Kompaktkurs Unix

Patricia Jung & Detlev Degenhardt
Rechenzentrum der Universität Freiburg

27. Juni 1997

Inhaltsverzeichnis

Intro

Hinweise zum Kurs

Dieser Kurs richtet sich an Anfänger. Es werden keine tieferen Betriebssystemkenntnisse von anderen Rechnerplattformen vorausgesetzt, wohl aber Erfahrung im allgemeinen Umgang mit Computern. Ziel des Kurses ist, prinzipielle Konzepte von UNIX zu vermitteln und den ersten Einstieg zu erleichtern. Es ist nicht das Ziel, möglichst viele UNIXbefehle kennenzulernen.

Hinweise zum Skript

Dieses Skript ist nicht als Lehrbuch gedacht. Es stellt vielmehr eine etwas erweiterte Version der Overhead-Folien dar, die im Kurs verwendet werden. Das Skript allein mag an manchen Stellen ohne den Besuch des Kurses und die Ausführung der Beispiele an einem Rechner des cipsun-Pools unverständlich sein. Eine Liste von ausführlicheren Skripten und Büchern findet sich im Anhang[1].

In diesem Skript gelten folgende Konventionen:

`Schreibmaschinenschrift`: Befehle, Programme und Optionen, die wortgetreu eingegeben werden müssen;

Kursiv : Argumente, Optionen und Variablen, die durch Benutzerangaben zu ersetzen sind.

⚠ : Das Achtungszeichen am Seitenrand weist auf Fallstricke hin.

[1]Der Kurs lehnt sich teilweise an [1] an; einige Textstellen wurden in dieses Skript übernommen.

1. Allgemeines

1.1. Geschichtliche Hintergründe – die wichtigsten Stationen

Historisches

1965 : Multics (Bell Telephone Laboratories)

- Ziel: Multiuser-Betriebssystem,
- Fehlschlag, Aufgabe nach ein paar Jahren

ab 1969 : Unix-Vorversion von ehemaligen Mitarbeitern des Multics-Projektes (Thompson, Kernighan & Ritchie); Entwicklung der Programmiersprache B

1973 : Unix

- Weiterentwicklung von B nach C,
- Unix ist fast ausschließlich in C geschrieben $\Longrightarrow$ Portabilität

1983 : Unix System V (AT&T);
parallele Entwicklung von BSD-Unix (Berkeley System Distribution) in Berkeley (University of California)

Mitte der 90er Jahre:

- Mittlerweile gibt es viele herstellerspezifische Varianten, aber die Hauptvarianten System V und BSD bleiben.
  Beispiel RZ:
  - SunOS (cipsuns): BSD,
  - Solaris (RZ-Suns): System V,
  - AIX (IBM-Maschinen des RZs)
- weitere Varianten: HP/UX, SGI, DEC ...
- Bestrebungen der Hersteller, die verschiedenen Unix-Strömungen wieder zu vereinen
- Entwicklung freier Unixe für den PC (Linux, FreeBSD u. a.)

1.2. Unix-Konzepte

Eigenschaften von Unix

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

⟹ Beispiel: (Nicht zur Abschreckung ;-)

```
$ tar cf - Verzeichnis1 | ssh sun2 '( cd temp; tar xvfp -)'
```

`tar`, `ssh` und `cd` sind UNIX-Befehle, die hier in einer Zeile zu *einem* Kommando zusammengesetzt wurden. (Wen es interessiert: In diesem Beispiel wird der Inhalt des gesamten Verzeichnisbaums namens `Verzeichnis1` „zusammengepackt", über das Netz auf einen anderen Rechner (sun2) kopiert und dort im Verzeichnis `temp` wieder „ausgepackt".)

Die große Flexibilität wird dadurch erreicht, daß

- in UNIX alles wie eine Datei behandelt wird, selbst technische Dinge wie Festplatten, Tastaturen, Drucker usw.,
- Programme durch feste Datenleitungen (pipes) verbunden werden können (siehe unten).

1.3. Vergleich mit anderen Betriebssystemen

UNIX ist ein *Multiuser/Multitasking*-Betriebssystem. Was bedeutet das?

Singleuser/Singletasking-Systeme

Das auf PCs weit verbreitete DOS ist ein Singleuser/Singletasking-System. Der Rechner kann nur von einem Benutzer (user) gleichzeitig bedient werden, und dieser Benutzer kann auch nur eine Aufgabe (task) bzw. ein Programm zur gleichen Zeit ausführen[2].

Nichtpräemptives Multitasking

Das ebenfalls weitverbreitete MS-Windows ist ebenfalls ein Singleusersystem. Unter Windows können zwar schon mehrere Programme gleichzeitig arbeiten, aber nur in Form des nichtpräemptiven[3] Multitaskings: Auf keinem Rechner mit nur einem Prozessor (egal, welches Betriebssystem) können mehrere Programme wirklich gleichzeitig laufen. Der Trick besteht darin, unmerklich für den Benutzer jedem Programm kurzfristig Rechenzeit zur Verfügung zu stellen und zwischen den Programmen sehr schnell hin- und herzuschalten. Bei Windows ist nun nicht das Betriebssystem für dieses Umschalten (scheduling) verantwortlich, sondern die einzelnen Applikationen müssen ihre Rechenzeit von sich aus wieder freigeben („nicht präemptiv"). Dies tun sie häufig nicht, was dann die bekannten „Hänger" verursacht.

Präemptives Multitasking

OS/2 geht hier schon einen Schritt weiter: Das Betriebssystem selbst sorgt für das Scheduling (also: präemptives Multitasking). Aber OS/2 ist immer noch ein Singleuser-Betriebssystem.

Multiuser-Systeme

Hier kommt jetzt die Stärke von UNIX zum Vorschein: Auf UNIX-Systemen können mehrere Benutzer mehrere Programme gleichzeitig ausführen. Dies zieht einen ganz wichtigen Punkt nach sich:

NIEMALS EINEN UNIX-RECHNER AUSSCHALTEN!!! ⚠

[2] Eine Ausnahme macht `print`, das auch im Hintergrund laufen kann.
[3] to pre-empt – *engl.:* vorwegnehmen

Nur der Systemadministrator darf den Rechner ausschalten, nachdem sie oder er geeignete Maßnahmen zur Datensicherung getroffen hat.

1.4. Die ersten Schritte mit Unix

UserID und Paßwort

Bevor man auf einerUNIX-Maschine arbeiten kann, muß man vom Systemadministrator als Benutzer eingetragen werden: Man braucht eine UserID (UID). Diese kann im Rechenzentrum bei der Benutzerinformation beantragt werden. Die UserID für diesen Kurs ist nur eine Woche gültig.

Wenn der Systemverwalter eine UserID einrichtet, belegt er sie mit einem Paßwort[4]. Erst, wenn Sie UserID und Paßwort wissen, können Sie sich auf demUNIX-Rechner einloggen.

Anmelden beim System

Das System meldet sich mit dem *Login-Prompt*.

⟹ Beispiel:

```
$ cipsun1 login:
```

Die Shell

Nach dem Einloggen (UserID und Paßwort eingeben; dabei wird das Paßwort nicht angezeigt) öffnet das System eine *shell* (Muschel, Schale), die den Kern des Betriebssystem („Kernel") umschließt und die Kommunikation zwischen Kernel und Benutzer ermöglicht.

