte für die Kommandodatei mit der Anweisung

```
% chmod u+x <Dateiname>
```

gegeben werden, sodaß die Datei mit ihrem Namen gestartet werden kann. In diesem Fall erfolgt die Ausführung ebenfalls unter einer neuen Shell, hier vom gleichen Type wie die LOGIN- Shell.

Wichtig:

In der ersten Zeile muß das erste Zeichen ein Doppelkreuz "#" sein (z.B. "#!/bin/csh") !

Beispiel: Die Datei demo enthält folgende Kommandos:

```
#!/bin/csh
/usr/5bin/echo "Programm : $0 , Prozess-Nr.: $$ \n"
ps -j
```

Der Datei demo wird mit dem folgenden Befehl das Ausführungsrecht gegeben,

```
% chmod u+x demo
```

und danach wie folgt aufgerufen:

```
% demo
Programm : demo , Prozess-Nr.: 1883

 PPID
  PGID   SID TT TPGID  STAT   UID  TIME COMMAND
 1772  1773  1773  1772 p0  1883    SE   102  0:00 -csh (csh)
 1773  1883  1883  1772 p0  1883    SE   102  0:00 /bin/csh demo
 1883  1889  1883  1772 p0  1883    RE   102  0:00 ps -jecho
```

Hier ist deutlich zu erkennen, daß für das Programm demo eine neue Shell erzeugt wurde, die einen Sohnprozeß der LOGIN - Shell darstellt. Diese Shell erzeugt ihrerseits neue Sohnprozesse bei der Bearbeitung der Kommandos in der Datei.

*Unter der **sh** muß nicht immer eine eigene Shell erzeugt werden. Wenn das im Beispiel angegebene Shell - Skript mit*

```
$ . demo
```

gestartet wird, werden die Kommandos dieser Datei in der LOGIN - Shell verarbeitet. Bei dem Prozeß - Listing würde die mittlere Zeile entfallen.

Wichtiger Hinweis

Niemals darf eine ausführbare Datei *test* heißen, da dies ein Kommando der Shell ist.

5.1 Positionsparameter

Genauso wie einzelne Kommandos können auch Shellprozeduren (Kommandofolgen) mit Argumenten aufgerufen werden.

Die Argumente werden von der Shell in den Positionsparametern $1, $2, $3, ... abgelegt.

```
$0              Name der Kommandoprozedur
$1              Text des 1. Parameters
...             ...
$n              Text des n. Parameters
$*              Text aller Paramter
$#argv          Anzahl der Parameter
$$              Prozeßnummer der Shellprozedur
$<              einlesen von der Standardeingabe (z.B. zum Einlesen in einem C-
                Shellskript)
```

Beispiel

```
$ demo /etc /etc/passwd
```

$0 = demo $1 = /etc $2 = /etc/passwd

5.2 Ablaufsteuerungen

Die Steuerung des Ablaufs von Shell - Programmen mit Shell- internen Kommandos erweitert die Anwendungsmöglichkeiten solcher Programme erheblich. Die Shell verfügt über nahezu alle Steuerkommandos, die auch in höherenen Programmiersprachen verfügbar sind, teilweise allerdings mit etwas anderer Syntax (auch zwischen den einzelnen Shells).

5.2.1 Ausdrücke und Operatoren

Einige der **csh** Kommandos, wie z.B. **if, set** und **while**, akzeptieren Ausdrücke, für die weitere Ausführung. Der Ausdruck liefert immer dann der EXIT- Status 0, wenn der Ausdruck den Wert *true* hat, andernfalls den EXIT- Status 1. Der <Ausdruck> kann sich auf Dateien, Zeichenketten oder numerischen Größen beziehen. Die wichtigsten Formen sind:

```
-r <Name>       true, wenn die Dateiobjekt vorhanden ist und gelesen werden kann
-w <Name>       true, wenn die Dateiobjekt vorhanden ist und beschrieben werden kann
-x <Name>       true, wenn die Dateiobjekt vorhanden ist und ausgeführt werden kann
-f <Name>       true, wenn es sich um eine normale Datei handelt
```

```
-d <Name>       true, wenn es sich um ein Verzeichnis handelt
-z <Name>       true, wenn die Datei die Länge 0 hat
-o <Name>       true, wenn die Datei dem User gehört
```

Alle numerischen Variablen können durch folgende Operatoren miteinander verknüpft werden:

```
(...)           Gruppierung
~               Einer Komplement
!               Logische Negierung
* / %          Multiplikation, Division, Modulo
+ -             Addieren, Subtrahieren
<<, >>          Shift nach links, shift nach rechts
==, !=          Zeichenketten vergleichen (gleich, ungleich)
=~, !~          Zeichenketten vergleichen (gleich, ungleich), wobei auf der rechten
                Seite Metazeichen ersetzt werden.
< > <= >=       Kleiner, größer, kleiner gleich und größer gleich
&               Bitweise UND
|               Bitweise ODER
&&              Logisches UND
||              Logisches ODER
```

Beispiel

```
#!/bin/csh
if ( -d $1 ) then
ls $1 | wc -l
else if ( -f $1 ) then
more $1
else
echo "Fehler\!"
endif
```

Das Beispiel - Skript stellt fest, ob ein als Parameter übergebener Name ein Verzeichnis oder eine Datei darstellt und führt entsprechende Kommandos aus.

5.2.2 Die if - Abfrage

Eine **if**- Anweisung beginnt immer mit einem **if** und endet bei einem **endif**. Dazwischen liegen optional ein oder mehrere **else-if**- Zweige und ein **else**- Zweig.

```
if ( expr )  then
  <kommando(liste)>
[ else if ( expr2 ) then
  <kommando(liste)> ]
[ else ]
endif
```

Wenn der Ausdruck nach dem if den EXIT- Status 0 hat (logisches *true*) dann werden die Kommandos nach dem *then* ausgeführt, andernfalls die nach dem *else* (soweit vorhanden).

