Eine Einführung in UNIX

Rolf Reinelt[1]

6. Juni 1996

[1] e-mail: reinelt@pi.TU-Berlin.DE

Inhaltsverzeichnis

Kapitel 1

Einführung

Unix ist ein multi-user, multi-tasking Betriebssystem, das sich als Standard auf allen Rechnern vom PC mit 386-Prozessor, über alle Workstations bis zu Superrechnern durchgesetzt hat.

1.1 Geschichte und Philosophie von Unix

Ein gewisses Hintergrundwissen zur Entstehung von Unix ist hilfreich, um die Funktionsweise von Unix zu verstehen und gewisse Umständlichkeiten von Unix zu akzeptieren.

Die Entwicklung von Unix begann 1969 durch Ken Thompson bei den Bell Labs, einer gemeinsamen Tochter der Firmen AT&T und Western Electric.

Zu dieser Zeit waren die meisten Rechner sogenannte Batchrechner mit Lochkarten- bzw. Lochstreifeneingabe. Ein typischer Arbeitsablauf zur Lösung eines Problems war folgender:

- Aufschreiben des Programms (z.B. in FORTRAN) und der Eingabedaten auf Papier.
- Eingabe des Programms am Lochkartenstanzer.
- Abgabe des Lochkartenstapels beim Operator.
- Abholen der Ergebnisse (normalerweise als Zeilendruckerausdruck) nach einigen Stunden.
- Sichten der Ergebnisse.

In den Bell Labs wurde zu dieser Zeit an dem moderneren Betriebssystem MULTICS (MULTiplexed Information and Computing System) gearbeitet, das noch stark von dieser Denk- und Arbeitsweise geprägt war.

Ziel von Ken Thompsons Entwicklung war es deshalb, ein System zu schaffen, auf welchem mehrere Programmierer im Team und im Dialog mit dem Rechner arbeiten, Programme entwickeln, korrigieren und dokumentieren konnten, ohne von einem Großrechner mit all seinen Restriktionen abhängig zu sein. Das Ziel war sicherlich nicht die Entwicklung eines Betriebssystems für Homecomputer und PC's, deren „Erfindung“ erst 10 Jahre später erfolgte. Als Wortspiel nannte Thompson dieses Betriebssystem Unics (UNiplexed Information and Computing System), aus dem dann Unix wurde.

Dieses erste Unixsystem lief auf einer PDP-7, einem Kleinrechner der Firma Digital Equipment Corporation (DEC). Die erste Version von Unix war dabei in der Assemblersprache der PDP-7 geschrieben. 1971 wurde Unix auf die PDP-11 (DEC) übertragen. Um bei zukünftigen Projekten die Maschinenabhängigkeit durch die maschinennahe Sprache zu umgehen, entwarf Thompson die Programmiersprache B, aus der dann Dennis Ritchie 1972 die Sprache C entwickelte. Unix wurde dann 1973 in C umgeschrieben und auf der PDP-11 implementiert. Von nun an erfolgte die Weiterentwicklung von Unix in der höheren Programmiersprache C. Im Jahre 1975 wurde die Version 6 fertiggestellt. Diese Version fand starke Verbreitung. Der C-Quellcode für Unix wurde von den Bell Labs für die Kopierkosten an Universitäten abgegeben. Dies hatte für die Computerentwicklung (dramatische) Folgen:

- Aufgrund der geringen Kosten erfolgte eine weite Verbreitung.
- Wegen der Formulierung in einer höheren Programmiersprache erfolgte eine Portierung auf viele verschiedene Hardwareplattformen.

- Wegen der Formulierung in einer höheren Programmiersprache und des modularen Aufbaus konnten andere Programmierer neue Funktionen zu Unix hinzufügen, was zu einer schnellen Weiterentwicklung des Betriebssystems führte.
- Allerdings entstanden damit auch verschiedene Dialekte, die sich aus rechtlichen Gründen nicht direkt Unix nennen dürfen (UNICOS, Ultrix, Linux, XENIX, SINIX etc.)

Ausgangspunkt für viele Derivate von Unix war allerdings die Version 7, welche im Jahre 1979 erschien. Für viele ist diese Version „the last true Unix, an improvement over all preceding and following Unices".

Von dieser Version stammt auch das Berkeley-Unix BSD ab, das neben der AT&T-Linie die wichtigste Entwicklungslinie von Unix darstellt. Auch das auf unseren DEC-Stations verwendete Ultrix baut auf dem Berkeley-Unix auf. Glücklicherweise sind Bestrebungen im Gange, die verschiedenen Entwicklungslinien wieder zur Konvergenz zu bringen. Aus dieser kurzen Historie ergeben sich somit die wesentlichen Vorteile von Unix:

- Multi-User, Multi-Tasking
- Weite Verbreitung, deshalb kurze Einarbeitungszeit bei Benutzung neuer Rechner. (login, fell at home)
- Großer Leistungsumfang, speziell für Programmentwickler
- Preisgünstige Portierung des Betriebssystems auf neue Hardware, da Unix (im wesentlichen) in der Hochsprache C geschrieben ist.

Nachteile hat Unix aber auch:

- Zum Teil ist Unix sehr kryptisch, da es ursprünglich für Systemprogrammierer gedacht war.
- Aufgrund des großen Funktionsumfangs verbraucht es relativ viel Resourcen.

1.2 Rechnerzugang

Unix ist ein multi-user, multi-tasking Betriebssystem. Es muß deshalb zwischen verschiedenen Benutzern unterscheiden können. Dazu erhalten die Benutzer einen sogenannten Username (Benutzername). Die Zugangskontrolle zum Computer erfolgt über ein sog. Password (Geheimwort). Jeder User (Benutzer) erhält einen eigenen Bereich auf der Festplatte des Computers, das sog. Homedirectory (Heimatverzeichnis). Username, Password und Homedirectory bilden den sog. Account (Benutzerzulassung), der beim Systembesitzer beantragt werden muß. Eingerichtet wird der Account vom Systemverwalter, der selbst einen speziellen Account auf dem Rechner hat. Sein Username ist root. Mit dem Password von root hat man totale Kontrolle über einen Unixrechner.

Der physikalische Zugang zu einem Unixrechner kann über die Konsole, angeschlossene Terminals oder über ein Netzwerk erfolgen.

Die Konsole besteht aus einer Tastatur und einem Bildschirm, die direkt an den Rechner angeschlossen sind. Jeder Unixrechner besitzt eine Konsole. Bei großen Unixrechnern hat normalerweise nur der Systemverwalter Zugang zur Konsole. Auf dem Bildschirm der Konsole erscheinen z.B. die Systemmeldungen beim Einschalten des Computers.

Besonders bei zentralen Unixrechnern älterer Bauart sind oftmals außer der Konsole noch weitere Terminals angeschlossen. Die Terminals haben prinzipiell keine eigene Rechenleistung oder Speicherkapazität, sondern dienen lediglich der Ein- und Ausgabe. Diese „dummen" Terminals werden zunehmend durch „intelligente" ersetzt. So kann z.B. ein MS-DOS Rechner mit einem sog. Terminalemulationsprogramm als Terminal betrieben werden.

Schließlich kann der Zugang zu einem Unixrechner über ein Netzwerk erfolgen. Dieses Netzwerk kann durch Wählleitungen des öffentlichen Telefonnetzes realisiert sein. Wesentlich häufiger ist die Vernetzung über spezielle Gebäudeverkabelungen z.B. über Koaxialkabel. Das Kabel wird an einen netzwerkfähigen Rechner angeschlossen, welcher z.B. unter Unix oder MS-DOS betrieben werden kann.

Die Beispiele in diesem Skript gelten für einen Benutzer Karl Hansen. Sein Username ist 'hansen'. Das Einloggen (Computerslang für das Anmelden bei einem Rechner) soll an dem Rechner 'weber' erfolgen.


```
ULTRIX V4.1 (Rev. 52) (weber)

login: hansen
Password:
Last login: Wed Sep 29 14:02:37 from nusselt.pi.TU-Be
ULTRIX V4.1 (Rev. 52) System #1: Thu Jan  9 06:47:55 EST 1992
UWS V4.1 (Rev. 197)

                Digital Equipment Corporation
                Nashua, New Hampshire

Type console unknown
Wed Sep 29 14:14:06 MET 1993
weber>
```

Die Eingabe des Passwords erfolgt unsichtbar. Das vom Systemverwalter vergebene Password sollte normalerweise vom Benutzer geändert werden. Dazu wird das Kommando

`passwd`

benutzt.

```
weber> passwd
Changing yp password for hansen
Old yp password:
New password:
Retype new password:
yellow pages passwd changed on w423zrz
weber>
```

Wiederum erfolgt die Eingabe der Passwörter unsichtbar. In unserem speziellen Fall wird die Benutzerverwaltung zentral auf der DEC-Station w423zrz durchgeführt und den anderen DEC-Stations zur Verfügung gestellt (Stichwort Yellow Pages yp). Für den Benutzer bedeutet dies lediglich, daß er auf allen angeschlossenen DEC-Stations das gleiche Password hat. Beendet wird eine Sitzung mit dem Kommando

`exit`

1.3 Dateisystem

1.3.1 Dateiarten

Eine Datei (File) ist aus Sicht des Benutzers im allgemeinen eine Folge logisch zusammengehöriger Informationen, aus Sicht des Systems hingegen ist sie zunächst nur eine endliche Folge von Bytes. Eine Datei wird über ihren Namen angesprochen, wobei zwischen Groß- und Kleinschreibung unterschieden wird. Intern benutzt Unix sog. Inodenummern (inode-numbers). Unix kennt im wesentlichen vier Dateiarten:

- normale Dateien (files)
- Verzeichnisse (directories)
- Gerätedateien (special files)
- Pipes (pipes)

Normale Dateien: Eine normale Unixdatei entspricht konventionellen, aus anderen Betriebssystemen bekannten Dateien. In diesem Sinn kann eine gewöhnliche Datei beliebige Daten aufnehmen, beispielsweise Programme und Texte. Unix erwartet keinerlei spezielle Strukturierung einer Datei in Sätze, Blöcke, Sektoren usw. Vom Standpunkt des Anwenders aus gesehen hat auch in Unix jede Datei eine interne Struktur; so besteht eine Textdatei beispielsweise aus einer Folge von Zeichen mit einer übergeordneten Zeilenstruktur, wobei die Zeilenenden durch besondere Zeilenendezeichen vermerkt sind.

Verzeichnisse: Verzeichnisse sind Dateien, die Verweise auf weitere Dateien und Verzeichnisse enthalten. Leere Verzeichnisse enthalten zumindest noch die Verweise auf sich selbst und ihr Vaterverzeichnis. Dies hängt mit dem hierarchischen Aufbau des Dateisystems zusammen. Die Anzahl der Einträge ist beliebig, natürlich durch die Größe des Datenträgers limitiert.
Gerätedateien: Gerätedateien sind Dateien, welche für physikalische Geräte stehen. Durch ihre den normalen Dateien gleiche Behandlung ergibt sich für den Benutzer kein Unterschied zwischen der Ein- und Ausgabe auf Dateien oder auf physikalische Geräte, und damit auch weitestgehende Geräteunabhängigkeit.
Pipes: Pipes sind systeminterne Dateien zur Prozeßkommunikation, siehe auch Kapitel 3.2.3.

1.3.2 Dateibaum

Da ein Unixsystem normalerweise mit mehreren tausend Dateien umgeht, würde der Benutzer den Überblick verlieren, wenn es nicht einen Mechanismus geben würde, der es erlaubt, Dateien zu Gruppen zusammenzufassen. Die Unix-Dateienstruktur ist hierarchisch und wird normalerweise als umgekehrter Baum veranschaulicht. Der Ausgangspunkt eines solchen Baumes ist die Wurzel (root), die unter Unix mit / bezeichnet wird. Die Wurzel selbst ist ein Verzeichnis, in dem Verweise auf weitere Dateien stehen, die wiederum selbst Verzeichnisse sein können. Auf diese Weise entsteht die Baumstruktur. Abbildung (1.1) zeigt einen Teil eines typischen Unix-Dateisystems.

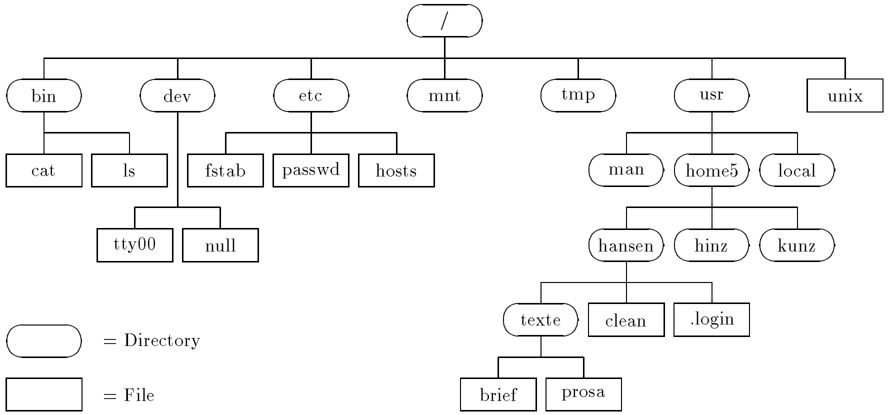

Abbildung 1.1: Beispiel eines Dateibaumes

Während der gesamten Sitzung ist dem Benutzer zu jedem Zeitpunkt genau ein Verzeichnis zugeordnet, auf das sich seine Kommandos beziehen. Dieses Verzeichnis nennt man das *aktuelle Verzeichnis* (current working directory). Der Benutzer kann sich im gesamten Dateibaum bewegen, soweit er dafür die entsprechenden Rechte besitzt.
Um sich im Dateibaum zu bewegen, benutzt man das Kommando `cd` (**c**hange **d**irectory).

`cd` [*verzeichnis*]

Zur Information, wo man sich zur Zeit befindet, benutzt man das Kommando `pwd` (**p**rint **w**orking **d**irectory).

`pwd`

Oftmals wird das aktuelle Verzeichnis auch im sogenannten Prompt (Eingabeaufforderung) angezeigt. Dann ist das `pwd`-Kommando überflüssig. Bei der Eingabe des `cd`-Kommandos ohne Angabe eines Arguments wird das Heimatverzeichnis (homedirectory) eingestellt. In Abb. (1.1) würde das Heimatverzeichnis des Benutzers hansen `/usr/home5/hansen` lauten. Die Eingabe von zwei Punkten als Argument des `cd`-Kommandos bewirkt

den Wechsel zum Vaterverzeichnis des aktuellen Verzeichnisses, also ein Verzeichnis zurück in Richtung Wurzelverzeichnis (root). Beispiel:

```
weber> pwd
/usr/home5/hansen
weber> cd texte
weber> pwd
/usr/home5/hansen/texte
weber> cd ..
weber> cd ..
weber> cd ..
weber> cd local
weber> pwd
/usr/local
weber>
```

Wichtig ist noch der Unterschied zwischen relativen und absoluten Pfadnamen. Für jedes Verzeichnis und jede Datei gibt es genau einen absoluten Pfadnamen, der immer im Wurzelverzeichnis (root) beginnt, also immer mit / anfängt. Der absolute Pfadname der Datei clean in Abb. (1.1) lautet z.B. `/usr/home5/hansen/clean`. Beim relativen Pfadnamen beginnt man im aktuellen Verzeichnis. Ist man z.B. im Verzeichnis `/usr/local`, so lautet der relative Pfadname der Datei clean `../home5/hansen/clean` . Verzeichnisse werden mit dem Kommando `mkdir` (**mak**e **dir**ectory) erzeugt

`mkdir` *verzeichnis*

und mit `rmdir` (**re**mo**v**e **dir**ectory) entfernt.