Benutzer ⟺ Shell ⟺ Betriebssystemkern

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

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

⟹ Beispiel:

```
kursse00cipsun1.ruf:/u/kurs/se/kursse00$
```

Fenstersystem

Jetzt könnte man eigentlich loslegen. Im allgemeinen wird man aber nicht an der sogenannten *Konsole* arbeiten, sondern ein komfortables Fenstersystem (hier OpenWindows) starten:

```
$ W ov
```

In diesem System kann man mit mehreren Fenstern arbeiten[5]. Wichtig ist erst einmal nur, daß man durch Anklicken eines Fensterrahmens mit der linken Maustaste dieses

[4]Dieses Paßwort sollten Sie schnell ändern! (Datensicherheit!!!)

[5]Nähere Informationen bietet das ausgegebene Faltblatt.

Fenster auf dem Bildschirm verschieben kann; mit der Tastenkombination `CTRL`+rechte Maustaste (innerhalb eines Fensters) erhält man ein Menü, in dem man die Schriftgröße verändern kann.

Die erste Aufgabe ist, die Paßwörter zu verändern, da im Augenblick jede Kursteilnehmerin und jeder Kursteilnehmer dasselbe Paßwort haben. Zur Änderung wird das UNIX-Kommando `passwd` benutzt: `passwd`

`$ ssh cipsun0`	(Dies gehört eigentlich nicht zur Paßwortänderung, sondern ist eine Besonderheit der cipsun-Rechner. Auf diesen Befehl wird erst in Kapitel 7.1 näher eingegangen.)
`$ passwd`	
`$ exit`	(siehe oben)

Mit der Eingabe eines neuen Paßwortes wird dieses verschlüsselt und in eine Paßwortdatei geschrieben. Eine Besonderheit des cipsun-Pools erfordert noch das Ausführen des Befehls `ssh cipsun0` vor der Ausführung des `passwd`-Kommandos. Außerdem ist das geänderte Paßwort erst nach der nächsten vollen Stunde aktiv!

Ein wichtiger Hinweis:

BEI UNIX MUSS MAN AUF GROSS- UND KLEINSCHREIBUNG ACHTEN! ⚠

Es gibt außer `passwd` noch ca. 400 Kommandos, die zu UNIX gehören. Niemand kann sich alle merken, deshalb gibt es auf allen UNIX-Systemen *Online-Hilfen*, die sogenannten *Manpages* (Kurzform für *manual pages*). `man`

⟹ Beispiel:

```
$ man passwd
$ man man
$ man -k Suchbegriff
```

Es gibt natürlich noch weitere Hilfefunktionen, die sich aber von System zu System unterscheiden, während man sich darauf verlassen kann, daß die Manpages immer vorhanden sind.

Wenn man mit der Arbeit fertig ist, darf man nicht vergessen, sich beim System abzumelden. Dazu dient das Kommando `exit`

```
$ exit
```

MAN IST ERST DANN ABGEMELDET, WENN SICH DAS SYSTEM WIEDER MIT DEM LOGIN-PROMPT MELDET!!! ⚠

Auf manchen Systemen heißt das Abmeldekommando `logout, logoff, quit` usw. Auf allen Systemen kann man sich jedoch mit der Tastenkombination `CTRL-D` abmelden. `logout, logoff, quit, CTRL-D`

2. Das Dateisystem

2.1. Einführung

Das Dateisystem ist die Basis eines jeden Betriebssystems. In UNIX werden nicht nur Textdateien, ausführbare Programme usw. als Dateien behandelt, sondern auch die Hardware. So ist zum Beispiel die Maus für UNIX lediglich eine Datei namens `/dev/mouse`. Diese Verfahrensweise macht UNIX so flexibel.

2.2. Dateien

Unter Dateien verstehen wir hier zunächst Textdateien. Textdateien werden mit *Editoren* erstellt bzw. verändert. Ein mächtiger und auf allen UNIX-Systemen vorhandener Editor ist der `vi`. Aufruf:

`vi`

`$ vi` *filename*

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

Es gibt zwei Modi des `vi`:

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

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

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

`h, j, k, l`	– Positionieren des Cursors (meist auch durch die Cursortasten möglich)
`CTRL-f`	– ganzer Bildschirm nach unten
`CTRL-b`	– ganzer Bildschirm nach oben
`G`	– bewegt den Cursor an den Anfang der letzten Zeile der Datei
`x`	– löscht das Zeichen unter dem Cursor
`dd`	– löscht die Zeile, in der der Cursor steht
`o`	– erzeugt eine neue Zeile hinter der aktuellen und wechselt in den Eingabemodus
`O`	– erzeugt eine neue Zeile vor der aktuellen und wechselt in den Eingabemodus
`:w`	– Speichern der Datei
`:q`	– Verlassen der Datei
`:wq`	– Speichern und Verlassen der Datei

Es gibt eine Reihe weiterer Befehle für den `vi`-Editor, die aber den Rahmen dieser Kurzeinführung sprengen würden. In jedem guten Skript oder Lehrbuch werden sie erklärt, und natürlich auch in der Manualpage zu `vi`.

Auf den meisten Rechnern des RZs ist ein Lernprogramm installiert. Aufruf:

`$ vilearn` `vilearn`

Mit Editoren kann man Textdateien erstellen, ändern oder auch nur ansehen. Wenn man lediglich wissen will, was in einer Datei steht, ohne die Absicht zu haben, den Inhalt zu ändern, gibt es auch andere UNIX-Kommandos: `cat, more`

`$ cat` *filename*
`$ more` *filename* (seitenweise Anzeige der Datei)

Tasten-Kommandos für more:

Leertaste	– blättert eine Seite vor
Entertaste	– blättert eine Zeile vor
`b`	– blättert eine Seite zurück
`h`	– Hilfe (zur Verfügung stehende Kommandos in `more`)
`q`	– beendet more

Wenn man mehrere Dateien erstellt hat, möchte man auch Informationen darüber haben, welche Dateien vorhanden sind. Das heißt, man möchte eine Dateiliste erzeugen. Dazu dient das Kommando

`$ ls` (listet den Inhalt eines Verzeichnisses auf) `ls`

Das `ls`-Kommando allein (ohne Optionen) gibt nur Informationen darüber, welche Dateien überhaupt vorhanden sind. Natürlich kann man sich auch die Größe der Dateien und weitere Informationen anzeigen lassen. Zu diesen Feinheiten von `ls` kommen wir weiter unten. Zunächst werden hier weitere wichtige Kommandos für den Umgang mit Dateien vorgestellt: `cp, rm, mv`

`$ cp` *file1 file2*	(copy) Kopieren von Datei *file1* nach *file2*
`$ rm` *file1*	(remove) Löschen der Datei *file1*
`$ mv` *file1 file2*	(move) Umbenennen der Datei *file1* in *file2*

2.3. Dateiverzeichnisse

Nach einer gewissen Zeit wird die Anzahl der Dateien so groß, daß eine Strukturierung bzw. Ordnung notwendig wird. UNIX bietet dafür – wie andere Betriebssysteme auch – die Möglichkeit, Dateiverzeichnisse (directories) zu erstellen.