Beispiel

```
#!/bin/csh
echo "Gib die Zahl z1: \c"
set z1 = $<
echo "Gib die Zahl z2: \c"
set z2 =  $<
if ( $z1 == $z2 ) then
echo "z1 = z2"
else if ( $z1 <= $z2 ) then
echo "z1 < z2"
else echo "z1 > z2"
endif
```

Bei diesem Programm erfolgt eine Fehlermeldung, wenn eine, der den read- Anweisungen eingegebene Zeichenfolge, keine ganze Zahl darstellt.

5.2.3 Die while- Schleifen

Die Syntax lautet:

```
while ( expr )
  <Kommandoliste>
end
```

Wie bei der **if**- Abfrage, ist auch bei dieser Schleife der Ausdruck nach der **while**- Anweisung maßgebend. Die Schleife wird ausgeführt, solange die Anweisung den EXIT- Status 1 hat.

Wie bei anderen Programmiersprachen, kann die Schleifenausführung auch bei Shell- Programmen durch Kommandos innerhalb der Schleife beeinflußt werden:

continue	Rücksprung zum Schleifenanfang, Prüfung der Schleifenbedingung.
break	Schleifenabbruch, Fortsetzung des Programms.

Bei geschachtelten Schleifen können diese Anweisungen noch mit einer ganzen Zahl versehen werden, die bei break angibt, über wieviele Schachtelungen die Anweisung ausgeführt werden soll.

Beispiel

```
#!/bin/csh
set z = 0
while ( 1 != $z )
  echo "Gib eine Zahle ein: \c"
  set z = $<
end
```

Mit diesem Shell- Programm sollen solange Zahlenwerte eingelesen werden, bis eine "1" eingeben worden ist.

5.2.4 Die foreach - Schleife

Diese Schleife entspricht der **for ...in**- Schleife in der **sh**.

```
foreach name ( wortliste )
  <Kommando(liste)>
end
```

Die Kommando(liste) bilden den Schleifenrumpf, der durch ein **end** begrenzt wird. Die Variable **name** nimmt zunächst den ersten Wert aus der **wortliste** an. Mit diesem Wert werden die Kommandos des Schleifenrumpfes abgearbeitet. Anschließend nimmt **name** den nächsten Wert aus der **wortliste** an, und der Rumpf wird erneut ausgeführt usw. Dieser Vorgang wiederholt sich, bis **name** alle Werte aus **wortliste** angenommen hat.

Beispiel

```
#!/bin/csh
foreach x ( `ls *.c` )
  echo $x
end
```

Hier werden also alle Name der C- Programme, des aktuellen Verzeichnisses, untereinander aufgelistet.

5.2.5 Die switch - Anweisung

Wie bei anderen Programmiersprachen, dient diese Anweisung der Mehrfachentscheidung. Sie hat die folgende Syntax:

```
switch ( str )
  case label1:
    <Kommandos>
    breaksw
  case label2:
    <Kommandos>
    breaksw
  default:
    <Kommanods>
    breaksw
endsw
```

Hier wird <*str*> mit dem nachfolgenden Mustern (Zeichenfolgen) verglichen. Stimmt eines dieser Muster mit <*str*> überein, werden die nachfolgenden Kommandos bis zum **breaksw** ausgeführt.

Fehlt das **breaksw**, werden alle weiteren Kommandos ausgeführt. Stimmen mehrere Muster mit <*string*> überein, werden nur die Kommandos nach dem ersten Auftreten dieses Muster ausgeführt. Stimmt kein Muster mit <*string*> überein, werden - wenn vorhanden - die Kommandos nach dem **default** ausgeführt. In den Mustern können die Sonderzeichen *, ?, und [ ] sowie $ zur Variablenersetzung benutzt werden.

5.2.6 Die repeat - Schleife

```
% repeat <Anzahl> <Kommando>
```

Diese Funktion führt das Kommando so oft, wie in Anzahl beschrieben, aus.

5.3 Funktionen mit dem alias - Kommando

In der **csh**- Shell können neue benutzereigene Kommandos auch über eine Anweisung der Form

```
% alias <Name> <Kommando(liste)>
```

definiert werden. So kann z.B. mit dem Kommando

```
% alias ls ´ls -F´
```

das **ls**- Kommando so modifiziert werden, daß immer eine Kennzeichnung der Dateiobjekte erfolgt. Normalerweise werden die **alias**- Kommandos in der Datei **.cshrc** geschrieben. Wird der Befehl ohne weitere Angeben aufgerufen, so werden alle **alias-** Definitionen angezeigt.

Um einen durch **alias** wieder zu löschen wird folgender Befehl benutzt:

```
% unalias <Kommando>
```

6 Prozesse

6.1 Definitionen

Ein Programm kann definiert werden als eine geordnete Folge von Anweisungen an den Prozessor. Die vom Betriebssystem dem Benutzer zur Verfügung gestelltes Programme werden meist als Kommandos bezeichnet. Insofern besteht unter UNIX kein wesentlicher Unterschied zwischen einem Kommando und einem Programm - Aufruf.

Ein gestartetes, d.h. ein sich in der Ausführung befindliches Programm, wird als **Prozeß** bezeichnet. Da in der Regel bei den meisten UNIX - Systemen nur ein Prozessor vorhanden ist, kann zu einem gegebenen Zeitpunkt auch nur ein Prozeß aktiv sein. Dennoch ist das Betriebssystem UNIX sowohl multiuser - fähig als auch multitasking - fähig.

Multiuser - Betrieb

Mehrere Benutzer sind gleichzeitig im Betreibssystem angemeldet und können Kommandos eingeben, bzw. Programme starten. Von gelegentlichen kurzen Verzögerungen bei der Kommandoausführung abgesehen, hat jeder Benutzer den Eindruck, der Prozessor stände nur ihm zur Verfügung.

