UNIX Basiskurs

ETH Eidgenössische Technische Hochschule Zürich

Informatikdienste

Sektion
Beratung & Schulung

Jürg Luthiger

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Inhalt

Dieses Skriptum dient als Unterlage zum UNIX Basiskurs der Informatikdienste an der ETH Zürich. Ziel ist es, dem Einsteiger grundlegende Kenntnisse über das UNIX-Betriebsystem zu vermitteln. Ein Verständnis für die Basisprinzipien moderner Computersysteme sollte vom Teilnehmer mitgebracht werden, ist aber keine notwendige Bedingung für die Teilnahme an dem Kurs. Viele UNIX Einsteiger haben schon Erfahrung mit anderen Systemen, seien es Personal Computer oder Grossrechner. Begriffe wie Computer, Diskette, Speicher werden hier nur insofern definiert, falls sie eine spezielle Bedeutung im Zusammenhang mit dem UNIX Betriebssystem besitzen. Sonst werden sie als substantieller Teil der Begriffswelt eines Bewohners Mitteleuropas des ausgehenden 20. Jahrhunderts betrachtet und somit nicht näher diskutiert.

Folgende Inhalte sollen dem Teilnehmer vermittelt werden und die Grundlage für ein weiteres selbständiges Arbeiten und Experimentieren mit UNIX Systemen darstellen:

- Grundprinzipien und „Philosophie“
- Hierarchisches Filesystem
- Prozesse
- Verwendung der Shell Kommandosprache
- Manipulation von Information mittels Kommandos
- Editieren von Files
- Grundlegende Kommandos

Die Beschreibung der einzelnen Befehle und Konzepte erhebt keinen Anspruch auf Vollständigkeit, sie sollte allerdings ein Gefühl dafür vermitteln, wie UNIX Informationen verwaltet und verarbeitet. Für eine vollständige Beschreibung der Syntax und Semantik von Befehlen möchte ich auf die Verwendung der Manual Pages verweisen, die standardmässig mit dem UNIX Betriebssystem installiert werden und mittels `man` abgerufen werden können. In dieser Online Dokumentation sind detaillierte Informationen über Befehle, Konzepte, Programmierschnittstellen des betreffenden Systems und vieles mehr hinterlegt. Dieses Skriptum kann und soll diese Information nicht duplizieren.

1.2 UNIX Versionen

Da zwischen verschiedenen UNIX Implementationen zum Teil oft Unterschiede in der Syntax und Semantik von Kommandos bestehen, wird, wo es sinnvoll erscheint, auf diese Unterschiede hingewiesen. Es würde aber den Rahmen des Kurses sprengen, alle möglichen Unterschiede explizit aufzuzählen.

Ein besonderes Dilemma wird durch die Unterschiede in der Verwendung verschiedener Standard Kommandosprachen (Bourne Shell und C Shell, seit kurzem auch Korn Shell) hervorgerufen. Einerseits stellt die Bourne Shell einen generellen UNIX Standard auf allen UNIX Systemen dar - fast alle System Shell Prozeduren sind in dieser Sprache verfasst - andererseits wird die C Shell vor allem auf Systemen mit Berkeley UNIX (4.3BSD) als Standard Benutzerschnittstelle verwendet. Die Kompatibilität zwischen den beiden Kommandosprachen ist begrenzt. Die Bourne Shell besitzt viele Vorteile bei der Programmierung von Shell Prozeduren, die C Shell bietet eine etwas komfortablere Benutzerschnittstelle.

Die Korn Shell ist voll aufwärtskompatibel zur Bourne Shell aber nicht kompatibel zur C Shell. Auf die Korn Shell wird in diesem Kurs nicht speziell eingegangen. Benutzer, die die Korn Shell verwenden wollen, können problemlos von der Bourne Shell umsteigen und die Vorteile der Korn Shell nutzen.

Ich habe mich aufgrund der Verbreitung der SUN-Systeme, welche als Berkeley UNIX Systeme die C Shell verwenden, für die Syntax dieses Kommandointerpreters entschieden. Bourne Shell Beispiele werden jedoch dort aufgeführt, wo es mir sinnvoll erscheint, die Unterschiede der Shell Implementierungen zu unterstreichen.

1.3 Schrifttypen

Für laufenden Text dieses Skriptes wird die Schriftart verwendet, die Sie gerade vor sich sehen.

Namen von Kommandos und anderen Systembegriffen im Text werden in dieser Schriftart angegeben.

Beispiele werden `in dieser Schriftart,` dargestellt.

Kommentare in Beispielen sind durch # *Kommentartext in Schrägschrift* gekennzeichnet.

Beispiele sind zusätzlich durch einen Balken am linken Rand hervorgehoben, wobei ein einfacher Balken allgemeine oder Bourne Shell spezifische Beispiele kennzeichnet. Ein doppelter Balken bezeichnet ein C Shell Beispiel.

```
C Shell Beispiel
% Beispiel fuer Eingabe
```

```
Bourne Beispiel
$ Beispiel fuer Eingabe
```

Diese Konventionen gelten nur innerhalb dieses Skriptes und werden von den beschriebenen UNIX Systemen nicht angewendet. Das Dokument wurde mit Hilfe von LaTeX und XFIG auf einer SUN-Sparcstation produziert.

Zürich, den 24. August 1995

Kapitel 2

Anmeldung

2.1 Zugang zum System

Um sich an einem UNIX System einloggen zu können müssen zumindest zwei Voraussetzungen erfüllt sein:

Physischer Zugang

Eine physische Verbindung zum System über ein Terminal oder über das Netz muss vorhanden sein. Wenn das der Fall ist, sollte auf dem Bildschirm der Prompt

```
|| login:
```

erscheinen.

Logischer Zugang

Ein *Benutzername* (Account) auf dem UNIX System muss vom Systemverwalter eingerichtet werden.

2.2 Login

Geben sie nun ihren Benutzernamen nach dem Login Prompt ein und schliessen sie die Eingabe mit einem Carriage Return (`<cr>` bezeichnet im folgenden die Carriage Return oder Enter Taste) ab.

```
login: kurs<cr>
```

Falls Sie vom Systemverwalter ein *Passwort* bekommen haben, antwortet das System mit dem Prompt:

```
Password:
```

Geben Sie nun das Passwort ein und schliessen Sie wieder mit `<cr>` ab. Das Passwort erscheint aus Sicherheitsgründen nicht auf dem Bildschirm. Wenn Sie das Passwort richtig eingegeben haben, wird Ihre Shell gestartet. Eventuell erscheinen noch einige mehr oder weniger informative Meldungen auf dem Bildschirm (message of today), dann gibt die Shell ihren Prompt aus Wie dieser Prompt aussieht, hängt davon ab, welche Shell Sie verwenden (das wird normalerweise vom Systemadministrator festgelegt) und ob der Shell Prompt in ihrem Login File neu definiert wurde. Die Bourne Shell oder Korn Shell antworten standardmässig mit

```
$
```

Die C Shell antwortet mit einem

```
%
```

2.3 Kommandos

Das System ist nun bereit, Kommandos von ihrem Bildschirm entgegenzunehmen. Kommandos bestehen aus einer Reihe von Worten oder Symbolen, die durch *Blank* (Leertaste oder Tabulatorzeichen) voneinander getrennt sind und mit einem `<cr>` abgeschlossen werden. Das erste Wort ist der Name des Kommandos, die weiteren Worte stellen Argumente für dieses Kommando dar.

Einige Symbole besitzen eine spezielle Bedeutung für die Shell. Diese werden zuerst von der Shell interpretiert, dann erst wird das Kommando mit seinen eigentlichen Argumenten aufgerufen.

Das `<cr>` wird in den weiteren Beispielen nicht mehr explizit angegeben. Der Shell Prompt (`$`, oder `%` bei C Shell) wird im Text immer mit angegeben, um Benutzer Eingaben zu kennzeichnen, stellt jedoch keinen Teil der Eingabe dar.

Zur Philosophie von UNIX gehört es, möglichst einfache Kommandos, die jeweils nur eine ganz spezifische Aufgabe ausführen zur Verfügung zu stellen. Komplexere Aufgaben werden oft durch die Verbindung mehrerer einfacher Kommando mittels Pipelines und in Shell Scripts gelöst. Die Shell dient dazu, solche einfachen Kommandos auf verschiedenste Arten miteinander zu verknüpfen.

2.3.1 whoami

Ein einfaches Kommando ohne Argumente ist whoami. Es dient dazu, ihren *Benutzernamen* ausgeben.

```
% whoami
kurs
%
```

2.3.2 who

Der Befehl who zeigt an, welche Benutzer gerade am System eingeloggt sind. Die einzelnen Spalten beinhalten folgende Informationen

- Benutzername
- Terminal
- Zeitpunkt des Logins
- Name der Maschine, von der der Benutzer mittels *remote login* angemeldet ist

```
% who
root     console Jun 18 17:00
luthiger tty45   Jun 25 17:22
kurs     ttyp1   Jun 25 14:53   (host:0.0)
%
```

2.3.3 date

Mit date kann man das Systemdatum anzeigen

```
% date
Thu Jul  4 11:06:14 MET DST 1991
%
```

2.3.4 passwd

Falls Sie noch kein *Passwort* vom Systemverwalter bekommen haben, ist nun der beste Zeitpunkt dieses selbst zu tun. Das Kommando dazu heisst passwd. Das System fragt Sie um die Eingabe des neuen Passworts. Das neue Passwort ist ebenfalls nicht auf dem Schirm sichtbar. Nachdem Sie das neue Passwort eingegeben haben, bittet Sie das System, die Eingabe zu wiederholen. Falls Sie einen Tippfehler begangen haben, bekommen Sie nun die Chance ihr Passwort ein drittes Mal einzugeben. Das geht so lange, bis Sie zweimal hintereinander das gleiche Passwort eingegeben haben.

```
% passwd
Changing password for kurs.
New password:
Retype new password:
%
```

Sie können den Befehl passwd auch verwenden, um sich ein neues Passwort zu vergeben. In diesem Fall bittet Sie das System zuerst um die Eingabe des alten Passworts:

```
% passwd
Changing password for kurs.
Old password:
New password:
Retype new password:
%
```

Manche Systeme verlangen dass das Passwort nicht zu kurz ist und eine bestimmte Anzahl verschiedener Zeichen enthält. Nur die ersten 8 Zeichen des Passworts sind signifikant. Das erste Zeichen muss ein Buchstabe sein. Sonderzeichen innerhalb des Wortes sind jedoch erlaubt.

Beispiel:

aus der Wortfolge

der sommer,87,war schlecht

kann folgendes Passwort abgeleitet werden

ds,87,ws

Die 5 goldenen Regeln für ein gutes Passwort:

- Das Passwort soll man sich merken können.
- Die Mindestlänge bei Passwörtern mit grossen und kleinen Buchstaben beträgt 5 Zeichen, bei Passwörtern bestehend aus nur kleinen Buchstaben 6 Zeichen.
- Keine Namen, Geburtstage, Telefonnummern oder andere leicht zu erratende Wörter verwenden.
- Passwort von Zeit zu Zeit ändern.
- Passwort nicht auf einem Zettel in der Nähe des Systems aufgeschrieben aufbewahren.

2.3.5 exit

Um die Shell zu verlassen gibt es mehrere Möglichkeiten:

Eingabe eines EOF Zeichens durch gleichzeitiges Niederhalten der Tasten Control und D (Ctrl-D, ^D)

```
|| % ^D
```

oder den Befehl exit

```
|| % exit
```

oder logout

```
|| % logout
```

Das System sollte nun wieder mit dem login Prompt antworten, wenn Sie über ein Terminal eingeloggt sind. Bei einer Netzwerk Verbindung wird mit dem Verlassen der Login Shell auch die Verbindung unterbrochen.

Übungen

Melden Sie sich wie beschrieben an Ihrem UNIX System an und probieren Sie die Befehle who, date, whoami und passwd aus. Melden Sie sich wieder ab und probieren Sie das neue Passwort aus.

Kapitel 3

Orientierung

Nachdem Sie sich erfolgreich das erste mal am UNIX System eingeloggt haben, ist es nützlich, sich über ihre momentane Umgebung zu informieren. Dieses Kapitel beschreibt, wie Sie sich in einem UNIX Filesystem zurechtfinden.

3.1 pwd

Das Kommando `pwd` (print working directory) dient dazu, sich das *Working Directory* anzuzeigen. Nach dem Login ist das aktuelle Working Directory ihr *Home Directory.*

```
% pwd
/export/home/kurs
%
```

Der Name **`kurs`** ist wieder beliebig gewählt. Sinnvollerweise werden auf UNIX Systemen die Home Directories aller gewöhnlichen Benutzer unter einem eigenen Directory (hier **`/export/home`**) zusammengefasst und erhalten den Namen des Benutzers, der dieses Directory als Home Directory besitzt.

3.2 Filenamen

Der Aufbau von Filenamen ist von System zu System unterschiedlich. Es gibt aber bestimmte Mindestanforderungen.

Ein Filename besteht generell aus maximal 14 Zeichen. Es gibt auch Systeme, die längere Filenamen erlauben. Mit der Einschränkung auf 14 Zeichen ist man aber sicher, dass der Name auch wirklich von allen Systemen verstanden wird.

Filenamen können aus beliebigen Zeichen ausser Schrägstrich `/` (slash) bestehen. Der Schrägstrich wird vom Filesystem benötigt, um Directorynamen in Pfadnamen zu trennen.

UNIX unterscheidet zwischen Gross- und Kleinschreibung!

Gültige Filenamen sind z.B:

home	
kurs	
KURS	bezeichnet nicht dasselbe File wie kurs !
passwd	
#._a9&	

3.3 Hidden Files

UNIX kennt auch sogenannte Hidden Files, Sie werden auch als *Dot Files* bezeichnet, da sie mit einem . (dot) beginnen. Diese Files werden normalerweise nicht angezeigt.

Spezielle Namen sind . und ..

.	ist ein Verweis auf das aktuelle Directory
..	ist ein Verweis auf das übergeordnete Directory

3.4 Pfadnamen

Der Text /export/home/kurs wird als Pfadname bezeichnet. Ein Pfadname dient dazu, ein File oder ein Directory relativ zum *Root Directory* oder relativ zum *Working Directory* zu benennen. Ein Pfadname besteht aus durch das Zeichen / getrennte Namen. Die Namen usr, home und kurs sind Namen von Directories. Der Pfadname /export/home/kurs wird nun folgendermassen interpretiert:

/export/home/kurs ist ein *absoluter Pfadname* und bezeichnet einen Pfad relativ zum Root Directory. Alle Pfadnamen, die mit / beginnen sind absolute Pfadnamen. Ein / allein steht für das Root Directory selbst.

kurs ist ein Subdirectory von /export/home und /export/home selbst ist ein Subdirectory, das unter dem Directory /export liegt.

Relative Pfadnamen beginnen niemals mit einem /, sind aber sonst wie absolute Pfadnamen aufgebaut. Vor einem / kann natürlich in jedem Fall nur ein Name stehen, der ein Directory bezeichnet.

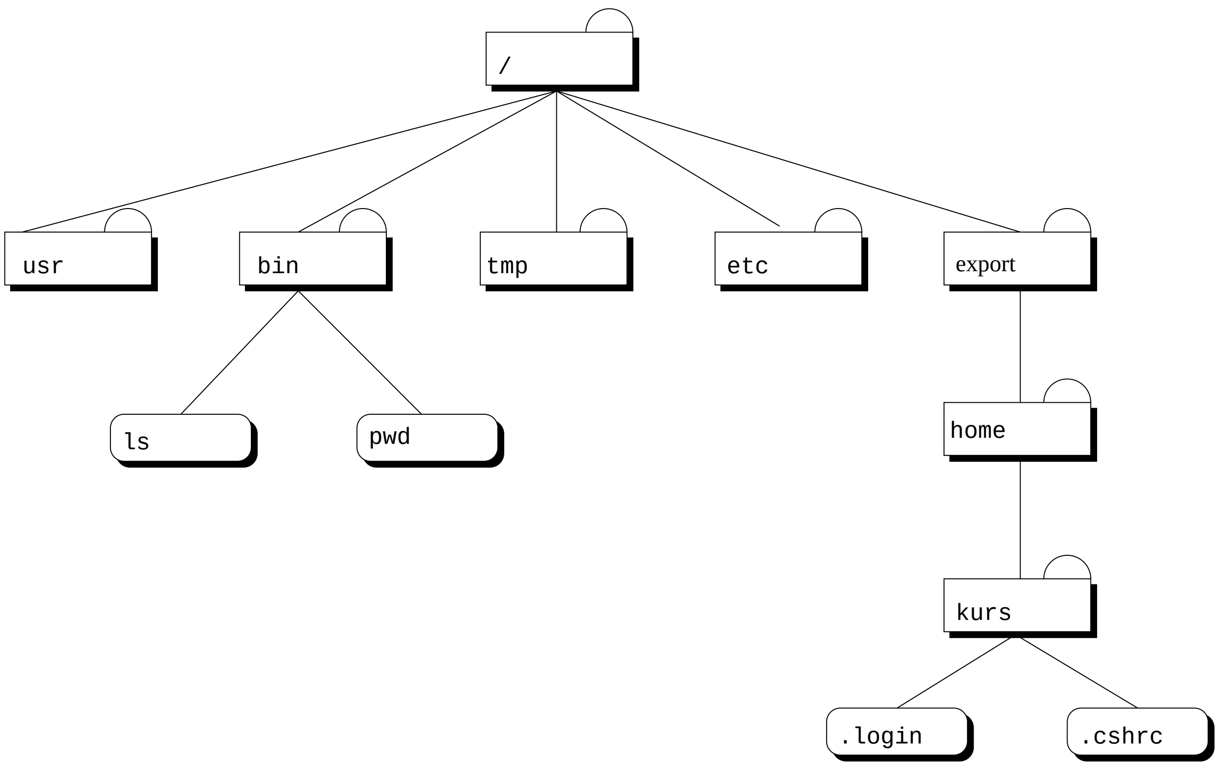
Gültige Pfadnamen sind z.B:

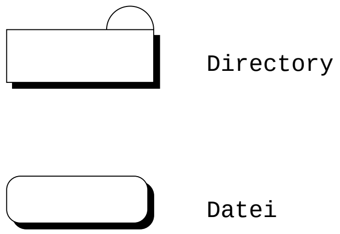
/	Root Directory
/export/home	Directory, das alle Home Directories von Benutzern enthält
/export/home/kurs	z.B. Ihr Home Directory
kurs	relativer Pfadname für Ihr Home Directory vom Directory `/export/home` aus gesehen
.	Ihr aktuelles Directory, auch Working Directory genannt
..	/export/home, falls Sie sich in einem Subdirectory von `/export/home` befinden
../kurs	Ihr Home Directory, von einem beliebigen Directory unter `/export/home` aus gesehen

Filenamen müssen innerhalb eines Directories eindeutig sein, können aber sonst in einem Filesystem beliebig oft vorkommen. Daraus folgt, dass Pfadnamen einen eindeutigen Namen eines File darstellen. Wie wir später sehen, kann aber ein und dasselbe File auch über mehrere absolute Pfadnamen angesprochen werden.