Verzeichnisse werden mit dem Kommando `mkdir`

`$ mkdir` *Verzeichnisname* (make directory) `ls` erstellt. Mit dem schon erwähnten Kommando `ls` kann man sich Dateien und Verzeichnisse auflisten lassen (Verzeichnisse sind auch Dateien):

`$ ls`	
`$ ls -F`	Die Option `-F` hängt einen `/` an den Namen von Verzeichnissen an (zur besseren Unterscheidung zwischen Dateien und Verzeichnissen)

Jetzt können vorhandene Dateien übersichtlich in Unterverzeichnissen abgelegt werden. Dazu kann man auch den `mv`-(move-)Befehl verwenden, der also nicht nur zum Umbenennen von Dateien dient.

`mv`

⟹ Beispiel:

`$ mv` *file1 dir1*	bewegt *file1* in das Verzeichnis *dir1*
`$ ls` *dir1*	Auflisten des Inhalts des Verzeichnisses *dir1*

`cp, rmdir, rm` Weitere Befehle:

`$ cp` *file2 dir1*	*file2* wird nach *dir1* kopiert, bleibt aber auch im ursprünglichen Verzeichnis erhalten.
`$ rmdir` *dir1*	Löschen von *leeren* Verzeichnissen
`$ rm -r` *dir1*	rekursives Löschen: Löschen von *dir1* einschließlich aller darin enthaltenen Dateien und Unterverzeichnisse

⚠ EINMAL GELÖSCHTE DATEIEN UND VERZEICHNISSE KÖNNEN UNTER UNIX NICHT WIEDERHERGESTELLT WERDEN!!!

2.4. Die Dateihierarchie

Das Verzeichnis, in dem man sich gerade befindet, wird *aktuelles Arbeitsverzeichnis* („working directory") genannt. Das Verzeichnis, in dem man sich nach dem Anmelden am System befindet, heißt *Homedirectory* des Benutzers/der Benutzerin.

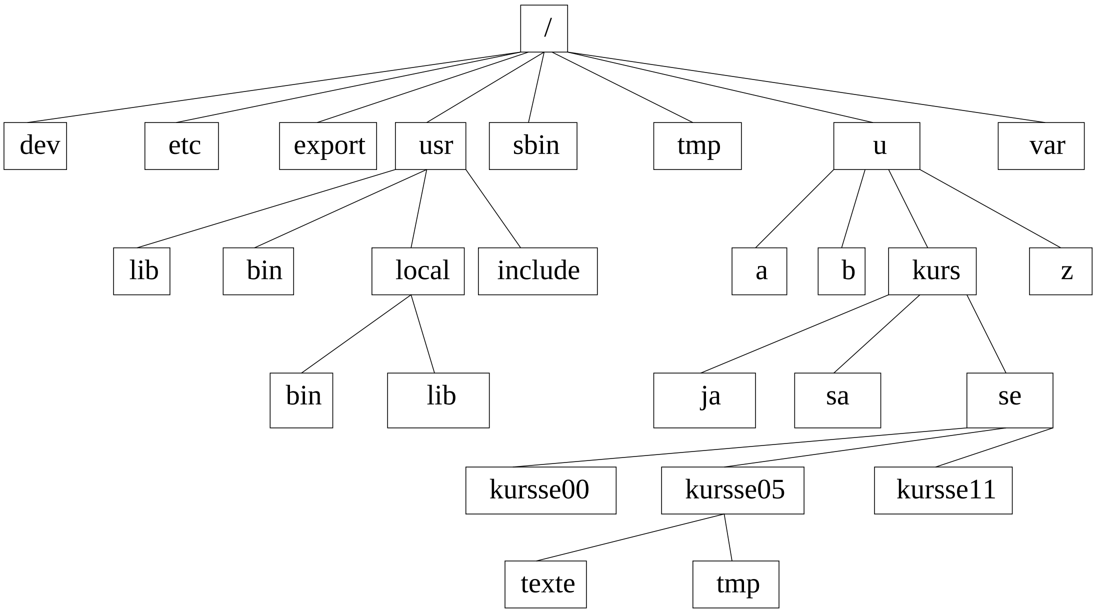
`cd` Wechseln des Verzeichnisses:

`$ cd` *dir1*	change working directory (macht *dir1* zum aktuellen Verzeichnis)
`$ cd ..`	geht eine Hierarchiestufe nach oben
`$ cd`	wechselt in das eigene Homedirectory

Man kann mit `mkdir` eine ganze Reihe von Unterverzeichnissen anlegen und mit `cd` hineinwechseln.

Verzeichnisstruktur

Insgesamt bilden die UNIX-Verzeichnisse eine baumartige Struktur (ähnlich wie auch bei DOS und OS/2) mit einer obersten Ebene, dem *Wurzelverzeichnis* (root directory) `/`. Das eigene Homeverzeichnis liegt relativ weit unten im Dateibaum:



Man kann ein bestimmtes Verzeichnis auf zwei verschiedene Arten angeben, nämlich durch *relative* und *absolute Pfadnamen*. Der absolute Pfad wird vom Wurzelverzeichnis aus angegeben, also zum Beispiel `/u/kurs/se/kursse05/texte`. Wenn man in dieses Verzeichnis wechseln will, gibt man ein:

relative und absolute Pfade

```
$ cd /u/kurs/se/kursse05/texte
```

Wenn man aber im Verzeichnis `/u/kurs/se/kursse05/tmp` ist und wieder in das oben genannte Verzeichnis wechseln will, kann man das auch relativ zu seinem/ihrem aktuellen Verzeichnis machen: Man geht zunächst eine Hierarchiestufe nach oben und wechselt dann in das gewünschte Verzeichnis.

```
$ cd ../texte
```

Wenn man sich im Dateibaum verlaufen hat und kein Fenstersystem besitzt, in dessen Fensterrahmen das aktuelle Verzeichnis angezeigt wird, kann man mit folgendem Kommando das aktuelle Verzeichnis anzeigen lassen:

`pwd`

`$ pwd` (print working directory)

2.5. Zugriffsrechte

Damit in einem Multiuser/Multitasking-Betriebssystem nicht das große Chaos ausbricht, muß genau geregelt sein, welcher Benutzer was darf. In Unix wird daher jeder Datei und jedem Verzeichnis ein Satz von Zugriffsrechten (permissions) zugeteilt. Diese regeln, welcher Benutzer welche Aktionen mit der jeweiligen Datei ausführen darf[6]. Mögliche Aktionen sind: Lesen (read), Schreiben (write) und Ausführen (execute). Dementsprechend heißen die dazu gehörenden Zugriffsrechte

Zugriffsrechte

[6]Zur Erinnerung: Unter Unix werden auch die Hardwarekomponenten (Hauptspeicher, Festplatten(partitionen), Drucker etc.) als Dateien angesprochen.

`r` wie readable,

`w` wie writeable und

`x` executable.

Diese Rechte können getrennt vergeben werden für

- den Besitzer einer Datei/eines Verzeichnisses (user, abgekürzt: `u`),
- die Mitglieder der Gruppe, der der Besitzers zugeordnet ist (group, abgekürzt: `g`) sowie
- alle anderen Benutzer (others, abgekürzt: `o`).

Ein Satz Zugriffsrechte sieht dann beispielsweise so aus:

```
r w x r - x r - -
```

Die linken drei Rechte bezeichnen dabei die Rechte des Users, die mittleren drei die der Gruppe und die rechten die aller anderen Benutzer. Ein Minus bedeutet, daß das entsprechende Recht nicht gesetzt ist.