Multitasking - Betrieb

Jeder Benutzer kann mehrere Programme starten. Die so erzeugten Prozesse ("*tasks*") werden entsprechend ihrer jeweiligen Prioritäten nacheinander vom Prozessor bearbeitet. In den Zwischenzeiten befinden sie sich in einem Wartezustand.

Sowohl für den Multi - User - als auch für den Multi - Tasking - Betrieb ist demnach eine mehr oder weniger aufwendige Prozessverwaltung notwendig. Dazu besitzt jeder Prozeß neben dem eigentlichen Programmtext noch eine Prozeß - Umgebung (*environment*), in der u.a. die Speicherbelegung, Registerinhalte und die Prozeß - Kenndaten registriert sind. Die Anzahl der verwalteten Prozesse (aktiv oder wartend) ist, je nach UNIX- System, begrenzt.

Die Priorität der wartenden Prozesse werden von einem Systemprozeß mit Namen **sched** (Scheduler) überwacht, der zugleich die Aktivierung, Terminierung und das zeitweilige Verdrängen eines Prozesse in den Wartezustand steuert. In regelmäßigen Abständen werden die Prioritäten aller wartenden Prozesse nach einem Algorithmus neu berechnet, der neben einer dem Prozeß eigenen Anfangspriorität (die auch als *nice* - Prioriät bezeichnet wird), die im letzten Zeitintervall verbrauchte CPU - Zeit, die Größe des Prozesses und die bisherige Wartezeit berücksichtigt.

Bei dem als *swapping* bezeichneten Verfahren wird der gesamte Prozess mit seiner Umgebung auf einen dafür reservierten Bereich des Externspeichers (System - Platte) kopiert, von dem aus er erneut in den Arbeitsspeicher geholt wird, sobald seine Priorität wieder entsprechend hoch ist.

Bei einem virtuellen System dagegen, wird das Programm in kleine Stücke (Programm - Seiten / *pages*) von 2-4 kByte unterteilt, die bei Bedarf in einen dafür reservierten Pagingbereich auf der Platte ausgelagert werden.

Jeder Prozeß erhält eine Prozeß - Nr., die als **PID** (*process identification*) bezeichnet wird. Wenn ein Prozeß beendet ist, wird seine PID nicht wieder vergeben, die PID- Werte wachsen also während des Betriebs laufend an, bis zu einem Maximalwert und beginnen erst dann wieder mit 0. Einige Prozesse laufen permanent durch (solange das System selbst aktiv ist).

Ein Prozeß kann selbst wieder einen Prozeß erzeugen, den man als Sohn- Prozeß bezeichnet (engl. *child*). Den erzeugenden Prozeß nennt man entsprechend den Vater- Prozeß (engl. *parent*). Die Prozeß-Nr. des Vaterprozesses wird auch als PPID (*parent- process- identification*) bezeichnet. Nahezu alle Systemprozesse haben die PPID 1, d.h. sie sind „Söhne“ des /**etc/init**, dem zweiten Prozess nach **sched**.

Mit der Anmeldung (dem LOGIN) eines Benutzers wird ein eigener Prozeß gestartet, der erst mit der Abmeldung (mit **^d** oder **exit**) beendet wird. Alle Kommandos des Benutzers, und die von ihm gestarteten Programme, sind Sohnprozesse dieses LOGIN- Prozesses, der auch als *LOGIN- Shell* bezeichnet wird.

6.2 Prozeß- Informationen

Um sich Informationen über die laufenden Prozesse zu holen, wird das folgende Kommando benutzt:

```
% ps [ <Optionen> ]
```

Ohne Optionen werden alle Prozesse des Benutzers mit Angabe der Prozeß-Nr. (PID), der Terminal-Leitung (TTY), der vom Prozeß bisher verbrauchten CPU- Zeit (in Sek.) und dem Namen des Programms bzw. des Kommandos aufgelistet.

```
-e          Gibt die Umgebung der Prozesse aus.
-a          Alle Terminal- Prozesse, jedoch ohne LOGIN- Prozeß.
-l          Liste mit zusätzlichen Infromationen zu den Prozessen (siehe man)
-u          user Ausgabe
```

Beispiel

```
% ps -l
      F UID   PID  PPID CP PRI NI  SZ  RSS WCHAN    STAT TT  TIME COMMAND
20408201 102   389   388  0  15  0  68  240 kernelma S    p2  0:00 -csh (csh)
20000001 102   412   389 20  30  0 216  452          R    p2  0:00 ps -l
```

6.3 Prozeß- Kommandos

Prozeß abbrechen.

Um einen laufenden Prozeß zu beenden, muß das folgende Kommando benutzt werden:

```
% kill [ -<signal> ] <PID(´s)>
```

Das Kommando sendet ein Unterbrechungssignal an die Prozesse, deren PID´s angegeben wurden. Da einige Prozesse dieses Signal abfangen, ist es zweckmäßig, mit der Option -9 das Killsignal Nr. 9 (KILL) zu verwenden, welches nicht abgefangen werden kann.

Mit dem **kill**- Kommando kann ein Benutzer nur Prozesse abbrechen, die unter seinem Namen laufen. Wird die PID 389 angegeben, so werden sämtliche Prozesse des Benutzers, also auch der LOGIN- Prozeß abgebrochen, d.h., das Kommando

```
% kill -9 389
```

führt zur Abmeldung des Benutzers vom System (LOGOUT). Der Befehl

```
% kill -l
```

listet alle möglichen Signale auf, die durch die Signalnummer (oder die Bezeichnung) an die Prozesse gesendet werden können.

Hintergrundprozesse

```
% <Kommando> &
```

Während der Ausführung von Kommandos, die vom Benutzer über die Tastatur eines Terminals (also im Dialog mit seinem LOGIN- Prozeß) gegeben werden, ist das Terminal für weitere Eingaben gesperrt. Das kann bei zeitaufwendigen Programmen bzw. Kommandos lästig sein. In solchen Fällen ist es zweckmäßig, den Prozeß im Hintergrund laufen zu lassen. Das erfolgt durch Anhängen des Zeichens "**&**" an das Kommando. Danach wird auf dem Bildschirm die lfd. Nummer für den Hintergrundprozeß und seine PID angegeben. Anschließend ist das Terminal frei für weitere Eingaben (Kommandos).