`rmdir` *verzeichnis*

Beim Entfernen eines Verzeichnisses muß es leer sein und darf nicht das aktuelle Verzeichnis sein. Beispiel:

```
weber> cd
weber> pwd
/usr/home5/hansen
weber> mkdir test
weber> cd test
weber> pwd
/usr/home5/hansen/test
weber> cd ..
weber> rmdir test
weber>
```

Um sich den Inhalt von Verzeichnissen anzuschauen, gibt es das Kommando

`ls` [`-al`] [*datei ...*]

Dieses Kommando hat sehr viele Optionen, die die Art der Anzeige steuern. Die wichtigsten sind `-l` und `-a`. Die Option `-l` (long) gibt wichtige Informationen für jede Datei aus. Die Option `-a` (all) gibt auch die Dateien aus, die mit einem Punkt beginnen. Beispiel:

```
weber> cd
weber> pwd
/usr/home5/hansen
weber> ls
clean  texte
weber> ls -a
.           .Xdefaults  .login      .profile    texte
..          .cshrc      .mailrc     clean
weber> ls -l
total 2
-rw-r--r--  1 hansen         9 Sep 29 15:22 clean
```

```
drwxr-xr-x  2 hansen        512 Sep 29 15:26 texte
weber> ls -la
total 13
drwxr-xr-x  3 hansen        512 Sep 29 15:31 .
drwxr-xr-x  7 root          512 Sep 29 15:09 ..
-rw-r--r--  1 hansen       5030 Sep 29 15:09 .Xdefaults
-rwxr-xr-x  1 hansen        261 Sep 29 15:09 .cshrc
-rwxr-xr-x  1 hansen        258 Sep 29 15:09 .login
-rwxr--r--  1 hansen         79 Sep 29 15:09 .mailrc
-rwxr--r--  1 hansen        182 Sep 29 15:09 .profile
-rw-r--r--  1 hansen          9 Sep 29 15:22 clean
drwxr-xr-x  2 hansen        512 Sep 29 15:26 texte
weber>
```

Der Eintrag '.' steht für das aktuelle Verzeichnis selber, während '..' für das Vaterverzeichnis steht, in diesem Fall also `/usr/home5`.

1.3.3 Umgang mit Dateien

Um sich den Inhalt einer Textdatei anzuschauen, gibt es in Unix verschiedene Möglichkeiten. Das gebräuchlichste Kommando ist

`cat` *datei ...*

(con**cat**enate). Bei langen Dateien kann man mit dem Kommando

`more` *datei ...*

eine seitenweise Ausgabe erreichen. Das Kommando

`head` *datei ...*

zeigt die ersten (10) Zeilen einer Datei und

`tail` *datei ...*

die letzten (10) Zeilen einer Datei an. Außerdem kann man sich Textdateien natürlich mit einem Editor ansehen, siehe Kapitel 2. Mit dem Kommando

`cp` *alte-datei neue-datei*

(**cop**y) erzeugt man eine Kopie einer Datei bzw. kopiert man mehrere Dateien in ein Verzeichnis.

`cp` *alte-datei ... verzeichnis*

Das Kommando `rm` (**rem**ove) entfernt bzw. löscht eine Datei.

`rm` *datei ...*

Mit dem Kommando `mv` (**mov**e) ändert man einen Dateinamen und/oder verschiebt die Datei in ein anderes Verzeichnis.

`mv` *alte-datei neue-datei*

`mv` *alte-datei ... verzeichnis*

```
weber> cd
weber> cd texte
weber> cat prosa
Hoffentlich kommt bald ein Birkenwald,
denn meine Pillen wirken bald.
weber> cp prosa dirty_prosa
```



```
weber> mv brief brief_an_bollmann
weber> ls
brief_an_bollmann  dirty_prosa        prosa
weber> rm dirty_prosa
weber> ls
brief_an_bollmann  prosa
weber>
```

Natürlich darf nicht jeder Benutzer eines Unixsystems alle Dateien manipulieren. Die entsprechenden Rechte werden über Attribute der Dateien und Verzeichnisse verwaltet. Wir betrachten dazu den Output des `ls`-Kommandos mit den Optionen `-lg`.

```
weber> cd
weber> ls -l
total 2
-rwxr-x---  1 hansen   system          9 Sep 29 17:20 clean
drwxr-xr-x  2 hansen   system        512 Oct 29 18:39 texte
weber>
```

Bei Ultrix bewirkt die Option `-g` die Ausgabe der zugehörigen Gruppe. Bei anderen Unix-Dialekten ist diese Option schon implizit in der Option `-l` enthalten. Die Ausgabe des Kommandos `ls -lg` bedeutet im einzelnen folgendes:

- In der ersten Zeile wird angegeben, wieviele Blöcke die Einträge in diesem Verzeichnis belegen. Ein Block sind normalerweise 512 Byte. Darauf folgt eine Liste aller Dateien, wobei pro Datei eine Zeile an Informationen ausgegeben wird.
- In der ersten Spalte stehen das Dateiartbit und die 9 Zugriffsrechtebits (protection bits), wobei jedes Bit durch ein druckbares Zeichen dargestellt wird. Das Dateiartbit kann folgende Einträge haben:
  - `d` Datei ist ein Verzeichnis (directory)
  - `l` Datei ist nur ein symbolischer Linkeintrag (Soft-Link) auf eine andere Datei. Ein Link wird mit dem Kommando `ln` erzeugt, siehe auch Kapitel 4.4 auf Seite 35.
  - `-` normale Datei
  - `b` Blockorientierte Gerätedatei
  - `c` Characterorientierte Gerätedatei
  - `p` FIFO-Puffer (named pipe special file)

  Die Zugriffsrechte werden gleich erläutert.
- In der dritten und vierten Spalte stehen der Name des Besitzers und der zugehörigen Gruppe.
- In der fünften Spalte wird die Dateilänge in Bytes angegeben.
- In den nächsten Spalten wird Datum und Uhrzeit der letzten Änderung angegeben. Bei sehr alten Dateien wird statt der Uhrzeit das Jahr angegeben.
- In der letzten Spalte steht der Name der Datei.

Für die Zugriffsrechte muß man die folgenden drei Benutzerklassen unterscheiden:

- u (**u**ser) Der Besitzer einer Datei.
- g (**g**roup) Die Gruppe, der der Besitzer einer Datei angehört.
- o (**o**ther) Alle Benutzer.

Für jede dieser drei Benutzerklassen werden drei verschiedene Zugriffsrechte gesetzt:

- r (**r**ead) Lesen
- w (**w**rite) Schreiben

- x (**e**x**e**cute) Ausführen

Insgesamt können also 9 Zugriffsrechte pro Datei gesetzt werden. Diese Zugriffsrechte gibt es für alle Dateien, also auch für Gerätedateien und Verzeichnisse. Mit der Vergabe von Rechten sollte aus Gründen des Datenschutzes und der Datensicherheit sehr vorsichtig umgegangen werden. An der Ausgabe des `ls -l` Kommandos im obigen Beispiel kann man die Zugriffsrechte ablesen. Die ersten drei Bits zeigen die Rechte des Eigentümers, die mittleren die Rechte der Gruppenmitglieder und die letzten drei Bits die Rechte aller anderen Benutzer an. Die Datei `clean` besitzt z.B. als protection mode `rwxr-xr--`. Dies bedeutet, daß der Eigentümer (also hansen) diese Datei lesen, auf ihr schreiben und sie ausführen darf. Alle Gruppenmitglieder dürfen diese Datei nur lesen oder ausführen, nicht aber löschen oder umbenennen. Andere Benutzer dürfen auf diese Datei nur lesend zugreifen.

Die Zugriffsrechte auf Verzeichnisse stellen eine Spezialität dar. Die protection bits sind gleich denen von normalen Dateien, jedoch besitzen die Zugriffsrechte eine etwas andere Bedeutung:

- Um ein Verzeichnis ansehen oder durchsuchen zu können, benötigt man nicht nur das Leserecht für dieses Verzeichnis, sondern auch das Ausführungsrecht. Es müssen also das r und das x Bit gesetzt sein.

- Ist das w-Bit eines Verzeichnisses für eine Benutzerklasse gesetzt, so kann jedes Mitglied dieser Klasse eigene Dateien in dieses Verzeichnis eintragen. Eigentümer dieser neuen Dateien ist der Benutzer, der die Datei anlegt. Es ist sehr gefährlich, diese Schreibberechtigung zu vergeben, da jeder, der das Schreibrecht für ein Verzeichnis hat, auch Dateien aus dem Verzeichnis löschen kann, und zwar unabhängig davon, ob er irgendwelche Zugriffsrechte auf diese Datei besitzt oder nicht.

Mit dem Kommando `chmod` (**ch**ange **mod**e) kann der Eigentümer einer Datei die Zugriffsrechte dieser Datei ändern. Für dieses Kommando gibt es zwei syntaktisch verschiedene Formen:

`chmod` *permissionlist datei ...*

`chmod` *ooo datei ...*

Im ersten Fall gibt der Benutzer eine *permissionlist* und den Namen der Datei an, deren protection mode geändert werden soll, und zwar durch gezieltes Hinzufügen oder Wegnehmen von Zugriffsrechten einzelner Benutzerklassen.

Die *permissionlist* hat dabei folgende Form: *benutzerklasse* ± *rechte*. Als Benutzerklasse ist außer den drei bekannten (u, g und o) noch eine Sammelklasse a (**a**ll) möglich. Beispiel:

```
weber> ls -l
total 2
-rwxr-xr--  1 hansen          9 Sep 29 15:22 clean
drwxr-xr-x  2 hansen        512 Sep 29 15:38 texte
weber> chmod o-r clean
weber> chmod go-rx texte
weber> ls -l
total 2
-rwxr-x---  1 hansen          9 Sep 29 15:22 clean
drwx------  2 hansen        512 Sep 29 15:38 texte
weber>
```

In der zweiten Form des Kommandos wird der protection mode als ganzer neu gesetzt. Dabei steht *ooo* für eine dreistellige Oktalzahl, die als Binärzahl gelesen genau dort die Einsen hat, wo das entsprechende Zugriffsrecht gesetzt sein soll. Beispiel:

```
weber> chmod 754 clean
weber> ls -l
total 2
-rwxr-xr--  1 hansen          9 Sep 29 15:22 clean
drwx------  2 hansen        512 Sep 29 15:38 texte
weber>
```


Die Oktalzahl 754 lautet in Binärdarstellung `111101100`, was genau den Zugriffsrechten `rwxr-xr--` der Datei clean entspricht.

Bei jeder Erstellung einer Datei werden automatisch vom System voreingestellte protection bits gesetzt. Diese Voreinstellung kann und sollte vom Benutzer bei Sitzungsbeginn auf seine eigenen Bedürfnisse hin mit dem Kommando `umask` (**u**ser **mask**) geändert werden.

`umask` [*ooo*]

ooo ist hierbei wieder eine Oktalzahl, die allerdings etwas anders wirkt als beim Kommando `chmod`. Die Bits, die in der Maske bestimmt sind, werden beim anlegen der Datei gerade *nicht* gesetzt. Man bestimmt also, welche Rechte nicht gewährt werden sollen. Beispiel:

```
weber> umask 177
weber> cp clean testdatei
weber> ls -l
total 3
-rwxr-xr--  1 hansen          9 Sep 29 15:22 clean
-rw-------  1 hansen          9 Sep 29 15:54 testdatei
drwx------  2 hansen        512 Sep 29 15:38 texte
weber> umask
177
weber>
```

Jede von diesem Zeitpunkt an erzeugte Datei wird read- und write-Berechtigung für den Eigentümer besitzen, andere Benutzer haben überhaupt keine Zugriffsrechte. Das `umask` Kommando ohne Argument liefert die momentan gültige Einstellung der Maske.

1.4 Prozesse

Ein wesentliches Leistungsmerkmal von Unix ist die Fähigkeit zum multi-tasking. Das bedeutet, daß der Rechner quasi-zeitgleich mehrere Programme bzw. Kommandos (tasks) ausführen kann. Bis jetzt wurde in diesem Skript immer ein Kommando gegeben und die Shell antwortete erst nach Beendigung des Kommandos mit einem Prompt. Alle Kommandos wurden also nacheinander abgearbeitet. Wie startet man aber mehrere Programme, die gleichzeitig laufen sollen? Es gibt dazu zwei wichtige Möglichkeiten:

- **Mehrere Terminals:** Bei Benutzung der X-Windows Oberfläche kann man mehrere Fenster (Windows) aufmachen. Die Tastatur ist dann exklusiv dem gerade aktiven Fenster zugeordnet. Man hat dann also die Möglichkeit, in jedem Fenster Prozesse zu starten, die dann parallel ablaufen.

- **Hintergrundprozesse:** Durch Anhängen des Zeichens & an ein Kommando wird dieses im Hintergrund abgearbeitet. Die Shell meldet sich in diesem Fall sofort wieder mit dem Prompt. Der Benutzer erhält auf dem Bildschirm eine Nachricht, wenn der Hintergrundprozeß beendet wurde.

In Abbildung 1.2 ist der zeitliche Ablauf von verschiedenen Prozessen dargestellt. Ausgangspunkt ist der Prozeß `csh`. Dies ist die C-Shell, also das Programm, das unsere Eingaben annimmt und entsprechende Betriebssystemaufrufe veranlaßt. Näheres siehe Kapitel 3. In der Abbildung wird im Hintergrund mit `f77` der Fortrancompiler aufgerufen. Da der Compiler im Hintergrund läuft, können sowohl `csh` als auch `f77` aktiv sein, was durch die durchgezogenen Linien dargestellt wird. Dagegen wird das Kommando `ps` im Vordergrund gestartet, so daß der Prozeß `csh` bis zur Beendigung von `ps` warten muß, was durch die gestrichelte Linie dargestellt ist.

Das `ps`-Kommando erlaubt die Ausgabe von Systeminformationen über alle laufenden Prozesse.

`ps` [`-axl`]

Ohne Optionen gibt `ps` nur die wichtigsten Informationen über die Prozesse aus, die vom Benutzer an diesem Terminal gestartet wurden. In der ersten Spalte wird dabei die eindeutige Nummer jedes Prozesses angezeigt (process identification number PID). Diese Nummer kann benutzt werden, um mit dem Kommando `kill` Prozesse zu stoppen, egal ob sie im Vorder- oder Hintergrund laufen.