`ls` Um sich über die Zugriffsrechte einer Datei zu informieren, verwendet man das schon bekannte Kommando `ls`, nun mit der Option `-l` („long“):

```
$ ls -l
```

Das Ergebnis sieht beispielsweise so aus:

```
total 9
drwxr-xr-x  2 tantzen       512 Apr 28  1992 tantzen
drwx------  4 test1         512 Nov 23  1993 test1
drwxr-xr-x  4 test2         512 Nov 23  1993 test2
drwxr-xr-x  4 test3         512 Nov 19  1993 test3
lrwxrwxrwx  1 root           14 Oct 16 14:42 testuser -> /u2/t/testuser
lrwxrwxrwx  1 root           14 Oct 16 15:05 theckert -> /u2/t/theckert
drwxr-xr-x  4 thies         512 Nov 22  1993 thies
drwxr-xr-x 16 tseng        1536 Oct 24 21:26 tseng
```

Die Angaben bedeuten im einzelnen:

Spalte 1 : Dateityp (z. B. `d` für directory)
Spalten 2–10: Zugriffsrechte für die Datei
Spalte 11 : Anzahl der Verweise (hier nicht wichtig)
Spalte 12 : Benutzer/Besitzer
Spalte 13 : Größe der Datei/des Verzeichnisses
Spalte 14–16: Datum bzw. Zeitpunkt der letzten Inhaltsänderung der Datei
Spalte 17 : Name der Datei

Das Recht executable bedeutet bei Verzeichnissen, daß man mit `cd` hineinwechseln darf. Bei Dateien macht dieses Recht natürlich nur Sinn, wenn dahinter auch ein ausführbares Programm steht. writeable bedeutet bei Verzeichnissen, daß man Dateien und Unterverzeichnisse anlegen und löschen darf.

Wie kann man Rechte verändern, um zum Beispiel zu verhindern, daß andere Benutzer in das eigene Verzeichnis hineinschauen können? Dazu dient das Kommando `chmod`. Die gewünschte Zugriffsänderung kann auf verschiedene Arten spezifiziert werden: `chmod`

- durch Angabe
  - der betroffenen Benutzergruppe,
  - der Änderungsoperation sowie
  - des Zugriffsrechts.

  $\Longrightarrow$ Beispiel:

  ```
  $ chmod g-r  filename
  $ chmod o+x  filename
  $ chmod o+x,g-r filename
  $ chmod go-r filename
  $ chmod go=rw filename
  ```

- durch Angabe einer Oktalzahl.

  Hierbei wird der gesamte Satz an Zugriffsrechten durch eine Oktalzahl neu spezifiziert. Die Oktalzahlen ergeben sich folgendermaßen:

  1 entspricht executable
  2 entspricht writeable
  4 entspricht readable

  Diese Zuordnung geschieht für jede Benutzergruppe einzeln. Dadurch entsteht insgesamt eine dreistellige Zahl.

  $\Longrightarrow$ Beispiel:

  Für die Datei *filename* sollen die Rechte `rwxr-xr--` gesetzt werden:

  $$\begin{array}{ccc} \mathtt{r\ w\ x} & \mathtt{r\ -\ x} & \mathtt{r\ -\ -} \\ \underbrace{4\ 2\ 1}_{7} & \underbrace{4\ 0\ 1}_{5} & \underbrace{4\ 0\ 0}_{4} \end{array}$$

  Also wäre das richtige Kommando:

  ```
  $ chmod 754  filename
  ```

3. Kommandos

3.1. Verständigung mit dem Betriebssystem

Wie kommuniziert man mit UNIX? Die meisten Betriebssystembefehle sind nichts anderes als C-Programme, die über die sogenannten system calls auf den UNIX-Kern (kernel) zugreifen.

Die Kommunikation zwischen Benutzer und UNIX läuft immer über einen Kommandointerpreter (shell). Das ist bei DOS genauso, nur fällt es da nicht so auf[7]. Die Shell hat die Aufgabe, Eingaben anzunehmen, eventuell selbst zu verarbeiten und an die entsprechenden Stellen weiterzuleiten. Es gibt verschiedene Shells; allerdings sind nicht immer alle auf jedem UNIX-Rechner vorhanden.

Auf den SUN- und IBM-Rechnern des RZs ist die Kornshell (`ksh`) voreingestellt.

3.2. Kommandoaufrufe

Nach dem Anmelden (login) wird vom Login-Prozeß die Shell gestartet. Sie meldet sich mit dem Prompt. Ab jetzt kann der Benutzer mit dem System arbeiten.

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

Manchmal dauert die Ausführung eines Programms sehr lange. Wenn dieses Programm keine Benutzereingaben benötigt, wäre es gut, wenn man gleichzeitig noch etwas anderes tun könnte. Hier hilft UNIX durch seine Mehrprozeßfähigkeit: Wenn man an

Hintergrundprozesse

ein Kommando ein `&` anhängt, wird das Programm im Hintergrund gestartet, und die Shell meldet sich sofort wieder mit dem Prompt zurück. Angenommen, wir haben ein Programm mit dem Namen `coffee`, dessen Ausführung viel Zeit in Anspruch nimmt, und ein Programm `tea`, das deutlich schneller geht. Dann würde man folgendermaßen verfahren:

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

Nach der Beendigung des Programms `coffee` erhält der Benutzer eine Nachricht.

`jobs`

Wenn man nicht mehr genau weiß, welche Programme noch im Hintergrund laufen, kann man sie sich durch das Kommando

```
$ jobs
```

ansehen.

[7] Unter DOS heißt die Shell `command.com`.

3.3. Argumente

Kommandos benötigen im allgemeinen Argumente (Ausnahmen sind Kommandos wie `pwd, exit`...). So kann man z. B. `mkdir` nicht ohne Angabe eines Verzeichnisnamens aufrufen[8].

Normalerweise ist das kein Problem. Dennoch muß man aufpassen, da bei einigen Befehlen die Wirkung sehr stark von der Art des Parameters abhängt.

⟹ Beispiel:

`$ mv` *file1 file2*	ändert den Dateinamen von *file1* nach *file2*
`$ mv` *file1 dir1*	verschiebt die Datei *file1* in das Verzeichnis *dir1*
`$ mv` *file1 dir1*`/`*file2*	verschiebt die Datei *file1* in das Verzeichnis *dir1* und ändert ihren Namen in *file2*
`$ mv` *dir1 dir2*	benennt das Verzeichnis *dir1* in *dir2* um (falls es *dir2* noch nicht gibt) oder verschiebt das Verzeichnis *dir1* (samt Unterverzeichnissen) in das bereits vorhandene Verzeichnis *dir2*
`$ mv` *dir1 dir2*`/`*dir3*	verschiebt einen ganzen Verzeichnisbaum in ein anderes (vorhandenes) Verzeichnis und ändert den Namen

3.4. Optionen

Wie bereits erwähnt, besteht eine Stärke von UNIX darin, daß es eine große Anzahl einfacher, hochspezialisierter Werkzeuge zur Verfügung stellt. Dabei ist *einfach* wohl eher ein Understatement, da die meisten Werkzeuge durch eine oftmals sehr große Reihe von Optionen in ihrer Wirkung gesteuert werden können. Diese Optionen, auch flags genannt, werden in der Regel mit einem Minuszeichen „`-`“ eingeleitet.

Auch bei der Angabe von Optionen muß man aufpassen, weil die Wirkung oft anders ist, als man erwartet, insbesondere, wenn man mehrere Optionen miteinander kombiniert.