Hintergrundprozesse können nicht mit der Tastenkombination **^c** abgebrochen werden, sondern nur über das **kill**- Kommando. Im Übrigen werden sie genauso behandelt wie normale (Vordergund-) Prozesse; sie unterliegen insbesondere auch den gleichen Prioritätenregeln.

```
% ( find . -name .exrc -print >&! erg ) &
[1] 23631
```

Werden die Ausgaben des Kommandos nicht umgeleitet, so werden sie in die Standardausgabe geschrieben.

Abbruchsignal ignorieren

```
% nohup <Kommando>
```

Mit der Abmeldung des Benutzers vom System (LOGOUT) wird sein LOGIN-Prozeß und alle Sohnprozesse beendet, also auch noch laufende Hintergrundprozesse. Das kann mit dem Zusatz **nohup** zum Kommando verhindert werden. Durch diesen Zusatz werden die beiden Signale HUP (*hangup*) und QUIT (*quit*), die zum Abbruch des Prozesses führen, ignoriert. Diese Signale werden in der Regel mit der Tastenkombination **^d**, dem kill Kommando (ohne den Zusatz -9) und vor allem die Unterbrechung der Terminalleitung, die z.B. auch durch die Abmeldung mit **^d** oder **exit** bewirkt wird, erzeugt

Wird ein Hintergrundprozeß mit Hilfe des **nohup** auch nach der Benutzerabmeldung noch weiterausgeführt, sind Ausgaben auf dem Bildschirm natürlich nicht mehr möglich. Sofern das Kommando nicht schon vom aufrufenden Benutzer mit einer Ausgabeumleitung versehen wurde, werden Ausgaben in einer Datei **$home/nohup.out** geschrieben. Die Standard- Fehlerausgabe wird auf die Standard- Ausgabe umgelenkt.

Prioritäten von Prozessen

```
% nice [ -<Nummer> ] <Kommando>
```

Weiter oben wurde die Priorität von Prozessen beschrieben. Mit dem **nice-** Kommando kann ein neu zu startender Prozeß mit einer niedrigeren Priorität, als normalerweise Eingestellt (bourne-Shell 10 und C- Shell 4), gestarte werden. Die Priorität kann zwischen 0 und 127 liegen, wobei 0 die höchste Priorität ist. Nur der Superuser kann Prozessen eine höhere Priorität zuweisen.

Prozessausführung zu einem späteren Zeitpunkt

```
% at [ <Optionen> ] <Zeit> <Skript(Kommando)>
```

Die auf diese Anweisung folgenden Kommandos werden zu der unter <Zeit> angegebenen Uhrzeit gestartet. Die Kommandoeingabe wird in einer neuen Zeile beginnend mit dem *eof-* Kommando "^**d**" abgeschossen.

Ist zum Zeitpunkt der Kommandoausführung die Standard- Ausgabe nicht bereit, werden alle Ausgaben des Prozesses als *mail* an den Benutzer geschickt. Jeder **at**- Auftrag erhält eine sogenannte „Job- Nr.“, die dem Benutzer mit der vorgesehenen Ausführungszeit auf dem Bildschirm mitgeteilt wird.

6.4 Signale

Bei besonderen Ereignissen sendet das Betriebssystem Signale an einen Prozeß, die je nach Anlaß zu einer Fehlermeldung oder zum Abbruch des Prozesses führen. Solche Signale wurden bereits im vorhergehnden Abschnitt im Zusammenhang mit dem **kill**- Kommando und Hintergrundprozessen erwähnt. Von den insgesamt 20 verschiedenen Signalen, sind in der folgenden Tabelle die in der Praxis wichtigsten aufgeführt:

Nr	Name	Anlaß
1	SIGHUP	Leitungsunterbrechung, LOGOUT
2	SIGINT	Interrupt, Taste <Entf> / ^C
3	SIGQUIT	quit - Signal, Abbruch / ^d
4	SIGILL	nicht ausführbare Anweisung
8	SIGFPE	Gleitkomma- Fehler, Division durch 0
9	SIGKILL	kill- Kommando, kann nicht abgefangen werden
11	SIGSEGV	Segmentverletzung, Fehler beim Speicherzugriff
15	SIGTERM	Software- Abbruch, Voreinstellung