3.5 Directory Struktur

Die Struktur eines UNIX *Filesystems* lässt sich als (verkehrter) Baum mit `/` als Wurzel (*Root Directory*), den einzelnen Directories als Verzweigungen und den Files als Blättern darstellen.





Absolute Pfadnamen bezeichnen nun einen Weg durch diesen Graphen, von der Wurzel aus beginnend, wobei die Namen von Knoten, die auf dem Weg passiert werden, durch `/` getrennt aufgeschrieben werden (z.B: **`/export/home/kurs/.login /usr /bin/pwd /export/home/kurs`**)

Relative Pfadnamen bezeichnen einen Weg durch den Graphen vom aktuellen Working Directory aus beginnend, wobei die Namen von Knoten, die auf dem Weg passiert werden, durch `/` getrennt aufgeschrieben werden. Ein Schritt in Richtung der Wurzel wird durch den Namen `..` angegeben (z.B: **`.login . .. ../../bin/ls`**).

3.6 Standard Directories

Neben den Home Directories existieren unter UNIX eine Reihe von Standard Directories, die unter der Root angelegt sind.

/	Root Directory. Von diesem Directory aus sind alle anderen erreichbar.
/bin	Dieses Directory enthält *System Kommandos* wie pwd, ls oder passwd.
/etc	Enthält weitere Kommandos, die vorwiegend für System Administration bestimmt sind und eine Reihe von Systemtabellen. Das File /etc/passwd enthält z.B. alle Benutzer, die Zugangsberechtigung zu dem System haben.
/usr/bin	Enthält weitere System Kommandos wie z.B. den vi Editor.
/usr/man	Enthält die Online System Dokumentation
/lib	Enthält verschiedene *Bibliotheken*
/usr/lib	Weitere Bibliotheken
/usr/local	Lokale Kommandos und Bibliotheken
/usr/include	*Headerdateien* für C Programme
/dev	*Gerätedateien* für Terminals, Drucker, Festplatten, Disketten usw.
/tmp	Ablage für *temporäre* Files
/home	Home Directories für "gewöhnliche„ Benutzer Kann auf verschiedenen Systemen anders benannt oder auch weiter strukturiert sein (z.B: /usr/home, /home/maschinenname, /home/abteilung/projekt usw.)
/usr/mail	Enthält die *Briefkästen* der einzelnen Benutzer
/vmunix	Das File /vmunix (oder /unix) enthält das Betriebssystem (Kernel.)

Die meisten Files und Directories, die nicht im Home Directory eines Benutzers liegen, können zwar von diesem gelesen werden, er hat jedoch in der Regel keine Möglichkeit in solchen Directories irgendwelche Änderungen vorzunehmen (Ausnahmen sind z.B. /usr/spool/mail, /tmp).

3.6.1 ls

Der Befehl ls (list names) gibt den Inhalt eines Directories aus. ls ohne Argumente erzeugt eine einfache Liste, die nur die Namen aller Files des Working Directories ausgibt, die nicht mit . beginnen.

```
% ls
%
```

Als neuer Benutzer besitzen Sie normalerweise keine Files, deshalb wird hier auch nichts ausgegeben.

Im allgemeinen verstehen UNIX Kommandos *Argumente*, die die Wirkung des Kommandos steuern. Argumente, die mit einem *Minuszeichen* beginnen werden auch als *Optionen* bezeichnet und dienen dazu, gewisse Unterfunktionen des Kommandos an- oder abzuschalten. Weitere Argumente sind Pfadnamen oder anderer Text , der angibt, auf welche Daten das Kommando angewendet werden soll.

ls besitzt verschiedene Optionen. Um alle Files in einem Directory zu sehen, verwenden Sie die Option -a.

```
% ls -a
. .. .login .cshrc
%
```

Die Einträge . und .. sind die besprochenen Verweise auf das aktuelle und das übergeordnete Directory. .login und .cshrc sind die Files, die von der C Shell beim Login ausgeführt werden.

Als Argumente kann für ls auch eine Liste von Namen mitgegeben werden, die entsprechenden Files und der Inhalt von angegebenen Directories werden dann angezeigt.

```
% ls /usr/bin /bin
...
% ls /etc/passwd
...
%
```

3.7 Abbrechen von Kommandos

Wenn Ihnen bei der Ausgabe des vorigen Beispiel das Warten etwas zu lange wird, können Sie durch gleichzeitiges Niederhalten der Tasten Control und der Taste C (*Control C,* Ctrl-C, ^C) das Kommando abbrechen.

In der C Shell und der Korn Shell können Sie Kommandos auch mit Ctrl-Z abbrechen. Diese können mit der in dieser Shell eingebauten *Job Control* dann wieder im Vordergrund oder im Hintergrund fortgesetzt werden.

Mit den Tasten Ctrl-S und Ctrl-Q kann die Ausgabe eines Kommandos vorübergehend abgebrochen werden. Wenn die Ausgabe eines Kommandos also einmal unvorhergesehen stehenbleibt, ohne dass der Shell Prompt erscheint, könnte eine versehentliche Eingabe von Ctrl-S die Ursache sein. Sie können dann mit Ctrl-Q die Ausgabe wieder fortsetzen.

3.7.1 Job Control (C Shell und Korn Shell)

Alle vom Benutzer gestarteten Kommandos werden als Prozesse in eine Liste eingefügt, die mit dem Befehl jobs angezeigt werden kann. Die mit Ctrl-Z unterbrochenen Prozesse können mit dem Befehl fg [%Jobnummer] in den Vordergrund oder mit dem Befehl bg [%Jobnummer] in den Hintergrund zurückgeholt werden. (mehr dazu im Kapitel “Prozesse„)

3.8 cat

Der Befehl cat (concatenate) dient dazu, den Inhalt von einem oder mehreren Files auf die Standard Ausgabe (Bildschirm) auszugeben.

```
% cat .cshrc
set path = (. /usr/ucb /usr/etc /bin /usr/bin /usr/local/bin)
set history=40
set notify
set savehist=40
set prompt="`whoami`%"
%
```

3.9 more

more ist ein sogenannter *Pager* und wird verwendet, um lange Files seitenweise auszugeben. Wenn also ein Kommando wie ls -l eine Ausgabe erzeugt, die nicht auf eine Seite passt, hat man zwei Möglichkeiten:

Verwendung von Ctrl-S und Ctrl-Q, um die Ausgabe händisch zu stoppen (mühsam und unsicher).

Verwendung von more.

```
% more LangeDatei
Zeile1
...
Zeile24
--More--(80%)
```

Mit der Leertaste kann man eine Seite weiterblättern, mit Return wird die nächste Zeile angezeigt. more hat einige Subkommandos, mit h kann man sich eine Hilfeseite anzeigen lassen. Nach der letzten Seite kehrt more wieder in die Shell zurück.

```
<Leertaste>
Zeile25
...
Zeile30
%
```

Der Vorteil von more ist, dass dieses Kommando, wenn keine Filenamen als Argumente angegeben werden, von der Standard Eingabe lesen wird. Dadurch kann nicht nur in Files, sondern auch durch die Ausgabe von Kommandos geblättert werden.

```
% ls -a /bin | more
.
..
STTY
[
ar
as
awk
cat
cc
chgrp
chmod
chpt
clri
cmp
cp
csh
date
dd
df
diff
dirname
--More--
```

3.10 cd

Mit dem Kommando `cd` (change directory) können sie ihr *Working Directory*, das ist der Ort im Filesystem, auf den alle *relativen* Pfadnamen bezogen werden, neu setzen. `cd` erwartet als Argument einen relativen oder absoluten Pfadnamen.

Ohne Argument wechselt `cd` immer in das eigene Home Directory.

```
% pwd
/export/home/kurs
% cd ..                         # Wechsel in übergeordnetes Directory
% pwd
/export/home
% cd ../..                      # Wechsel in übergeordnetes Directory
% pwd
/
% cd /usr                       # Absoluter Pfadname
% pwd
/usr
% cd                            # ohne Argument...
% pwd                           # gelangen Sie wieder...
/export/home/kurs               # in ihr Home Directory
%
```

3.11 man

Der wichtigste Befehl überhaupt - jeder UNIX Benutzer sollte mit ihm vertraut sein. Mit man erhält man Zugang zur Online System Dokumentation. Wenn Sie einmal vergessen haben sollten, wie ein bestimmtes Kommando anzuwenden ist oder zusätzliche Informationen benötigen, verwenden Sie

```
% man Befehl
```

und Sie erhalten Angaben über Syntax und Semantik des Befehls (auf Ihrem System).

mit

```
% man man
```

erhalten sie Informationen über den Befehl man selbst. Die *Manual Pages* sind üblicherweise in 8 Abschnitte unterteilt.

(1)	User Commands
(2)	System Calls
(3)	Subroutines
(4)	Devices
(5)	File Formats
(6)	Games
(7)	Miscellaneous
(8)	Sys. Administration

Ein Eintrag für ein Kommando beinhaltet seinen Namen mit einer Kurzbeschreibung, die Schreibweise (Syntax) und eine detaillierte Beschreibung aller möglichen Argumente und ihrer Bedeutung (Semantik). Weiters gibt es Hinweise auf eventuelle Fehler und Einschränkungen und Verweise auf weitere Informationen, die Sie online abrufen können oder auf Bücher, die detailliertere Angaben enthalten (z.B. vi - Editor, Programmiersprachen). Die Syntax der Befehle wird in einer normiertenForm angegeben, wobei folgende Symbole und Attribute eine besondere Bedeutung haben:

Blank	Leerzeichen oder Tabulatorzeichen trennt einzelne Worte
Wort	Das Wort muss in der angegeben Form eingegeben werden
[ ]	Die eingeschlossen Begriffe sind fakultativ
...	Wiederholung des vorangegangenen Begriffs
{ | }	Wahlweise Verwendung eines der durch | getrennten Begriffe
<u>**Wort**</u>	Ein unterstrichenes (oder anders hervorgehobenes) Wort bezeichnet ein Argument das vom Benutzer gewählt wird. Das Wort selbst gibt in der Regel an, welche Art von Argument erwartet wird.

Beispiel:

```
% man ls

ls(1)

  Name
       ls - list and generate statistics for files

  Synopsis
       ls [ options ] name ...

  Description
       For each directory argument ls lists
       ...

  Options
       -1   Displays one entry per line
            ...

  Restrictions
       The output device ...

  Files
       /etc/passwd
       ...
%
```

Übungen

Übung 1: Wie heisst ihr Working Directory. Finden Sie heraus wieviele Ebenen unter dem Root Directory sich Ihr Working Directory befindet und wechseln sie mittels eines relativen Pfadnamens in das Root Directory.

Wechseln Sie mittels relativen und absoluten Pfadnamen in verschiedene Directories, die Sie im Filebaum in der Abbildung finden und überprüfen Sie das Ergebnis.

Übung 2: Das ls Kommando versteht als Argumente auch die Namen von Files oder Directories. Wechseln Sie in ihr Home Directory

```
% cd
```

und probieren Sie:

```
% ls ..
```

und

```
% ls /export/home
```

um sich den Inhalt des Directories /export/home anzusehen.

Was ist der Unterschied zwischen .. und /export/home ?

Übung 3: Probieren sie auch den Befehl ls aus, wenn ihr Working Directory gerade auf /home gesetzt ist. Was sehen Sie ?

Probieren Sie auch

```
% ls /etc/passwd
% ls /etc
% ls etc
% ls passwd
```

Übung 4: Spielen Sie mit dem Befehl man um sich mit den Manual Pages vertraut zu machen. Sehen Sie sich zumindest die Einträge für ls, cd, echo, cat, more und man kurz an.

Freunden Sie sich mit dem Hilfesystem an !

Kapitel 4

Environment

Jeder Prozess - also auch die Shell - besitzt ein Environment (Umgebung). Bestimmte Werte (Environment Variable) werden vom System automatisch beim Login gesetzt. Andere Variable können vom Benutzer nachträglich gesetzt oder verändert werden. Welche Variable das sind und welche Werte sie besitzen hängt vom System und von der verwendeten Shell ab.

4.1 env

Das Kommando env (bei C Shell auch printenv) dient dazu, das aktuelle Environment anzuzeigen.

```
% env
HOME=/export/home/kurs
SHELL=/bin/csh
TERM=vt100
USER=kurs
PATH=/usr/ucb:/bin:/usr/bin:/usr/local/bin:
/export/home/kurs:.
%
```

HOME Home Directory des Benutzers

SHELL Die Login Shell

TERM Wird von verschiedenen Editoren und Programmen benötigt, um eine richtige Ausgabe auf unterschiedlichen Terminals zu erzeugen

USER Benutzername

PATH Diese Variable gibt an, in welchen Directories die Shell nach ausführbaren Programmen suchen soll, wenn der Name des Kommandos nicht durch einen Pfadnamen relativ zu einem Directory angegeben wurde

4.2 echo

Die Shell ersetzt ein $ gefolgt vom Namen einer Variable durch ihren Wert. Das Kommando echo dient dazu seine Argumente auf der Standard Ausgabe auszugeben. Das kann man z.B. benutzen, um sich den Wert einer Variable ausgeben zu lassen.

```
% echo $USER
kurs
% echo "Terminal: $TERM"
Terminal: vt100
%
```

4.3 .cshrc und .login

Beim Login liest die C Shell das File .cshrc und das File .login ein und führt deren Inhalt aus, bevor sie die Eingabe von Ihrem Bildschirm entgegennimmt. Mit dem File .login können Sie Aktionen definieren, die bei jedem Login automatisch ausgeführt werden sollen. Typischerweise das Setzen von Environment Variablen.

4.4 .profile (Bourne und Korn Shell)

analog .login und .cshrc

4.5 set und setenv

Die C Shell besitzt neben dem Environment auch eigene Shell Variable. Shell Variable werden mit dem Befehl set gesetzt und angezeigt, Environment Variable werden mit dem Befehl setenv gesetzt. Der Befehl unset löscht eine Shell Variable, unsetenv löscht eine Environment Variable.
Die C Shell liest immer zuerst ihre Shell Variablen, dann erst die globalen Environment Variablen. Das folgende Beispiel zeigt den Effekt, wenn derselbe Name im Environment und als Shell Variable verwendet wird.

```
% setenv VAR 1000                    # Setzen der Environment Variable VAR
% echo $VAR
1000
% set VAR=wert                       # Setzen der Shell Variable VAR
% echo $VAR
wert                                 # Shell überschreibt Environment
% unset VAR                          # Löschen der Shell Variable
% echo $VAR
1000                                 # alter Wert
% unsetenv VAR                       # Löschen der Environment Variable
% echo $VAR
VAR: Undefined variable.
%
```

Es hat sich eingebürgert für Environment Variable Grossschreibung und für C Shell Variable Kleinschreibung zu verwenden. Die C Shell Variablen können über den Befehl set auch angezeigt werden.

Bourne Shell Variablen können ebenfalls über den Befehl set angezeigt werden.

```
% set
argv    ()
cdpath  ()
cwd     /export/home/kurs
filec
history 64
home    /export/home/kurs
lcd     ()
notify
path    (/usr/ucb /bin /usr/bin /usr/local/bin
/export/home/kurs .)
prompt  %
savehist        40
shell   /bin/csh
status  0
term    vt100
user    kurs
%
```

4.6 Variablen exportieren in der Bourne- und Korn Shell

In der Bourne Shell werden Variable gesetzt indem man mittels des Zeichens = eine Zuweisung durchführt.

```
% sh                                    # Bourne Shell starten
$ VAR=wert
$ echo $VAR
wert
$
```

Um die Variable auch für andere Programme zu setzen muss sie noch explizit (in das Environment) exportiert werden. Sonst ist sie nur lokal zur derzeit laufenden Shell.

```
$ export VAR
$
```

Der Wert der Variablen VAR ist nun auch für alle von dieser Shell aus gestarteten Prozesse im Environment enthalten. Nicht aber in übergeordneten Shells. Dieser Befehl ist unbedingt notwendig, wenn Sie z.B. erreichen wollen, dass der Editor den Terminaltyp kennt.

```
$ echo $TERM
vt100
$ TERM=xterm
$ vi                                      # vi arbeitet mit TERM=vt100
$ export TERM
$ vi                                      # vi arbeitet mit TERM=xterm
```

Bemerkung: Um den vi-Editor zu verlassen muss man :q! eingeben.