Der einfachste Fall ergibt sich, wenn die Wirkungen der Optionen voneinander unabhängig sind.

⟹ Beispiel: `ls`

`$ ls -l`	zeigt ein langes Listing (die Zugriffsrechte werden mitangezeigt)
`$ ls -F`	hängt einen `/` an Verzeichnisse an
`$ ls -lF`	zeigt ein langes Listing mit zusätzlichem `/` am Ende von Verzeichnisnamen

[8] Ausprobieren! :-)

Bei voneinander abhängigen Optionen kann es zu Unklarheiten kommen. So zeigt `ls -l` auch den Zeitpunkt der letzten Inhaltsänderung der Datei an.

```
$ ls -lc
```

zeigt hingegen den Zeitpunkt der letzten allgemeinen Änderung an. Dies muß ja keine Inhaltsänderung sein, sondern kann z. B. auch eine Änderung der Rechte durch das Kommando `chmod` sein. Die Option `c` verändert also die Wirkung der Option `l` beim Kommando `ls`!

Die Option `c` allein bewirkt gar nichts. Sie wirkt nur im Zusammenhang mit der Option `l`. Entsprechendes gilt für die Option `u`:

```
$ ls -lu
```

zeigt den Zeitpunkt des letzten Zugriffs an. Lesen ist zum Beispiel ein Zugriff, der keine Änderung hervorruft. Jetzt ist offensichtlich, daß sich die Optionen `u` und `c` beim `ls`-Kommando gegenseitig ausschließen.

Auch bei der Reihenfolge der Optionen muß man im allgemeinen aufpassen und vorsichtshalber in den Manpages nachlesen. Bei `ls` ist die Reihenfolge unerheblich (`ls -ulF, ls -lFu, ls -Flu` usw. ergeben den gleichen Output), aber das muß nicht immer so sein[9]!

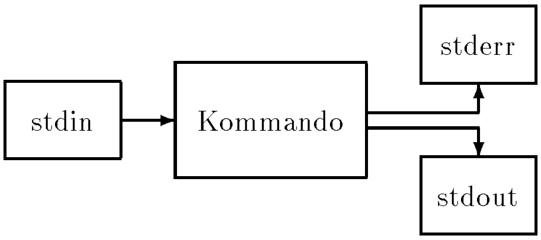
Desweiteren sei darauf hingewiesen, daß sich die möglichen Optionen zu einem Kommando von UNIX-Derivat zu UNIX-Derivat unterscheiden können. Auf einige Beispiele wird in den Kapiteln 4.2 und 5.1 näher eingegegangen.

3.5. Datenumlenkung

Standardkanäle

UNIX-Kommandos besitzen normalerweise drei Kommunikationskanäle:

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



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

Datenumleitung

[9] Beispielsweise ist GNU-Implementation von `tar` sehr empfindlich, was die Reihenfolge der Optionen betrifft.

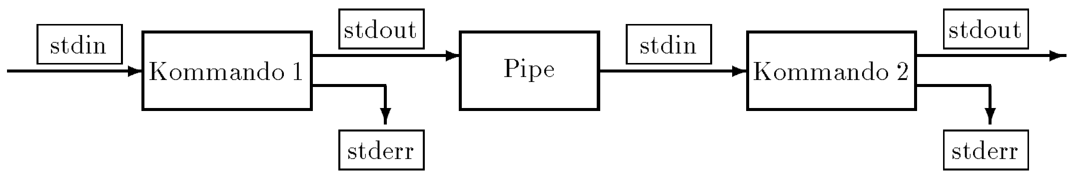
Kommunikationskanäle in andere Dateien umzulenken:

$ *Unixkommando* > *Ausgabedatei*	lenkt stdout in Ausgabedatei um
$ *Unixkommando* >> *Ausgabedatei*	lenkt stdout in Ausgabedatei um und hängt die Information an eine eventuell bestehende Datei an
$ *Unixkommando* < *Eingabedatei*	lenkt stdin in Eingabedatei um
$ *Unixkommando* 2> *Fehlerdatei*	lenkt die Fehlermeldungen in eine Datei um

⟹ Beispiel:

`$ ls -lF >` *xxx*	Die erzeugte Liste wird in *xxx* abgelegt.
`$ mkdir` *test*	
`$ rm` *test* 2> *fehler*	In der Datei *fehler* steht jetzt die Fehlermeldung, daß *test* ein Verzeichnis ist und folglich nicht mit `rm` gelöscht werden kann.

Häufig möchte man die Ausgabe eines Kommandos als Eingabe für das nächste Kommando verwenden, ohne daß die Zwischenergebnisse in Form einer Datei hinterher noch benötigt werden. Dafür bietet UNIX die Pipes (Röhre, Rohrleitung). Damit kann man die Standardausgabe eines Kommandos direkt auf die Standardeingabe des nächsten Kommandos umleiten. Als Zeichen für die Pipe dient der senkrechte Strich: | . Pipes



⟹ Beispiel:

```
$ ls -lF /dev | more
$ cat /etc/passwd | more
$ cat /etc/passwd | grep ddegen
```

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

Programme, die ihre Eingabe von stdin bekommen, Operationen mit diesen Eingaben ausführen und das Ergebnis nach stdout schreiben, heißen *Filter* (z. B. `grep, cat, more, sort`[10]). Filter

[10] `man grep; man sort` ;-)

4. Prozesse

4.1. Definition

Was ist ein Prozeß?

Ein Prozeß ist die Folge von Aktivitäten, die zwischen dem Aufruf eines Programms und dessen Ende stattfinden. Von jedem Kommando wird ein neuer Prozeß gestartet.

4.2. Prozeßabbruch

Angenommen, man hat ein Programm gestartet, das in eine Endlosschleife gerät. Wie kann man den dazugehörigen Prozeß trotzdem beenden?

Läuft der Prozeß im Vordergrund, so läßt er sich normalerweise[11] mit `CTRL-C` abbrechen. Diese Tastenkombination ist das Signal für jeden Prozeß, einen *geordneten Abbruch* durchzuführen. Das bedeutet zum Beispiel, daß geöffnete Dateien noch geschlossen werden u. a.

Wenn ein Prozeß nicht mehr auf `CTRL-C` reagiert, ist das dennoch kein Grund zur Panik: Wir arbeiten schließlich mit einem Multitasking-System. Man geht dann einfach in ein neues Fenster oder an ein anderes Terminal und sucht die *Prozeßidentifikationsnummer PID* des hartnäckigen Prozesses. Dazu verwendet man das Kommando `ps` (process status):

`ps`

```
$ ps
```

`ps` listet die eigenen Prozesse tabellarisch auf. Um eine ausführliche Liste aller auf der Maschine laufenden Prozesse zu bekommen, verwendet man auf BSD-UNIX-Systemen (z. B. cipsuns) `ps` mit den Optionen `-aux`; auf System-V-Systemen (z. B. RZ-Suns, mibms) benötigt man dagegen die Optionen `-ef`. Jetzt muß man in der Spalte `COMMAND` den Prozeß suchen, den man „abschießen“ möchte. In der Spalte `PID` findet man die zugehörige Prozeßidentifikationsnummer. Jetzt kann man den hängengebliebenen Prozeß durch den Befehl

`kill`

$ `kill` *pid*

abbrechen. Führt das immer noch nicht zum Erfolg, so kann man `kill` mit der Option `-9` aufrufen – kein Prozeß kann

$ `kill -9` *pid*

ignorieren.