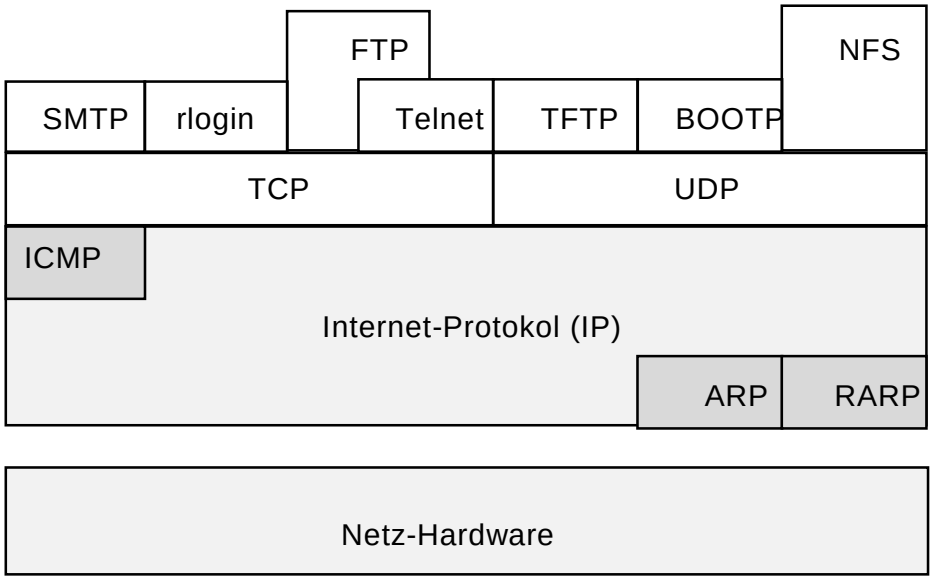
7 Anwendungen und Tools

7.1 Netzwerk TCP/IP

Mit der Einführung von UNIX 4.2BSD durch Berkeley Software Distribution im September 1983, erlangte eine umfassende Palette von Kommunkationsprotokollen namens TCP/IP (Transmission Control Protocol / Internet Protocol), einen erheblichen größeren Verbreitungs- und Bekanntheitsgrad als zuvor. Dies war kein Zufall, weil die US-Regierung diesen Teil von UNIX 4.2BSD finanzierte. Die TCP/IP-Protokolle basieren auf Standards, die für die US-Regierung und die Forschungszentern der USA entwickelt wurde.

In den Vergangenen Jahren sind die Fähigkeiten des TCP/IP weit über die USA hinaus bekannt geworden. Durch die Spezifkation für offene Netze ONC (Open Network Computering) der Firma Sun Microsystems, auch unter NFS (Network File System) bekannt, erhielt das TCP/IP-Protokoll Ende der 80´ziger Jahre einen weiteren Anschub.

Dieses Protokoll ist nicht auf das UNIX-System beschränkt, sondern es zeichnet sich gerade dadurch aus, daß auf allen gängigen Computer-Systemen ein "TCP/IP-Plattform" erhältlich ist.



ARP	Adress Resolution Protocol
RARP	Reverse Adress Resolution Protocol
ICMP	Internet Control Message Protocol
SMTP	Simple Mail Transfer Protocol
rlogin	remote login
FTP	File Transfer Protocol
Telnet	remote terminal login
TFTP	Trivial File Transfer Protocol
BOOTP	Boot Protocol
NFS	Network File System
UDP	User Datagram Protocol
TCP	Transmission Control Protocol

Abbildung 8 Die Architektur von TCP/IP aus [TCP/IP]

Die bekannteren Protokolle der Anwendungschicht sind FTP, Telnet und SMTP. NFS ist ein Dienst zur gemeinsamen Nutzung von Datei- und Festplattensystemen. BOOTP stellt die Basis für festplattenlose Arbeitsplatzrechner dar. An dieser Stelle sollte noch SNMP (Simple Network Management Protocol) aufgeführt werden, ein Internet-Standard für die Netzwerkverwaltung.

7.1.1 telnet

telnet ist eine Application, die den Fernzugriff vom eigenen Computer auf andere, sich im Netzwerk befindliche Computersysteme ermöglicht. Im Gegesatz zu **rlogin**, weches nur das Einloggen auf sich im Netz befindlichen UNIX-Systemen ermöglicht, kann mit telnet von verschiedenen Systemen gearbeitet werden. Nach dem Starten der telnet-Software und der Eingabe des **hostname** (oder im Internet der IP-Adresse "Domain-Name"), des gewünschten Zielsystems, baut telnet unmittelbar die gewünschte Verbindung auf. Das Zielsystem wird dabei allgemein als Server oder als Host (Gastgeber) bezeichnet, der auf den Server zugreifende Computer als Client. Da alle telnet-Versionen, gleichgültig ob sie für MS-DOS-basierende PC´s, für Appel oder UNIX-Workstations erstellt wurden, sich nach außen hin gleich verhalten, kann telnet Hardwareübergreifend eingesetzt werden. Das heißt, ein UNIX-Client kann genauso auf einem PC-Server zugreifen wie auf einem Appel-Client und umgekehrt. Leistungsfähige Multiuser-Systeme wie UNIX-Workstations sind, wenn sie als Server fungieren, in der Lage, eine Vielzahl von sogenannten telnet-Sessions gleichzeitig zu bedienen. Aus diesem Grund sind Computersysteme, die per Definition primär als Server fungieren, in der Regel UNIX-Workstations.

Gestartet wird telnet mit dem Befehl:

```
$ telnet [ <hostname> ]
```

Beispiel

```
privat3:/home/sb43 % telnet privat4
Trying 193.174.94.4 ...
Connected to privat4.
Escape character is '^]'.


SunOS UNIX (privat4)



login: sb43
Password:
Last login: Wed Nov  9 10:15:14 from gateway
SunOS Release 4.1.3 (GENERIC) #1: Fri Apr 29 10:10:39 MET DST 1994
privat4:/home/sb43 %
privat4:/home/sb43 % exit
```

7.1.2 FTP

Während telnet nach dem Aufbau der Verbindung zwischen Client und Server lediglich das interaktive Ausführen von Programmen auf dem Zielsystem ermöglicht, können mit dem File Transfer Protocol Daten zwischen den Systemen kopiert werden. Man ist also in der Lage, Text,

Bild, Ton, Video oder Programmdateien von einem Fremdsystem auf den eigenen lokalen Computer zu übertragen und umgekehrt. Der komplette Vorgang der Dateiübertragung wird dabei von der Client-Seite gesteuert. Voraussetzung für eine erfolgreiche Übertragung, ist eine Zugangsberechtigung für das Zielsystem, die im Rahmen des Verbindungsaufbaues mit FTP mittles User-Identifikation und Paßwort überprüft wird.