`kill` [`-9`] *prozeßnummer*

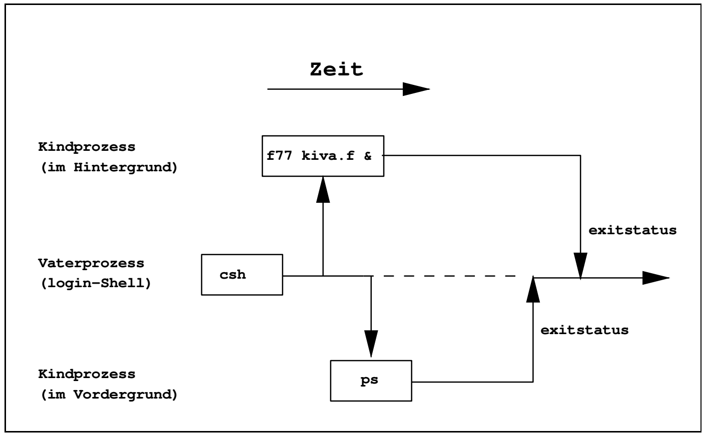

Abbildung 1.2: Zeitlicher Verlauf von Prozessen

```
weber> cd
weber> clean &
[1] 6031
weber> ps
  PID TT STAT  TIME COMMAND
 5825 p0 S     0:02 -csh (csh)
 6031 p0 S     0:00 /bin/sh clean
 6032 p0 S     0:00 sleep 99
 6033 p0 R     0:01 ps
weber> kill -9 6031

[1]    Killed               clean
weber> ps
  PID TT STAT  TIME COMMAND
 5825 p0 S     0:02 -csh (csh)
 6037 p0 R     0:01 ps
weber>
```

1.5 Online-Dokumentation

Unixsysteme sind normalerweise mit einer Online-Dokumentation aller Kommandos, den sogenannten Man-pages, ausgestattet. Zum Lesen dieser Dokumentation steht das Kommando `man` zur Verfügung.

`man` *kommandoname*

```
weber> man apropos

                                                                 apropos(1)


     Name
```



```
          apropos - locate commands by keyword lookup

     Syntax
          apropos keyword...

     Description
          The apropos command shows which manual sections contain instances
          of any of the given keywords in their title.  Each word is con-
          sidered separately and the case of letters is ignored.  Words
          that are part of other words are listed.  Thus, looking for the
          word compile hits all instances of 'compiler' also.

          If the line starts 'name(section) ...' you can do 'man section
          name' to get the documentation for it.  The following command
          line lists all commands that have to do with formatting:

          apropos format

--More--
```

Auf die Aufforderung –More– hin drückt man z.B. die Leertaste, woraufhin eine weitere Seite angezeigt wird.

```
          To then access the reference page for the printf subroutine that
          you see listed, type:

          man 3s printf

          The apropos command is actually just the -k option to the man
          command.

     Files
          /usr/lib/whatis          data base

     See Also
          man(1), whatis(1), catman(8)


                                                                       1

weber>
```

Kapitel 2

Editoren

Unix stellt eine Reihe sehr leistungsfähiger Editoren zur Verfügung. Es sind dies:

- `ed` ist ein Zeileneditor mit mächtigen Such- und Ersetzungsmechanismen.
- `ex` ist ebenfalls ein Zeileneditor und stellt eine Obermenge von `ed` dar, d.h. `ex` versteht alle `ed`-Kommandos, kann aber noch mehr.
- `vi` ist der Unix-Fullscreeneditor. `vi` und `ex` sind tatsächlich nur 2 verschiedene Modi desselben Editors.
- `sed` ist ein batchorientierter Editor, der speziell für gleichartige Änderungen an vielen Dateien nützlich ist. Außerdem können mit diesem Editor beliebig große Dateien bearbeitet werden.

Für die tägliche Arbeit ist selbstverständlich ein Fullscreeneditor zu verwenden. Da der `vi` auf *allen* Unixsystemen vorhanden und außerdem sehr mächtig ist, hält der Autor alle anderen Editoren für überflüssig!

2.1 vi

vi arbeitet immer mit einer lokalen Kopie der Datei im Arbeitsspeicher. Solange nicht abgespeichert wird, bleibt also die Datei auf der Festplatte unverändert.

2.1.1 Aufruf und Verlassen des Editors

Der Editor wird aufgerufen mit

`vi` [`-r` *recoverdatei*] [`-x`] [`-R`] [*datei ...*]

Die Option `-r` *recoverdatei* wird benutzt, wenn während des Editierens der Editor oder das ganze System abgestürzt sind. Durch die Option `-x` erfolgt eine Verschlüsselung der Daten. Der Benutzer wird dabei interaktiv nach einem Schlüssel gefragt. Für diesen erscheint bei der Eingabe kein Echo. Durch die Option `-R` (**r**ead only) wird die Datei nur zum Lesen geöffnet.

Der Editor wird mit dem Kommando

`ZZ`

verlassen. Wurde die Datei verändert, so wird vorher abgespeichert. Mit dem Kommando

`:w` [*datei*]

wird nur abgespeichert. Ohne Angabe von *datei* wird der beim Aufruf angegebene Dateiname benutzt. Den Editor zu verlassen, ohne Änderungen abzuspeichern, ist mit dem Kommando

`:q!`

möglich. Wurde beim Aufruf des `vi` mehr als eine Datei angegeben, so wechselt man mit

`:n`

zur nächsten Datei und mit

`:rew`

kommt man wieder in die erste Datei. Eine Übertragung von Textteilen ist mit benannten Puffern möglich, siehe Abschnitt 2.1.5.

2.1.2 Cursorpositionierung

Zuerst die schlechte Nachricht: Cursorpositionierung per Maus ist nicht möglich. Die gute Nachricht: Alles per Tastatur denkbare ist möglich. Es gibt dabei 5 Arten der Verschiebung:

- In festen Bildschirmeinheiten wie Zeichen, Zeile, halbe Ausschnittsgröße oder um eine ganze Ausschnittsgröße (Seite)
- In Objekten wie Wort, Satz oder Absatz
- Durch Suchen nach einem Textmuster, siehe Abschnitt 2.1.4.
- Benutzung von vorher gesetzten Marken, siehe Abschnitt 2.1.5.
- Korrespondierende Klammerpaare

Die Objekte Wort, Satz und Absatz sind wie folgt definiert:

- Ein **Wort** ist eine Folge von Buchstaben und Ziffern ohne Zwischenraum. Wird bei den Wort-Operationen der kleine Buchstabe verwendet, so wird ein Sonderzeichen als eigenes Wort interpretiert. Bei den großen Buchstaben werden Worte nur durch Leerzeichen, Tabulator oder Neue-Zeile getrennt.
- Ein **Satz** ist eine Folge von Worten, die durch '.', '!' oder '?' terminiert wird. Diesem Zeichen müssen entweder eine neue Zeile oder zwei Leerzeichen folgen.
- Ein **Absatz** beginnt nach jeder Leerzeile.

Die Objekte sind nicht nur zur Cursorpositionierung wichtig, sondern auch beim Löschen und Kopieren etc. Die Objekte (`<obj>`) werden ausgewählt durch

`< >`	(Leerzeichen) für ein einzelnes Zeichen.
`w`	für ein Wort ohne Sonderzeichen.
`W`	für ein Wort mit Sonderzeichen.
`b`	für ein vorhergehendes Wort ohne Sonderzeichen.
`B`	für ein vorhergehendes Wort mit Sonderzeichen.
`G`	für bis zum Ende des Puffers.
`^`	für bis Anfang der Zeile.
`$`	für bis zum Ende der Zeile.

Einfache Cursorbewegungen

`h`	1 Zeichen nach links.
`j`	1 Zeichen nach oben.
`k`	1 Zeichen nach unten.
`l`	1 Zeichen nach rechts.
`<cr>`	Erste Zeichen der nächsten Zeile.
`-`	Erste Zeichen der vorherigen Zeile.
`G`	Erste Zeichen der letzten Zeile.

Dabei sind Wiederholungsfaktoren möglich. Die Eingabe von `5l` würde z.B. den Cursor 5 Stellen nach rechts verschieben. Beim `G` wird die Zeile mit der entsprechenden Zeilennummer angesprungen.

Cursorbewegungen auf der aktuellen Zeile

0 Anfang der aktuellen Zeile

^ Erste sichtbare Zeichen der aktuellen Zeile

$ Letzte Zeichen der aktuellen Zeile

n| Bewegt den Cursor auf die n-te Spalte der aktuellen Zeile. Dies ist sehr nützlich für Fortranprogrammierer.

Cursorbewegungen auf Worte

w Anfang des nächsten Wortes

W Anfang des nächsten Wortes

b Anfang des aktuellen oder vorangehenden Wortes

B Anfang des aktuellen oder vorangehenden Wortes

e Ende des aktuellen oder folgenden Wortes

E Ende des aktuellen oder folgenden Wortes

Wiederholungsfaktoren sind möglich.

Cursorbewegungen auf Sätze und Absätze

) Anfang des nächsten Satzes

(Anfang des vorhergehenden Satzes

} Anfang des nächsten Absatzes

{ Anfang des vorhergehenden Absatzes

Cursorbewegungen Bildschirmweise

H Erste Zeichen der ersten Zeile des aktuellen Bildschirms

M Erste Zeichen der mittleren Zeile des aktuellen Bildschirms

L Erste Zeichen der letzten Zeile des aktuellen Bildschirms

<ctrl+f> Ein Bildschirm vorwärts

<ctrl+b> Ein Bildschirm rückwärts

Positionieren des Cursors auf Klammerpaare

% Bewegt den Cursor auf die korrespondierende Klammer (,),{,},[,].

2.1.3 Einfügen, Verändern und Löschen

Um etwas Einfügen zu können, muß vom Kommandomodus in den Einfügemodus umgeschaltet werden. Der Einfügemodus wird mit <esc> verlassen. Man beachte, daß Cursorpositionierungen etc. nur im Kommandomodus möglich sind.

Einfüge- und Ersetzungskommandos

i Einfügen vor dem Cursor.

I Einfügen am Zeilenanfang.

a Einfügen nach dem Cursor.

A Einfügen am Zeilenende.

o Einfügen von Text nach der aktuellen Zeile.

O Einfügen von Text vor der aktuellen Zeile.

r Ersetzen eines Zeichens durch ein anderes.

R Überschreiben des Textes.

c<obj> Ersetzen des Objektes <obj>.

C Ersetzen des Rests der Zeile.

s Ersetzen des Zeichens durch den eingegebenen Text. Dabei sind Wiederholungsfaktoren möglich.

S Ersetzen der ganzen Zeile.

Im Eingabemodus haben u.a. folgende Kontrollzeichen eine Sonderbedeutung:

Kontrollzeichen im Einfügemodus

<ctrl+w> Lösche das letzte eingegebene Wort.

<bs> Lösche das letzte eingegebene Zeichen.

<ctrl+v> Eingabe eines Sonderzeichens.

Mit

:r *datei*

wird die Datei *datei* nach der aktuellen Zeile eingefügt.

Löschen von Text

x Löscht das Zeichen an der Cursorposition.

X Löscht das Zeichen vor der Cursorposition.

d<obj> Löscht das Objekt <obj>.

dd Löscht die aktuelle Zeile.

D Löscht die Zeile von der aktuellen Cursorposition bis zum Zeilenende.

dG Löscht die aktuelle und alle nachfolgenden Zeilen.

2.1.4 Suchen und Ersetzen, reguläre Ausdrücke

Reguläre Ausdrücke

Häufig möchte man Unix-Kommandos wie z.B. Löschen oder Kopieren auf eine Reihe von Dateien anwenden oder in Texten, statt nach einer festen Zeichenkette, mit einem Muster suchen, in dem noch gewisse Freiheitsgrade vorhanden sind. Für beide Anforderungen stehen unter Unix sogenannte „reguläre Ausdrücke“ (regular expression) zur Verfügung, die weitgehend einheitlich von den verschiedenen Unix-Programmen interpretiert werden.

Ein regulärer Ausdruck ist dabei eine Folge von normalen Zeichen und Metazeichen. Ein normales Zeichen steht für das entsprechende ASCII-Zeichen selbst. Ein Metazeichen ist ein Zeichen, welches nicht das entsprechende Zeichen darstellt, sondern eine erweiterte Bedeutung besitzt. Das Fragezeichen ? z.B. steht in der Angabe von Dateinamen für ein beliebiges (auch nicht druckbares) Zeichen. Meint man in einem regulären Ausdruck das Zeichen selbst und nicht die Metabedeutung, so muß man das Metazeichen maskieren. Dies geschieht durch das Voranstellen des Fluchtzeichens \. Der Editor vi stellt folgende Metazeichen zur Verfügung.

Suchmuster	Bedeutung
.	beliebiges einzelnes Zeichen
.*	beliebige Zeichenkette (auch die leere)
*	beliebige Wiederholung des vorangestellten Zeichens
[...]	eines der Zeichen aus ...
[a-e]	eines der Zeichen von a bis e
[a-eh-x]	eines der Zeichen von a bis e oder h bis x
[^...]	ein beliebiges Zeichen außer ...
\	Fluchtsymbol
^muster	Zeichenkette muster am Anfang der Zeile
muster$	Zeichenkette muster am Ende der Zeile
^muster$	Zeile, die nur aus der Zeichenkette muster besteht
\<muster	Zeichenkette muster am Anfang eines Wortes
muster\>	Zeichenkette muster am Ende eines Wortes

Kommandos zum Suchen

/<reg-exp> Suche von der aktuellen Cursorposition aus vorwärts nach dem regulären Ausdruck <reg-exp>.

?<reg-exp> Suche von der aktuellen Cursorposition aus vorwärts nach dem regulären Ausdruck <reg-exp>.

n Wiederhole das letzte Suchkommando.

N Wiederhole das letzte Suchkommando in umgekehrter Richtung.

:set ic Unterscheide nicht zwischen Groß- und Kleinschreibung.

:set noic Unterscheide zwischen Groß- und Kleinschreibung.

Suchen und Ersetzen

Sollen mehrere gleiche Textteile gesucht und ersetzt werden, so geschieht dies am besten mit dem Substitute-Kommando des ex-Modus.

:[*bereich*]s/*suchmuster*/*ersetzungsmuster*/[gc]

Die Angabe *bereich* wählt dabei die Zeilen aus, die bearbeitet werden sollen:

bereich	Bedeutung
weggelassen	nur die aktuelle Zeile
%	der gesamte Text
a,e	Zeile a bis e
a,$	Zeile a bis zum Ende der Datei
.,e	von der aktuellen Zeile bis Zeile e
e	nur die Zeile e

Die Option g (global) erlaubt mehr als eine Ersetzung pro Zeile. Durch die Option c (confirmation) wird bei jeder Ersetzung nachgefragt.

2.1.5 Puffer und Markierungen

Puffer

vi arbeitet mit einer ganzen Reihe von Puffern:

Pufferbezeichnung	Bedeutung
genereller Puffer	hier wird die zu editierende Datei gehalten
unbenannter Puffer	der temporäre Puffer
"a	die benannten Puffer a bis z
"1	die numerierten Puffer 1 bis 9

Das Zeichen " ist dabei Teil der Syntax. Die benannten Puffer bleiben auch beim Übergang in die nächste Datei erhalten. Die numerierten Puffer enthalten die zuletzt gelöschten Textteile. Folgende Befehle benutzen die Puffer

<puffer>y<obj> Kopiere das Objekt <obj> in den Puffer <puffer>.

<puffer>Y Kopiere die aktuelle Zeile in den Puffer <puffer>.

<puffer>p Füge den Inhalt von Puffer <puffer> nach der aktuellen Cursorposition ein.

<puffer>P Füge den Inhalt von Puffer <puffer> vor der aktuellen Cursorposition ein.

<puffer>d<obj> Löscht das Objekt <obj> und rettet es in den Puffer <puffer>.