[11] ... aber eben nicht immer

5. Peripherie

Zur Peripherie gehört alles, was nicht direkt Bestandteil des Rechners selbst ist, beispielsweise Drucker, Bandmaschinen, Streamer usw.

5.1. Drucker

Für UNIX stellt das Drucken einfach nur das Kopieren einer Datei in eine andere Datei, nämlich den Drucker, dar. Das Druckkommando lautet `lp, lpr`

`$ lpr -P`*druckername druckfile*	(BSD)
`$ lp -d`*druckername druckfile*	(System V[12])

Welche Drucker im System zur Verfügung stehen, erfährt man durch das Kommando `lpstat`

`$ lpstat -p`

`-P` und `-p` sind zwei völlig verschiedene Optionen! ⚠

⟹ Beispiel:

Im cipsun-Pool gibt es die Drucker `ascii` für einfache Textdateien und `ps` für Postscriptdateien[13]. Eine ASCII-Datei würde man also mit

`$ lpr -Pascii` *ASCII-Datei*

ausdrucken, während man für eine Postscriptdatei den Befehl

`$ lpr -Pps` *Postscriptdatei*

verwenden würde.

UNIX bearbeitet Druckaufträge nicht direkt, sondern stellt sie in eine Warteschlange (queue), denn es kann ja sein, daß mehrere Benutzer zur etwa gleichen Zeit Druckaufträge abgegeben haben. Für die Weiterverarbeitung ist dann ein ständig im Hintergrund laufendes Systemprogramm (der *line printer daemon* `lpd`) zuständig. Mit Hilfe des Kommandos `lpd`

`$ lpq -P`*druckername*

kann man sich die Druckerwarteschlange ansehen und überprüfen, ob der eigene Druckauftrag eingereiht worden ist. Dieses Kommando zeigt einem auch die *Auftragsnummer* des Druckjobs. Das ist nützlich, wenn man einen Druckauftrag wieder aus der Warteschlange herausnehmen will: `lprm, cancel`

`$ lprm -P`*druckername auftragsnummer*	(BSD)
`$ cancel` *auftragsnummer druckername*	(System V)

[13] Beide Drucker werden in Wirklichkeit durch einen einzigen Laserdrucker repräsentiert. Man bezeichnet sie daher als *virtuelle Drucker*.

5.2. Externe Speicher

5.2.1. Bandgeräte

Für Backups (Datensicherungen) werden häufig Bandgeräte benutzt. Als Benutzer(in) des Rechenzentrums der Universität Freiburg wird man jedoch nicht damit konfrontiert, da für die Datensicherung die jeweiligen Systemadministratoren und Systemadministratorinnen verantwortlich sind[14]. Lediglich ein Kommando soll in diesem Zusammenhang erwähnt werden, da es sich auch dazu eignet, ganze Dateibäume innerhalb des Filesystems zu verschieben oder zu kopieren. Es handelt sich hierbei um das schon in Kapitel 1.2 erwähnte Kommando `tar`[15].

`tar`

5.2.2. Diskettenlaufwerke

Manche Workstations sind mit Diskettenlaufwerken zum Speichern kleinerer Datenmengen ausgestattet. Im Gegensatz zu PC-Diskettenlaufwerken besitzen sie in der Regel keinen mechanischen Auswurf, sondern werden ausschließlich über Software gesteuert (Der Befehl „Diskettenauswurf" lautet `eject`.).

`eject`

Der fehlende Auswurfknopf ist im übrigen keine Schikane, sondern eine sehr sinnvolle Angelegenheit: UNIX kennt bekanntlich keine Laufwerke. Möchte man zum Beispiel eine Datei auf eine Diskette kopieren, dann kann man zwar die Diskette ins Laufwerk schieben – das UNIX-Filesystem weiß jedoch noch nichts von der Existenz eines Dateisystems auf Diskette. `cp` kann aber nur innerhalb eines Dateisystems kopieren. Folglich muß man das Filesystem der Diskette erst in das UNIX-Dateisystem einbinden. Diesen Vorgang nennt man *mounten*[16]. Drückte man jetzt den Auswurfknopf, gäbe es einen Bereich im Filesystem, der kein physikalisches Äquivalent hätte – Schäden am Dateisystem wären die Folge[17].

Mounten

Es gibt noch eine weitere Möglichkeit, auf (V)FAT-formatierte DOS-Disketten zuzugreifen. Man benötigt dazu die *mtools*, eine Sammlung von Werkzeugen, die wie DOS-Kommandos mit einem *m* davor heißen (`mcd, mdir, mcopy, mdel, mmd` etc.) und in etwa das tun, was man vom entsprechenden DOS-Kommando erwarten würde. Das Diskettenlaufwerk wird in der Regel mit `a:` angesprochen, als Trennzeichen wird der Slash `/` verwendet.

mtools

⟹ Beispiel:

`$ mcopy` *unixfile* `a:`*DOS-Unterverzeichnis*`/`*Dateiname_unter_DOS*

---

[14] Sollte dennoch Bedarf bestehen (zum Beispiel für einen Datentransport), so wende man sich an den Systemverwalter.

[15] `man tar`

[16] Der entsprechende Befehl `mount` darf in der Regel nicht von „normalen" Benutzern ausgeführt werden.

[17] Aus diesem Grund sollte man auf einem Linux-PC höllisch aufpassen, wenn man Disketten mountet: Der Auswurfknopf darf erst dann betätigt werden, wenn die Diskette wieder *unmountet* wurde.

6. Eigenschaften der Shell

Die *Shell* stellt, wie erwähnt, die Schnittstelle zwischen Kernel und User dar. Da im Rechenzentrum die Kornshell (`ksh`) die am meisten genutzte (da auf den cip- und RZ-Suns sowie den IBM-Maschinen voreingestellte) Shell ist, sollen einige ihrer Eigenschaften im folgenden besprochen werden.

6.1. Startdateien

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

Dies ist zunächst die Datei `/etc/profile`, die vom Systemadministrator gewartet wird und auf die die Benutzer keinen Schreibzugriff haben. Anschließend wird die Datei `.profile` im Homeverzeichnis des Benutzer abgearbeitet. Somit sind die Einstellungen festgelegt, die für die gesamte Sitzung gelten sollen (Es sei denn, sie werden während der Sitzung vom Benutzer ausdrücklich geändert.). `/etc/profile` `.profile`

Anschließend werden die benutzerdefinierten Voreinstellungen für die Kornshell geladen, indem die Datei `.kshrc`, so im Homeverzeichnis des Benutzers vorhanden, ausgeführt wird. Diese Datei wird übrigens jedesmal ausgewertet, wenn eine neue Kornshell aufgerufen wird – sei es durch das Kommando `ksh` oder die Eröffnung eines neuen Fensters. `.kshrc`

Dateien wie `.profile` oder `.kshrc`, die mit einem Punkt beginnen, nennt man auch *versteckte Dateien*, da sie normalerweise vom Kommando `ls` nicht angezeigt werden. Es handelt sich dabei in der Regel um Konfigurationsdateien für verschiedene UNIX-Programme, die im Normalbetrieb meistens weniger von Interesse sind und die anzuzeigende Liste nur unnötig vergrößern. Wenn man `ls` jedoch mit der Option `-a` aufruft, werden auch die versteckten Dateien wieder mit aufgelistet. `ls`

$\Longrightarrow$ Beispiel:

```
$ ls -al
```

6.2. Umgebungsvariablen

In den o. g. Systemdateien werden insbesondere *Umgebungsvariablen* gesetzt. Man kann sich mit dem Befehl `env`

```
$ env
```

alle gesetzten Variablen ausgeben lassen – sie werden in der Regel groß geschrieben. Man kann sich die Definition jeder einzelnen Umgebungsvariablen aber auch mit dem Kommando `echo`

$ echo $*umgebungsvariable*

ansehen. Dabei muß man ein $-Zeichen direkt vor den Variablennamen schreiben.