Gestartet wird das FTP mit den Befehl:

```
$ ftp [ <hostname> ]
```

Typische FTP-Benutzerbefehle sind:

user	Einen neuen Benutzer anmelden
ls	Den Inhalt eines Verzeichnisses anzeigen
pwd	Das aktuelle Verzeichniss ausgeben
cd	Das Verzeichnis wechseln
lcd	Lokales Verzeichnis wechseln
get	Eine Datei empfangen
put	Eine Datei senden
mget	Mehrere Dateien empfangen
mput	Mehrere Dateien senden
send	Eine Datei senden
receive	Eine Datei empfangen
binary	In den Binärmodus umschalten
text	In den Textmodus umschalten
?	Hilfe
!	Zum lokalen Betriebssystem umschalten, ohne FTP zu beenden
open	Eine Verbindung aufbauen
quit	Eine Verbindung schließen
status	Den Status einer Verbindung anzeigen
exit	ftp beenden

7.2 Datensicherung

Um Dateien auf ein Magnetband oder andere Dateinträger sichern zu können, kann der Befehl:

```
% tar <optionen> <name> <Dateiliste>
```

-c	Erzeugt eine neue Datei.
-r	Die genannten Dateien werden an das Ende der angegebenen Datei angeängt.
-t	Listet den Inhalt der angegebenen Datei auf.
-u	Die Dateien werden nur dann in die angegebene Datei geschrieben, wenn sie noch nicht vorhanden sind, oder neueren Datums sind.
-x	Die Daten sollen aus der angegebenen Datei ausgepackt werden.
-b n	Gibt die zu verwendende Blockgröße an.
-f <name>	Der name gibt die Datei (oder Gerät) an, in der die Daten gesichert werden sollen.
-v	Die Namen der zu sichernden Dateien werden ausgegeben.
-w	Interaktives sichern durch Bestätigung mit "y".

eingesetzt werden. Mit den Optionen kann angegeben werden, wie dies geschehen soll. Der Parameter name gibt an, welche Dateien oder Dateibäume herausgeschrieben oder wieder eingelesen werden sollen. Wird dabei ein Verzeichnis angegeben, so wird der gesamte, darin enthaltene Verzeichnisbaum, übertragen. Die Namen der zu erzeugenden Dateien, sollten auf ".tar"

enden.

Beispiel

```
% tar cvf sicherung.tar * .[a-z,A-Z]*
```

Hier werden alle normalen und versteckten Dateien (auch in den Unterverzeichnissen) gesichert.

Hinweis

Leere Unterverzeichnisse werden nicht berücksichtigt.

Eine weitere Möglichkeit der Datensicherung ist der Befehl:

```
% cpio <-o/i/p> [ <name> ]
```

Dies ist ein recht universelles Programm zum Sichern und Wiedereinlagern von Dateien. Man arbeitete dabei in der Regel mit *raw*-Ein-/Ausgabe (d.h. mit Gerätenamen, die mit **/dev/r...** beginnen). Mit Hilfe von Pipes können so gezielt Dateien gespeichert werden.

Mit **cpio -o** (output) werden Datein in der Regel auf ein Sicherungsmedium (Standardausgabe) hinaus kopiert. Die Namen der zu transferierenden Dateinen liest **cpio** von der Standardeingabe. Die vollständige angegebene Namensangabe wird zusammen mit der Statusinformation der Datei (wie Zugriffsrechte und Modifikationsdatum) gesichert.

In der Form **cpio -i** (input) list cpio Dateien von der Standardeingabe. Welche Dateien gelesen werden sollen, kann durch Namensmuster (Metazeichen wie bei der Shel) angegeben werden. Fehlt ein solches Muster, so werden alle Dateien zurückgelesen.

Mit **cpio -p** (in u. out) wird zuerst hinaus kopiert und danach wieder eingelesen.

7.3 Programmentwicklung

UNIX wurde als ein System entworfen, welches speziell für die Programmentwicklung geeignet sein sollte. Aus diesem Grund enthält es eine ganze Reihe von Hilfsmitteln.

Für die Entwicklung von Programmen stellt das UNIX-System eine Anzahl von komplexen Tools zur Verfügung. Zu diesen Tools gehörne u.a.

adb	interaktives Testhilfe- und Analyseprogramm (auf Maschinenebene) mit dem Programme kontrolliert abgearbeite werden können. Ist auch zur analyse von *core dumps* geeignet.
ar	Archivierer, damit können Dateien zu einer Bibliothek zusammengefaßt werden.
cc	C-Compiler
cpp	Präprozessor, erlaubt die Substitution von Textstrings in Dateien.
lint	Ein C-Programm *verifier.*
ld	Binder, erlaubt es, mehrere Objektdateien zu einer neuen Datei zusammenzubinden
make	Tool zur automatischen Generierung bzw. Neuerstellung eines Programmsystems (aus mehreren Dateien bestehendes Objekt)

7.4 Terminaleinstellungen

Um die aktuellen Terminaleinstellungen zu verändern wird das Kommando

```
% stty [ <option> ]
```

```
-a              zeigt alle Einstellungen am Bildschirm an.
```

benutzt. Damit können z.B. die Symbole für die erase-Funktion, den Interrupt, dem EOF oder die Baud-Rate der Terminalleitung eingestellt werden.

Hinweis

Manchmal kann es passieren, daß das Terminal so verstellt ist, daß man keine Eingabe mehr machen kann. Selbst die RETURN-Taste scheint nicht mehr zu funktionieren. In diesem Fall drückt man mehrmals CONTROL j, bis der Cursor allein dasteht. Dann existieren folgende Reperaturmöglichkeiten:

```
a)   % stty icanon icrnl opost echo CONTROL j
b)   % stty sane CONTROL j
c)   % stty -raw CONTROL j
d)   % stty cooked CONTROL j
```

Mindestens eine dieser Möglichkeiten sollte das Terminal wieder gebrauchsfähig machen. Man kann auch versuchen, von einem anderen Terminal aus das eigene (z.B. /dev/tty2) mit obigen Möglichkeiten zu reparieren:

```
% stty sane < /dev/tty2
```

Das Terminal läßt sich mit der folgenden Anwiesung in den raw-Mode setzen.

```
% stty raw
```

In diesem Mode werden Steuerzeichen nicht mehr interpretiert.