4.7 csh und sh

Mit sh (csh) können Sie eine neue Shell starten (die Shell ist ein Programm, das wie jedes andere auch aufgerufen werden kann).

Diese Shell erbt von der übergeordneten Shell das Environment, die Standard Ein/Ausgabe und die Zugriffsrechte.

Sie können mit dieser Shell arbeiten wie mit einer Login Shell. Das kann dazu verwendet werden, wenn Sie z.B. von der C Shell in die Bourne Shell oder umgekehrt wechseln wollen.

```
% sh                              # Wechsel in andere Shell (Bourne Shell)
$ echo $SHELL                     # Anzeigen der Login Shell
/bin/csh
$ exit                            # Verlassen der BourneShell
%
```

Übungen

Übung 1: Welche Werte haben die Variablen TERM, USER, PATH ?

Was ist Ihre Login Shell ?

Übung 2: Setzen Sie eine neue Shell Variable und wechseln Sie die Shell (csh).

Zeigen Sie sich den Wert der Variable an und wechseln Sie wieder in ihre Login Shell zurück (exit). Setzen Sie eine Environment Variable und starten Sie wieder eine neue Shell. Zeigen Sie dieVariable dort an. Weisen Sie der EnvironmentVariable (setenv) nun einen neuen Wert zu. Verlassen Sie nun abermals die neue Shell und zeigen Sie den Wert der Variable in ihrer Login Shell an. Was können Sie daraus für setenv schliessen ?

Kapitel 5

Shell Syntax

Neben der Variablen Substitution führt die Shell auch noch weitere Textersetzungen und Expandierungen durch. Leider bestehen hier zwischen Bourne Shell und C Shell einige Unterschiede.

Folgende Definitionen sind nützlich, um die Interpretation von Kommandos durch die Shell zu verstehen:

Blank	Ein Leerzeichen oder ein Tabulatorzeichen
Bezeichner	Jede Folge von Buchstaben, Ziffern oder Unterstrich beginnend mit Buchstabe oder Unterstrich
Kommentar	Jedes Wort, das mit einem # beginnt. Der Kommentar erstreckt sich bis zum Ende der Zeile
Kommando	Eine Folge von Zeichen in der Shell Syntax. Die Shell führt die Aktionen direkt oder durch Aufruf der entsprechenden Werkzeuge aus
Metazeichen	Die Zeichen * ? [ ] $; & ( ) | < > ‘ newline space tab
Wort	Jede Folge von Zeichen, die kein Blank enthalten.
Einfaches Kommando	Eine Folge von durch Blank getrennten Wörtern. Das erste Wort bezeichnet den Befehl, der auszuführen ist. Die folgenden Wörter sind Argumente für das Kommando und werden von 1 aufsteigend numeriert. Das erste Wort wird dem Parameter mit der Nummer 0 zugewiesen. Ein Kommando kann einen Wert erhalten, der 0 ist, wenn das Kommando erfolgreich ausgeführt wurde, ungleich 0 sonst.
Spezielles Kommando	Ein Kommando, das direkt von der Shell ausgeführt wird.
Pipeline	Eine Folge von einem oder mehreren einfachen Kommandos getrennt durch das Pipeline Symbol |.
List	Eine Folge einer oder mehrerer Pipelines getrennt durch && || &.
Maskierung	Die Zeichen \ " ' werden verwendet, um die Interpretation von Metazeichen zu unterdrücken
Variable	Ein Name oder Parameter, dem ein Wert zugewiesen werden kann. Variable werden entweder durch die Shell oder durch den Benutzer gesetzt. Ein Wort, das mit einem $ beginnt, wird durch die Shell durch den Wert der Variable ersetzt

Subshell Eine Shell, die als SohnProzess der aufrufenden Shell ausgeführt wird.

5.1 Kommandointerpretation

Die Shell führt auf einem eingegeben Kommando eine Reihe von Substitutionen und Expandierungen durch, bevor tatsächlich Aktionen durchgeführt werden.

Aktion	**Zeichen**
Maskierung	`" " ' ' \`
Aufspaltung des Kommandos	`| && || & ( )`
Variablensubstitution	`$`
Kommando Substitution	`` ` ` ``
Eingabe/Ausgabe Umleitung	`< << > >>`
Filenamen Expandierung	`* []`
Kommentar	`#`

Die Reihenfolge dieser Substitutionen ist essentiell für die Bedeutung des Kommandos und daher in der angegebenen Reihenfolge genau definiert !

History und Alias Substitution werden nur in C Shell und Korn Shell durchgeführt. Sie werden im Anhang B und C vorgestellt.

5.1.1 Variablen Substitution

Alle Wörter, die mit einem $ beginnen, werden durch den Wert der Shell oder Environment Variable ersetzt, deren Name das entsprechende Wort ist.

```
% echo Mein Terminal ist ein $TERM
Mein Terminal ist ein vt100
%
```

5.1.2 Kommando Substitution

Die Ausgabe eines Kommandos kann als Argument für ein anderes Kommando weiter verwendet werden. Dazu wird das Subkommando in Backquotes (‘) eingeschlossen an die Stelle der Argumente geschrieben.

In den folgenden Beispielen, wird angenommen, dass sich in ihrem Working Directory drei Files mit den Namen dylan, madonna und prince befinden. Diese können mit dem Kommando touch erzeugt werden.

```
% set WD=`pwd`
% echo $WD
/export/home/kurs

% touch dylan prince madonna
% ls
dylan madonna prince
% set FILES="(`ls`)"
% echo $FILES
(dylan madonna prince)
```

Substitutionen innerhalb des Subkommandos werden lokal angewendet.

```
% echo `set TERM=neuerWert; echo $TERM` $TERM
neuerWert vt100
%
```

5.1.3 Maskierung

Metazeichen können mittels Maskierung vor der Interpretation durch die Shell versteckt werden.

```
% echo \$TERM
$TERM
```

Dabei ist zu beachten, dass innerhalb der einfachen Hochkommas (') alle Metazeichen maskiert werden. Innerhalb doppelter Anführungszeichen (") werden jedoch Kommando- und Variablensubstitution durchgeführt und die einfachen Hochkommas verlieren ihre Maskierungsfunktion.

```
% echo $TERM
vt100
% echo '$TERM'                # Zeichen $  wird zum ASCII Zeichen
$TERM
% echo "$TERM"
vt100
% echo "'$TERM'"              # Hochkommas verlieren Maskierungsfunktion
'vt100'
% echo '"$TERM"'              # Alle Metazeichen (" und $) werden zu ASCII Zeichen
"$TERM"
%
```

Die Interpretation doppelter Anführungszeichen (") ist in der Bourne Shell und in der C Shell verschieden. Dazu ein Beispiel:

In der C Shell wird die Variablen Substitution innerhalb von Anführungszeichen (") immer durchgeführt.

```
% echo "\$TERM"
\vt100
%
```

In der Bourne und Korn Shell behält der Backslash (\ ) seine Maskierungsfunktion

```
$ echo "\$TERM"
$TERM
$
```

5.1.4 Zuweisung von Variablen

Wörter der Form NAME=wert werden als Variablen Zuweisung behandelt. Die Shell Variable NAME erhält den Wert wert. In der C Shell muss dazu der Befehl set verwendet werden.

```
$ NAME=wert
$
```

```
% set NAME=wert
%
```

5.2 Eingabe und Ausgabe Umleitung

Mit den Metazeichen < und > können Standard Eingabe und Standard Ausgabe eines Kommandos durch die Shell auf Files umgelenkt werden. Dabei ist zu beachten, dass nur je eine Eingabe Umlenkung und eine Ausgabe Umlenkung je Kommando erlaubt ist.

```
% echo $TERM > datei                       # Ausgabe Umlenkung
% cat datei                                # Ausgabe von "datei"
vt100
%
```

Der Befehl cat ohne Filenamen erwartet seine Befehle von der Standard Eingabe. So kann auf einfache Weise mittels Ausgabe Umlenkung ein File eingegeben werden. Die Eingabe muss mit einem *End of File (EOF)* abgeschlossen werden. Das erreicht man durch Eingabe von Control D (^D) am Beginn einer Zeile.

```
% cat >lennon
imagine there is no country
and no religion too
^D
% cat lennon
imagine there is no country
and no religion too
%
```

Mit dem Metazeichen >> kann die Standard Ausgabe an ein File angehängt werden. Das File wird in diesem Fall nicht überschrieben, sondern erweitert.

```
% echo $TERM >> lennon
% cat lennon
imagine there is no country
and no religion too
vt100
```

5.3 Filenamen Expandierung

Die Shell ersetzt Wörter, die die Metazeichen `*` und `[ ]` - csh auch `{}` - enthalten, durch eine Liste von Filenamen. Dabei wird folgendes *Pattern Matching* (Mustererkennung) angewendet.

*	Das Muster * wird durch einen beliebigen Substring erfüllt. Ein * allein passt auf alle Namen im entsprechenden Directory ausgenommen derer die mit einem . beginnen.
?	Ein einzelnes Zeichen
[Zeichenfolge]	Erkennt eines der in Zeichenfolge enthaltenen Zeichen. Zeichenfolgen der Form Zeichen1-Zeichen2 werden als Zeichenbereiche interpretiert.
Anderes Zeichen	Erkennt das Zeichen selbst
{ss1, ss2, ...}	Erkennt einen der angegebenen Teilstrings (nur csh !)

```
% echo *
dylan madonna prince
% echo d*
dylan
% echo ?????
dylan
% echo [Dd]ylan
dylan
% echo [a-m]*
dylan madonna
% echo .*
. ..
```

Da viele Befehle die Bearbeitung mehrerer Files gleichzeitig unterstützen, ist dieser Expandierungsmechanismus ein mächtiges Werkzeug, um komplexe Kommandos auf eine Liste von Files anzuwenden, ohne dass man alle Namen explizit anschreiben muss.

So löscht zum Beispiel der Befehl

```
% rm a*
%
```

alle Files im Working Directory, die mit einem a beginnen.

Achtung: Die C Shell führt auch für die Eingabe- und Ausgabe Umlenkung eine Filenamen Expandierung durch, nicht die Bourne Shell.

```
% sh

$ cat <len*
len*: cannot open
$ cat len*
imagine there is no country
and no religion too
$
```

aber in der C Shell geht's auch so:

```
% cat <len*
imagine there is no country
and no religion too
%
```

5.4 Pipelines

Das Metazeichen | *(Pipe)* dient dazu, die *Standard Ausgabe* eines Kommandos auf die *Standard Eingabe* des folgenden Kommandos umzulenken.

Eine *Pipeline* besteht nun aus einem einzelnen Kommando oder mehreren Kommandos, die durch das Zeichen | getrennt sind.

sh und csh erwarten ihre Kommandos von der Standard Eingabe. Sie könnten also die Ausgabe eines Kommandos als Befehl verwenden, indem Sie diese Ausgabe direkt mittels | auf eine neue Shell umlenken.

```
% echo ls | csh
dylan lennon prince madonna
%
```

more ist ein Programm, das dazu dient, seine Standard Eingabe seitenweise auf den Bildschirm zu kopieren. Mittels einer Pipeline kann man also die Standard Ausgabe eines beliebigen Kommandos Seite für Seite ausgeben.

```
% ls /bin | more
....
%
```

Ohne Pipe Mechanismus müssen Sie, falls Sie seitenweise auflisten möchten, zuerst ein File erstellen, die anschliessend mit dem more Kommando angeschaut werden kann.

```
% ls /bin > inhalt
% more inhalt
....
%
```

5.5 Komplexe Kommandos

Die Metazeichen & && ; werden verwendet, um mehrere Kommandos zu verknüpfen. Diese Verknüpfung wird auch *list* genannt.

;	Trennt zwei Kommandos. Die beiden Kommandos werden sequentiell eines nach dem anderen ausgeführt.
&	Trennt zwei Kommandos. Die beiden Kommandos werden asynchron (gleichzeitig) ausgeführt. Ein & am Ende einer Zeile bewirkt, dass die Shell nicht auf das Ende der Ausführung wartet. Der Befehl wird im Hintergrund ausgeführt.
&&	Trennt zwei Kommandos. Das zweite Kommando wird nur ausgeführt, wenn das erste keinen Fehler liefert.

```
% cd;echo ok
ok
% cd && echo ok
ok
% cd songs && echo ok
songs: bad directory
%
```

5.6 Subshells

Mit den Metazeichen ( ) kann eine Subshell eröffnet werden. Das ist oft dann nützlich, wenn man die Ausgabe einer Reihe von Kommandos als zusammenhängende Ausgabe für eine *Ausgabe Umlenkung oder Pipeline* verwenden will.

```
% (echo r.e.m.; echo patti) > singers
% cat singers
r.e.m.
patti
%
```

Im Gegensatz zu

```
% echo stones; echo beatles > singers
stones
% cat singers
beatles
%
```

Man kann eine Subshell auch verwenden, um temporär das Workingdirectory für ein Kommando zu setzen:

```
% (cd $HOME/..; pwd)
/export/home
% pwd
/export/home/kurs
%
```

5.7 Mehrzeilige Eingaben

Die Bourne Shell erkennt bei Abschluss einer Zeile, wenn die Eingabe eines Paars von Metazeichen (wie z.B: ( ) ") nicht vollständig erfolgte und antwortet darauf mit einem zweiten Prompt > , damit die Eingabe abgeschlossen werden kann. Das erfolgt solange, bis entweder ein Fehler auftritt oder die Eingabe komplett ist.

```
$ echo "time
> time
> sign o the time"
time
time
sign o the time
$
```

Die C Shell erlaubt das nicht. Hier muss man die neue Zeile mit dem Metazeichen \ maskieren:

```
% echo "time\
time\
sign o the time"
time
time
sign o the time
% echo "troubles
Unmatched ".
%
```

Übungen

Übung 1: Erzeugen Sie ein ein- oder mehrzeilige File mittels echo und Ausgabe Umlenkung.

Übung 2: Erzeugen Sie ein File, die die Namen aller Files und Directories im Directory /usr/bin enthält.

Geben Sie dieses File mittels cat aus. Was fällt Ihnen auf ?

Übung 3: Listen Sie alle Files im Directory /etc, die mit einem r beginnen und einem e enden.

Übung 4: Erzeugen Sie ein File mit dem (gültigen) Namen "a*. Überprüfen Sie mit ls, ob Ihnen das auch gelungen ist, geben Sie den Inhalt des Files aus und löschen Sie sie wieder.

Übung 5: Speichern sie den String * in der Variable FILES.

```
% set FILES="*"
```

Benutzen Sie die Variable in Verbindung mit echo, um sich die Files ihres Home Directories anzuzeigen. Erzeugen Sie nun mittels touch ein neues File und geben Sie wiederum die Filenamen mittels der Variable aus.

Welches Ergebnis erhalten Sie ?

Übung 6: Listen Sie alle Files im Directory /etc unter Verwendung der Kommandos more seitenweise auf.

Übung 7: Listen Sie alle Files im Directory /etc auf, die mit einem u beginnen und als zweiten Buchstaben ein n oder ein a haben.

Kapitel 6

Filesystem

6.1 Files

Ein File ist eine sequentielle Folge von Zeichen. UNIX kennt keine weitere Strukturierung wie Records oder Blöcke. Blöcke unter UNIX dienen nur dazu, das File auf dem Speichermedium zu organisieren und sind für den Benutzer transparent.

Ein File wird im Normalfall sequentiell gelesen und geschrieben. So kann z.B. jedes beliebige File mittels Eingabe Umlenkung als Standard Eingabe für einen Befehl verwendet werden, oder von der Standard Ausgabe erzeugt werden. Der Erfolg hängt natürlich vom Inhalt des Files ab.

6.2 Filesysteme

Files sind in heutigen UNIX Systemen aus verschiedenen Gründen (Sicherheitsüberlegungen, Plattenkapazitäten) nicht auf einer einzigen Festplatte gespeichert. Meistens werden eine oder mehrere Festplatten weiter in sogenannte Partitions unterteilt. Jede solche Partition kann nun ein Filesystem enthalten.

Achtung: Filesystem bezeichnet im weiteren eine strukturierte Information auf einem Speichermedium (Festplatten Partition, CD-ROM oder Diskette). Im Unterschied zum Filesystem als die Summe aller Programme und Routinen, die den Zugriff auf Files verwalten.

Jedes Filesystem besteht aus bestimmten Strukturinformationen (Superblöcke, Boot Blöcke, I-Node Tabelle) und den eigentlichen Datenblöcken, die die Files enthalten.