<puffer>dd Löscht die aktuelle Zeile und rettet sie in den Puffer <puffer>.

Marken

vi erlaubt, im Text (unsichtbare) Marken zu setzen, die dann zur Cursorpositionierung und zur Auswahl von Textteilen benutzt werden können.

mx Setze die Marke x, wobei x im Bereich a bis z liegt.

‘x Setze den Cursor auf die Marke x.

"by‘a Speichere im Puffer b den Text zwischen der aktuellen Cursorposition und der Marke a.

:’a,’ed Löscht den Text von der Zeile mit der Marke a bis zur Zeile mit der Marke e.

Man beachte, daß bei den ex-Kommandos (die Kommandos mit :) das Zeichen ’ (single quote), bei den vi-Kommandos jedoch das Zeichen ‘ (back quote) benutzt wird.

2.1.6 Weitere Kommandos

Zum Schluß noch einige Kommandos, die in keine Rubrik passen.

J Füge die nachfolgende Zeile an die aktuelle an.

u Mache die letzte Änderung rückgängig.

U Hebe die letzten Änderungen der aktuellen Zeile wieder auf.

:set nu Zeige Zeilennummern an.

:set nonu Zeige keine Zeilennummern an.

:!<com> Führe das Unixkommando <com> aus. Der Output des Kommandos wird am unteren Rand des Bildschirms angezeigt.

. Führe das letzte vi-Kommando noch einmal aus.

<ctrl+g> Zeige den Dateinamen, die aktuelle Zeilennummer und die Gesamtzeilenanzahl an.

Kapitel 3

Die Shell

Der Benutzer kommuniziert in der Regel nicht direkt mit dem Betriebssystem Unix, da dessen Schnittstellen nach außen zu kompliziert und benutzerunfreundlich sind. Über diesen Schnittstellen liegt eine Schicht, welche die Benutzeranweisungen interpretiert und in Betriebssystemaufrufe umsetzt. Außerdem überprüft diese Schicht die Rückmeldungen des Systems und gibt im Fehlerfall entsprechende Fehlermeldungen aus. Diese Schicht heißt unter Unix Shell, weil sie sich wie eine Schale um den Betriebssystemkern legt. Abbildung (3.1) veranschaulicht das grafisch.

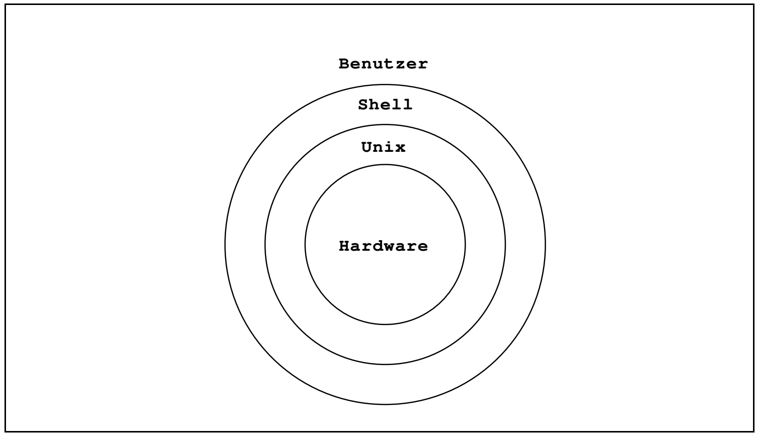

Abbildung 3.1: Die verschiedenen Schichten zwischen Benutzer und Hardware

Die Shell ist kein Bestandteil des Betriebssystemkerns, sondern den Benutzerprogrammen gleichgestellt und leicht austauschbar. Die Shell ist kommandoorientiert und auf die Bedürfnisse eines Programmentwicklers ausgerichtet, d.h. die Kommandosprache ist sehr knapp gehalten und fragt beim Löschen etc. nicht nach. Da die Shell als Programm einfach austauschbar ist, gibt es eine Vielzahl von Varianten:

- `sh`: Die Bourne-Shell ist die Standardshell des Unixsystems.
- `csh`: Die C-Shell ist komfortabler als die Bourne-Shell. Sie enthält Elemente analog der Syntax der Sprache C.
- `ksh`: Die Korn-Shell.
- `bash`: Die Born-Again-Shell.

In diesem Kurs werden wir die C-Shell benutzen, da sie komfortabler als die Bourne-Shell ist. Es soll deshalb auch nur diese Shell erklärt werden. Neben der einfachen Kommandoausführung stellt die Shell eine ganze Reihe weiterer Konzepte zur Verfügung. Hierzu gehören Umlenkungsmechanismen für Ein- und Ausgabe, Dateinamenexpansion bei Verwendung von Metazeichen sowie programmiersprachenähnliche Kontrollstrukturen.

Die Kommandosyntax ist immer gleich:

kommando-name [*optionen*] [*parameter*]

Dabei ist *kommando-name* der Bezeichner eines Shell-internen Kommandos oder der Dateiname eines Programms. Art und Anzahl der Optionen und Parameter sind programmabhängig. Optionen werden normalerweise mit einem Minuszeichen eingeleitet. Beispiel:

```
weber> ls -l clean
-rwxr-xr--  1 hansen          9 Sep 29 17:20 clean
weber>
```

Dabei ist `-l` die Option und `clean` der Parameter. Mehrere Kommandos werden durch Semikolon getrennt

kommando1; *kommando2*; *kommando3*

3.1 Komfortable Kommandoeingabe

Die C-Shell stellt umfangreiche Möglichkeiten zur komfortablen Kommandoeingabe zur Verfügung. Über den sogenannten History-Mechanismus merkt sich die Shell die letzten Kommandos. Auf den meisten Systemen ist es möglich, diese Kommandos zu editieren, wobei die Syntax der vi-Kommandos benutzt wird. Die Erzeugung des endgültigen Kommandos ist dann also ein zweistufiger Vorgang:

1. Erzeugung der Kommandozeile bis zur Übergabe an die Shell mit der Return-Taste.
2. Interpretation der Kommandozeile durch die Shell mit Aliasing, Quoting, Historymechanismus, Variablenauswertung und Dateinamenexpansion.

3.1.1 Der History-Mechanismus der C-Shell

Mit dem Kommando

`set history=` *n*

kann man dafür sorgen, daß sich die C-Shell die n letzten Kommandos merkt. Bei dem Aufruf

`history`

werden einem die letzten n Kommandos angezeigt. Die Kommandos sind fortlaufend durchnumeriert. Mit folgenden Aufrufen kann man diese Kommandos wieder ansprechen:

`!!`	Das letzte Kommando wird nochmal ausgeführt.
`!2`	Das Kommando mit der Nummer 2 wird ausgeführt.
`!v`	Das letzte Kommando, das mit einem 'v' anfängt, wird ausgeführt.

Es können auch Teile der Kommandos angesprochen oder ersetzt und Zusätze gemacht werden.

```
weber> set history=3
weber> ls
clean      test       testdatei  texte
weber> pwd
/usr/home5/hansen
weber> history
    44  ls
    45  pwd
```

```
    46  history
weber> !p
pwd
/usr/home5/hansen
weber> !44 -l clean
ls -l clean
-rwxr-x---  1 hansen          9 Sep 29 17:20 clean
weber>
```

3.1.2 Editieren von Kommandos

Bei den meisten Unix-Systemen mit C-Shell ist ein vi-ähnliches Editieren auch der alten Kommandos möglich. Die Shell verhält sich dann folgendermaßen:

- Die alten Kommandos stehen in einer (imaginären) Datei, ein Kommando pro Zeile.
- Wenn die Shell dem Benutzer das Prompt anzeigt, befindet man sich im Einfügemodus auf der letzten (noch freien) Zeile.
- Durch Esc kommt man in den Kommandomodus.
- Es wird immer nur eine Zeile angezeigt.

Zeigt z.B. das `history`-Kommando folgendes an,

```
weber> history
    81  set history=4
    82  ls
    83  cd texte
    84  history
weber>
```

so würde beim Prompt folgende Tastenfolge [Esc] [k] [k] [A] [␣] [-] [l] [a] [Return] folgendes erzeugen:

```
weber> ^[
ls -la
total 4
drwx------  2 hansen        512 Sep 29 15:38 .
drwxr-xr-x  3 hansen        512 Oct 15 12:00 ..
-rw-r--r--  1 hansen         99 Sep 29 15:26 brief_an_bollmann
-rw-r--r--  1 hansen         70 Sep 29 15:24 prosa
weber>
```

Bitte ausprobieren!

3.1.3 Alias-Funktion der C-Shell

Die C-Shell erlaubt eine Namensersetzung, d.h. für jedes Kommando (erstes Wort einer Kommandostruktur) wird vor seiner Ausführung untersucht, ob eine Ersetzung vorgenommen werden soll. Dieser Mechanismus wird Aliasing genannt. Die Definition einer Ersetzung erfolgt durch

`alias` *kürzel kommandotext*

Kommt nun in einem Kommando *kürzel* als Kommandoname vor, so wird stattdessen von der Shell *kommandotext* eingesetzt und die Interpretation beginnt von vorne. Es können damit geschachtelte Ersetzungen vorkommen, wobei Endlosschleifen erkannt werden. Nach der Alias-Ersetzung wird die History-Ersetzung durchgeführt. Das Aliasing wird nur auf den Kommandonamen angewendet und ist kein allgemeiner Substitutionsmechanismus. Eine Alias-Zuweisung kann durch

`unalias` *kürzel*

wieder aufgehoben werden. Beispiel:


```
weber> alias dir ls -la
weber> dir
total 4
drwx------  2 hansen        512 Sep 29 15:38 .
drwxr-xr-x  3 hansen        512 Oct 15 12:00 ..
-rw-r--r--  1 hansen         99 Sep 29 15:26 brief_an_bollmann
-rw-r--r--  1 hansen         70 Sep 29 15:24 prosa
weber>
```

3.1.4 Dateinamenexpansion bei Verwendung von Metazeichen

Eine der nützlichsten Eigenschaften der Shell ist die Expansion von Dateinamen bei Verwendung von Metazeichen. Dateinamen können also reguläre Ausdrücke sein. Die Verwendung von Metazeichen wurde schon im Abschnitt 2.1.4 erklärt. Bei der Dateinamenexpansion sind die Möglichkeiten etwas eingeschränkter und wie folgt gegeben.

`~` wird von der C-Shell durch den Namen des Heimatverzeichnisses ersetzt.

`*` steht für eine beliebige Zeichenfolge (auch die leere), jedoch nicht für Dateinamen, die mit einem Punkt beginnen.

`?` steht für ein beliebiges einzelnes Zeichen.

`[...]` steht für eines der in Klammern angegebenen Zeichen.

Beispiel:

```
weber> ls -l
total 4
-rwxr-xr--  1 hansen          9 Sep 29 17:20 clean
-rw-r--r--  1 hansen         68 Oct 13 17:50 test
-rw-------  1 hansen          9 Sep 29 15:54 testdatei
drwx------  2 hansen        512 Sep 29 15:38 texte
weber> ls *a*
clean      testdatei
weber>
```

3.1.5 Variablen

Die C-Shell kennt wie die Bourne-Shell Variablen vom Typ Text, Zahl und logischer Wert. Variablennamen dürfen bis zu 20 Zeichen lang sein. Shellvariablen können, soweit sie nicht bereits vordefiniert sind, auf drei Arten deklariert und einen Wert zugewiesen bekommen. Dabei ist darauf zu achten, daß Zwischenräume zwischen '`set`', '`setenv`', '`@`', '`=`', '*name*' und dem Wert stehen müssen.

`set` *name* `=` *text*

Hierdurch wird die Shellvariable *name* deklariert und erhält als Wert die Zeichenkette *text*. Der Gültigkeitsbereich ist lokal. Kommen in *text* Trennzeichen vor, so ist die Form "*text*" oder '*text*' zu wählen.

`@` *name* `=` *numerischer-ausdruck*

Hierdurch wird die Shellvariable *name* deklariert und erhält als numerischen Wert das Ergebnis der Auswertung von *numerischer-ausdruck*. Der Gültigkeitsbereich ist lokal.

`setenv` *name* `=` *text*

Hierdurch wird die Shellvariable *name* deklariert und erhält als Wert die Zeichenkette *text*. Der Gültigkeitsbereich ist global.

Der Inhalt der Variablen wird durch `$`*name* bzw. `${`*name*`}`, falls in *name* Sonderzeichen vorkommen, angegeben. Die C-Shell ermöglicht sogar die Deklaration von Feldern, d.h. Shellvariablen mit mehreren Elementen. Diesen Elementen müssen bei der Definition sogleich Werte zugewiesen werden.

`set` *name* `= (` *text1 text2 ...*`)`

Mit `$`*name* ist dann das ganze Feld gemeint. Die Shellvariable `$#`*name* gibt die Anzahl der Elemente des Feldes `$`*name* an. Der Zugriff auf die einzelnen Feldelemente geschieht wie bei der Programmiersprache C durch eckige Klammern.

Die C-Shell kennt neben den vom Benutzer definierten Shellvariablen unter anderem die nachfolgend beschriebenen Variablen.

`$argv` In dieser Variablen befinden sich die Argumente (Parameter) des Prozeduraufrufs. Analog zu einem C-Programm gilt:

- `$argv[0]` Dies ist der Name des aufgerufenen Programms selbst. Dies kann entsprechend der Bourne-Shell mit `$0` abgekürzt werden.
- `$argv[1]` Hier steht der 1. Parameter. Dies kann mit `$1` abgekürzt werden.
- `$argv[`*n*`]` Hier steht der n-te Parameter. Dies kann mit `$`*n* abgekürzt werden.
- `$argv[*]` Diese Variable enthält die gesamte Parameterliste und ist äquivalent zu `$*`.
- `$#argv` Hier ist die Anzahl der Parameter beim Aufruf zu finden.

`$cwd` Dies ist der volle Pfadname des aktuellen Verzeichnisses.

`$history` Der Wert der Variablen legt fest, wieviele Kommandos im History-Speicher festgehalten werden sollen.

`$home` Wird `cd` ohne Parameter aufgerufen, so wird das in `$home` stehende Verzeichnis zum aktuellen Verzeichnis. Außerdem substituiert die C-Shell das Metazeichen `~` in Dateinamen durch dieses Verzeichnis.

`$path` Die Variable enthält eine Reihe von Verzeichnissen, die der Reihe nach durchsucht werden, wenn die Shell ein externes Kommando ausführen soll und kein absoluter Pfadname benutzt wurde.

`$prompt` Kommt im Prompt ein '`!`' vor, so wird dafür von der C-Shell eine fortlaufende Nummer eingesetzt.

`$<` Es wird eine Zeile von Standard-Eingabe gelesen und als Text in dieser Variablen zurückgegeben.

Zur Ausgabe von Text und Variableninhalten wird das Kommando

`echo` [*argumente*]

benutzt. Die *argumente* werden von der Shell nach deren Regeln expandiert.

3.1.6 Quoting

Manchmal ist es notwendig, gewisse Ersetzungsmechanismen der Shell außer Kraft zu setzen. Dies wird unter Unix als „Quoting“ bezeichnet, wobei folgende Zeichen benutzt werden.

\ Als Fluchtsymbol für das nachfolgende Zeichen.

`$` Erzeugt eine Parametersubstitution, d.h. der Wert einer Variablen wird eingesetzt.