8 Anhang

8.1 Beispieldateien

8.1.1 .cshrc

```
default .cshrc for DPN NMS Workstation users

set path = ( . )
set path = ( $path $home/bin )
set path = ( $path /usr/5bin )
set path = ( $path /usr/ucb )
set path = ( $path /bin )
set path = ( $path /usr/bin )
set path = ( $path /etc )
set path = ( $path /usr/etc )
set path = ( $path /usr/bin/X11 )
set path = ( $path /iws/ipc )
set path = ( $path /iws/usm )
set path = ( $path /iws/dpn/macros/user )
set path = ( $path /iws/dpn/macros/bnr )
set path = ( $path /iws/sur )

setenv SPID $$
setenv CWD = `pwd`
set prompt = "`hostname` ! # "
set ignoreeof
set history=60
set noclobber
#set notify
set filec
set savehist=40
set time=100

stty rows 24
stty columns 80

umask 022

alias cd      'cd \!*;echo $cwd'
alias mv      'mv -i'
alias rm      'rm -i'
alias pwd     'echo $cwd'
alias la      'ls -Fa'
alias ls      'ls -F'
alias l       'ls -Fl'
alias h       'history'
alias echo    /usr/5bin/echo
alias cd      'cd \!*; setp'
alias mypwd   ''
alias setp    'set prompt = "•[1m`hostname`:`pwd` ! %•[0m "'
setp
```

```
if ( -f $HOME/.alias ) source $HOME/.alias

if( $?USER == 0 ) exit
if( $?TERM == 0 ) exit
if( $?HOST == 0 ) setenv HOST `hostname`

if( "$TERM" == "dialup" )   setenv TERM vt100
if( "$TERM" == "unknown" )  setenv TERM xterm
if( "$TERM" == "network" )  setenv TERM vt100

switch( "$TERM" )
  case "vt100":
    set term = vt100
    breaksw

  case "xterm":
    set term = xterm
    breaksw

  default:
    breaksw
endsw

alias mypwd 'echo -n "•]2;$HOST":" $CWD•";echo -n "•]1;$HOST•"'
alias cd 'cd \!*;setenv CWD `pwd`;mypwd'
set prompt = "•[1m`hostname` ! #•[0m "

mypwd

#         skip remaining setup if not an interactive shell

if ($?USER == 0 || $?prompt == 0) exit
```

8.1.2 .exrc

```
set ai
set number
set showmode
set tabstop=2
```

8.1.3 .plan

```
UNIX-Systemadministrator
```

8.2 Bücherliste

Die folgende Bücherliste soll nur um ersten Einstieg in die große weite UNIX-Welt dienen!

[GULB] Jürgen Gulbins: „UNIX“ , Version 7, bis System V.3 / Springer- Verlag / ISBN 3-540-19248-4 / ca. 90,- DM

[RICHT] Hauke Richter: „UNIX V.4“, Systemverwaltung / Addison- Wesley / ISBN 3-89319-617-X / 79,90DM

[WASHB] Kevin Washburn: „TCP/IP“, Aufbau und Betrieb einse TCP/IP-Netzes / Addison-Wesley / ISBN 3-89319-658-7 / 99,90 DM

[STRAN] John Strang : „Termcap & Terminfo“ / Addison- Wesley / ISBN 0-937175-22-6 / ?

[WINSO] Janice Winsor: „Solaris System Administrator´s Guide“ / tewi / ISBN 3-89362-817-7 / 89,-DM

[HANDS] T. Handschuch: „Solaris 2 für den Systemadministrator“ / iwt / ISBN 3-88322-532-0 / 98,-DM

[RUSSO] M. Russo: „The New User´s Guide to the Sun Workstation“ / Springer- Verlag / ISBN 0-387-97249-8 / ca. % 35,-

[HALL] M. Hall, J. Rohner: „The Sun Technology Papers“ / Springer- Verlag / ISBN 3-540-97145-9 / 58,-DM

[LEVEN] L.A. Leventhal, J. Rohner : „The SPARC System Developer´s Guide“ / Springer-Verlag / ISBN 3-540-97251-X / ?

[FEIG] Michael Feig: „UNIX von Anfang an“ / Computer Fischer / ISBN 3-596-11456-X / 16,90DM

[BRAUN] Christopher Braun: „UNIX-Systemsicherheit“ / Addison- Wesley / ISBN 3-89319-640-4 / 39,90DM

9 Index

10 Übungen

Aufgaben

Aufgabe 1.1

Aus der Umgebung wird die Zeile extrahier, in welcher der Pfad festgelegt ist und einer Variable p zugewiesen. Man beacht, daß in der Umgebung möglicherweise auch die Varable MAILPATH definiert ist.

Aufgabe 1.2

Das Kommando tty gibt den Pfadnamen des Arbeitsterminals an. Es soll nur der Name des Terminals (ohne /dev/) angezeigt werden.

Aufgabe 1.3

Aus der Ausgabe des Kommandos "who am i" soll der Login-Name und der Terminalname extrahiert werden.

Aufgabe 1.4

Aus der Datei /etc/passwd sollen alle Benutzernamen ausgegeben werden, die nicht zu Ihrer Gruppe gehören.

Aufgebe 1.5

Es soll die Anzahl aller angemeldeten Benutzer angezeigt werden. Ist ein Benutzer mehrfach angemeldet, wird er nur einmal gezählt.

Aufgabe 1.6

Es sollen die Namen der angemeldeten Benutzer ausgegeben werden. Vor dem Benuterznamen steht eien Zahl, die angibt, wie oft er angemeldet ist.

Aufgabe 1.7

Aus der Passwortdatei sollen die Benutzer und ihrer Anwendungen ausgegeben werden.

Aufgabe 1.8

Die Benutzer und ihre Anbeldezeiten sollen sortiert, am Bildschirm ausgegeben werden.

Aufgabe 1.9

Erzeugen Sie eine Tabelle aus zwei Spalten, woobei in der ersten Spalte die Dateigröße in Bytes und in der zweiten Spalte die Dateinamen aus /etc stehen. Die Tabelle ist nach Dateigröße angeordnet.