6.3 I-Nodes

Zusätzliche Information über ein File ist in den sogenannten I-Nodes (Informationsknoten) gespeichert. Diese Information beinhaltet:

- Typ
- Anzahl Verweise
- Zugriffsrechte
- Eigentümer

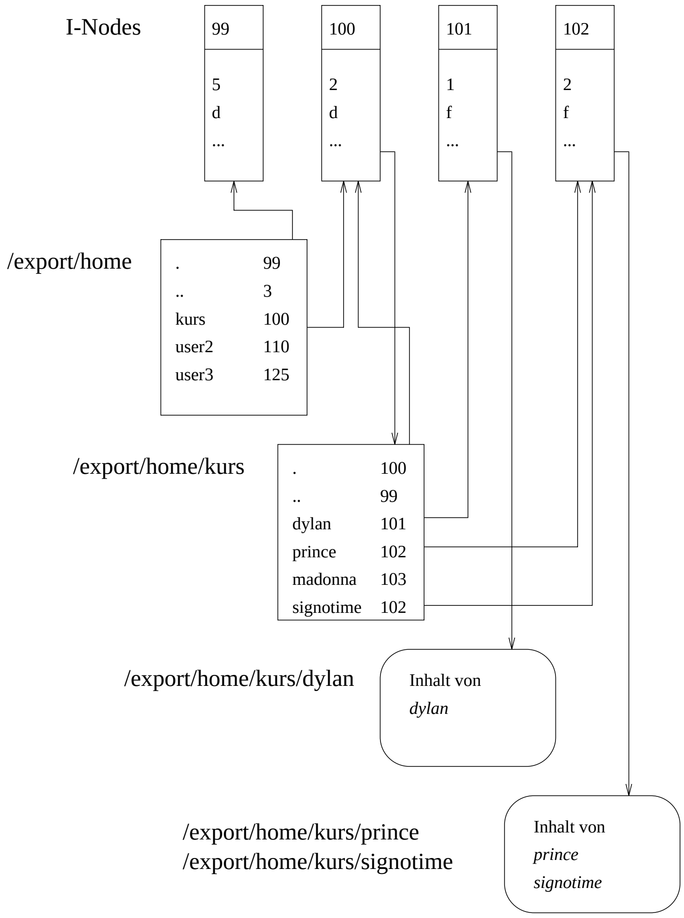
- Zugriffsdatum
- Änderungsdatum
- Verweise auf Datenblöcke

Jedes Filesystem hat eine eigene Liste von I-Nodes, die von eins beginnend durchnumeriert sind.

6.4 Directories

Directories sind spezielle Files, die eine Liste der Namen der Files und ihre I-Node Nummer enthalten.

Die folgende Abbildung soll veranschaulichen, wie Directories, I-Nodes und Files zusammenhängen.



Diese Aufteilung ermöglicht einerseits die völlige Trennung von Information über das File von ihrem Inhalt, andererseits ist es möglich, mehrere Namen für ein und dieselbes File mittels Links zu vergeben (z.B: *prince* und *signotime*).

UNIX kennt eine grosse Anzahl von Befehle zur Manipulation von Files. Die wichtigsten werden in diesem Kapitel vorgestellt.

6.5 mkdir

Directories werden nicht wie normale Files behandelt und müssen mit einem speziellen Befehl erzeugt werden.

Um ein Directory songs in ihrem Home Directory zu erstellen geben Sie folgenden Befehl ein:

```
% mkdir songs
% ls
dylan lennon madonna prince songs
%
```

Um sich den Typ eines File anzuzeigen kennt ls die Option -F. Die Option -i dient dazu, zusätzlich die I-Node Nummer anzuzeigen.

```
% ls -iF
101 dylan    103 madonna         102 prince
140 songs/
%
```

Der / hinter songs zeigt an, dass es sich bei songs um ein Directory handelt. In dieses Directory können Sie nun mit cd wechseln und dort wie in ihrem Home Directory Files anlegen.

```
% cd songs
% pwd
/export/home/kurs/songs
% touch jagger
% ls
jagger
% cd ..
% pwd
/export/home/kurs
% ls songs
jagger
%
```

6.6 cp

Copy kennt mehrere Formate. Das erste dient dazu, den Inhalt eines File in ein anderes zu kopieren, Wenn die Files noch nicht existiert, wird sie neu erstellt, andernfalls wird sie überschrieben.

cp [-i] File1 File2

```
% cp prince purple_rain
%
```

Mit der Option -i kann man sich davor schützen, Files versehentlich zu überschreiben:

```
% cp -i prince purple_rain
overwrite purple_rain: no
%
```

Das zweite Format dient dazu, mehrere Files in ein Zieldirectory zu kopieren.

cp [-i] [-r] File ... Directory

```
% cp dylan prince songs
% ls songs
dylan jagger prince
%
```

oder mehrere Directories in ein Zieldirectory zu kopieren.

cp [-i] -r Directory ... Directory

```
% cp -r songs groups
% ls -F
dylan madonna prince songs/ groups/
% ls groups
dylan jagger prince
%
```

6.7 mv

Mit move kann man Files umbenennen.

mv [-i] [-f] [-] File1 File2

```
% ls -i madonna
103 madonna
% mv madonna material_girl
% ls -i material_girl
103 material_girl
% mv madonna material_girl
mv: cannot access madonna
%
```

Move kann auch verwendet werden, um Files zwischen verschiedenen Directories zu verschieben.

mv [-i] [-f] [-] File ... Directory

```
% ls
dylan material_girl prince songs groups
% mv -i material_girl prince
remove prince? no
% mv material_girl songs
% ls
dylan prince songs groups
% ls songs
dylan jagger material_girl prince
%
```

Befinden sich altes und neues Directory im selben Filesystem, so wird nur der Name verändert, I-Node und Fileinhalt bleiben stehen, wo sie sind.

6.8 rm

Mit rm kann man Directory Einträge löschen. rm dekrementiert den Zugriffszähler im I-Node. Das File wird erst dann wirklich gelöscht, wenn auch der letzte Verweis auf den zugehörigen I-Node gelöscht wird.

rm [-f] [-r] [-i] [-] Name ...

```
% rm -i prince
rm: remove prince? yes
% ls
dylan songs groups
% rm *
rm: songs directory
rm: groups directory
% ls
songs groups
%
```

Directories können mit rm -r gelöscht werden. Da rm aber standardmässig einfach alles löscht ohne nachzufragen sollte man auch die Option -i verwenden, wenn man nicht 1000% sicher ist.

```
% rm -ri groups
rm: remove groups/dylan? yes
rm: remove groups/jagger? yes
rm: remove groups? yes
% ls
songs
%
```

6.9 rmdir

rmdir ist eine andere Möglichkeit ein Directory zu löschen. Dazu muss es allerdings zuerst leer sein. Leer bedeutet, keine Einträge ausser . und .. .

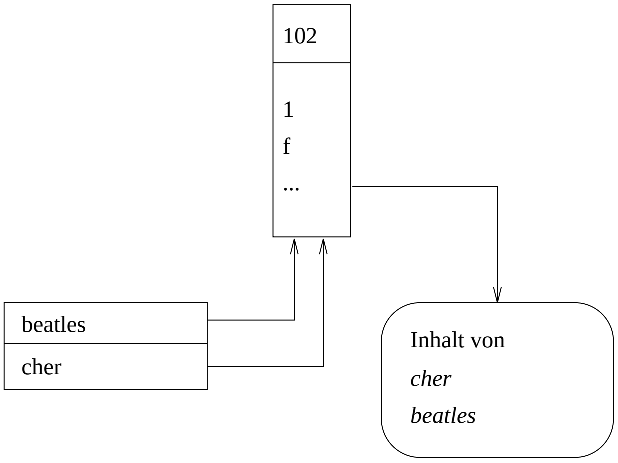
```
% rmdir songs
rmdir: songs: Directory not empty
% rm songs/*
% ls -a songs
. ..
% rmdir songs
%
```

6.10 ln

Mit Hilfe des ln Kommandos können Sie einem File mehrere Namen zuweisen. Das erste Format dupliziert einen Namen in das Working Directory bzw. erstellt einen Eintrag mit dem zweiten Namen, der auf denselben I-Node verweist.

ln [-s] name1 [name2]

```
% touch santana
% mkdir dir
% cd dir
% ln ../santana
% ls -i santana ../santana
102 ../santana   102 santana
% mv santana beatles
% ls -i beatles
102 beatles
% ln beatles cher
% ls -i
% 102 beatles   102 cher
%
```



Das zweite Format dupliziert alle angegebenen Namen in ein Directory.

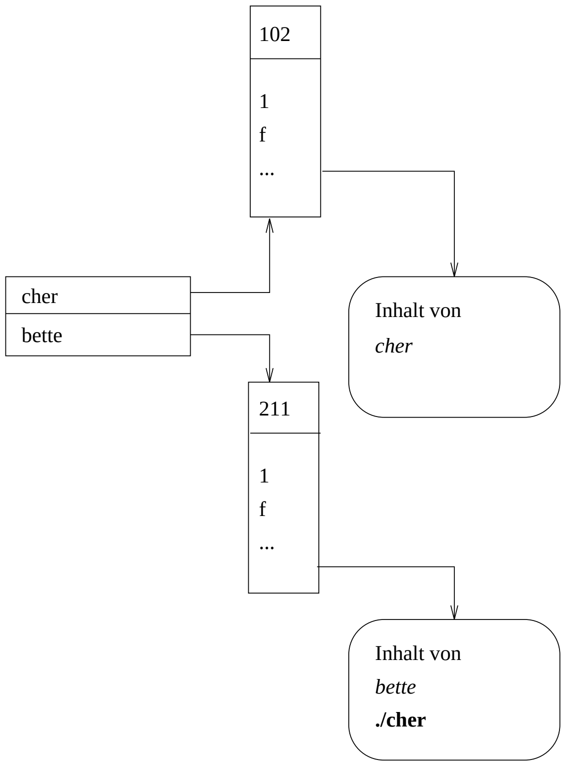
ln [-s] name ... directory

```
% ln * ..
% rm *
% cd ..
% ls -i
102 beatles   102 cher
%
```

Links, die einen direkten Verweis auf den I-Node enthalten, werden als Hard Links bezeichnet. Hard Links sind nur innerhalb eines Filesystems möglich und können nicht für Directories erzeugt werden. Es gibt auch sogenannte Symbolic Links, die einen Verweis auf einen Filenamen erzeugen.

```
% ln -s cher bette
% ls -F
santana   beatles   cher  bette@
%
```

Das Zeichen `@` nach `bette` kennzeichnet einen solchen Symbolic Link. Symbolische Links werden verwendet, um auch Verweise auf Directories oder in andere Filesysteme erzeugen zu können.



Übungen

Übung 1: Erklären Sie den Zusammenhang zwischen `mv`, `rm`, `ln` und `cp`.

Lässt sich einer der Befehle mit Hilfe der anderen simulieren ? Wenn ja, welcher? Wenn nein warum ?

Experimentieren Sie mit den Befehlen, um ihre Theorie zu belegen. Verwenden Sie nur das einfache Format mit zwei Filenamen für ihre Versuche.

Übung 2: Warum wohl können zwischen verschiedenen Filesysteme keine Hard Links erzeugt werden ?

Erzeugen Sie einen symbolischen Link auf ein File in ihrem Home Directory und versuchen Sie, den Inhalt des File mittels Ausgabe Umlenkung zu verändern. Löschen Sie den symbolischen Link.

Erzeugen Sie einen symbolischen Link auf das File `/etc/passwd` und versuchen Sie den Inhalt mittels Ausgabe Umlenkung zu verändern. Löschen Sie den symbolischen Link wieder.

Versuchen Sie sich die Fehlermeldungen zu erklären.

Übung 3: Erstellen Sie einen Hard Link auf ein File und einen Symbolic Link auf dieselbes File. Erfassen sie mittels `cat` einige Zeilen Text, wobei Sie einen der Links als Name verwenden.

Löschen Sie die Originaldatei.

Geben Sie den Inhalt des File sowohl über den Hard Link als auch über den Symbolic Link aus.

Erklären Sie die Fehlermeldung.

Übung 4: Erzeugen Sie ein neues Directory namens `d1` in ihrem Home Directory und einige Files in diesem neuen Directory.

Kopieren Sie das Directory samt Inhalt auf ein neues Directory Namens `d2`.

Erzeugen Sie einen Link namens `d3` auf `d2`.

Zeigen Sie den Inhalt der Directories `d1`, `d2` und `d3` an.

Versuchen Sie den Inhalt von d3 inklusive d3 mittels `rm -r` zu löschen.

Was wurde gelöscht ?

Verwenden Sie für die gesamte Aufgabe keinen `cd`.

Kapitel 7

Zugriffsrechte

Um Daten vor unberechtigtem Zugriff zu schützen, kann man unter UNIX Zugriffsrechte für Files und Directories vergeben. Die Zugriffsrechte sind unterteilt in Rechte für *Eigentümer*, *Gruppe* und *Andere* für Lesen, Schreiben und Ausführen eines File. Die Information über Zugriffsrechte ist jedem File eindeutig zugeordnet und im I-Node des Files abgespeichert.

Jedes File ist genau einem Eigentümer und einer Gruppe zugeordnet. Jeder Benutzer ist einer Primary Group zugeordnet und kann zusätzlich noch zu mehreren anderen Gruppen *(Group Set)* gehören

> Ein Benutzer besitzt die Zugriffsrechte des Eigentümers, wenn seine Benutzer Identifikation der des Eigentümers des Files entspricht.
>
> Ist der *Benutzer* nicht der Eigentümer eines Files, so hat er die Zugriffsrechte der Gruppe, wenn eine der Gruppen Identifkationen seines Group Sets der Gruppe des Files entspricht.
>
> Trifft keine der obigen Bedingungen zu, so gehört der Benutzer zum Rest der Welt und die Zugriffsrechte für den Benutzerkreis *Andere* treffen zu.
>
> Der *Super User* (**root**) hat alle Rechte für Lesen und Schreiben auf alle Files. Zugriffsrechte für Ausführen sind gleichzeitig ein Kennzeichen, ob es sich um ein ausführbares File handelt und werden auch für den Super User geprüft.

Mit diesem Konzept lassen sich sehr detaillierte und individuelle Zugriffskontrollen realisieren.

7.1 /etc/passwd

Die Information über Benutzer Identifikation (*Userid*) und Gruppen Identifikation (*Groupid*) eines Benutzers sind in der *Passwortdatei* /etc/passwd hinterlegt.

```
% cat /etc/passwd
root:x:0:0:Super User:/:/bin/sh
daemon:x:1:1:Mr. Daemon:/etc:
bin:x:2:2:Mr. Binary:/bin:
sys:x:3:3:Mr. Kernel:/usr/sys:
kurs:x:101:100:Kurs Test:/home/kurs:/bin/csh
%
```

Das File enthält für jeden Benutzer eine Zeile, mit folgenden durch Doppelpunkt getrennten Informationen:

- Benutzername
- Verschlüsseltes Passwort (bei System V (Solaris 2.x) sind die Passwöerter in der Datei /etc/shadow, welche nur für den Superuser lesbar ist)
- Benutzer Identifikation
- Gruppen Identifikation (Primary Group)
- Benutzer Information
- Home Directory
- Ausführbares Programm (Shell)

Dieses File wird beim Anmelden vom Login Prozess gelesen und dient einerseits dazu, den Zugang mittels Passwort zu überprüfen. Hat sich der Benutzer erfolgreich angemeldet erhält er das angegebene Home Directory und das ausführbare Programm (meistens eine Shell, könnte aber auch z.B. ein CAD Paket sein) wird gestartet.

/etc/passwd wird auch von anderen Programmen (z.B. ls) verwendet, um den vollen Namen eines Benutzers zu finden.

7.2 /etc/group

Dieses File enthält für jede Gruppe

- Name der Gruppe
- Verschlüsseltes Gruppen Passwort
- Gruppen Identifikation
- Benutzerliste

Die Benutzerliste enthält die Namen der Benutzer, die zu dieser Gruppe gehören. Die *Primary Group* eines Benutzers steht schon in /etc/passwd und muss hier nicht mehr angegeben werden. Das *Group Set* eines Benutzers besteht somit aus seiner Primary Group und allen Gruppen in /etc/group, in denen er in der Benutzerliste aufscheint.

```
% cat /etc/group
system:*:0:
bin:*:2:root,bin
sys:*:3:root,bin,sys
usr:*:100:
%
```

Aus dieser Liste kann man z.B. ersehen, dass root zu den Gruppen system, bin und sys gehört, wobei system die primary group ist und im File /etc/passwd dem Benutzer root zugeordnet wurde.

7.3 ls -l

Mit ls -l erhält man eine Liste mit genaueren Informationen über ein File. Diese beinhalten, durch Leerzeichen getrennt folgende Information:

- Zugriffsrechte
- Anzahl der Verweise auf das File
- Benutzername
- Gruppenname (nicht alle Systeme)
- Grösse des File in Bytes
- Datum der letzten Änderung
- Filename

```
% ls -al
total 36
drwxr-xr-x 2 kurs   kurse    512  26 Jun 10:25 .
drwxr-xr-x 5 bin    bin      512  21 Jun 10:32 ..
-rwxr----- 1 kurs   kurse    243  21 Jun 10:32 .login
-rwxr----- 1 kurs   kurse    243  21 Jun 10:32 .cshrc
-rwxr-xr-x 1 kurs   kurse     10  21 Jun 16:41 santana
% ls -l /etc/passwd
-rw-rw-r-- 1 root   other    953  25 Jun 13:56 /etc/passwd
%
```

Die erste Spalte enthält zehn Zeichen mit Informationen über *Filetyp* und *Zugriffsrechte*. Die Zugriffsrechte sind als Schalter im I-Node abgespeichert und werden oft als Oktalzahlen dargestellt.

