Nutzung des Betriebssystems UNIX™
Grundkurs

32. Überarbeitung, September 2018 | **Autoren:** Mitarbeiter des JSC |

JÜLICH Forschungszentrum | JÜLICH SUPERCOMPUTING CENTRE

Inhalt

Inhalt

Inhalt

UNIX-Historie

- Grundlagen in den 60-er Jahren
  (BCPL, Pagingstrategien, Scheduling, Dateisysteme)
- 1967 Dennis Ritchie $\rightarrow$ AT & T (Bell Labs)
- 1968 Ken Thompson $\rightarrow$ AT & T (Bell Labs)
- 1969 Unics entsteht in Bell Labs
  UNiplexed **I**nformation and **C**omputing **S**ystem
  - Entwicklung eines Dateisystems
  - Optimierung eines Space Travel Program

  (DEC-PDP/7, Assembler, Single/Dual User, Shell, Prozesse)
- 1970 Brian Kernighan prägt UNIX Begriff
- 1971 Version 1
  ($\rightarrow$ DEC-PDP/11, Assembler, Textverarbeitung + Patente)
- 1972 Version 2
  (Pipelines, C-Compiler)

UNIX-Historie

- 1973 Version 4
  Neuimplementierung in C:
  - Prozeßverwaltung (Thompson)
  - Ein-/Ausgabesystem (Ritchie)
- 1975 Version 6
  (erste verfügbare Version außerhalb Bell Labs)
- 1978/79 Version 7
  (Dateisystem überarbeitet, Bourne-Shell, BOS portabel)

  Aufspaltung in 2 Linien:
  - System V (AT & T)
    (konservativ, solide, kommerzielle Produkte)
  - BSD (Berkely Software Distribution)
    (fortschrittlicher, mutiger, Universitäten + Forschung,
    Neuerungen: C-Shell, TCP/IP, VI-Editor, virtual memory)

  → **Zersplitterung in viele Derivate**

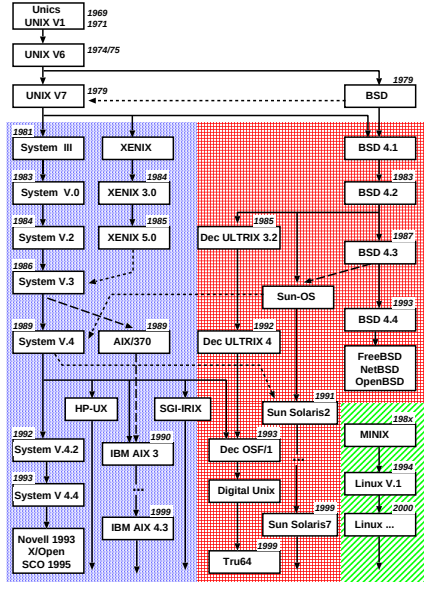
UNIX-Historie

- 1984 X-Oberfläche durch MIT (X11)
- 1987 Versuch der Vereinheitlichung durch Sun und AT & T (1991 wieder aufgelöst)
  - 1989 System V Revision 4 (SVR4)
  - Grundstein für viele kommerzielle Anbieter
- 1988 Gründung von OSF als Gegenpart zu Sun und AT & T (Open Software Foundation: IBM, DEC, HP)
- 1990 Die Skriptsprache Perl (Larry Wall) gewinnt neben den Shells mehr an Bedeutung
- 1991 Freigabe von OSF1 (Standard bei DEC-UNIX)
- 1992 Sun entwickelt UNIX-Derivat Solaris
  (erste SMP-Architektur)

UNIX-Historie

- 1994 Linux (Linus Torvald), ein im Sourcecode frei verfügbares UNIX-Derivat, gewinnt an Bedeutung. Es wurde in 1. Linie für Intel-Hardware entwickelt, ist aber auch für DEC Alpha bzw. Sun Sparc verfügbar
- 1997 KDE, ein freies Desktop-System unter Linux, entsteht, da CDE lizenzpflichtig ist
- 1999 Zur Unterstützung neuer Hardware werden 64-bit Versionen der gängigen Betriebssysteme entwickelt
  (AIX 4.3, Solaris 2.7, Tru64)
- Beispiele für derzeitige Betriebssysteme:
  openSUSE Leap 42.3, AIX 7.2, Oracle Solaris 11, CentOS 7