Aufgabe 1.10

Es soll die selbe Tabelle wie unter Aufgabe 9 ausgegeben werden, jedoch soll die Tabelle nur solche Einträge enthalten, in denen die Dateigröße mindestens 1000 Bytes beträgt.

Aufgabe 2.1

Ein Shell-Script soll zwei UNIX-Kommands enthalten und als Ausgabe die Anzahl der dem System angemeldeten Benutzer liefern.

Aufgabe 2.2

Ein Shell-Script soll überprüfen, ob eine Datei existiert und, wenn ja, sie dann ausführbar machen.

Aufgabe 2.3

Einem Kommando wird als Parameter ein Zeichen übergeben. Dieses einzelne Zeichen soll zehnmal auf dem Bildschirm erscheinen wenn es ein Vokal ist, ansonsten nur fünfmal.

Aufgabe 2.4

Es sollen 25 leere Dateien mittels eines Script´s angeleget werden. Die Dateien sollen die Namen tmp11, tmp12,..tmp15, tmp21, tmp22,...,tmp25, tmp31.....tmp45, tmp51,...,tmp55 erhalten.

Aufgabe 2.5

Einem Shell-Script werden als Parameter Dateinamen übergeben. Mittels einer for-Schleife sollen die Datei-Inhalte am Bildschirm ausgegeben werden.

Aufgabe 2.6

Ein Shell-Script soll drei Worte (Strings) einlesen. Dann sollen diese drei Worte auf Gleichheit getestet werden!.

Aufgabe 2.7

In einem Shell-Script soll solange eine Zahl eingelesen werden, bis diese Zahl im Intervall [70,80] liegt.

Aufgabe 2.8

Schreiben Sie ein Shell-Script mit dem Namen Vergleich, welches solange eine Eingabe anfordert, bis diese Eingabe mit dem Text "UNIX" übereinstimmt. Formulieren Sie dieses Script rekursiv.

Aufgabe 2.9

Ein Shell-Script bekommt beim Aufruf Argumente mitgeliefert. Das erst Argument ist ein Dateiname. Innerhalb des Scripts wird dann diese Datei automatisch editiert und die restlichen Argumente in die letzte Zeile der Datei eingefüght. (HINWEIS: Inline-Schriebweise).

Lösungen

Lösung 1.1

```
% set p = `env | grep ^PATH` ; echo $p
```

Lösung 1.2

```
% tty | cut -d'/' -f3
```

Lösung 1.3

```
% who am i | tr -s ' ' ' ' | cut -d' ' -f1,2
```

Lösung 1.4

```
% grep -v `id | cut -d'(' -f2 | cut -d'=' -f2` /etc/passwd | cut -d: -
f1
```

Lösung 1.5

```
% who | tr -s ' '' ' | sort | cut -d' ' -f1 | uniq | wc -l
```

Lösung 1.6

```
% who | tr -s ' '' ' | cut -d' ' -f1 | uniq -c
```

Lösung 1.7

```
% cut -d':' -f1,7 /etc/passwd | tr ':' '<TAB>'
```

Lösung 1.8

```
% who | tr -s ' '' ' | cut -d' ' -f1,5 | sort -n +1
```

Lösung 1.9

```
% ls -l /etc | tr -s ' '' ' | sort -n +4 -5 | cut -d' ' -f5,9
```

Lösung 1.10

```
% ls -l /etc | tr -s ' '' ' | sort -n +4 -5 | cut -d' ' -f5,9 | grep
'[1-9]\{4\}'
```

Lösung 2.1

```
#
set x = `who | wc -l`
echo "$x Benutzer sind eingeloggt\!"
```

Lösung 2.2

```
#
if ( -f $1 ) then
        chmod ugo+x $1
endif
```

Lösung 2.3

```
#
if ( $#argv == 1 ) then
switch ( $1 )
        case [aAeEiIoOuU] :
                echo $1$1$1$1$1$1$1$1$1$1
                breaksw
  default :
                echo $1$1$1$1$1
endsw
else echo "Nur ein Parameter\!"
endif
```

Lösung 2.4

```
#
foreach i ( 1 2 3 4 5 )
  foreach j ( 1 2 3 4 5 )
    cat /dev/null > tmp$i$j
  end
end
```

Lösung 2.5

```
#
foreach index ( $* )
  cat $index
        echo "Weiter mit RETURN\c" ; $<
end
```

Lösung 2.6

```
#
echo "Erstes Wort ->\c";set a = $<
echo "Zweites Wort ->\c";set b = $<
echo "Drittes Wort ->\c";set c = $<
echo "\nFolgende Wort wurden eingegeben $a $b $c\n"
if ( ( $a == $b ) && ( $a == $c ) ) then
        echo "Alle Worte sind gleich\!"
else if ( $a == $b ) then
        echo "Erstes und zweites Wort sind gleich\!"
else if ( $a == $c ) then
        echo "Erstes und drittes Wort sind gleich\!"
else if ( $b == $c ) then
        echo "Zweites und drittes Wort sind gleich\!"
else
        echo "Keine Uebereinstimmung\!"
endif
```

Lösung 2.7

```
#
echo "Eine Zahl ->\c"
set x = $<
while ( ! ( ( $x >= 70 ) && ( $x<= 80 ) ) )
  echo "Die Zahl muss zwischen 70 und 80 liegen\!"
  echo "Eine Zahl ->\c"
  set x = $<
end
```

Lösung 2.8

```
#
echo "Ein Wort ->\c"
set x = $<
if ( "UNIX" == $x ) then
        exit
else 2.8
endif
```

Lösung 2.9

```
#
if ( $#argv == 0 ) then
  echo "Keine Argumente\!"
  exit
else
vi $1 <<+
G
A
$*<CONTORL>V<ESC>ZZ
+
endif
```