Typ	Eigentümer			Gruppe			Andere		
-	r	w	x	r	w	x	r	w	x

Das erste Zeichen bezeichnet den Filetyp

- Normales File

d Directory

l Symbolischer Link

Die nächsten neun Zeichen sind in Dreiergruppen mit Zugriffsrechten für *Eigentümer*, *Gruppe* und *Andere* aufgeteilt.

Für Files haben diese Zeichen folgende Bedeutung:

- Kein Zugriff

r Lesezugriff

w Schreibzugriff

x Ausführbares File

Für Directories haben diese Zeichen folgende Bedeutung:

- Kein Zugriff

r Lesezugriff (ls)

w Schreibzugriff (Files anlegen, löschen, umbenennen)

x Change Directory in dieses Directory erlaubt

7.4 chmod

Mit chmod kann man die Zugriffsrechte (Modus) von Files neu setzen.

chmod [-fR] mode File...

Der Modus kann auf zwei Arten repräsentiert werden. Als *Oktalzahl* oder symbolisch.

Die Oktalzahl erhält man, indem man die Zugriffsrechte für Benutzer, Gruppe und Andere in der Reihenfolge, wie sie von ls geliefert werden aufschreibt und für jede Berechtigung eine 1 und für jedes Zugriffsverbot eine 0 aufschreibt. Die so erhaltene Binärzahl kann in eine Oktalzahl umgewandelt werden und als Argument für chmod verwendet werden.

Um z.B. das File springsteen selbst lesen und schreiben zu können, aber den Zugriff für die Gruppe auf Lesen einzuschränken und für alle anderen zu verbieten, kann man wie folgt vorgehen:

Aufschreiben der *Zugriffsbits* und umwandeln in Oktalzahl

r	w	-	r	-	-	-	-	-
1	1	0	1	0	0	0	0	0
	6			4			0	

```
% chmod 640 springsteen
%
```

Der symbolische Modus hat folgendes Format:

[wer] op Zugriff [op Zugriff]

Für *wer* kann eine Kombination von *u* für User, *g* für Gruppe und *o* für alle anderen (others) oder *a* für *ugo* stehen.

Für *op* kann + für Hinzufügen, - für Wegnehmen, = für Setzen eines Rechts stehen.

Für *Zugriff* kann eine Kombination aus *r* für Lesen, *w* für Schreiben und *x* für Ausführen stehen.

Um für *datei* nun z.B. nachträglich Rechte für Ausführung zu erteilen, kann folgendes Kommando verwendet werden:

```
% chmod ug+x springsteen
% ls -l springsteen
-rwxr-x--- 1 kurs   usr  0 Jun 28 14:37 springsteen
%
```

7.5 umask

Das Kommando umask dient dazu, die sogenannte User Mask zu setzen. Diese Maske gibt an, welche Zugriffsbits für neu angelegte Files gesetzt werden sollen. Die Maske wird als Oktalzahl wie in chmod angegeben. Darin gesetzte Bits werden von den Bits in 777 abgezogen, bei gewöhnlichen Files ist jedoch das x Bit nicht gesetzt.

```
% umask
077
% touch lenny
% ls -l lenny
-rw-------  1 kurs        0 Jul 16 14:59 lenny
% umask 027
% touch jackson
% ls -l jackson
-rw-r-----  1 kurs        0 Jul 16 15:00 jackson
% mkdir rock-n-roll
% ls -ld rock-n-roll
drwxr-x---  2 kurs      512 Aug 23 12:08 rock-n-roll
%
```

Übungen

Übung 1: Finden Sie ihre Benutzer und Gruppen Identifikation heraus.

In welchen Gruppen befindet sich ihr Benutzername und wie heissen Sie?

Wer darf das File /etc/passwd lesen und wer darf auch schreiben ?

Können Sie ein cd nach /etc machen ? Benutzen sie nicht den Befehl cd, um das herauszufinden!

Übung 2: Was bedeuten die Zugriffsrechte der Files in ihrem Home Directory, und wie können Sie als Oktalzahlen für chmod verschlüsselt werden ?

Übung 3: Setzen sie die Zugriffsrechte für ihr Login File (.login) so, dass sie nur von Ihnen selbst gelesen, geschrieben und ausgeführt werden kann, von ihrer Gruppe gelesen und ausgeführt werden kann und von allen anderen weder gelesen, geschrieben noch ausgeführt werden kann.

Übung 4: Angenommen in ihrer Institution gibt es drei Projekte, die auf demselben UNIX System Informationen hinterlegt haben.

Benutzer sollen nur Zugriff auf die Files erhalten, die zu Projekten gehören, in denen sie gerade arbeiten. Es gibt Benutzer die nur an einem der Projekt arbeiten, aber auch welche, die in mehreren Projekten tätig sind.

Wie könnte eine solche Organisation in den Zugriffsrechten für die Projektdaten repräsentiert werden ?

Welche Files müssten dazu geändert werden und welche Benutzer sind berechtigt, das zu tun ?

Kapitel 8

vi-editor

Der Standard *Text Editor* unter UNIX ist vi. Der Vorteil von vi ist, dass er für alle UNIX Systeme identische Grundfunktionen besitzt. vi ist eine seitenorientierte Erweiterung von ex der seinerseits auf ed aufbaut. Die beiden letzteren sind zeilenorientierte Editoren. Der Umfang von vi ist sehr mächtig und beinhaltet neben den Standard Funktionen zum Einfügen, Verändern, Verschieben, Suchen oder Löschen von Text auch Möglichkeiten, Macros zu definieren und durch Optionen das Verhalten an Benutzerbedürfnisse anzupassen.

In diesem Abschnitt sollen nur die grundsätzlichen Funktionen des vi angeschaut werden.

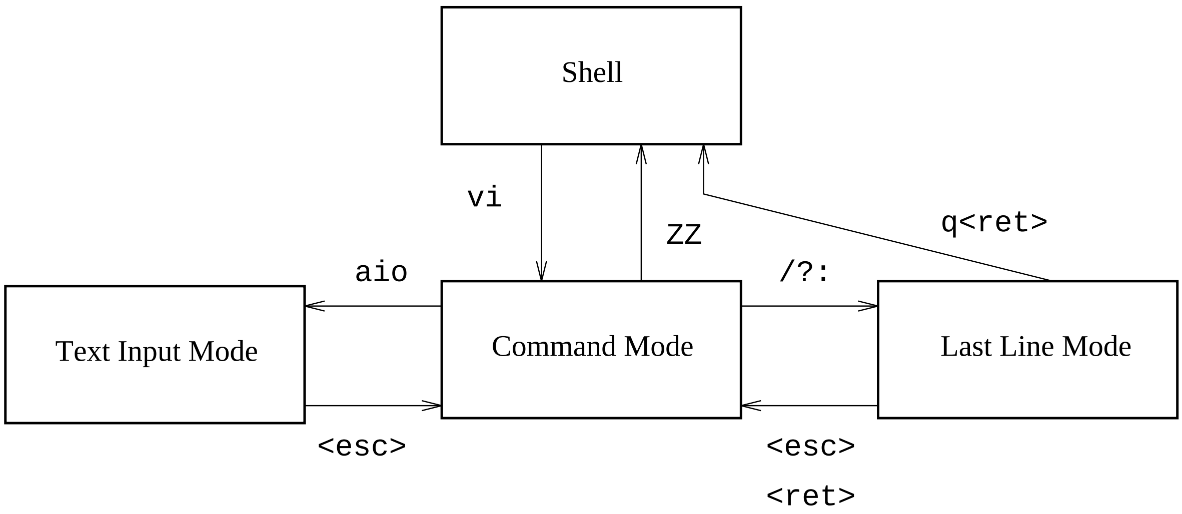
vi kennt drei Modi

Kommando Modus Dieser Modus ist gesetzt, nachdem der Editor aufgerufen wurde. In diesem Modus können Subkommandos eingegeben werden.

Eingabe Modus In diesem Modus kann Text eingegeben werden. Mit der Taste escape <esc> kommt man wieder in den Command Mode.

Letzte Zeile Modus In diesem Modus können ex Kommandos oder Suchstrings in der letzten Zeile eingegeben werden.

Diese Modi hängen auf die folgende Art zusammen:



8.1 Aufruf von vi

Das einfachste Format für den Aufruf ist:

vi Filename

Falls das File existiert, wird sie eingelesen und die erste Seite angezeigt, falls das File nicht existiert, wird ein leerer Schirm angezeigt. Leerzeilen, die nicht mehr zum File gehören, werden durch ˜ (Tilde) markiert. Die Cursorposition wird in den Beispielen durch eine Unterstreichung gekennzeichnet,.

```
% vi dylan
```

```
_
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"dylan" [New file]
```

8.2 Texteingabe

Mit folgenden Kommandos gelangt man nun in den Eingabe Modus, der es ermöglicht Text in das File einzufügen:

a	Text wird hinter der Cursorposition angefügt
i	Text wird vor der Cursorposition angefügt
o	Eine leere Zeile wird nach der Cursorposition geöffnet

Während man sich im Eingabe Modus befindet, können folgende Zeichen verwendet werden, um Text zu ändern:

Ctrl-H	
Backspace	Letztes Zeichen wird gelöscht
Ctrl-W	Letztes Wort wird gelöscht

Geben Sie nun folgenden Text, so wie er in der nächsten Zeile erscheint, ein:

aThere must be way out of here<cr>
Said the joker to the thief<esc>

Der Bildschirm sollte danach folgendes Aussehen haben:

```
There must be way out of here
Said the joker to the thief
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"dylan" [New file]
```

8.3 Verlassen von vi

Es gibt mehrere Möglichkeiten, vi wieder zu verlassen.

:q	Verlassen von vi, wenn keine Änderungen gemacht wurden, oder schon mit :w zurückgeschrieben wurden
:wq	Zurückschreiben der Änderungen und Verlassen von vi
ZZ	Zurückschreiben der Änderungen und Verlassen von vi
:q!	Verlassen von vi ohne Zurückschreiben der Änderungen

```
:wq
% cat dylan
There must be way out of here
Said the joker to the thief
%
```

Falls sich während des Editierens Probleme mit der Bildschirmanzeige ergeben, kann man mit Ctrl-L die Anzeige erneuern (*refresh*). Sollten die Probleme nicht verschwinden, so liegt das oft an einer falsch gesetzten TERM Variable im Environment. In diesem Fall sollte man vi verlassen die Variable neu setzen und einen entsprechenden Eintrag im .profile oder .login vornehmen.

8.4 Files verändern

Um den Text zu verändern rufen wir vi erneut auf.

```
% vi dylan
```

```
There must be way out of here
Said the joker to the thief
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"dylan" 2 lines, 63 characters
```

Der Cursor steht nun auf dem ersten Zeichen.

8.5 Cursor bewegen

Im Kommando Modus kann man den Cursor zeichenweise, zeilenweise oder wortweise bewegen. Wenn man unsicher ist, in welchem Modus man sich gerade befindet, kann man die *Escape* Taste drücken. Wenn man schon im Kommando Modus war, antwortet vi mit einem beep und bleibt im Kommando Modus.

h	Ein Zeichen nach links
Ctrl-H	
←	
j	Eine Zeile nach unten
Ctrl-J	
Ctrl-N	
↓	
k	Eine Zeile nach oben
Ctrl-P	
↑	
l	Ein Zeichen nach rechts
→	
ˆ	Zum ersten Zeichen ungleich Blank
0	Zum Anfang der Zeile
$	Zum Ende der Zeile
w	Zum Beginn des nächsten Worts
b	Zum Anfang des Worts oder vorigen Worts
Ctrl-U	Rollt den Schirm eine halbe Seite nach oben
Ctrl-D	Rollt den Schirm eine halbe Seite nach unten
Ctrl-F	Einen ganzen Schirm nach vorwärts
Ctrl-B	Einen ganzen Schirm nach rückwärts
Ctrl-E	Rollt den Schirm ein Zeile nach unten
Ctrl-Y	Rollt den Schirm ein Zeile nach oben
G	Geht zum Ende des Dokuments
nG	Eine ganze Zahl gefolgt von einem G geht zu der Zeile n.

Den meisten vi Kommandos kann wie bei G eine ganze Zahl vorangestellt werden. Diese Zahl gibt an, wie oft das nächste Kommando ausgeführt werden soll. Diese Zahlen erscheinen ebenso wie die Kommandos nicht auf dem Schirm!

3w

```
There must be way out of here
Said the joker to the thief
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"dylan" 2 lines, 63 characters
```

Geht 3 Wörter weiter.

isome <esc>
Fügt den Text „some “ vor der Cursorposition ein.

```
There must be some way out of here
Said the joker to the thief
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"dylan" 2 lines, 63 characters
```

Der Cursor bleibt nach dem Escape auf dem letzten Zeichen der vorangehenden Eingabe stehen (und nicht ein Zeichen danach)!

8.6 Text suchen

Die Kommandos / ? n N $ werden verwendet, um Text zu suchen.

/**Muster**	Geht zur nächsten Zeile, die das Muster enthält. Das Muster wird in der letzten Zeile eingegeben und muss mit einem <cr> abgeschlossen werden.
?**Muster**	Geht zur vorigen Zeile, die das Muster enthält
n	Wiederholt den letzten Suchbefehl
N	Wiederholt den letzten Suchbefehl in die entgegengesetzte Richtung

/he<cr>

```
There must be some way out of here
Said the joker to the thief
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"dylan" 2 lines, 63 characters
```

n

```
There must be some way out of here
Said the joker to the thief
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"dylan" 2 lines, 63 characters
```

8.7 Text löschen

Mit den Tasten **x d D** kann man Text löschen.

x	löscht ein Zeichen
dObjekt	löscht ein Objekt.
dd	löscht eine Zeile
D	löscht bis zum Ende der Zeile
J	Zusammenfügen zweier aufeinanderfolgender Zeilen

Das Objekt wird durch das entsprechende Cursorbewegungs Kommando angegeben. Beispiele:

dw	löscht das nächste Wort.
7dh	löscht die vorigen 7 Zeichen
dG	löscht alles bis zum Ende des Dokuments
1dG	löscht alles bis zum Beginn des Dokuments

dd

```
There must be some way out of here
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"dylan" 2 lines, 63 characters
```

6w2dw

```
There must be some way out_
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"dylan" 2 lines, 63 characters
```

8.8 Eingaben widerrufen

Ungewollte Textveränderungen können mit dem *Undo* (u) widerrufen werden. Dabei kann vi immer nur genau das letzte Kommando zurücknehmen. Ein Kommando kann mit . wiederholt werden.

u	Die letzte Eingabe oder Ersetzung wird widerrufen. Durch ein erneutes u wird sie wiederholt.
.	Das letzte Kommando wird wiederholt.
p	Der zuletzt gelöschte Text wird nach dem Cursor eingefügt (*put*).
P	Der zuletzt gelöschte Text wird vor dem Cursor eingefügt
"np	Der Text aus dem Buffer n wird nach der Cursorposition eingefügt. Gelöschte Texte werden in eigenen Speicherbereichen (*Buffers*) abgelegt. Diese sind von 1 bis 9 numeriert, wobei "1 immer den zuletzt gelöschten Text, "2 den davor gelöschten bezeichnet usw.

8.9 Text ersetzen

s	Ein oder mehrere Zeichen werden durch den eingegebenen Text ersetzt.
S	Die ganze Zeile wird durch den eingegebenen Text ersetzt
cObjekt	Ein Objekt wird durch den eingegeben Text ersetzt.
C	Der Text von der Cursorposition bis zum Ende der Zeile wird ersetzt

Das Ende des zu ersetzenden Textes wird von vi durch das Zeichen $ markiert.

bC

```
There must be some way out$
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"dylan" 2 lines, 63 characters
```

Der Cursor steht auf dem o von out, das $ Zeichen markiert das Ende des zu ersetzenden Textes (Ende des zweiten Worts).

in<esc>

```
There must be some way in
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"dylan" 2 lines, 63 characters
```

rZeichen	Ersetzt das Zeichen unter dem Cursor durch das angegebene Zeichen
RText	Überschreibt die Zeile ab der Cursorposition durch den angegebenen Text

0RNo reason to get excited
the thief he kindly spoke<esc>

```
No reason to get excited
the thief he kindly spoke
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"dylan" 2 lines, 63 characters
```

8.10 Shell Kommandos

Man kann auch Shell Kommandos aus vi heraus aufrufen, oder vi temporär verlassen und eine Shell eröffnen.

:!cmd	Führt das Shell Kommando cmd aus
:shell	Eröffnet eine neue Shell, die mit exit oder Ctrl-D wieder verlassen werden kann.

```
~
~
~
~
~
~
~
~
~
~
~
~
~
:!pwd
/export/home/kurs
[Hit return to continue]
```

```
~
~
~
~
~
~
~
~
~
~
~
~
~
:shell
% pwd
/export/home/kurs
% exit
```

8.11 Setzen von Optionen

Mit dem ex Kommando set können Optionen für vi gesetzt werden.

set	Zeigt gesetzte Optionen an
set all	Zeigt alle Werte von Optionen an
set opt	Setzt Option opt
set noopt	Setzt Option opt zurück
set opt=wert	Setzt Option opt auf wert

Beispiele für Optionen sind

autoindet	
ai	Der Cursor bleibt im Eingabe Modus in neuen Zeilen auf der Einrückung der vorigen Zeile. Mit Ctrl-D kommt man wieder auf die vorige Spalte.
number	
nu	Die Zeilennummern werden für alle Zeilen angezeigt.
ts=	Setzt den Abstand zwischen Tabulatorstopps. Default ist ts=8.
term=	Setzt den Terminaltyp. Default ist der Wert der Environment Variable TERM
sw=	Setzt den Abstand um den mit > eingerückt wird. Default ist sw=8.