UNIX-Stammbaum



Eigenschaften von UNIX

- implementiert in höherer Programmiersprache und daher portabel
  (UNIX ist auf den meisten Plattformen - von PC bis Höchstleistungsrechner - verfügbar)
- Multiuser- und Multiprocess-System
  (mehrere Benutzer mit jeweils mehreren Logins)
- spezielle Maschineneigenschaften bleiben dem Benutzer verborgen
  (Blockung der Platte, Art des Datenträgers)
- baumartig aufgebaute, hierarische Verzeichnisstruktur
- einfache Filestruktur (ohne Recordlänge und -formate)
- Hilfsmittel zum Aufbau komplexer Anwendungen
  (mächtige Shells z.B. bsh, ksh, csh, ssh, bash, ...)
- Vielzahl von Anwendungen und Utilities
- graphische Benutzeroberflächen
  (X11, OSF Motif Manager, OpenWindows, XDM, CDE, KDE)

weitere Eigenschaften von UNIX

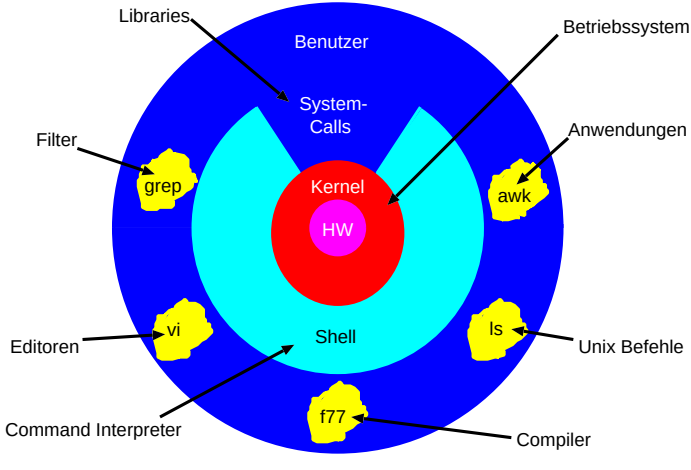
Anwendungen, die in den meisten UNIX-Betriebssystemen integriert sind:

- Internet-Dienste
  - bieten weltweite Kommunikation
    (E-Mail, Datenaustausch, Login, Informationsservice, ...)
- Network File System (NFS)
  - ermöglicht den Zugriff auf Daten anderer Rechner, so als wären sie im eigenen System
- X11 (graphische Benutzeroberfläche)
  - Window-basiert
  - auf X11 aufbauend existieren eine Vielzahl menügesteuerter Anwendungen
    (Editoren, Graphik-Tools, Mail-Tools, ...)
  - netzwerkfähig Client-Server Prinzip:
    Rechnen auf einem entfernten Rechner,
    Ausgabe auf dem Schirm des lokalen Rechners

UNIX-Systeme im Forschungszentrum

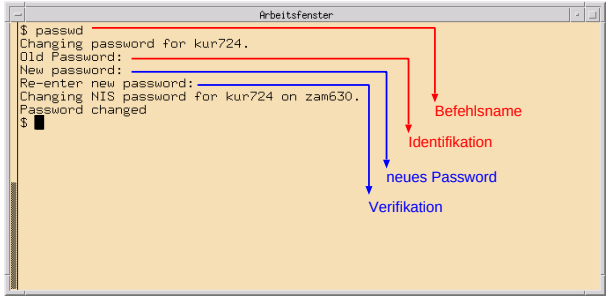
UNIX-System	Hersteller
Linux	u.a. Novell(SuSE), Red Hat, ...
AIX	IBM
TRU64 (früher Digital UNIX) ULTRIX	Compaq (früher DEC)
Solaris (früher SunOS)	Sun-Microsystems
HP-UX	Hewlett Packard
IRIX	Silicon Graphics