''*text*'' In der Zeichenkette *text* wird die Dateinamenexpansion verhindert. Andere Ersetzungen sind möglich.

‘*kommando*‘ Das Kommando *kommando* wird ausgeführt und sein Output als Text eingesetzt.

'*text*' In der Zeichenkette *text* wird keinerlei Substitution vorgenommen.

Beispiel:

```
weber> set history = 4
weber> set prompt = "\!:‘hostname‘:‘pwd‘> "
3:weber:/usr/home5/hansen> ls
clean          test           testdatei      texte
4:weber:/usr/home5/hansen>
```


3.2 Umlenkungsmechanismen, Filter und Pipes

3.2.1 Umlenkungsmechanismen

Unix behandelt Geräte wie Dateien. Alle Gerätedateien sind im Verzeichnis `/dev` registriert.

Normalerweise liest ein Programm von Standard-Input und schreibt auf Standard-Output. Die Fehler werden auf Standard-Error geschrieben. Dabei identifiziert die Shell den Standard-Input mit der Tastatur, den Standard-Output und Standard-Error mit dem Bildschirm. Diese Standardzuweisungen können aber auch umgelenkt werden, so daß von einer Datei gelesen bzw. auf eine Datei geschrieben werden kann. Zur Umlenkung des Standard-Input wird das Zeichen `<` benutzt.

kommando `<` *infile*

Zur Umlenkung des Standard-Output wird das Zeichen `>` benutzt.

kommando `>` *outfile*

In diesem Fall wird eine eventuell schon vorhandene Datei *outfile* überschrieben! Soll der Standard-Output an eine Datei angehängt werden, so benutzt man die Zeichen `>>`.

kommando `>>` *outfile*

In beiden Fällen würden Fehlermeldungen trotzdem auf dem Bildschirm erscheinen. Die C-Shell bietet die Möglichkeit, Standard-Output und Standard-Error in eine gemeinsame Datei umzuleiten.

kommando `>&` *outfile*

Schließlich gibt es noch eine andere Art der Eingabe, die als „Here Document“ bezeichnet wird.

kommando `<<` *endstring*

Statt von Standard-Input werden jetzt die nachfolgenden Zeilen gelesen, und zwar solange, bis eine Zeile lediglich *endstring* enthält. Beispiel:

```
weber> cat > test << EOF
Dies ist ein Test.
So koennte man ohne Editor einen Text eintippen.
EOF
weber> cat test
Dies ist ein Test.
So koennte man ohne Editor einen Text eintippen.
weber>
```

In der Regel werden solche Konstruktionen in Kommandoprozeduren eingesetzt.

3.2.2 Filter

Filter sind Programme, die Text von Standard-Input lesen und irgendwie modifiziert auf Standard-Output schreiben. Wird einem Filter als Parameter ein Dateiname gegeben, so wird statt von Standard-Input von der Datei gelesen. Die wichtigsten Filter sind `wc`, `sort`, `grep` und der Editor `sed`.

Das Kommando

`wc` [*datei ...*]

zählt die Zeilen, Wörter und Zeichen von Dateien. Das Kommando

`sort` [*datei*]

sortiert die Zeilen von *datei* lexikalisch. Das Kommando

`grep` [`-i`] *regular-expression* [*datei ...*]

sucht in den angegebenen Dateien nach dem regulären Ausdruck *regular-expression* und gibt jede Trefferzeile aus. Mit der Option `-i` wird nicht zwischen Groß- und Kleinschreibung unterschieden.

3.2.3 Pipes

Soll in einer Kommandosequenz die Ausgabe des einen Kommandos als Eingabe des nachfolgenden Kommandos benutzt werden, so kann dies durch das Pipe-Symbol | geschehen.

kommando1 | *kommando2* | ...

Pipes werden oft in Kombination mit Filtern benutzt. Beispiel:

```
weber> ls -l | sort
-rw-------  1 hansen          9 Sep 29 15:54 testdatei
-rw-r--r--  1 hansen         68 Oct 13 17:50 test
-rwxr-xr--  1 hansen          9 Sep 29 17:20 clean
drwx------  2 hansen        512 Sep 29 15:38 texte
total 4
weber> ls -l | wc
       5      34     215
weber>
```

3.3 Hintergrundprozesse

Bereits im Abschnitt 1.4 wurde das Zeichen `&` zum Absetzen von Hintergrundprozessen eingeführt.

kommando `&`

Es soll jetzt erklärt werden, wie man beliebige Prozesse vom Vordergrund in den Hintergrund und zurück schickt. Betrachten wir zunächst die Möglichkeiten, einen Prozeß anzuhalten. Mit

`<ctrl+d>`

wird an einen auf Input wartenden Job ein Dateiendezeichen gesandt. Mit

`<ctrl+c>`

wird an einen Job ein Interrupt-Signal gesendet. Manche Kommandos werden daraufhin (vorzeitig) beendet, andere ignorieren dies. Mit

`<ctrl+z>`

wird ein Kommando nur angehalten, dies aber sicher. Wir haben bereits erfahren, daß mit dem `ps`-Kommando ermittelt werden kann, welche Prozeßnummer der Prozeß hat. Mit dem `kill`-Kommando kann man dann diesen Prozeß entfernen. Die C-Shell selber verwaltet gestoppte Prozesse und Hintergrundprozesse unter der Vergabe von Jobnummern. Diese kann man mit dem Kommando

`jobs`

auflisten. Mit dem Kommando

`fg` [`%`*jobnummer*]

wird der Job in den Vordergrund geholt und mit

`bg` [`%`*jobnummer*]

in den Hintergrund geschickt. Beispiel:

```
35:weber:/usr/home5/hansen> set prompt = "`hostname`> "
weber> clean
^Z
Stopped
weber> jobs
[1]  + Stopped              clean
weber> bg
[1]    clean &
weber> jobs
[1]    Running              clean
weber>
[1]    Done                 clean

weber>
```


3.4 Shellscripts

Shellscripts sind ASCII-Dateien, die Unix-Befehle enthalten. Sie werden benutzt, wenn bestimmte (kompliziertere) Aufgaben, die durch immer wiederkehrende Kommandofolgen bearbeitet werden, des öfteren erledigt werden müssen. Es muß dann lediglich der Name des Shellscripts aufgerufen werden, um die Aufgabe zu lösen.

Eine spezielle Bedeutung haben dabei die folgenden im Heimatverzeichnis abgelegten Dateien

- `.profile` Diese Datei wird automatisch nach dem Einloggen ausgeführt, wenn die Bourne-Shell (`/bin/sh`) die für den jeweiligen Benutzer in der Datei `/etc/passwd` eingetragene Login-Shell ist.
- `.cshrc` Diese Datei wird automatisch nach dem Start einer C-Shell (`/bin/csh`) ausgeführt.
- `.login` Ist die C-Shell auch die Login-Shell, so wird diese Datei nach `.cshrc` ausgeführt.
- `.logout` Ist die C-Shell auch die Login-Shell, so wird diese Datei beim Beenden der Shell ausgeführt.

Da es Unterschiede zwischen Bourne- und C-Shell bzgl. der Abarbeitung von Shellscripts gibt, müssen alle Shellscripts (außer den obigen) entsprechend gekennzeichnet werden. Steht in der ersten Zeile des Shellscripts ein `#!/bin/sh` oder ist die erste Zeile eine Leerzeile, wird das Shellscript in der Bourne-Shell abgearbeitet. Steht in der ersten Zeile des Shellscripts ein `#!/bin/csh`, so wird das Shellscript in der C-Shell abgearbeitet. Um ein Shellscript zu starten, muß es Lese- und Ausführungsrecht haben, siehe Abschnitt 1.3.3.

Kommentare in Shellscripts werden mit dem Zeichen `#` eingeleitet. Es ist jeweils ein `#` pro Zeile zu verwenden.

3.4.1 Kontrollstrukturen für C-Shellscripts

Die C-Shell verfügt (natürlich) auch über Kontrollstrukturen, um variable Shellscripts zu schreiben. Die Syntax lehnt sich eng an die Programmiersprache C an. Zuweisungen an Variablen erfolgen mit den im Abschnitt 3.1.5 gegebenen Befehlen. Zur Bildung von Ausdrücken werden unter anderem folgende Operatoren verwendet.

`+`	Addition
`-`	Subtraktion
`*`	Multiplikation
`/`	Divison
`<`	Vergleich, ob kleiner
`<=`	Vergleich, ob kleiner oder gleich
`==`	Vergleich auf textuelle Gleichheit
`!=`	Vergleich auf textuelle Ungleichheit
`=~`	Vergleich auf textuelle Gleichheit, wobei der linke Ausdruck ein regulärer Ausdruck sein darf
`&&`	logisches UND
`||`	logisches ODER
`!`	logische Negation
`(...)`	Durch die Klammerung kann die Reihenfolge der Operationen vorgegeben werden.
`-s` *datei*	Liefert den Wert *wahr*, falls *datei* vorhanden und größer Null Byte ist.
`-d` *datei*	Liefert den Wert *wahr*, falls *datei* ein Verzeichnis ist.
`-e` *datei*	Liefert den Wert *wahr*, falls *datei* existiert.
`-f` *datei*	Liefert den Wert *wahr*, falls *datei* eine normale Datei ist.

`-o` *datei* Liefert den Wert *wahr*, falls *datei* dem Benutzer gehört.

`-r` *datei* Liefert den Wert *wahr*, falls *datei* gelesen werden kann.

`-w` *datei* Liefert den Wert *wahr*, falls *datei* modifiziert werden kann.

`-x` *datei* Liefert den Wert *wahr*, falls *datei* ausgeführt werden kann.

`-z` *datei* Liefert den Wert *wahr*, falls *datei* die Länge Null hat.

Ist ein Ausdruck *wahr*, so hat er den Wert 1. Es ist möglich zu testen, ob ein Programm bzw. Kommando fehlerfrei beendet wird. Der Ausdruck '`{ command }`' hat den Wert *wahr* bzw. 1, wenn `command` fehlerfrei beendet wird, also den Exit-Status Null liefert. Liefert `command` einen von Null verschiedenen Exit-Status, so erhält der Ausdruck den Wert 0. Damit sind die nun folgenden Kontrollstrukturen möglich:

Die IF-Anweisung
Die if-Anweisung existiert in drei Varianten:

```
if (ausdruck) kommando
```

```
if (ausdruck) then
kommandofolge
endif
```

```
if (ausdruck) then
kommandofolge-1
else
kommandofolge-2
endif
```

Die *kommandofolge* kann aus mehreren Befehlen bestehen, die durch Semikolon oder Neue Zeile getrennt sind.

Die FOREACH-Schleife
Die Syntax dieser Anweisung sieht wie folgt aus:

```
foreach index ( argumente )
kommandofolge
end
```

Für jedes der in *argumente* angegebene Elemente wird die *kommandofolge* einmal durchlaufen, wobei der Schleifenindex *index* nacheinander die Werte in der Liste *argumente* annimmt.

Die WHILE-Schleife
Die Syntax dieser Anweisung sieht wie folgt aus:

```
while ( ausdruck )
kommandofolge
end
```

Die Kommandos in *ausdruck* oder der dort stehende Vergleich wird ausgeführt. Ist das Ergebnis Null, so werden die Anweisungen in *kommandofolge* ausgeführt und der Zyklus beginnt von vorne.

Bei den Schleifenkonstruktionen `foreach` und `while` kann mit

```
break
```

die Kommandoschleife verlassen werden. Das Kommando

```
continue
```

erlaubt, an das Ende der Schleife zu springen.

Die REPEAT-Anweisung
Die Syntax dieser Anweisung sieht wie folgt aus:

```
repeat  n anweisung
```


Hierbei wird das Kommando in *anweisung* *n* mal ausgeführt.

Fallunterscheidung mit SWITCH
Die Syntax dieser Anweisung sieht wie folgt aus:

```
switch ( text )
case muster1: kommandofolge [;breaksw]
.
.
.
case mustern: kommandofolge [;breaksw]
[default: kommandofolge]
endsw
```

Hierbei wird *text* (in der Regel der Inhalt einer Shellvariablen) mit den verschiedenen Mustern (*muster1* ... *mustern*) verglichen. Paßt *text* auf ein Muster, so wird die dahinterstehende *kommandofolge* ausgeführt. Wie bei der Sprache C wird dann beim nächsten Muster fortgesetzt, es sei denn, daß der Vergleich durch die `breaksw`-Anweisung explizit abgebrochen wird. Mit `default` kann eine Anweisungssequenz angegeben werden, die ausgeführt wird, wenn keines der Muster zutrifft.

Die Sprunganweisung GOTO
Die Anweisung

```
goto marke
```

erlaubt einen expliziten Sprung zu der angegebenen Marke. Diese Marke wird wie in C geschrieben.

```
marke:
```

3.4.2 Beispiele für Shellscripts

Beispiel 1:
Das folgende Beispiel demonstriert unter anderem den rekursiven Aufruf von Shellscripts. Dabei muß das Shellscript `baum` von allen Stellen her als Programm aufrufbar sein. Die Datei `baum` liegt deshalb im Verzeichnis `~/bin`, welches im Suchpfad, d.h. in der Variablen `$path` enthalten sein muß.

```
weber> cd
weber> cd bin
weber> cat baum
#!/bin/csh
# Dieses Shellscript erlaubt das Durchlaufen von Dateib"aumen.
#
if ( $#argv != 0 ) cd $1
set katalog = "`pwd`"
echo "Dateien im Verzeichnis ${katalog}:"
ls $katalog
foreach i (*)
  if ( -d $i ) then
     cd $i
     baum
     cd $katalog
  endif
end
weber> cd
weber> baum
Dateien im Verzeichnis /usr/home5/hansen:
benutzer        clean           testdatei
bin             test            texte
Dateien im Verzeichnis /usr/home5/hansen/bin:
```

```
baum
Dateien im Verzeichnis /usr/home5/hansen/texte:
brief_an_bollmann       prosa
weber>
```

Man beachte, daß bei jedem rekursiven Aufruf von `baum` eine neue Shell gestartet wird, so daß unter Umständen sehr viele Prozesse erzeugt werden.

Beispiel 2:
Das folgende Beispiel demonstriert unter anderem eine numerische Berechnung.

```
weber> cat countdown
#!/bin/csh
@ i = $1
while ( $i )
        echo $i
        @ i = $i - 1
        sleep 1
end
echo Ignition!
weber> countdown 3
3
2
1
Ignition!
weber>
```

Kapitel 4

Drucken, Datensicherung und andere nützliche Kommandos

4.1 Drucken

Da Unix ein multi-user, multi-tasking Betriebssystem ist, sollte man normalerweise Dateien nicht direkt zum Drucker schicken, sondern das sog. Druckerspoolsystem benutzen. Dieses verwaltet eine Warteschlange, damit die verschiedenen Dateien nicht gemischt werden. Außerdem kann der Druckerspooler so konfiguriert werden, daß für jeden Druckjob ein Deckblatt mit dem Benutzernamen ausgegeben wird.

Die Unterschiede zwischen dem System V (AT&T) und dem Berkeley Druckservice sind sehr groß. System V bietet den lp und Berkeley den lpr.