Es gibt noch einige weitere Optionen, die nach Bedarf den Handbüchern entnommen werden können.

8.12 Tastenbelegung

Mit dem Kommando map kann man Tasten umbelegen. Das erste Argument für map ist die zu belegende Taste, der Rest der Zeile wird als Reihe von Kommandos interpretiert. Mit map! belegt man Tasten für den Eingabe Modus.

map ^X dd	Belegt die Taste Ctrl-X mit der Zeilenlöschfunktion
map ^Y {>}	Belegt die Taste Ctrl-Y mit dem Einrücken eines Absatzes

8.13 .exrc

vi liest beim Aufruf das File .exrc aus dem Working Directory oder dem Home Directory ein und führt die darin enthaltenen Kommandos aus. Das kann dazu verwendet werden, um Optionen und Tastenbelegungen permanent zu setzen.

```
% cat .exrc
set ts=4
set sw=4
set autoindent
map ^X dd
%
```

Übungen

Übung 1: Rufen Sie den Editor auf, um ein neues File zu editieren. Erfassen Sie einen Text, der länger als eine Seite ist. Beobachten Sie, wie der Text beim Eingeben über den Bildschirm rollt (scrolling). Positionieren Sie den Cursor auf verschiedene Zeilen mittels direkter Adressierung (G) und relativer Cursorbewegung. Rollen Sie den Text mittels Cursorbewegung (Pfeiltasten, j k) oder Control Tasten (Ctrl-B, Ctrl-U). Beobachten Sie, wie sich der Cursor verhält.

Probieren Sie alle Cursorbewegungsfunktionen für Zeichen, Wörter, Zeilen, Sätze, Absätze und Suchen aus und überlegen Sie sich, welche Sie wohl am häufigsten brauchen können. Verwenden Sie auch die Möglichkeit der Wiederholung mittels Zahlenpräfix.

Gewöhnen Sie sich auch an die Möglichkeit, die Tasten h j k l anstatt der Pfeiltasten zu verwenden.

Übung 2: Löschen Sie Zeichen, Wörter, Zeilen und Sätze.

Ersetzen Sie Zeichen, Wörter, Zeilen und Sätze durch neuen Text.

Versuchen Sie die Funktionen für Widerrufen und Wiederholen von Änderungen.

Kapitel 9

Prozesse und Signale

Jeder Art von Benutzertätigkeit ist ein Prozess zugeordnet.

9.1 ps

Prozesse sind durch eine Reihe von Zuständen und Parametern charakterisiert, die mittels ps angezeigt werden können.

```
% ps -f
  UID   PID  PPID  C    STIME TTY        TIME COMD
 kurs 22985 22924  8 11:53:41 pts/0      0:00 /usr/bin/ps -f
 kurs 22924 22922 80 11:27:40 pts/0      0:01 -csh
%
```

Die Ausgabe Felder haben hier folgende Bedeutung:

UID	Benutzer
PID	Prozess ID
PPID	Parent Prozess ID
C	Prozessor Auslastung
STIME	Startzeit des Prozesses
TTY	Terminal, an dem der Prozess läuft
TIME	Verbrauchte Zeit
COMD	Kommando, mit dem der Prozess aufgerufen wurde

ps kennt eine Reihe von Argumenten und Ausgabe Feldern (siehe man ps).

9.2 kill

Signale sind die einfachste Form der Inter Prozess Kommunikation unter UNIX. Ein Prozess wird vom Kernel durch ein Signal unterbrochen und es wird eine Routine aufgerufen, die für dieses Signal definert ist. Eine Reihe von Signalen haben vordefinierte Bedeutung. Hier eine Auswahl :

HUP	**1**	hangup
INT	**2**	interrupt
QUIT	**3**	quit
KILL	**9**	kill (cannot be caught or ignored)
TERM	**15**	software termination signal from kill
STOP	**17**	sendable stop signal not from tty
CONT	**19**	continue a stopped process

Mit kill kann man ein Signal an einen Prozess schicken.

kill [-Signal] ProzessID...

Signal ist eines aus der obigen Liste oder die entsprechende Signalnummer. Wird kein Signal angegeben, so sendet kill das Signal TERM (15) zur Beendigung des Prozesses. Das Signal KILL (9) kann vom Prozess nicht abgefangen werden und der Prozess wird bedingungslos abgebrochen.

```
% kill 2317              # Sendet TERM an Prozess mit PID=2317
% kill -9 4711           # Sendet KILL an Prozess mit PID=4711
%
```

Ein Benutzer kann nur an seine eigenen Prozesse Signale senden, der *Super User* an alle.

9.3 Job Control

Die C Shell und die Korn Shell kennen auch eine sog. Job Control. Damit können mit Ctrl-Z unterbrochene Kommandos verwaltet und im Vordergrund oder im Hintergrund fortgesetzt werden. Das Shell Kommando jobs dient dazu, die laufenden und unterbrochenen Jobs anzuzeigen. Mit fg %Jobnummer und bg %Jobnummer kann man Jobs fortsetzen. Mit kill können laufende und unterbrochene Jobs abgebrochen werden.

```
% sleep 99999
^Z
Stopped
% jobs
[1]  + Stopped              sleep 99999
% bg
[1]    sleep 99999 &
% jobs
[1]    Running              sleep 99999
% fg %1
sleep 99999
^C
%
```

Siehe man csh, man ksh.

Übungen

Übung 1: Starten Sie mit sleep 9999 & einen Prozess im Hintergrund und senden Sie an diesen Prozess verschiedene Signale aus der obigen Liste.

Übung 2: Sehen Sie sich die Manual Pages für ps und kill an.

Kapitel 10

Drucken

Drucker werden unter UNIX wie alle Geräte als normale Files angesprochen. Da alle Benutzer aufgrund der Zugriffsrechte theoretisch gleichzeitig auf die Drucker zugreifen können, ist es nicht sinnvoll, Daten direkt auf einen Drucker auszugeben.
Stattdessen werden *Druckaufträge (Print Jobs)* in sogenannte *Drucker Warteschlangen (Print Queues)* eingereiht, die von einem eigenen Drucker Prozess *(Queue Daemon)* in einer vordefinierten Reihenfolge abgearbeitet werden.
Der Benutzer hat Möglichkeiten, Druckaufträge in Warteschlangen einzureihen, diese wieder abzubrechen und den Status der Warteschlange abzufragen.
In heutigen verteilten Systemen kommt es oft vor, dass Drucker nicht lokal an jedem einzelnen Rechner, sondern an einem eigenen *Drucker Server* angeschlossen sind, der die Druckaufträge einer ganzen Netzgruppe bedienen kann. Daneben kann es auch lokale Drucker geben. Für den Benutzer ist diese Verteilung normalerweise transparent, er kann alle Warteschlangen auf die gleiche Art verwenden.

10.1 lpr

Mit lpr wird ein Druckauftrag aufgegeben. Das einfachste Format ist:

```
% lpr dylan
%
```

Das File wird auf den Standard Drucker des Systems ausgegeben. Wird kein File angegeben, so nimmt lpr die Standard Eingabe. Will man einen anderen Drucker verwenden, so kann man mit

```
% lpr -Plaser1 dylan
%
```

den Namen einer anderen Warteschlange (z.B. laser1) angeben. Wenn man diese Warteschlange permanent verwenden will, kann man im Environment die Variable PRINTER setzen. Am besten geschieht das im .login mit:

```
%setenv PRINTER laser1
%
```

10.2 lpq

Mit lpq können Sie den Status der Drucker Warteschlangen abfragen. Sie erhalten damit Information über

- Laufende Druckaufträge
- Wartende Druckaufträge
- Benutzer
- Auftragsnummer
- Filename
- Grösse der Druckdateien

Mit lpq ohne Argumente können Sie ihre Druckaufträge ansehen.

```
% lpq
laser1 is ready and printing
Rank   Owner   Job  Files             Total  Size
active kurs     23  /home/kurs/dylan  63 bytes
%
```

Für die Argumente zu lpq siehe man lpq

10.3 lprm

Mit lprm können Sie Druckaufträge unterbrechen. Dazu geben Sie die Auftragsnummer ein, die Sie mit dem Kommando lpq anzeigen können.

```
% lprm 23
%
```

Mit lprm können Sie nur Druckaufträge aus der Warteschlange entfernen, die Ihnen gehören.

10.4 vpp

Auf der ETH Zürich wird die Verteilung von Druckaufträgen mittels vpp (Verteiltes Printen und Plotten) gesteuert. Hierzu gibt es eigene vpp Stationen, die die Druckaufträge über das Netzwerk empfangen und lokal verwalten. Aufträge können nur auf den vpp Stationen unterbrochen werden. lpq und lprm haben hier keinen Zugriff. vpp kann von jedem Rechner (nicht nur UNIX) auf die vpp Drucker der ETH zugreifen

```
% vpp dylan
%
```

Für nähere Informationen siehe man vpp.

Übungen

Übung 1: Sehen Sie sich die Manual Pages für lp lpr pr lpq lprm vpp an. Drucken Sie die Manual Page für ein beliebiges Kommando aus.

Kapitel 11

Kommunikation

11.1 wall

Mit wall kann man eine *Botschaft (message)* an alle angemeldeten Benutzer senden.

```
% wall
ich gehe jetzt nach hause
^D

Broadcast message from kurs on tty1 ...

ich gehe jetzt nach hause

%
```

11.2 write

Mit write kann man seine Standard Eingabe auf den Bildschirm eines anderen Benutzers kopieren. Wenn ein Benutzer nur einmal angemeldet ist, genügt der Benutzername als Argument, ansonsten sollte man auch einen Bildschirmnamen mit angeben.

```
% write root console
Ich gehe heute doch etwas spaeter.
Squash auf Morgen verschoben.
^D
%
```

11.3 talk

Mit talk kann man auch über das Netzwerk kommunizieren. Der Bildschirm wird in einen Bereich für Senden und einen für Empfangen von Botschaften unterteilt. Im oberen Bereich erfasst man seine Botschaften im unteren kann man die des Partners lesen. So kann man über das Netz anrufen, falls das Telefon einmal ausgefallen sein sollte. Die Konversation wird aufgebaut indem man mit

```
% talk partner
```

den Gesprächspartner einmal anruft. Dieser wird auf das Gespräch aufmerksam gemacht und kann mit

```
% talk anrufer
```

die Verbindung herstellen. Dabei müssen die Argumente partner und anrufer das Format

Benutzername[@Maschinenname] [Schirm]

haben. Der Text @Maschinenname wird verwendet, wenn der Gesprächspartner auf einem anderen Rechner angemeldet ist und muss mit dem *Hostname* dieses Rechners identisch sein. Der Bildschirm sollte dann ungefähr folgendes Aussehen haben:

[**Connection established**]

Mit Ctrl-C können Sie die Konversation wieder abbrechen.

11.4 mesg

Mit mesg kann festgelegt werden, ob man bereit ist, Botschaften mittels write,wall oder talk entgegenzunehmen.

mesg [y][n]

```
% mesg y                          # Ok, Meldungen erlaubt
% mesg n                          # keine Meldungen erlaubt
% mesg                            # Status ausgeben
%
```

11.5 mail

Wenn der Empfänger der Botschaft gerade nicht angemeldet ist und mit write oder talk nicht erreicht werden kann, können Sie ihm immer noch einen Brief schicken. Dazu dient mail. Mit mail können Sie sowohl auf dem eigenen System als auch an Benutzer, die ihre Post auf anderen Systemen lesen, Nachrichten senden.

Senden

Um einen Brief an einen Benutzer zu senden rufen Sie mail mit dem Namen des Benutzers auf. Wenn der Benutzer auf einem anderen System arbeitet, so muss der Name die Form user@hostname besitzen. mail erwartet den Brief dann von der Standard Eingabe.

```
% mail root
Subject: Kontaktaufnahme
Hallo da bin ich
.
Cc: kurs
%
```

Mit Subject: kann man eine Überschrift angeben, die in den Kopfzeilen in der Übersicht über empfangene Post angezeigt wird. Mit Cc: (Carbon Copy) kann man den Brief an mehrere Benutzer gleichzeitig versenden. Mit Punkt oder Ctrl-D wird der Brief abgeschlossen. Man kann auch einen z.B. mit vi vorbereiteten Brief direkt als Standard Eingabe für mail verwenden:

```
% vi Brief
% mail user -s "Wie geht es Dir ?" <Brief
%
```

Lesen

Um angekommene Post zu lesen, ruft man mail ohne Argumente auf.

```
% mail
Mail version 2.18 5/19/83.  Type ? for help.
"/usr/spool/mail/kurs": 1 message 1 new
>N  1 kurs@host Thu Jul  4 16:20  10/247
"Kontaktaufnahme"
&
```

Folgende mail Kommandos sind unter anderem nun möglich

Return	Nächsten Brief anzeigen
h	Anzeigen der Kopfzeilen (Headers)
n	Anzeigen des Briefes mit der Nummer n
d message	Die in der Liste message angeführten Briefe werden gelöscht.
!cmd	Shell Kommando
m user	Sende Post an user
x	mail verlassen
q	mail verlassen, nicht gelöschte Post wird in $HOME/mbox gespeichert

```
& <cr>
Message  1:
From kurs Thu Jul  4 16:20:42 1991
Return-Path: <kurs@host>
To: kurs
Subject: Kontaktaufnahme
Date: Thu, 4 Jul 1991 16:20:39 +0200
From: Kursteilnehmer <kurs@host>
Sender: kurs@host
Status: R

Hallo da bin ich

& q
Saved 1 message in mbox
%
```

Mit mail -f [File] kann man sich die Post in File ansehen. Wird File nicht angegeben, so wird $HOME/mbox verwendet.

Übungen

Übung 1: Sehen Sie sich die Manual Pages für talk write mail an.

Suchen Sie sich einen Partner und schnurren Sie ein wenig mit ihm.

Übung 2: Schicken Sie sich selbst einen Brief.

Sehen Sie sich ihren Brief an und speichern Sie ihn in ihrem persönlichen Briefkasten (mbox). Geben Sie mbox einen neuen Namen und rufen Sie mail mit diesem File auf.

Kapitel 12

Shell Scripts

Um immer wiederkehrende Folgen von Kommandos effizient aufzurufen, ist es möglich diese Kommando Sequenzen in eigenen Files zu hinterlegen. Solche Files nennt man *Shell Scripts*. Die Shell liest und interpretiert diese Files so, als ob die Kommandos vom Bildschirm eingegeben würden.

Die Kommandos des Shell Scripts werden sequentiell in der Reihenfolge, in der sie in dem File stehen, abgearbeitet. Die Shell kennt spezielle Kommandos wie if...then...else...fi oder for...in...do...done, um Verzweigungen oder Schleifen zu programmieren. Diese Möglichkeiten werden an einer anderen Stelle diskutiert.

Shell Scripts können wie System Kommandos verwendet werden. Beim Aufruf eines Kommandos prüft die Shell, ob es sich um ein *ausführbares File* (*Execute Bit* gesetzt) handelt. Ist dieses File eine *Binärdatei*, so wird sie geladen und ausgeführt, ist es ein Shell Script, so wird ihr Inhalt von der Shell selbst interpretiert.

12.1 Parameter

In Shell Scripts haben einige Zeichen nach $ eine besondere Bedeutung:

$0	Name des Kommandos
$n	Argument n des Kommandos
$*	Alle Argumente $1 bis $n
$$	Prozessnummer des Shell Prozesses
$?	exit Status des letzten Kommandos
$#	Anzahl der Argumente
${wort}	
$wort	Wert der Variablen wort

Diese Parameter können in einem Script als normale Variablen verwendet werden. Folgendes Beispiel soll dies veranschaulichen:

```
% ptest A B C
```

$0	ptest
$1	A
$2	B
$3	C
$*	A B C
$#	3

oder

```
% cat ECHO
echo $0: PID=$$ ARG=$*
echo STATUS=$? \$1=$1
% chmod +x ECHO
% ECHO Das ist ein Test
ECHO: PID=4711 ARG=Das ist ein Test
STATUS=0 $1=Das
%
```

Das Shell Script wird per Default von der Bourne Shell abgearbeitet. Will man aber die C Shell verwenden, so muss man dies explizit wie folgt festlegen:

```
% cat ECHO
#!/bin/csh
echo $0: PID=$$ ARG=$*
echo STATUS=$? \$1=$1
%
```

12.2 Shell Script "ll"

Sie wollen beispielsweise ls immer mit den Argumenten -alF aufrufen und sind es müde alle Argumente inklusive der Pipeline mit more einzugeben. Sie können für dieses Kommando ein Shell Script erstellen.

```
% cat >ll
/bin/ls -alF $* | more
^D
% chmod +x ll
%
```

Mit ll können Sie nun eine beliebige Liste von Files oder Directories seitenweise ausgeben.

WICHTIG:

Soll ein Shell Script ablauffähig sein, muss das Execute Flag gesetzt sein. Sie erhalten sonst die Fehlermeldung "Permission denied". Das Execute Flag wird mit dem Kommando chmod gesetzt (Kapitel "Zugriffsrechte").