⟹ Beispiel:

PATH, HOME

```
$ echo $PATH
```

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

export

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

Das abschliessende :$PATH stellt sicher, daß der alte Pfad zusätzlich erhalten bleibt; die Variable HOME enthält den Pfad des eigenen Homeverzeichnisses. ⚠ Wenn man diese Zeile eingibt, gilt der neue Pfad natürlich nur für die aktuelle Shell in der aktuellen Sitzung. Beim nächsten Einloggen ist er vergessen. Deswegen kann man das Kommando als zusätzliche Zeile in die Datei .profile im eigenen Homeverzeichnis eintragen.

6.3. Aliase

alias

So angenehm die Möglichkeit, mit Hilfe von Kommandooptionen, Pipes etc. mächtige Befehle zu erzeugen, auch sein mag – sie ist jedesmal mit einer gewaltigen Schreibarbeit verbunden. Deshalb gibt es unter UNIX das Kommando alias, mit dem man sich eigene Abkürzungen für häufig gebrauchte Befehle definieren kann.

⟹ Beispiel:

Wenn man auf das aus der DOS-Welt bekannte Kommando dir zum Auflisten von Verzeichnissen nicht verzichten will, wäre eine Möglichkeit

```
$ alias dir='ls -al | more'
```

Jedesmal, wenn man jetzt dir eingibt, wird der Inhalt des entsprechenden Verzeichnisses in Langform inklusive aller versteckter Dateien seitenweise angezeigt – das definierte Kommando entspricht also eher dem DOS-Befehl dir /p.

⚠ Aliase gehen ebenso wie neudefinierte Umgebungsvariablen mit Beendigung der Shell, in der sie definiert wurden, verloren. Benötigt man einen Alias auch für zukünftige Sessions, so muß man ihn in einer Konfigurationsdatei (.profile, .kshrc) setzen.

6.4. Dateinamenexpansion

Wildcards

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

* steht für eine beliebige Zeichenkette,

`?` steht für ein einzelnes Zeichen und

`[`*Zeichenmenge*`]` steht für einen bestimmten Zeichensatz.

⟹ Beispiel:

`$ cat p*`	zeigt den Inhalt aller Dateien an, die mit einem kleinen `p` beginnen
`$ cat p?`	zeigt den Inhalt aller zweibuchstabigen Dateien an, die mit einem kleinen `p` beginnen
`$ cat [pP]*`	zeigt den Inhalt aller Dateien an, die mit einem kleinen `p` oder einem großen `P` beginnen
`$ cat p*[1-3]`	zeigt den Inhalt aller Dateien an, die mit einem kleinen `p` beginnen und mit einer `1`,`2` oder `3` enden

Die Kornshell besitzt überdies einen Mechanismus, der versucht, angefangene Dateinamen zu vervollständigen. Er wird durch zweimaliges Drücken der `ESC`-Taste in Gang gesetzt:

Komplettierung von Dateinamen

⟹ Beispiel:

Man möchte sich die Datei `Datei_mit_superlangem_Dateinamen` seitenweise ausgeben lassen. Dazu tippt man ein

```
$ more Dat
```

und drückt anschließend zweimal die `ESC`-Taste.

6.5. Kommandowiederholung

Die Kornshell verfügt über einen sogenannten *History-Mechanismus*, d. h., sie „merkt" sich alle abgeschickten Kommandozeilen. Mit Hilfe der *Pfeiltasten* ↓ und ↑ kann man diese Kommandos „wiederverwenden".

Der History-Mechanismus

7. Unix-Rechner im Netzwerk

Allein die Tatsache, daß UNIX ein Multiuser-Betriebssystem ist, bedeutet, daß man auch „von außen" Zugriff auf das Rechnersystem haben muß. Sonst könnte doch wieder nur ein User den Rechner benutzen, nämlich der, der gerade davor sitzt. Der zeitgleiche Zugriff mehrerer Benutzer(innen) auf einen Rechner wird über das Netzwerk ermöglicht.

7.1. Anmelden an entfernten Unix-Rechnern

`telnet` Mit dem Kommando `telnet` kann man sich auf einem entfernten (remote) Rechner im Internet anmelden – vorausgesetzt, man besitzt eine UserID auf diesem Rechner:

`$ telnet` *rechnername*

rechnername ersetzt man entweder durch den symbolischen Internetnamen oder die entsprechende IP-Nummer (Internet-Protokoll-Nummer) des Remoterechners.

⟹ Beispiel:

```
$ telnet cipsun0
$ telnet cipsun0.ruf.uni-freiburg.de
$ telnet 132.230.1.80
```

Man wird dann ganz normal nach UserID und Paßwort gefragt. Die UserID auf dem entfernten Rechner muß jedoch nicht mit der auf dem lokalen Rechner übereinstimmen. Je nach Betriebssystem auf dem Remoterechner meldet man sich wie gewöhnlich mit `exit`, `quit`, `logoff` etc. ab.

`ssh` Ähnlich wie `telnet` funktioniert das Kommando

`$ ssh` *rechnername*

Wenn man auf dem Remoterechner eine andere UserID hat, muß man diese mit der Option `-l` angeben.

⟹ Beispiel:

`$ ssh -l` *UserID* `sun2`

`ssh` steht dabei für Secure Shell: Beim Einloggen auf Remoterechner via `telnet` o. ä. werden sämtliche Daten (einschließlich des Paßworts!) im Klartext übertragen, d. h., sie können von Unbefugten abgehört werden. Dem beugt `ssh` vor, indem der gesamte Datenverkehr verschlüsselt übertragen wird. Netter Nebeneffekt: man kann (sofern man lokal unter X arbeitet) auf dem Remoterechner Programme mit graphischer Oberfläche starten und bekommt sie auf dem lokalen Bildschirm angezeigt.

7.2. Dateitransfer auf entfernte Unix-Rechner

Um Dateien von einem Rechner auf einen anderen zu übertragen, kann man auf das *File Transfer Protocol* `ftp` zurückgreifen. Da `ftp` – wie alle klassischen Internetdienste – sämtliche Daten im Klartext überträgt, man sich aber beim Remote-Rechner mit Paßwort identifizieren muß, sollte `ftp` nur dann zum Einsatz kommen, wenn es keinen sicheren Übertragungsweg gibt[18]. `ftp`

Stattdessen bietet das Kommando `scp` („Secure Copy") die Möglichkeit, Dateien sicher verschlüsselt von einem Rechner auf einen anderen zu kopieren – vorausgesetzt, man hat auf beiden Rechnern einen Account. `scp`

`$ scp` *filename remote_UserID*`@`*Remote-Rechner* : *Pfad/und/Dateiname*

Wenn man auf beiden Rechnern dieselbe UserID hat, kann man sie auch weglassen:

`$ scp` *filename Remote-Rechner* : *Pfad/und/Dateiname*

7.3. Andere Internetdienste

UNIX hat – im Gegensatz zu anderen Betriebssystemen – viele Programme integriert, die die Nutzung von Internetdiensten erlauben. Außer den schon erwähnten Befehlen `telnet`, `ssh`, `ftp` und `scp` wird man auf UNIX-Machinen auch Programme zur Echtzeitkommunikation zwischen Benutzern (z. B. `talk`), *Newsreader, Webbrowser* und selbstverständlich *Mailreader*[19] finden. Internetdienste unter UNIX

[18] Dies gilt selbstverständlich nicht für den anonymen Zugriff auf FTP-Server. Hier wird man sicherlich auch in Zukunft seinen FTP-Clienten benutzen.