System V Druckservice
Mit dem Kommando

`lp [-d`*drucker*`] [`*datei ...*`]`

wird die Datei *datei* auf dem Drucker mit dem Namen *drucker* ausgegeben. Fehlt die Angabe *datei*, so wird von Standard-Input gelesen, siehe Kapitel 3.2.1. Informationen über die Warteschlange erhält man mit

`lpstat`

Druckjobs löschen kann man mit dem Kommando

`cancel` *druckjobnummer*

Berkeley Druckservice
Mit dem Kommando

`lpr [-P`*drucker*`] [`*datei ...*`]`

wird die Datei *datei* auf dem Drucker mit dem Namen *drucker* ausgegeben. Fehlt die Angabe *datei*, so wird von Standard-Input gelesen, siehe Kapitel 3.2.1. Informationen über die Warteschlange erhält man mit

`lpq [-P`*drucker*`]`

Druckjobs löschen kann man mit dem Kommando

`lprm [-P`*drucker*`]` *druckjobnummer*

Auf den DEC-Stations wird unter Ultrix der Berkeley Druckservice benutzt!

4.2 Datensicherung

Das wahrscheinlich mächtigste Unix-Kommando ist `find` *verzeichnis ... ausdruck*

`find` durchsucht rekursiv die angegebenen Verzeichnisse *verzeichnis ...* nach Dateien, die in *ausdruck* ausgewählt werden. In *ausdruck* wird auch spezifiziert, was bei Treffern geschehen soll. In den folgenden Beschreibungen der Parameter für *ausdruck* ist *n* ein dezimaler Integer, wobei *+n* mehr als *n*, *-n* weniger als *n* und *n* exakt *n* bedeutet.

`-atime` *n*	Wahr, falls die Datei vor *n* Tagen benutzt wurde.
`-ctime` *n*	Wahr, falls die Datei vor *n* Tagen angelegt wurde.
`-exec` *command*`;`	Führt das angegebene Kommando aus, wobei die fehlerfreie Ausführung den Exitstatus 0 (wahr) liefert. Die aktuelle Datei kann mit den Zeichen '{}' angegeben werden.
`-mtime` *n*	Wahr, falls die Datei vor *n* Tagen modifiziert wurde.
`-name` *filename*	Wahr, falls die Datei den Namen *filename* hat. Bei Verwendung von Metazeichen muß '*filename*' benutzt werden, um die Expansion durch die Shell zu verhindern.
`-newer` *file*	Wahr, falls die Datei später als *file* modifiziert wurde.
`-perm` *ooo*	Wahr, falls die Datei die Zugriffsrechte der Oktalzahl *ooo* hat, siehe auch den Befehl `chmod` im Kapitel 1.3.3 auf Seite 10.
`-print`	Gibt den Dateinamen mit vollständigem Pfad aus.
`-size` *n*	Wahr, falls die Datei *n* Blocks groß ist, wobei ein Block 512 Byte hat.
`-user` *uname*	Wahr, falls die Datei *uname* gehört.

Auf diese Auswahlmöglichkeiten können außerdem noch logische Verknüpfungen angewandt werden.

- Klammerung
  Es können Klammern gesetzt werden, wobei diese vor der Shell geschützt werden müssen, also z.B. `\( \)`
- Negierung
  Für die Negierung (not) wird das Zeichen '!' benutzt.
- UND-Verknüpfung
  Die UND-Verknüpfung wird implizit angewandt.
- ODER-Verknüpfung
  Die ODER-Verknüpfung wird durch den Operator `-o` erreicht.

Die Fülle der Optionen dieses Kommandos bedarf sicherlich einiger Beispiele.

```
weber> find ~ \( -name a.out -o -name '*.o' \) -atime +7 -exec rm {} \;
weber>
```

Löscht alle Dateien, die den Namen `a.out` oder `*.o` haben und mindestens eine Woche nicht mehr benutzt wurden. Es werden die Dateien berücksichtigt, die im Heimatverzeichnis oder deren Unterverzeichnissen liegen.

```
weber> find . -size +5 -exec ls -l {} \;
-rw-r--r--  1 hansen       5030 Sep 29 15:09 ./.Xdefaults
-rw-r--r--  1 hansen      83551 Oct 27 14:32 ./printscreen.ps
weber>
```

Suche alle Dateien, die größer als 5 Blöcke sind, und gebe sie mit `ls -l` aus.

Das Kommando

`compress [-c]` [*datei*]

komprimiert Dateien nach dem recht effektiven Lempel-Ziv Algorithmus. Die komprimierte Datei erhält den Namen *datei*`.Z`. Wird *datei* nicht angegeben, so wird von Standard-Input gelesen und nach Standard-Output geschrieben. `compress` ist also auch ein Filter. Die Entkomprimierung erfolgt durch das Kommando

`uncompress [-c]` [*datei*`.Z`]

Die Option `-c` läßt `compress` und `uncompress` nach Standard-Output schreiben. Das Kommando

`tar` *key* [*datei ...*]

(**t**ape **ar**chiver) liest/schreibt Dateien von/nach einer einzelnen Datei. Diese Datei ist standardmäßig die Gerätedatei für das Bandlaufwerk. Der Ausdruck *key* ist eine Zeichenkette, die einen Buchstaben zur Auswahl der Funktion und weitere Optionen und Dateinamen enthält. Dabei meint ein Verzeichnisname den ganzen Baum ab diesem Verzeichnis.

Die Funktionsauswahl in *key* erfolgt über folgende Buchstaben:

`c`	Neuanlegen eines Archivs und schreiben.
`r`	Schreibe die Dateien an das Ende des Archivs.
`t`	Liste die Dateinamen im Archiv.
`u`	Schreibe die angegebenen Dateien in das Archiv, wenn sie dort noch nicht existieren oder seit dem letzten Schreiben modifiziert wurden.
`x`	Lies die angegebenen Dateien vom Archiv. Default sind alle Dateien des Archivs.

Außerdem sind noch folgende Optionen wichtig:

`f`	Benutze statt `/dev/rmt0h` das nächste Argument als Name des Archivs. Ist dies '`-`', so liest `tar` von Standard-Input bzw. schreibt auf Standard-Output. `tar` kann also mit Pipes benutzt werden.
`v`	Schreibe die Namen der Dateien nach Standard-Error.

Das Kommando `cpio` ist auf den ersten Blick dem `tar`-Kommando recht ähnlich. Es wird in 3 verschiedenen Formen benutzt:

`cpio -o`[*optionen*]

liest von Standard-Input Dateinamen und schreibt diese Dateien nach Standard-Output.

`cpio -i`[*optionen*] [*namensmuster*]

liest von Standard-Input Dateien, die durch *namensmuster* ausgewählt werden können.

`cpio -p`[*optionen*]

wirkt wie `-o`, wobei jedoch nach dem Schreiben sofort wieder eingelesen wird. Der wesentliche Unterschied zwischen `tar` und `cpio` ist, daß `cpio -o` von Standard-Input **Dateinamen** liest, während `tar` die **Dateien** liest. Deshalb erhält `cpio -o` normalerweise die Dateinamen über eine Pipe vom Kommando `find`, während `tar` komplette Verzeichnisbäume sichert. Beispiel:

```
weber> tar -cvf - texte | compress > texte.tar.Z
a texte/
a texte/prosa 1 blocks
a texte/test 1 blocks
a texte/brief_an_bollmann 1 blocks
weber> rm texte/*
weber> rmdir texte
weber> uncompress -c texte.tar.Z | tar -vxf -
x texte/
x texte/prosa, 70 bytes, 1 blocks
x texte/test, 68 bytes, 1 blocks
x texte/brief_an_bollmann, 99 bytes, 1 blocks
weber> find . -atime -2 -print | cpio -o | compress > sicher.cpio.Z
14 blocks
weber>
```

In diesem Beispiel wird zuerst das komplette Verzeichnis `texte` in die Datei `texte.tar.Z` gesichert. Danach wird das Verzeichnis `texte` gelöscht, was im Anwendungsfall versehentlich geschehen würde. Anschließend wird die Sicherungsdatei `texte.tar.Z` wieder ausgepackt. Das letzte Kommando sichert alle Dateien, die in den letzten zwei Tagen benutzt wurden.

4.3 Andere Benutzer

Als Werkzeug für im Team arbeitende Programmentwickler stellt Unix mehrere Kommandos zur Kommunikation zwischen Benutzern zur Verfügung. Das Kommando

`who`

zeigt an, welche User eingeloggt sind, was nicht notwendigerweise heißt, das diese auch arbeiten `;-)`. Das Kommando

`who am i`

zeigt an, welcher User eingeloggt ist. Dies kann z.B. nützlich sein, wenn man ein Terminal findet, bei dem sich der vorherige Benutzer nicht ausgeloggt hat. Mit dem Kommando

`write` *benutzer*

kann man einem eingeloggten Benutzer eine Nachricht auf den Bildschirm schreiben. Dieses Kommando liest von Standard-Input. Derartigen Unfug kann der angesprochene durch das Kommando

`mesg n`

unterbinden oder mit

`mesg y`

erlauben. Ist der Benutzer nicht eingeloggt, so kann man statt dessen einen elektronischen Brief (E-Mail) verschicken. Dazu wird das Kommando

`mail` *benutzer*

Dieses Kommando liest ebenfalls von Standard-Input. Fehlt die Angabe *benutzer*, so werden die Briefe im eigenen Briefkasten angezeigt. Im Kapitel 6 wird gezeigt, daß E-Mail weltweit verschickt werden kann.

4.4 Weitere Kommandos

Das Kommando

`cal` [*monat*] [*jahr*]

gibt einen Kalender für den angegebenen Monat auf dem Bildschirm aus. Das Kommando

`date`

gibt das aktuelle Datum und die Uhrzeit aus. Nur der Systemverwalter kann Datum und Uhrzeit verändern. Das Kommando

`file` *datei*

versucht, den Typ einer normalen Datei zu bestimmen. Der Typ kann z.B. 'ASCII text', 'PostScript (v2.0) text', 'compressed data' oder 'English text' sein.

In Kapitel 1.3.2 wurde gezeigt, daß unter Unix die Dateien in einem hierarchischen Dateisystem organisiert sind. Beim Bezug auf eine Datei stellt die Shell zuerst den absoluten Pfadnamen fest. Unix wird dann an der entsprechenden Stelle auf der Festplatte die Information finden, an welcher Stelle auf der Festplatte (Kopf, Spur, Sektor) der eigentliche Dateiinhalt steht.

Unix bietet nun die Möglichkeit, auf diesen Dateiinhalt mit weiteren Dateinamen zuzugreifen, die sogenannten Links. Es gibt zwei Arten von Links:

- **Hard-Links**
  Hier wird unter einem anderen Dateinamen dieselbe Adresse für den Dateiinhalt eingetragen. Danach kann nicht mehr zwischen Original und Linkeintrag unterschieden werden. Bei dieser Art des Links müssen sich die Pfadnamen auf demselben physikalischen Gerät befinden (genauer auf demselben Filesystem). Um den Dateiinhalt zu löschen, müssen sowohl der Originaleintrag als auch der Hard-Link gelöscht werden.

- **Soft-Links**
  Hier wird unter dem neuen Dateinamen der Dateiname der Originaldatei abgespeichert. Hier unterscheidet sich also der Linkeintrag vom Originaleintrag. Dafür kann der Linkeintrag auch auf einem anderen physikalischen Gerät erfolgen. Der Soft-Link kann mit `rm` wieder entfernt werden. Wird dagegen die Originaldatei gelöscht, so zeigt der Linkeintrag auf eine Karteileiche.

Der Link wird mit dem Kommando

`ln` [`-s`]*originaldatei linkeintrag*

hergestellt, wobei die Option `-s` einen Soft-Link erzeugt.

Das Kommando

`stty`

zeigt Terminaleinstellungen an und erlaubt deren Veränderung. So wird z.B. mit

`stty erase ^H`

die Backspace-Taste richtig eingestellt, wobei '`^H`' ein Zeichen ist, das beim Drücken der Backspace-Taste entsteht. Das Kommando

`sleep` *sekunden*

verschiebt die Ausführung der nachfolgenden Kommandos um die angegebene Zeit.

Kapitel 5

Programmentwicklung unter Unix

Unix wurde als ein System entworfen, welches speziell für die Programmentwicklung geeignet sein sollte. Aus diesen Gründen enthält es für die eigentliche Implementierungsphase eines Programms oder Programmsystems eine ganze Reihe von Hilfsmitteln.

Hierzu gehören auch zahlreiche Compiler und Interpreter. Eine Sonderstellung nimmt dabei die Sprache C ein, da Unix (fast ausschließlich) in C geschrieben ist. Der Zugriff auf die Systemfunktionen erfolgt deshalb bei allen Programmiersprachen über C-Funktionen. Damit ist natürlich ein großer Synergieeffekt zwischen Unix und C gegeben. Allerdings erzielen bei numerischen Anwendungen in Fortran implementierte Programme oftmals ein besseres Laufzeitverhalten, speziell auf Hochleistungsrechnern.

Fast alle angebotenen Compiler, sicher aber C, Fortran und Assembler, erlauben die getrennte Übersetzung einzelner Moduln. Diese müssen anschließend mit Hilfe des Binders (Linker) zusammengebunden werden. Der Linker trägt den Namen `ld` (**l**inking e**d**itor).

Der Aufruf von Präprozessor, Compiler und Linker ist normalerweise in einem Kommando zusammengefaßt. Der Aufruf des C-Systems lautet

`cc` [*optionen*] *datei ...*

und für Fortran wird auf den meisten Systemen der Befehl

`f77` [*optionen*] *datei ...*

benutzt, so daß im allgemeinen der Linker nie explizit aufgerufen wird.

Die Größe eines Programms erhält man mit Hilfe des Kommandos

`size` [*optionen*] *datei ...*

Dies gibt jedoch die statischen Größen an. Viele Programme vergrößern selbständig ihren Datenbereich zur Laufzeit entsprechend den verarbeiteten Daten. Die tatsächliche Größe läßt sich dann mit dem `ps`-Kommando ermitteln.

Das Kommando

`time` *program*

mißt die Ausführungszeit eines Programms, wobei sowohl die Verweildauer des Programms im Rechner (elapsed time), als auch die verbrauchte CPU-Zeit im Benutzer- und Systemmodus ermittelt werden. Darüber hinaus gibt es Hilfsmittel, die eine genauere Analyse nach einzelnen Unterprogrammaufrufen etc. erlauben.

5.1 Präprozessor

Häufig hat man bei der Programmentwicklung die Situation, daß eine Reihe von Konstantendefinitionen und Makros in mehreren Moduln verwendet werden. Unter Unix ist es dabei üblich, diese Definitionen (z.B. auch Common-Blöcke) in eigenen sogenannten Includedateien abzulegen und sich in den Moduln mit einer Anweisung

`include '`*datei*`.h'`

auf die entsprechende Datei zu beziehen. Die textuelle Einfügung der Datei erfolgt dann durch den `cpp`-Präprozessor, der von den meisten Compilern vor dem eigentlichen Übersetzungslauf aufgerufen wird.

Speziell bei der Programmierung in C werden weitere Präprozessorfunktionen benutzt. Es sind dies die Definition von Konstanten und Makros. Außerdem sind bedingte Übersetzungen möglich.