Für einfache Kommandofolgen kann in der C Shell und der Korn Shell, anstelle eines Shell Scripts, der Alias Mechanismus eingesetzt werden. Dieser ist im Anhang B und C beschrieben.

Übungen

Übung 1: Editieren Sie Ihr Login File so, dass sie nach dem Login eine Begrüssungsmeldung ihrer Wahl ausgibt und anzeigt, wer momentan am System angemeldet ist

Übung 2: Erstellen Sie mit vi ein Shell Script namens stat, das folgende Ausgabe erzeugt.

```
% stat
Name: <Ihr Name>
Processes:
<Ausgabe von ps>
Working Directory: <Ihr Working Directory>
Es ist nun: <Datum und Zeit>
%
```

Übung 3: Sehen Sie sich die Manual Pages für csh an.

Anhang A

UNIX

A.1 UNIX Systeme

Der Begriff UNIX steht heute für eine ganze Familie von Betriebssystemen, die auf AT&T UNIX zurückgehen. UNIX umfasst sowohl den Kern des Betriebssystems als auch eine Reihe von Kommandos, die auf diesem Kern aufsetzen. Der Name UNIX selbst ist ein Markenzeichen von AT&T. Zur Zeit existieren zwei Hauptlinien von UNIX Systemen, nämlich UNIX System V und 4.3BSD der University of California Berkeley, die beide auf UNIX Version 7 zurückgehen. Versionen anderer Hersteller sind unter anderen Namen bekannt.

UNIX System V	AT&T
SunOS	Sun Microsystems
AIX	IBM
A/UX	Apple Computer
HP-UX	Hewlett Packard
Sinix	Siemens
XENIX	Microsoft
ULTRIX	DEC
ConvexOS	Convex
SCO UNIX	The Santa Cruz Operation

Derzeit gibt es mehrere Vereinigungen zur Standardisierung von UNIX, die alle im Prinzip auf System V.3 und 4.3BSD basieren:

Unix International

Zusammenschluss von Sun und AT&T um ein gemeinsames UNIX System V Release 4 herauszugeben, das die Richtungen System V.3 und 4.3BSD vereinigt.

Open Software Foundation (OSF)

Zusammenschluss anderer Firmen zur Standardisierung von UNIX und UNIX Software zur Entwicklung hardwareunabhängiger Programme zur Erlangung eines Industriestandards.

Gründungsmitglieder sind unter anderen: Apollo, Hewlett Packard, IBM, DEC, Siemens und Philips.

X/Open, POSIX Lockerer Zusammenschluss verschiedener Firmen zur Definition von Richtlinien für eine Standardisierung von UNIX artigen Betriebssystemen. Ziel ist auch hier möglichst hohe Hardwareunabhängigkeit von UNIX Software.

A.2 Geschichte

1968 Ken Thompson und Dennis Ritchie entwickeln für AT&T Bell Laboratories ein neues Filesystem. Eine Simulation wird auf einem GECOS System implementiert. Erstes Single User UNIX auf PDP 7 mit Filesystem, Assembler und Kommandointerpreter.

1970 Kernighan prägt den Begriff UNIX. Erste Werkzeuge für Filebearbeitung. Gleichzeitiges Arbeiten von zwei Benutzern möglich. Portierung auf PDP 11/20.

1971 Version 1. Enthält alle UNIX Ideen ausser Pipelines

1972 Version 2. Einführung von Pipelines. Ken Thompson entwickelt Sprache "B".

1973 UNIX wird in "C" neu implementiert

1975 Version 6. Erste ausserhalb Bell Laboratories verfügbare Version. Abgabe gegen nominelle Gebühr an Universitäten. Entwicklung der Bourne Shell.

1977 Entwicklung einer portablen UNIX Version. Erweiterungen von "C" um Unions und Datentypen.

1979 UNIX Version 7 verfügbar. Später UNIX System III.

1983 UNIX System V wird von AT&T herausgegeben.

1984 UNIX 4.2BSD (Berkeley Software Distribution)

1986 UNIX 4.3BSD (Berkeley Software Distribution)

A.3 Highlights

- Echtes Multiuser und Multiprocessing Betriebssystem.
- Übersichtliches hierarchisches Filesystem mit verschiedenen Zugriffsrechten.
- Fast das ganze Betriebssystem ist in C geschrieben.
- Swapping Mechanismus für Prozesse.
- Buffer Cache Mechanismus für Block Devices.
- Prozess System mit Prozesskommunikation (IPC und Sockets).
- Philosophie: Alles möglichst nur einmal zu tun und modular zu verwenden. Es können z.B. komplexe Befehle oder Programme aus einfacheren aufgebaut werden.
- Durch hohe Portabilität von Betriebssystem und Anwendungsprogrammen auf vielen Plattformen vom PC bis Mainframe verfügbar.
- Grosse Auswahl an Software.

A.4 Filesystem

Ein Filesystem dient dazu, Informationen in einem Computersystem den Benutzern über Namen zur Verfügung zu stellen, und Details über die zugrunde liegende Hardware vor dem Benutzer zu verbergen. Weiters regelt es Zugriffsrechte auf einzelne Files. Das Filesystem von UNIX ist einfach. Es gibt nur drei verschiedene Typen von Files. Die Ein/Ausgabe Schnittstellen für alle Files sind einheitlich. Geräte werden im Filesystem wie einfache Files behandelt.

Files

Ein gewöhnliches File enthält ein Dokument (Text, Graphik, Datenbankfile) oder ein Programm. Die Files besitzen keine Satzstruktur, sondern werden als sequentielle Ströme von einzelnen Zeichen behandelt. Eine Blockstruktur wie sie auf Disketten oder Festplatten vorhanden ist, ist für den Benutzer nicht erkennbar, also transparent.

Directories

Directories (Kataloge, Verzeichnisse) beinhalten Verweise auf andere Files oder Directories. Alle Files eines UNIX Filesystems können von einem Root Directory (Wurzel) aus über Pfade erreicht werden. UNIX besitzt somit ein *hierarchisches Filesystem.* Directories können beliebig erzeugt und gelöscht werden, sofern die Zugriffsrechte für einen bestimmten Benutzer das zulassen.

Gerätedateien

Gerätedateien stellen die *Schnittstelle* zu Ein/Ausgabegeräten dar. Der Zugriff erfolgt über dieselbe Schnittstelle, wie bei gewöhnlichen Files. *Gerätetreiber* stellen die Verbindung von der Gerätehardware zum UNIX Filesystem her. Es wird zwischen *Block Devices* und *Character Devices* unterschieden. Typische Block Devices sind *Festplatten* oder *Disketten*, Character Devices sind z.B. *Streamer Tapes* oder *Terminals.* Jedes Block Device ist auch als Character Device ansprechbar (*"raw devices"*).

Zugriffsrechte

UNIX bietet die Möglichkeit alle Files (also auch Directories und Geräte) vor unberechtigtem Zugriff zu schützen. Das geschieht über die Vergabe von Zugriffsrechten für den Besitzer eines File, für seine Gruppe und für den Rest der Welt. Zugriffsrechte bestehen für das Schreiben, Lesen und Ausführen von Files.

Filesysteme

Ein UNIX Filesystem besteht logisch aus einem Baum, in dem alle Files und Directories von einer Wurzel aus erreicht werden können. Typischerweise sind diese Daten jedoch nicht auf einem einzigen Gerät abgelegt, sondern können über mehrere Festplatten (oder Unterteilungen von Festplatten) aber auch über mehrere Rechner (*Network File System*) verteilt sein. Diese Unterteilung ist für den Benutzer (ausser bei Hardwareausfällen oder Netzunterbrechungen) völlig transparent. Es ist möglich durch anhängen (`mount`) weiterer Filesysteme an beliebigen Punkten im Directory-Baum diesen im laufenden Betrieb zu erweitern aber auch einzuschränken.

Standard Directories

Einige Directories in einem UNIX Filesystem beinhalten spezielle Information für das System wie z.B. Systemtabellen, Kommandos, Gerätedateien. Diese haben festgelegte Namen und können normalerweise von einem gewöhnlichen Benutzer nicht verändert werden.

A.5 Prozesse

Prozesse repräsentieren in UNIX jede Art von Benutzertätigkeit.

„Ein Prozess ist eine einzige Folge von Ereignissen und besteht aus einem Hauptspeicherbereich und aus Files, auf die zugegriffen wird.“

Da UNIX ein Multi-User-Multi-Tasking Betriebssystem ist, können mehrere Prozesse mehrerer Benutzer gleichzeitig aktiv sein. Unter einem Single-Prozessorsystem kann jedoch immer nur ein Prozess zu einem bestimmten Zeitpunkt tatsächlich laufen. Ein Prozess kann sich selbst durch einen anderen Prozess ersetzen (*exec*) oder in einen Vaterprozess und einen Sohnprozess verzweigen (*fork*). Der Sohn erbt vom Vater sowohl Programmtext als auch Hauptspeicherdaten und zugeordnete Files.

Prozesse können über verschiedene Mechanismen (*Signale, Pipelines, Sockets, Semaphore*) miteinander kommunizieren.

Bestimmte Prozesse werden von UNIX beim Hochfahren des Systems automatisch gestartet (*Dämonen*) und laufen normalerweise, bis das System gestoppt wird. Andere Prozesse werden aufgrund von Benutzereingaben oder anderen Ereignissen dynamisch erzeugt.

Jedem Prozess sind standardmässig drei Files zugeordnet. Standard Eingabe (*stdin*), Standard Ausgabe (*stdout*) und Fehler Ausgabe (*stderr*). Oft sind diese Files mit dem Bildschirm verbunden, von dem aus ein Prozess gestartet wurde, es können aber auch beliebige andere Files sein.

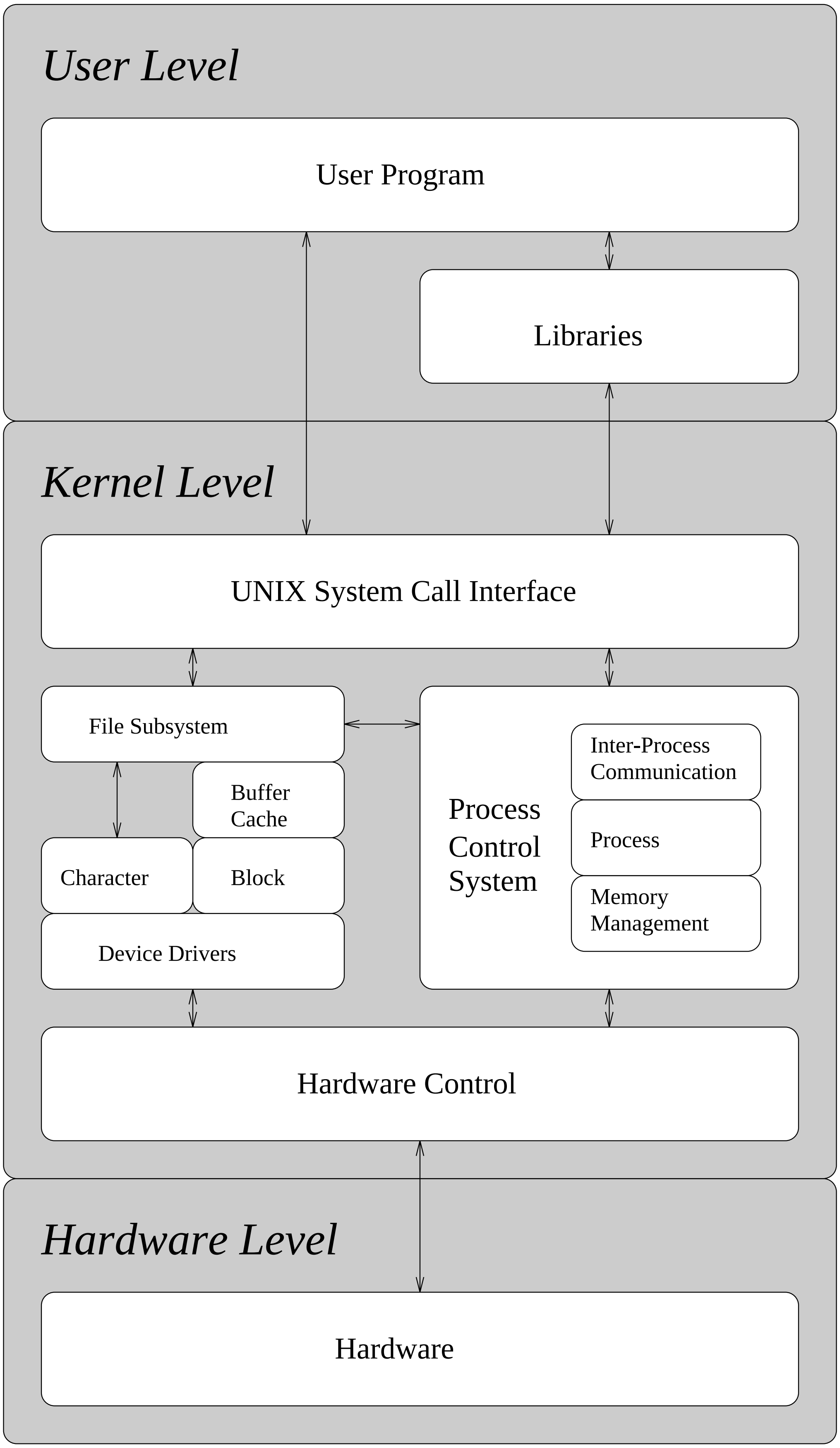
Jeder Prozess besitzt ein *Root Directory* und ein *Working Directory.* Alle Filenamen können relativ zum Root oder zum Working Directory angegeben werden.

Jedem Prozess ist ein *Benutzer* zugeordnet. Die Zugriffsrechte eines Prozesses werden vom VaterProzess geerbt und sind in der Regel identisch mit den Zugriffsrechten des Benutzers, der den Prozess gestartet hat.

Prozesse besitzen ein *Environment* (Umgebung), das eine Liste von Name=Wert Zuordnungen enthält und ebenfalls vererbt wird.

A.6 Schichtenmodell

Prozesse kommunizieren nicht direkt mit der zugrundeliegenden Hardware, sondern können nur über den Kernel darauf zugreifen. Der Kernel ist ein Programm, das zum Systemstart in einen eigenen geschützten Speicherbereich geladen wird und beinhaltet *Systemtabellen, Gerätetreiber, Prozesskontrollsystem* und *File Subsystem.* Ein Prozess kann entweder im *User Modus* oder im *Kernel Modus* ablaufen. In den Kernel Modus gelangt der Prozess nur durch Auslösen eines sogenannten Traps mittels eines *System Calls.* Das gewährleistet einerseits die hohe Hardwareunabhängigkeit von Benutzerprozessen andererseits eine sichere Abschirmung der Hardware und Systemdaten von den Benutzerprozessen.


User Level
User Program
Libraries
Kernel Level
UNIX System Call Interface
File Subsystem
Buffer
Cache
Character
Block
Device Drivers
Process
Control
System
Inter-Process
Communication
Process
Memory
Management
Hardware Control
Hardware Level
Hardware

A.7 Die Shell

Unter UNIX existieren verschiedene Shells, die untereinander mehr oder weniger kompatibel sind. Folgende Prinzipien sind diesen jedoch gemeinsam.

Die Shell ist ein Kommandointerpreter, der eine Schnittstelle zum Betriebssystem (Filesystem, Prozesse) bereitstellt. Durch Absetzen von Kommandos werden neue Prozesse erzeugt. Eingabe oder Ausgabe können durch Umlenkung von Standard Eingabe und Standard Ausgabe auf andere Files oder Geräte umgelenkt werden. Die Shell übernimmt auch die Generierung von Filenamen.

Die Shell kann ihre Eingabe sowohl von einem *Terminal* als auch von Files (*Shell Scripts*) einlesen. Die Shell ist mittels Steuerbefehlen (z.B. if...then...else) voll programmierbar. Dadurch können neue Kommandos definiert werden, die gleich wie existierende System Kommandos verwendet werden können.

Nach dem Login besitzt die Shell ein *Working Directory (= Home Directory)*, zu dem alle Pfadnamen relativ angegeben werden können. Das *Root Directory* der Shell ist in der Regel identisch mit dem Root Directory des Filesystems. Der Benutzer hat prinzipiell die Möglichkeit auf alle Files unter diesem Root Directory zuzugreifen, ausser wenn ihm das durch spezielle Zugriffsrechte für einzelne Files explizit verboten ist. Das ist notwendig, damit der Benutzer Kommandos, die normalerweise nicht in seinem Home Directory stehen, ausführen kann. Schreibzugriff besitzt ein gewöhnlicher Benutzer jedoch bis auf wenige Ausnahmen nur unter seinem Home Directory.

Der Shell sind nach dem Login die drei *Standarddateien* zugeordnet, die alle das Terminal repräsentieren, an dem sich der Benutzer angemeldet hat.

Die Zugriffsrechte der Shell sind die des Benutzers, der ein Login auf diese Shell durchgeführt hat und werden im Normalfall auf die von der Shell erzeugten Prozesse vererbt.

Die Shell hat wie jeder andere Prozess ein *Environment*. Daneben gibt es sogenannte *Shell Variable* und *Parameter*, die über Namen und Nummern angesprochen werden können. Diese Variablen können vom Benutzer gelesen und verändert werden. Shell Variable sind lokal zu einem Shell Prozess, können aber auch explizit an das Environment exportiert werden.