[19] *Webbrowser* nennt man Programme, die es einem ermöglichen, sich im *World Wide Web* zu bewegen; *Mailreader* helfen beim Lesen, Versenden und Verwalten von *electronic mail*, kurz *e-mail*; *Newsreader* tun dasselbe mit den *Netnews*, den elektronischen Diskussionsgruppen.

Extro

Dieser Kompaktkurs konnte nur die ersten Schritte in UNIX vermitteln. Viele weitergehende Möglichkeiten dieses Betriebssystems wurden nicht erwähnt. So kann man beispielsweise die Shell als Programmiersprache verwenden (Shellprogrammierung). Auch die Frage, wie man unter UNIX Internetdienste (E-Mail, News, WWW usw.) nutzt, wurde lediglich angerissen[20]. Und das ist erst die Spitze des Eisberges ...[21]

Auf vielen Rechnern des Rechenzentrums findet sich ein Programm mit dem Namen `hilfe`, das Informationen über UNIX im allgemeinen und über UNIX im speziellen Rechner-Pool erteilt. Darüber hinaus bietet die Anwenderberatung (Raum -103) bei den meisten Problemen fachkundige Hilfe an.

[20] In diesem Zusammenhang sei auf den Internetkurs des Rechenzentrums verwiesen.
[21] Achtung, UNIX kann süchtig machen ;-)

Anhang: Die 20 nächstwichtigen Unixkommandos

Die Überschrift sollte man nicht allzu wörtlich nehmen: Die folgende Liste wichtiger UNIX-Programme wurde völlig willkürlich[22] erstellt. Darin sind die Kommandos in alphabetischer Reihenfolge *ohne* Hinweise zu Argumenten und Optionen aufgeführt. Mit einem `*` sind dabei diejenigen Kommandos gekennzeichnet, die man auf *jedem* UNIX-system finden wird.

Die Liste soll anregen zum Selbststudium. Wichtigstes Hilfsmittel dazu ist das nun schon hinlänglich bekannte Kommando `man`.

`archie`	Von welchem ftp-Server kann ich mir Programm XY holen? Archies Bruder mit graphischer Bedienoberfläche läßt sich mit `xarchie` aufrufen, vorausgesetzt, man arbeitet auf einer graphischen X-Window-Oberfläche (Slang: „unter X").
`clear *`	löscht den Bildschirm
`cc`	ein C-Compiler
`date *`	Wie spät ist es; welches Datum haben wir heute?
`elm`	ein komfortabler Mailreader
`emacs`	ein äußerst mächtiger Editor (für manche Leute *der* Editor überhaupt)
`find *`	Suche nach bestimmten Dateien im Verzeichnisbaum
`finger`	Der Spion: Ist Mr. X. auf Maschine Y eingeloggt? (Ist auf vielen Maschinen im Internet disabled.)
`grep *`	Suche nach bestimmten Textmustern
`ln *`	Wenn ein- und dieselbe Datei an verschiedenen Stellen im Dateisystem benötigt wird, muß man sie nicht vielfach ablegen – es reicht, einen Verweis („link") zu setzen.
`lynx`	ein Webbrowser auf Textbasis – keine bunten Bildchen, aber dafür schnell
`pico`	ein einfach zu bedienender Texteditor
`pine`	noch ein Mailreader (besonders anfängerfreundlich, da exzessiv menügeführt; in neueren Versionen auch als Newsreader zu mißbrauchen)
`quota *`	Wieviel Platz ist noch auf meinem account? (Auf den IBM-Maschinen des RZs gibt es dieses Kommando selbstverständlich auch – es nützt einem aber nichts, da deren Dateisystem Teil des weltweiten Filesystems *AFS* ist. Mehr zum AFS und wie man dort seine Quota abfragt, findet man in [5].)

[22] d. h. nach persönlichen Präferenzen der Autorin ;-)

`tar *` (tape archiver) ein Archivierwerkzeug, das nicht nur auf Bänder, sondern auch in Dateien schreiben kann. Zu sichernde Dateien, Verzeichnisse und Dateibäume lassen sich damit in einer Sicherungsdatei, einem *tar-Archiv*, gemeinsam abspeichern und daraus auch wieder originalgetreu extrahieren.

`tin` ein komfortabler Newsreader

`uuencode` Zur Datenübertragung sind nur Zeichen zugelassen, die dem 7-Bit-ASCII-Zeichensatz angehören. Ausführbare Programme, Bilder, Sounddateien u. a. enthalten aber in der Regel noch zusätzliche Zeichen. Damit diese Sonderzeichen beim Datentransport (z. B. per e-mail) nicht verlorengehen, sollte man solche Dateien vor dem Versand so kodieren, daß sie nur noch 7-Bit-ASCII-Zeichen enthalten. Diese Aufgabe übernimmt `uuencode`. Dekodiert wird mit `uudecode`.

`unalias *` das Gegenteil von `alias` (löscht Aliase)

`who *` Wer arbeitet noch auf derselben Maschine wie ich?

`who am i *` Wer bin ich? (für Vergeßliche ;-)

Literatur

[1] CBT[23]-Kurs UNIX der Fernuniversität Hagen.
Fernuniversität Hagen
Prof. Dr. G. Schlageter
Praktische Informatik I
Postfach 940
Hagen

[2] Manfred Heisterkamp: UNIX-*Leitfaden* (Kursskript des Rechenzentrums der Universität Freiburg).

[3] UNIX – *eine Einführung*, Regionales Rechenzentrum Hannover,
ca. 180 Seiten, 8,- DM, erhältlich im RZ.

[4] Daniel Gilly et. al. : UNIX *in a Nutshell*, O'Reilly & Associates,
39,- DM, deutsche und englische Ausgaben erhältlich.

[5] Werner Beiser, Roman Drahtmüller: *Einführung in die Benutzung der IBM-Workstations*[24] (Kursskript des Rechenzentrums der Universität Freiburg).

[6] Jens Vieler: *Einführung in Unix* (PDF-Dokument),
`http://hyperg.informatik.uni-freiburg.de/0x84e69832_0x00062b6c`

[7] *Unix-Einführung der Fakultät für Informatik der TU München*,
`http://www.informatik.tu-muenchen.de/rechenbetrieb/documents/anleitung/anl.toc.html`

[8] Tatu Ylönen: *SSH (Secure Shell) Remote Login Program*,
`http://www.cs.hut.fi/ssh/`

[9] Thomas König: *Secure Shell – Frequently Asked Questions*,
`http://www.uni-karlsruhe.de/~ig25/ssh-faq/`

[23] Computer Based Training
[24] z. Z. in Revision