5.2 Compiler

Jede der Sprachen erwartet eine bestimmte Endung im Namen der Quelltextdateien. Dies sind

`.c`	C-Quelltext
`.f`	FORTRAN-77 Quelltext
`.p`	Pascal-Quelltext
`.s`	Assemblerprogramm

Die Compiler produzieren sogenannte Objektdateien, deren Namen die Endung `.o` tragen und zusammen mit Objektbibliotheken die Eingabe für den Linker `ld` darstellen. Der Aufruf des FORTRAN-77 Compilers lautete

`f77` [*optionen*] *datei ...*

Das `f77`-Kommando selbst untersucht die angegebenen Optionen und Dateien und erkennt, ob und mit welchen der Optionen der Präprozessor, Compiler und Linker aufzurufen sind.

Quelltextdateien, d.h. Dateien mit der Endung `.f`, werden dem Compiler übergeben. Die wichtigsten Optionen für den Compiler sind:

`-C`	Das ausführbare Programm prüft, ob Bereichsgrenzen von Indizes überschritten werden. Allerdings werden Programme dadurch merklich langsamer.
`-c`	Der Linker wird *nicht* aufgerufen.
`-g`	Schreibt in die Objektdateien Informationen über Zeilennummern und symbolische Namen des Quelltextes, die vom Debugger benutzt werden. Das Programm wird dadurch größer. Außerdem wird normalerweise die Optimierung abgeschaltet, so daß das Programm auch langsamer wird.
`-O`*n*	Steuert die Optimierung, wobei n=0 die Optimierung abschaltet. Je größer n, desto höher die Optimierung. Allerdings wird damit auch die Compilationszeit größer und die Gefahr von falscher Übersetzung steigt. Es empfiehlt sich also, ab und an zu überprüfen, ob optimierter und nicht optimierter Code gleiche Ergebnisse erzeugen! `-O2` ist default.
`-o`*datei*	Die ausführbare Datei soll *datei* heißen. Default ist `a.out`. Diese Information wird an den Linker durchgereicht.
`-r8`	Übersetzt ein Single-Precision Programm als Double-Precision Programm.

5.3 Linker

Der Linker ist im Programmerstellungszyklus das letzte Glied. Er bindet die einzelnen Moduln zu einem einzigen ablauffähigen Modul zusammen: der Programmdatei. Dabei werden in der Regel noch die verwendeten Funktionen des jeweiligen Laufzeitsystems hinzugefügt. Der Linker ist in der Lage, Objektbibliotheken nach Symboldefinitionen zu durchsuchen und die entsprechenden Moduln aus der Bibliothek selektiv in die Ausgabe einzubinden. Der Linker verarbeitet also zwei Arten von Input-Dateien:

`.o`	Objektdateien
`.a`	Objektbibliotheken

Der Aufruf des Linkers erfolgt explizit mit dem Kommando

`ld` [*optionen*] *datei ...*

oder implizit mit

`f77` [*optionen*] *datei ...*

Die wichtigsten Optionen sind:

`-l`*name* Ist eine Abkürzung für die Bibliothek `/lib/lib`*name*`.a`

`-o`*datei* Die ausführbare Datei soll *datei* heißen. Default ist `a.out`.

5.4 Bibliotheken

Mit Hilfe von `ar` können mehrere Dateien zu einer Bibliothek (library) zusammengefaßt werden. Zumeist werden dabei Objektdateien zu einer Objektbibliothek angeordnet, welche als Eingabe für den Linker dienen kann. Dies hat folgende Vorteile:

- geringfügige Platzersparnis
- bessere Übersicht
- erhöhte Bindegeschwindigkeit, da nur eine Datei geöffnet werden muß
- weniger Parameter beim Aufruf des Linkers

Der Aufruf von `ar` hat folgenden Aufbau:

`ar` *funktion* [*position*] *bibliothek datei* ...

Der Parameter *funktion*, dem kein Minuszeichen vorangestellt wird, gibt an, was zu tun ist. Die wichtigsten Funktionen werden durch folgende Zeichen aufgerufen:

`d` Die angegebene *datei* soll aus der *bibliothek* gelöscht werden.

`r` In der *bibliothek* soll die angegebene *datei* ersetzt werden.

`t` Es wird ein Inhaltsverzeichnis der Bibliothek ausgegeben. Fehlt die Angabe *datei*, so werden alle Moduln ausgegeben.

`x` Die angegebene *datei* wird extrahiert.

5.5 Programmgenerierung mit make

`make` ist ein Programm, welches die weitgehend automatische Generierung bzw. Neuerstellung eines Programmsystems erlaubt. Unter Programmsystem ist hier ein aus mehreren Moduln zusammengesetztes Programm zu verstehen. Das `make`-Konzept ist jedoch so allgemein gehalten, daß auch ein System aus mehreren Programmen oder anders gearteten Dateien damit erstellt werden kann. `make` benutzt zwei Informationsquellen:

1. Eine Beschreibung von Abhängigkeiten (in der Datei `makefile`)
2. Das Modifikationsdatum der aus 1) ermittelten Dateien

Betrachten wir folgendes Beispiel:

```
weber> ex makefile
"makefile" 22 lines, 472 characters
:1,$nu
     1  # makefile fuer "programm"
     2  F77=f77
     3
     4  # Compiler Options for Dec-Station
     5  OPTION= -c -r8 -O2
     6
     7  # Compiler Options for Silicon-Graphics
     8  # OPTION= -c -static -r8 -O2 -trapuv
     9
    10  programm: main.o sub1.o \
    11        grafik.o mathe.o
    12          $(F77) -o programm main.o sub1.o grafik.o mathe.o
    13          size programm
    14
    15  main.o: main.f common.h
    16          $(F77) $(OPTION) main.f
    17  sub1.o: sub1.f common.h
    18          $(F77) $(OPTION) sub1.f
    19  grafik.o: grafik.f
    20          $(F77) $(OPTION) grafik.f
    21  mathe.o: mathe.f
    22          $(F77) $(OPTION) mathe.f
:q
weber> touch common.h
weber> make
f77 -c -r8 -O2 main.f
f77 -c -r8 -O2 sub1.f
f77 -o programm main.o sub1.o grafik.o mathe.o
size programm
text    data    bss     dec     hex
118784  20480   16416   155680  26020
weber>
```


In diesem Beispiel wurde die `ex`-Variante des Editors `vi` benutzt, um die Datei `makefile` mit Zeilennummern auszugeben.

Die Zeilen 1, 4, 7 und 8 sind Kommentarzeilen. Kommentare werden durch `#` eingeleitet. In Zeile 2 wird die Variable '`F77`' definiert. Die Benutzung von Variablen kann viel Arbeit sparen, wenn auf verschiedenen Systemen gearbeitet wird.

Das Ziel dieses makefiles ist die Generierung des Programms `programm`. In der Zeile 10 und der Fortsetzungszeile 11 wird definiert, von welchen Dateien das Ziel abhängt. In Zeile 12 und 13 stehen die Kommandos, die zur Generierung von `programm` notwendig sind. In den Zeilen mit den Kommandos (12, 13, 16, 18, 20, 22) muß als erstes Zeichen ein Tabulator stehen! Die Datei `programm` hängt also von 4 Objektdateien ab, die wiederum von den FORTRAN-Quelltextdateien abhängen. Die Dateien `main.f` und `sub1.f` können z.B. ein Include auf die Datei `common.h` enthalten, siehe Kapitel 5.1.

Wird nun die Datei `common.h` modifiziert, so müssen die Dateien `main.f` und `sub1.f` neu compiliert und anschließend die 4 Objektdateien gelinkt werden. Im Beispiel wurde die Modifikation einer Datei durch das Kommando

`touch` *datei* ...

simuliert. `touch` setzt die Modifikationszeit einer Datei auf die aktuelle Zeit. Existiert die Datei noch nicht, so wird sie mit 0 Byte Länge erzeugt.

Die Vorteile von `make` sind also:

- Zusammenfassung vieler Befehle zu einem Kommando
- Zeitersparnis, da nur die notwendigen Compilationen etc. durchgeführt werden.

5.6 Debugger

Debugger[1] sind Testhilfen, die Programme unter interaktiver Kontrolle ablaufen lassen. Man kann Haltepunkte (breakpoints) setzen, einzelne Programmschritte ausführen lassen und Register- und Speicherinhalte anschauen.

Symbolische Debugger bzw. Source-Level-Debugger kennen die Zeilennummern und symbolischen Namen des Quelltextes, wenn beim Compilieren entsprechende Listen in den Objektdateien angelegt wurden. Dies geschieht normalerweise durch die Option `-g` beim Compileraufruf, siehe Kapitel 5.2. Beim Programmieren in Hochsprachen wird man heute nur noch symbolische Debugger verwenden.

[1] Der Begriff Debugger stammt aus der Steinzeit der Computer. Die Legende erzählt, daß sich ein besonders hartnäckiger Fehler schließlich als ein kleines Insekt (bug) zwischen den Kontakten eines Relais herausstellte.

Daneben erlauben die Debugger das Analysieren von `core`-Dateien, den sogenannten core dumps. Eine `core`-Datei ist der Speicherabzug eines Programms, der vom System beim Absturz eines Programms erzeugt wird.

Da Debugger u.a. mit Registern und Speicherzellen arbeiten, sind sie relativ maschinenabhängig. Den Namen des jeweiligen Debuggers kann man mit dem Kommando

`apropos debug`

ermitteln. Der Aufruf des Debuggers lautet z.B. unter Ultrix

`dbx` *program* [*corefile*]

Bei Angabe von *corefile* wird der Arbeitszeiger auf die Zeile gesetzt, in der der Fehler auftrat, sonst auf die erste ausführbare Anweisung des Programms. Breakpoints werden durch

`stop at` *line*

oder durch

`stop in` *procedure*

gesetzt. Ein Listing des Quelltextes am Arbeitszeiger erhält man mit

`list`

und Variableninhalte läßt man mit

`print` *expression*

ausgeben, wobei bei Feldern die C-Syntax benutzt wird, siehe späteres Beispiel. Zum Starten des Programms wird der Befehl

`run`

benutzt und zur Fortsetzung nach einem Breakpoint dient der Befehl

`cont`

Zur Ausführung von Einzelschritten gibt es zwei verschiedene Befehle. Der Befehl

`step` [*lines*]

führt *lines* Quelltextzeilen aus, wobei auch in Unterprogramme verzweigt wird, während der Befehl

`next` [*lines*]

Unterprogramme nur ausführt, aber nicht bei der Angabe *lines* mitzählt. Default für *lines* ist 1. Schließlich wird der Debugger mit

`quit`

verlassen. Beispiel:

```
weber> cat test.f
      program test
      real x(1000)
      common /vektor/ x

      anzahl=100
      do 10 i=1,anzahl
        x(i)=sin(i*17.5)
 10   continue
      call loop(anzahl)
      end
weber> cat loop.f
      subroutine loop(n)
      real x(1000)
```



```
      common /vektor/ x
      xmax=0.0
      do 10 i=1,n
        xmax=max(xmax,x(i))
 10   continue
      print*,'xmax= ',xmax
      end
weber> f77 -g test.f loop.f
weber> a.out
Segmentation fault (core dumped)
weber> dbx a.out
dbx version 2.0
Type 'help' for help.
reading symbolic information ...
[using test.test]
test:   5  anzahl=100
(dbx) list
     6        do 10 i=1,anzahl
     7          x(i)=sin(i*17.5)
     8   10   continue
     9        call loop(anzahl)
    10        end
(dbx) stop at 9
[2] stop at "test.f":9
(dbx) run
[2] stopped at   [test.test:9 ,0x4002e4]        call loop(anzahl)
(dbx) print x[1]
-0.97562599182128906
(dbx) print x[100]
-0.13250114023685455
(dbx) step
>*[loop, 0x400310]      addiu   sp,sp,-64
(dbx) step
  [loop.loop:4 ,0x400330]       xmax=0.0
(dbx) step
  [loop.loop:5 ,0x400338]       do 10 i=1,n
(dbx) step
  [loop.loop:6 ,0x400368]       xmax=max(xmax,x(i))
(dbx) print xmax
0
(dbx) print x[1]
-0.97562599182128906
(dbx) print i
1
(dbx) print n
1120403456
(dbx) quit
weber>
```

5.7 Source Code Control System SCCS

SCCS ist ein System zum Management von Quelldateien. Dabei werden die Unterschiede von verschiedenen Versionen der Dateien, inklusive von Kommentaren zum Grund der Änderung und wer sie wann gemacht hat, aufgezeichnet. Jede ältere Version kann zurückgeholt werden. Es können auch verschiedene Versionen gleichzeitig verwaltet werden. Wenn in einem Projekt mehr als eine Person arbeiten, sorgt SCCS dafür, daß eine Datei nicht gleichzeitig von zwei Personen editiert wird.

Alle Versionen einer Datei und die Veränderungsgeschichte werden im sogenannten `s-file` gespeichert. Mit dem `s-file` können 3 wesentliche Operationen durchgeführt werden.

- Datei für Compilation holen
  Durch diese Operation wird eine Version der Datei aus dem `s-file` extrahiert. Default ist die neueste Version. Die Datei soll nur zum Compilieren oder Ausdrucken dienen. Die Datei hat deshalb nur die Leseerlaubnis gesetzt.
- Datei zum Editieren holen
  Die Datei hat auch Schreiberlaubnis gesetzt. Sie kann verändert werden. Nur eine Person kann eine Datei zu einem Zeitpunkt editieren.
- Zurückschreiben einer editierten Datei
  SCCS vergibt eine neue Versionsnummer und speichert neben den Änderungen an der Datei auch Kommentare zum Grund der Änderungen.

5.7.1 SCCS Terminologie

Folgende Begriffe sollten verstanden worden sein, bevor man SCCS benutzt:

- **s-file**
  Das s-file ist die Datei, die alle Informationen über die verschiedenen Versionen der Quelltextdatei enthält. Dazu werden die Deltas abgespeichert.
- **Deltas**
  Der Benutzer entscheidet, nach wievielen Änderungen an einer Datei eine neue Version erzeugt werden soll. Der Unterschied zwischen der vorhergehenden und der aktuellen Version ist das Delta.
- **SCCS IDs** (Version Number)
  Jedes Delta erhält eine SCCS ID (SID), die aus zwei Teilen besteht, nämlich der Release Number und der Level Number. Die Release Number wird nur geändert, wenn größere Änderungen vorgesehen sind, z.B. bei Änderungen am Projekt.
- **ID-Schlüsselwörter** (ID-keywords)
  SCCS expandiert bestimmte Schlüsselwörter im Quelltext, wenn eine Datei zur Compilation geholt wird. Es stehen u.a. folgende Schlüsselwörter zur Verfügung.

  `%Z%` Expandiert zu `@(#)` und signalisiert den Beginn eines Schlüsselwortes.

  `%M%` Der Name der Datei.

  `%I%` Die SID.

  `%W%` Abkürzung für `%Z%%M% <tab> %I%`.

  `%G%` Das zu `%I%` korrespondierende Datum.

5.7.2 Benutzung von SCCS

Zur Benutzung von SCCS muß ein Verzeichnis `./SCCS` angelegt werden. Danach stehen u.a. folgende Befehle zur Verfügung:

`sccs admin -i`*file* *file*	Legt das s-file für die Datei *file* an.
`sccs get` *file*	Holt *file* zum Compilieren.
`sccs edit` *file*	Holt *file* zum Editieren.
`sccs delta` *file*	Speichert die Änderungen an *file* in einem Delta ab.
`sccs info`	Zeigt an, welche Dateien momentan von wem editiert werden.
`sccs prs` *file*	Zeigt die Kommentare zu den verschiedenen Deltas an.
`sccs diffs` *file*	Zeigt die Änderungen an der gerade editierten Datei an, die seit dem letzten Delta gemacht wurden.