Nach dem Login wird von der Shell noch eine *Login File* eingelesen und interpretiert, die jedem Benutzer gestattet seine eigenen Voreinstellungen für das Environment vorzunehmen. Der Name dieses File ist je nach verwendeter Shell unterschiedlich (*.login = C Shell, .profile = Bourne und Korn Shell*).

Die Bourne Shell

Die Bourne Shell (`sh`, `bsh`) ist der ursprüngliche UNIX Kommandointerpreter. Sie besitzt eine einfache Kommando Syntax und ist relativ einfach zu erlernen. Sie ist auf jedem UNIX System verfügbar, da die meisten Shell Scripts in Bourne Shell Syntax geschrieben sind. Sie ist (noch) die Standard Shell auf System V Systemen.

Die C Shell

Die C Shell (`csh`) ist eine Entwicklung der Universität Berkeley und lehnt sich syntaktisch an die Programmiersprache C an. Sie ist die Standard Shell für BSD Systeme. Sie besitzt gegenüber der Bourne Shell einige Vorteile wie z.B. Vergabe von Synonymen (alias) für häufig verwendete Kommandos, einen History Mechanismus (Aufzeichnung und Wiederholung von früher eingegebenen Kommandos) und spezielle Befehle für die Kontrolle von Kommandos, die im Hintergrund ausgeführt werden (Job Control). Der Nachteil der C-Shell besteht darin, dass die Verarbeitungsgeschwindigkeit von Scripts wesentlich langsamer ist als bei der Bourne Shell. Sie wird deshalb seltener zur Programmierung von Scripts verwendet, da sie nicht kompatibel zur Bourne Shell ist. Die C Shell ist auf den meisten UNIX Systemen verfügbar.

Die Korn Shell

Die Korn Shell (`ksh`) ist relativ jung und versucht die Vorteile von Bourne Shell und C Shell zu kombinieren. Sie ist jedoch nur aufwärtskompatibel zur Bourne Shell. Die Korn Shell soll zur neuen Standard Shell für System V und OSF Systeme werden, ist jedoch noch nicht auf allen Systemen verfügbar.

Anhang B

C Shell

B.1 History C Shell

Die C Shell verfügt über eine interne Tabelle, in der die letzten Kommandozeilen gespeichert werden. Die Anzahl der zu speichernden Kommandos kann durch die history Variable definiert werden, welche am besten im File .cshrc gesetzt wird.

```
%cat .cshrc
....
set history=40                    #Speichert die letzten 40 Kommandos
....
%
```

History-Tabelle auflisten

```
%history
1 whoami
2 date
3 ls
4 history
%
```

Die History Tabelle wird immer mit dem Ausrufezeichen ! angesprochen.

Das zuletzt aktivierte Kommando wiederholen

```
%!!
history
1 whoami
2 date
3 ls
4 history
5 history
%
```

Das 3. Kommando wiederholen

```
%!3
ls
....
%
```

Das letzte Kommando wiederholen, welches mit da beginnt

```
%!da
date
.....
%
```

Fehlerkorrektur in Kommandozeilen letztes Kommando: ^

```
%rm war
rm: war: no such file or directory
%^war^wer
rm wer
%
```

beliebiges Kommando: substitute

```
%history
44 date
45 rm war
46 ls
% !45:s/war/wer
rm wer
%
```

B.2 Alias C Shell

Mit Hilfe der Funktion alias können Kommandos gekürzt oder anderweitig benutzt und aufgerufen werden. Zum Beispiel können Sie ein Alias ll definieren, welches das Kommando ls -l ausführen soll.

```
%alias ll 'ls -l'
%ll
-rw-rw-rw-  1 kurs       21768 Jan  8 10:22 File1
-rw-rw-rw-  1 kurs       12661 Jan  9 10:22 File2
%
```

Hinweis:

Alias, welche immer wieder verwendet werden sollen, können im File .cshrc definiert werden und stehen dann dem Benutzer immer zur Verfügung.

Anhang C

Korn Shell

C.1 History Korn Shell

Die Korn Shell speichert per Default die letzten 128 Kommandos in ein history File. Mit dem Kommando fc kann dieses File aufgelistet werden oder können Kommandos neu editiert werden.

fc [-l] [-e Editor]

Wird fc alleine aufgerufen, so wird ein Editor gestartet, welche über die Variable FCEDIT definiert werden kann und mit dem das letzte Kommando editiert werden kann.

```
$cat .profile
....
export FCEDIT=/usr/ucb/vi                    #vi als FC Editor definiert
....
$ls
$fc
<vi wird gestartet mit dem Kommando ,,ls'' als Inhalt>
....
```

Beim Verlassen des Editors wird das korrigierte Kommando ausgeführt. Mit dem Argument -l kann das History-File aufgelistet werden.

```
$fc -l
1 whoami
2 date
3 ls
4 history
$
```

Mit dem Argument -e kann der Editor angegeben werden

```
$fc -e /usr/ucb/vi
<vi wird gestartet>
```

Wird als Editor das Zeichen „-“ eingegeben, so kann die Editor-Phase unterdrückt und mit folgender Syntax direkt die Substitution ausgeführt werden.

```
$ls
bad good
$ls bad
bad
$alias
....
$fc -e - bad=good l
ls  good
good
$
```

Das fc Kommando ersetzt im letzten mit „l“ startendem Kommando den String „bad“ mit dem String „good“ und führt anschliessend das Kommando aus.

Tip: Setzen Sie folgende Alias im File .profile oder .kshrc

```
$cat .kshrc
...
history=fc -l
r=fc -e -
...
$
```

C.2 Alias Korn Shell

Mit Hilfe der Funktion alias können Kommandos gekürzt oder anderweitig benutzt und aufgerufen werden. Zum Beispiel können Sie ein Alias ll definieren, welches das Kommando ls -l ausführen soll.

```
$alias ll 'ls -l'
$ll
-rw-rw-rw-  1 kurs       21768 Jan  8 10:22 File1
-rw-rw-rw-  1 kurs       12661 Jan  9 10:22 File2
$
```

Hinweis:

Alias, welche immer wieder verwendet werden sollen, können im File .profile oder .kshrc definiert werden und stehen dann dem Benutzer immer zur Verfügung. Um das File .kshrc zu benutzen, muss die Environment Variable ENV gesetzt sein.

```
$cat .profile
....
export ENV=$HOME/.kshrc
....
$
```

Anhang D

UNIX Kommandos

Hier eine Zusammenfassung der wichtigsten UNIX Kommandos zum Nachschlagen. Einige davon sind in diesem Kurs nicht genauer erklärt, sind aber wie alle Kommandos in den Manual Pages dokumentiert.

D.1 Benutzer und Login

login	Login ausführen
logout	Sitzung beenden
exit	Shell verlassen
su	Benutzer wechseln
newgrp	Gruppe wechseln
passwd	Passwort setzen
whoami	Benutzername ausgeben
id	Benutzer und Gruppen Identifikation ausgeben
groups	Gruppen Zugehörigkeit anzeigen

D.2 System Information

df	Information über Filesysteme
date	Datum anzeigen und setzen
du	Platten Auslastung anzeigen
last	Information über letzte Logins
uname	Zeigt Namen des Operating Systems
w	Systemauslastung
who	Wer ist an System angemeldet
hostname	Maschinenname

D.3 Files und Directories

cd	Working Directory wechseln
pwd	Working Directory anzeigen
ls	Anzeige des Directory Inhaltes
rm	Files löschen
cp	Files kopieren
ln	Files linken
mv	Files umbenennen
mkdir	Directory anlegen
rmdir	Directory löschen
chgrp	Gruppe eines File ändern
chmod	Eigentümer eines File ändern
find	Files finden
touch	Zugriffs und Änderungsdatum eines File setzen
cpio	Files auf Band kopieren
tar	Files auf Band archivieren
dd	Files kopieren und konvertieren

D.4 File Inhalte

cat	Files anzeigen und verbinden
pg	Files auf Bildschirm ausgeben
more	Files auf Bildschirm ausgeben
od	Oktal oder Hexadezimal Dump
head	Anzeige der ersten Zeilen eines File
tail	Anzeige der letzten Zeilen eines File
cmp	Files vergleichen
compress	Daten komprimieren
awk	Programmiersprache für Mustererkennung und Filemanipulation
diff	Textdateien vergleichen
cut	Ausgewählte Felder einer File ausgeben
egrep	Durchsuchen eines File nach einem Muster
grep	Durchsuchen eines File nach einem Muster
fgrep	Durchsuchen eines File nach einem Muster
join	Verbindet Zeilen von zwei Files
sort	Files sortieren
tr	Zeichen umwandeln
wc	Zeichen, Wörter oder Zeilen in einem File zählen
tee	Standard Eingabe anzeigen und auf File kopieren
ed	Zeilen Editor
ex	Zeilen Editor
vi	Bildschirm Editor
sed	Stream Editor

D.5 Prozesse

ps	Prozess Status anzeigen
kill	Signal an Prozess senden
env	Environment anzeigen
nice	Kommando mit anderer Priorität starten
nohup	Kommando nicht unterbrechbar aufrufen
sleep	Ausführung für bestimmte Zeit unterbrechen
time	Ausführungszeit eines Kommandos anzeigen
batch	Kommando zu späterer Zeit ausführen
at	Kommando zu bestimmter Zeit ausführen
crontab	Kommandos automatisch ausführen

D.6 Shell

sh	Bourne Shell
csh	C Shell
ksh	Korn Shell
expr	Argumente als Ausdruck ausführen
echo	Argumente ausgeben
basename	Filename ohne Suffix und Pfad anzeigen
test	Bedingung auswerten
if	Bedingte Ausführung von Shell Kommandos
then	
else	
fi	
for	Liste abarbeiten
do	
done	
while	Schleife
do	
done	
case	Auswahl von Kommandos
esac	

D.7 Drucken

lp	Files drucken
lpr	Files drucken
lpq	Status der Warteschlangen
lpstat	Status der Warteschlangen
lprm	Druckauftrag aus der Warteschlange entfernen
cancel	Druckauftrag abbrechen
pr	Druckaufbereitung von Files
banner	Strings in grosse Buchstaben umwandeln

D.8 Kommunikation und Netzwerk

wall	Botschaft an alle senden
write	Botschaft an anderen Benutzer senden
talk	Konversation über Netzwerk
mail	Post senden und ansehen
mesg	write oder talk zulassen und verbieten
ftp	Files auf anderen Rechner übertragen
telnet	Login auf fremdem Rechner
rlogin	Login auf anderem UNIX Rechner
rsh	Kommando auf anderem UNIX Rechner ausführen
rcp	Files auf anderen UNIX Rechner kopieren
ping	Verbindung zu anderem Rechner testen
finger	Benutzer Information

D.9 Programmierung

cc	C Compiler
as	Assembler
ld	Linken von Object Modulen
ar	Archivierung von Object Modulen
make	Projekt auf neuesten Stand bringen
sdb	Symbolischer Debugger
sccs	Source Code Control System
what	Identifikation von Modulen anzeigen
lex	Lexikalische Analyse
yacc	Parser Generator
lint	Überprüft C Programme
nm	Symboltabelle anzeigen
size	Grösse von Object Files anzeigen

D.10 Diverse Kommandos

bc	Taschenrechner
cal	Gibt einen Kalender aus
pic	troff PräProzessor für Zeichnen von Bildern
tbl	troff PräProzessor für Tabellen
nroff	Textaufbereitung
troff	Textaufbereitung
stty	Terminal Eigenschaften setzen
tty	Terminal Name anzeigen
tset	Terminal setzen
tput	Terminfo Datenbank lesen
man	Manual Pages ausgeben
whatis	Kurzinformation über Kommando
apropos	Sucht Text in Manual Pages Titeln

Anhang E

Anhang zu X-Windows und Netzwerk

E.1 Arbeiten in einem Netz unter X-Windows

Damit in einem Netzwerk unter der grafischen Oberfläche von X-Windows ein Progamm auf einem anderen Rechner ausgeführt und die Bildschirmausgabe auf dem eigenen Rechner erfolgen kann, muss wie folgt vorgegangen werden (im folgenden bedeutet “lokaler Rechner” die Maschine, an welcher der Benutzer sitzt, “remote-Rechner” die Maschine, welche via Netzwerk angesprochen wird):

- Auf dem lokalen Rechner muss ein windowmanager laufen
- Der “remote-Rechner” kriegt durch
  `xhost + remote_rechnername`
  die Erlaubnis, auf den Bildschirm des lokalen Rechners *schreibend* zuzugreifen
- Einloggen auf den remote-Rechner
- Die *Environmentvariable* `DISPLAY` auf dem remote-Rechner setzen auf
  `lokaler_Maschinenname:0`
- Nach Beenden der Arbeit auf dem remote-Rechner auf der lokalen Maschine durch
  `xhost - rechnername` Schreiberlaubnis des remote-Rechners auf den Bildschirm der lokalen Maschine wieder entfernen

ACHTUNG:

Falls einfach `xhost +` eingegeben wird, kann von *jeder* Maschine am Netz auf den Bildschirm der lokalen Maschine geschrieben werden.

Dieser Mechanismus ist Maschinenspezifisch, *nicht* Benutzerspezifisch. Das bedeutet, nach `xhost + wig` kann jeder Benutzer der Maschine `wig` auf den Bildschirm der lokalen Maschine schreiben.

Bei Maschinen, die mit mehr als einer Ethernet-Karte betrieben werden, muss bei Zugriffen ausserhalb des lokalen Netzes natürlich ein anderer Name bei `xhost + ...` verwendet werden als innerhalb des Subnetzes. (Der Sysadmin weiss im allgemeinen, wieviele Ethernetkarten in seiner Maschine drin sind und wie die Maschine innerhalb und ausserhalb des eigenen Subnetzes heisst...)

E.2 Arbeiten mit r*Kommandos

Um mit `r*kommandoname` arbeiten zu können, kann der Benutzer in seinem *Home-directory* ein File mit dem Namen `.rhosts` anlegen. Darin existiert pro Maschine eine Zeile mit remote-Maschinenname und Benutzername. Die Datei soll nur für den owner lesbar sein!!

Inhalt meiner Datei `.rhosts` auf meiner Maschine (saana):

```
linux pm
wig pm
```

Damit kann der Benutzer `pm` von den Maschinen `linux` und `wig` aus sämtliche `r*Kommandos` auf meiner Maschine ausführen.

Der gesamte Mechanismus der `r*Kommandos` kann durch den Sysadmin ausgeschaltet werden.

Literaturverzeichnis

[1] Paul Abrahams and Bruce Larson; *Unix for the Impatient*, Addison Wesley, 1992
Comment: **** Highly Recommended **** A new, comprehensive, in-depth reference to Unix ...

[2] Kaare Christian; *The Unix Operating System*, Wiley, 2nd ed. 1988
Comment: A classic overview of Unix commands ... very good ...

[3] Mark Sobell; *A Practical Guide to the Unix System*, Benjamin / Cummings, 1990
Comment: Similar to Christian's book ... slightly easier to read ... There is a new edition for System V Release 4 ...

[4] Mitchell Waite, Donald Martin and Stephen Prata; *The Waite Group's Unix System V Primer*, Hayden, 2nd ed. 1992
Comment: **** Highly Recommended **** A very very good hand-holding tutorial-type book for Unix/SVR4 ...

[5] Brent Heslop, David Angell; *Mastering SunOs*, Sybex, 1990
Comment: A good, comprehensive hand-on text to SunOs and OpenWin ...

[6] Peter Norton, Harly Hahn; *Peter Norton's Guide to Unix*, Bantam Computer, 1991
Comment: One of the many books by Peter Norton ... Good coverage ... but some errors ...

[7] Daniel Gilly, O'Reilly staff; *Unix in a Nutshell*, O'Reilly, 2nd ed. 1992 (for System V and Solaris 2)
Comment: **** Highly Recommended **** An excellent desktop reference to almost all Unix commands ... Probably the most inexpensive Unix book around ... Also an edition for 4.3. BSD ...

[8] Don Libes, Sandy Ressler; *Life with Unix - A Guide for Everyone*, Prentice Hall, 1990
Comment: **** Highly Recommended **** An everything-you-want-to-know-about-Unix book ... It includes info you might not find elsewhere ...

[9] James Gardner; *Learning Unix*, Sams, 1991
Comment: With disks containing MSDOS stimulation of Unix (MSK Tools) ... A good tutorial / reference book for those without constant access to Unix ...

[10] Pete Holsenberg; *Unix Desktop Guide to Tools*, Sams, 1992
Comment: A new and comprehensive guide to numerous Unix utilities ...

[11] Gail Anderson, Paul Anderson; *The Unix C Shell Field Guide*, Prentice Hall, 1986
Comment: The C-Shell bible - everything you need to know to use Unix effectively ...

[12] Martin Arick; *Unix C Shell - Desk Reference*, QED Technical, 1992
Comment: A more recent text on C-Shell ...

[13] John Valley; *Unix Desktop Guide to the Korn Shell*, Sams, 1992

Comment: This one is, IMHO, even better and easier to read than the authoritative work by Korn and Bolsky ...

[14] Maurice Bach; *The Design of the Unix Operating System* , Prentice Hall, 1986

Comment: An excellent reference on the internals of System V ... This book and the next one are indeed highly technical ...

[15] Samuel Leffler et al; *The Design and Implementation of the 4.3 BSD Unix Operating System* , Addison-Wesley, 1990

Comment: An authoritative description of the design of BSD Unix ...

Index