5.7.3 Beispiel zur Benutzung von SCCS

```
weber> mkdir SCCS
weber> chmod 777 SCCS
weber> cat main.f
      program main
      print*,'Interaktiver Gittergenerator'
      print*,'Version %I% vom %G%'
      call sub1
      call mathe
      call grafik
      end
weber> ls -l main.f
-rw-r--r--  1 hansen        159 Oct 25 18:39 main.f
weber> sccs admin -imain.f main.f
weber> rm main.f
weber> sccs get main.f
1.1
7 lines
weber> f77 -c main.f
weber> cd SCCS
weber> ls -l
total 1
-r--r--r--  1 sccs          304 Oct 25 18:43 s.main.f
weber> cd ..
weber> sccs edit main.f
1.1
new delta 1.2
7 lines
weber> ex main.f
"main.f" 7 lines, 159 characters
:1,$nu
     1        program main
     2        print*,'Interaktiver Gittergenerator'
     3        print*,'Version %I% vom %G%'
     4        call sub1
     5        call mathe
     6        call grafik
     7        end
:3a
      call readdt
.
:w
"main.f" 8 lines, 177 characters
:q
weber> sccs info
      main.f: being edited: 1.1 1.2 hansen 93/10/25 18:49:05
weber> sccs delta main.f
Comments? (^D to end)
Rufe Subroutine zum Lesen.
^D
1.2
1 inserted
0 deleted
7 unchanged
weber> sccs get main.f
1.2
```

```
8 lines
weber> cat main.f
      program main
      print*,'Interaktiver Gittergenerator'
      print*,'Version 1.2 vom 10/25/93'
      call readdt
      call sub1
      call mathe
      call grafik
      end
weber>
```

Kapitel 6

Vernetzung

Ein wesentlicher Vorteil von Unix ist, daß die Software für die Vernetzung Teil des Betriebssystems ist. Außerdem ist zumindest bei Workstations die entsprechende Hardware auch bei Minimalausstattung Standard.

Das der Software zugrunde liegende Protokoll ist TCP/IP. Hardwaremäßig erfolgt die Verkabelung meist über Ethernet (mit Koaxialkabeln). Der Trend geht jedoch zunehmend zu leistungsfähigeren Netzen, z.B. FDDI (Fiber Distributed Digital Interface).

Aufbauend auf TCP/IP sind die Rechner der TU Berlin durch WoTaN (Workstations der TU am Netz) vernetzt. WoTaN selbst ist Teil des weltweiten Netzwerkes Internet. Jeder an diesem Netz hängende Rechner besitzt eine eindeutige Internetadresse. Sie besteht aus vier durch Punkte getrennten Zahlen im Bereich 0-255. Der in den Beispielen benutzte Rechner weber hat z.B. die Internetadresse

130.149.47.6

Der weltweit eindeutige Name von weber ist

weber.pi.TU-Berlin.DE

In diesem Fall besteht eine eins-zu-eins Zuordnung von Name und Internetadresse. Es gilt weber$\hat{=}$6, pi$\hat{=}$47, TU-Berlin$\hat{=}$149 und DE$\hat{=}$130. Diese eins-zu-eins Zuordnung gilt aber nicht immer! Bei den meisten Kommandos kann man entweder die Internetadresse oder den Rechnernamen benutzen. Unix arbeitet letztlich immer mit der Internetadresse. Folgende Dienste sind besonders interessant:

- telnet
  ermöglicht das Einloggen auf einem anderen Rechner im Netz. Man benötigt einen Account auf dem anderen Rechner.
- ftp
  ermöglicht die Übertragung von Dateien zwischen Rechnern.
- e-mail
  ermöglicht das Versenden von „elektronischen“ Briefen an beliebige User auf beliebigen Rechnern. Das Standardkommando für e-mail ist `mail`, siehe Kapitel 4.3. Sofern vorhanden, sollte man jedoch das komfortablere Tool `elm` benutzen.
- nfs
  ermöglicht die transparente Benutzung von Festplatten auf anderen Rechnern. Transparent heißt, daß ein Benutzer normalerweise nicht merkt, ob eine Datei auf einer lokalen oder einer entfernten Festplatte liegt.

6.1 Einloggen auf entfernten Rechnern

Mit dem Kommando

`telnet` *host*

baut man eine Verbindung zum Rechner *host* auf, um sich dort einzuloggen. Für *host* kann der Rechnername oder die Internetadresse benutzt werden. Die Verbindung wird abgebaut, indem auf dem entfernten Rechner `exit` eingegeben wird. Kann `telnet` keine Verbindung aufbauen, so muß man `telnet` mit `quit` verlassen.

Oftmals stimmt auf dem entfernten Rechner die Terminaleinstellung nicht. Im allgemeinen hilft dann

```
set term = vt100
```

6.2 Elektronische Post e-mail

Es soll noch einmal erklärt werden, wie man mit dem Standardkommando `mail` Post versenden und empfangen kann. Das Kommando wurde bereits im Kapitel 4.3 vorgestellt. Interessant ist e-mail natürlich besonders beim Anschluß an das weltweite Internet.

6.2.1 Post verschicken

Am einfachsten ist es, den Brieftext zuerst mit einem Editor in eine Datei zu schreiben und dann die Datei zu verschicken. Dies geschieht mit dem Kommando

`mail` [`-s` *subject*] *user*[`@`*host*] `<` *datei*

subject ist ein kurzes Stichwort, das dem Empfänger angezeigt wird, bevor er die eigentliche Mail liest. Entfällt die Angabe `@`*host*, so ist der lokale Rechner gemeint.

6.2.2 Post empfangen

Um Post zu lesen, wird einfach

```
mail
```

ohne Parameter benutzt. Es wird angezeigt, welche Briefe im Briefkasten vorhanden sind. Diese Briefe kann man dann mit folgenden Kommandos bearbeiten:

- `?` Anzeige der möglichen Kommandos.
- `q` Beenden von `mail`.
- `+` Anzeige der nächsten Nachricht (gleiche Wirkung bei Return).
- `s` *datei* Speichert den Brief in der Datei *datei*.
- `d` Löscht den Brief.

Kapitel 7

X-Windows

7.1 Einführung

Das X-Windows System ist ein leistungsfähiges und portables Fenstersystem für Grafikbildschirme, das am Massachusetts Institute of Technology (MIT) entwickelt wurde. Das X-Windows System (im folgenden einfach X genannt) läuft auf zahlreichen Rechnersystemen. Es ist unabhängig von Betriebssystem, Prozessor, Hersteller und Programmiersprache. Allerdings hat es die größte Verbreitung auf Unix-Workstations und hat sich dort zum Standard entwickelt. X hat folgende wesentliche Vorteile:

- X ist netzwerktransparent. Anwendungsprogramme können gleichzeitig und ebenso einfach auf Bildschirmen anderer Rechner irgendwo im Netzwerk eine Ausgabe machen wie auf ihrem eigenen Bildschirm.
- Das benutzte Kommunikationsverfahren ist sprachen- und rechnerunabhängig. Die anderen Rechner müssen weder vom selben Typ sein noch unter demselben Betriebssystem laufen.
- X ist nicht Teil des Betriebssystems. Für das Betriebssystem ist es lediglich ein weiteres Anwendungsprogramm.
- X ist für jederman zu den Vertriebskosten frei verfügbar.

7.2 Server, Client und Kommunikationskanal

Im Gegensatz zu den meisten anderen Fenstersystemen ist X kein homogenes Einzelprogramm. Es besteht vielmehr aus drei zusammenhängenden Teilen:

1. Der Server ist ein Programm, das das physikalische Display und die Eingabegeräte steuert.
2. Die Clients sind Programme, die vom Server die Ausführung spezieller Fensteroperationen anfordern.
3. Der Kommunikationskanal wird von Server und Clients benutzt, um untereinander Verbindung zu halten.

Der Server
Die erste Grundkomponente von X ist der Server, die Software, die den physikalischen Bildschirm und die Eingabegeräte Tastatur und Maus kontrolliert. Der Server erzeugt auf Anforderung eines Client-Programms die Fenster und zeichnet Bilder und Texte in sie hinein. Von sich aus macht der Server nichts, er führt lediglich Aktionen aus, die von Client-Programmen angefordert wurden.

Jeder Bildschirm hat genau einen Server.

Der Server wird vom Hersteller des X-Systems geliefert und normalerweise vom Benutzer nicht geändert. Für das Betriebssystem ist der Server jedoch nur ein weiteres Benutzerprogramm.

Client-Programme
Die zweite Grundkomponente von X bilden die Client-Programme. Das sind Anwendungsprogramme, die von den Fensterfähigkeiten des Systems Gebrauch machen. Anwendungsprogramme unter X werden Clients

genannt, da sie tatsächlich Kunden des Servers sind. Sie bitten den Server, bestimmte Aktionen für sie auszuführen. Sie können kein Fenster und keinen Bildschirm direkt beeinflussen. Sie können lediglich den Server bitten, das zu tun, was sie wünschen. Typische Anfragen wären z.B.: „Schreibe die Zeichenkette 'Hallo, Welt' in das Fenster XYZ“ oder „Zeichne eine Linie einer bestimmten Farbe im Fenster ABC von D nach E“.

Selbstverständlich bildet das Anfordern von Fensteroperationen nur einen Teil eines Client-Programms. Der andere Teil ist der Programmcode, der sich auf die Aufgabe bezieht, die der Benutzer ausführen will. Als Benutzer verwendet man meist Client-Programme aus verschiedenen Quellen. Einige wesentliche werden zusammen mit dem System geliefert, andere erhält man von Fremdanbietern und manche Benutzer schreiben Ihre eigenen Client-Programme für bestimmte Anwendungen.

Der Kommunikationskanal
Die dritte Grundkomponente von X ist der Kommunikationskanal. Die Client-Programme nutzen ihn, um Anforderungen an den Server zu schicken, und der Server sendet über ihn Status- und andere Informationen an die Client-Programme zurück.

Die genaue Beschaffenheit des Kommunikationskanals ist nicht so wichtig, solange sowohl Server als auch Client wissen, wie er zu benutzen ist. Die auf einem bestimmten System oder Netzwerk benötigten Kommunikationstypen werden durch die X-Window Bibliothek (library) für dieses System unterstützt. Die gesamte Kommunikation zwischen Client und Server wird über diese Bibliothek, die „Xlib“ heißt, abgewickelt. Als Folge davon können alle Client-Programme automatisch sämtliche Kommunikationsmethoden benutzen, indem sie auf die Bibliothek zurückgreifen.

Die Client/Server-Kommunikation gliedert sich in zwei große Klassen, die die beiden wesentlichen Operationsweisen im X-System widerspiegeln:

1. Server und Client laufen auf demselben Computer. Hier können sie jede auf diesem Rechner verfügbare Form der Kommunikation zwischen Prozessen (IPC=inter-process-communication) nutzen. In diesem Modus verhält sich X letztendlich wie zahlreiche konventionelle Fenstersysteme.

2. Server und Client laufen auf verschiedenen Computern. In diesem Fall müssen sie über ein Netzwerk kommunizieren, indem sie ein gegenseitig vereinbartes Protokoll benutzen. Zur Zeit sind TCP/IP und DECnet die gebräuchlichsten Protokolle.

Die Fähigkeit einer Anwendung, genauso problemlos innerhalb eines Netzwerks wie auf lokalen Rechnern zu funktionieren, wird Netzwerktransparenz genannt und findet sich fast nur bei X. Durch diese Besonderheit ist X für den Aufbau von anpassungsfähigen Vielzweck-Netzwerken mit zusammenarbeitenden Rechnern geeignet.

Da Server und Client völlig getrennt sind, hat sich ein neuer Terminaltyp entwickelt, der X-Terminal genannt wird. Ein X-Terminal ist eine abgespeckte Workstation, auf der nur der X-Server laufen soll und sonst nichts. Diese verfügt über eine Tastatur, eine Maus, einen Bildschirm und kann im Netz kommunizieren. Sie verfügt jedoch nicht über ein eigenes Dateisystem und unterstützt keine Mehrzweckprogramme. Diese müssen irgendwo anders im Netzwerk laufen.

7.3 Windowmanager

Das X-Grundsystem, also der Server, bietet nur die elementaren Fensterfunktionen:

- Fenster erzeugen
- In Fenster schreiben und zeichnen
- Tastatur und Mauseingaben verarbeiten
- Fenster entfernen

Die komfortable Manipulation der Fenster wird bei X von einem separaten Programm wahrgenommen, dem sogenannten Windowmanager. Er bietet mindestens folgende Funktionen, die alle mit der Maus gesteuert werden:

- Verschiedene Anwendungsfenster erzeugen.
- Bewegen der Fenster auf dem Bildschirm.

- Ändern der Fenstergröße.
- Verdeckte Fenster nach oben holen und umgekehrt.
- Anwendungen starten und beenden.
- Bildschirm neu zeichnen.
- Fenster in Icons und Icons wieder in Fenster umwandeln.

Ähnlich wie bei Unix verschiedene Shells zur Auswahl stehen, so gibt es auch viele verschiedene Windowmanager. Der Windowmanager, der vom MIT im sogenannten Core-Release mitgeliefert wird, ist der `uwm`.

7.4 Benutzung von X

Um X benutzen zu können, muß man mindestens den Server starten. Oftmals (speziell bei unseren DEC-Stations) sind Unix-Systeme so konfiguriert, das X immer läuft. Startet man jedoch im einfachen Textterminalmodus, so muß der einzelne Anwender X starten. Dazu dient das Kommando

```
xinit
```

das den X-Server startet und danach ein **`xterm`** Fenster öffnet. In diesem **`xterm`** Fenster, das sich wie ein normales Textterminal verhält, kann man mit dem Kommando

```
uwm &
```

z.B. den Standardwindowmanager starten. Der Windowmanager wird das Starten von weiteren Textterminalfenstern ermöglichen. Man kann aber auch über die Tastatur das Kommando

```
xterm &
```

eingeben. Oftmals werden der Befehl `xinit` und der Aufruf des Windowmanager in einem Kommando zusammengefaßt, z.B.

```
startx
```

Damit Clients von entfernten Rechnern Fenster auf dem Bildschirm öffnen können, muß diesen mit dem Kommando

`xhost +` [*rechnername*]

Zugriffsrecht gewährt werden. Fehlt die Angabe *rechnername*, so findet keine Zugangskontrolle mehr statt. Zugriffsrechte werden entzogen mit dem Kommando

`xhost -` [*rechnername*]

Man beachte, daß das X-System Zugriffsrechte an Rechner verteilt, und nicht an bestimmte Benutzer! Um auf einem entfernten Rechner einen Client zu starten, der ein Fenster auf dem lokalen Bildschirm öffnen soll, muß auf dem entfernten Rechner die Variable 'DISPLAY' entsprechend gesetzt werden.

`setenv DISPLAY` *rechnername*`:`*displaynummer*

rechnername ist der Name des Computers, auf dem der X-Server läuft. Der normale Bildschirm hat die *displaynummer* 0.