Aufbau des UNIX-Systems



Inhalt

Ändern des Passwort-Eintrages



- 6-8 Zeichen lang
- Groß- und Kleinschreibung
- mindestens eine Ziffer und/oder Sonderzeichen
- keine persönlichen Daten (Vorname, Nachname, Geburtstag, KFZ-Kennzeichen, ...)
- keine Begriffe aus dem Lexikon
- Abkürzung eines Satzes (Beispiel: *4tPoL. = For the Power of Love ...)*
- Sinnvoll ist die Benutzung eines Password-Managers, z.B. KeePassX

Informations-Befehle

uname	listet Betriebssystemnamen
news	listet News und Infos
cat /etc/motd	listet Hotnews
users	listet logged-in Benutzer
who	listet logged-in Benutzer
whoami	listet eigenen Benutzernamen
id	listet Informationen zu sich selbst (numerische uid und gid, Gruppenzugehörigkeit)
hostname	listet Namen des Rechners
date	listet Datum und Uhrzeit

Inhalt

Grundsätzliches

- Ein Kommando ist eine Anweisung an das UNIX etwas zu tun.
- Die Eingabe erfolgt normalerweise auf dem Shell-Prompt.
- UNIX-Systeme sind prinzipiell *full duplex*, d.h. die eingegebenen Zeichen werden zum System geschickt und von da zum Bildschirm.
- UNIX unterscheidet zwischen Klein- und Großbuchstaben.
  Vorsicht! Benutzernamen nicht groß schreiben!
- *Type-Ahead* ist möglich, d.h. es braucht nicht auf die Beendigung des vorhergehenden Kommandos gewartet zu werden.
- Die Interpretation der Tasten, insbesondere der Control-Keys, ist konfigurierbar *(stty-Kommando)*.

Grundsätzliches

- *Control-Keys* bestimmen (abhängig von der Shell) den Ablauf:

RETURN/ENTER	Verarbeitet die Eingabe
BACKSPACE	Löscht das letzte Zeichen der Eingabe
Ctrl-c	Bricht die Ausführung des Befehls ab
Ctrl-d	Beendet die Eingabe (*EOF*)
Ctrl-z	Suspendiert die Ausführung des Befehls
	fg setzt den Befehl im Vordergrund fort $\Rightarrow$ kein Shell-Prompt mehr
	bg schickt den Befehl in den Hintergrund $\Rightarrow$ Shell-Prompt; weitere Kommandos werden sofort bearbeitet
Ctrl-s	Hält die Ausgabe auf dem Bildschirm an
Ctrl-q	Setzt die Ausgabe auf dem Bildschirm fort

Hinweis: Ctrl entspricht Strg auf deutscher Tastatur.

Shell-Überblick I

- Die Shell ist die Schnittstelle zwischen dem UNIX-System und dem Benutzer.
  Die Shell interpretiert die eingegebenen Kommandos. (Shell=Kommandointerpreter)
- Die Shell führt dann die geforderte Aktion aus.
- Typische Shell-Funktionen sind:
  - Expansion von **Wildcard-Zeichen** in Dateinamen
  - Ein- / Ausgabeumlenkung (**I/O Redirection**)
  - Bildung von Kommandoketten (**Pipes**)
  - Bereitstellen von **Ersetzungsmechanismen** für Kommandos und Shell-Variablen (**‘...‘,$x**)
  - geordnete Fehlerbehandlung
  - Zusammenfassen von Operationen zu Kommandoprozeduren (**Shell Scripts**)
  - Zurückholen früherer Befehle (**History**)
  - Definition von Abkürzungen (**Aliasnamen**)

Shell-Überblick II

- es gibt 3 "Basis-Shells":
  - **Bourne-Shell**
    - älteste größere UNIX Shell
    - eingeschränktes Angebot an Funktionen
    - Shell Scripts einfach zu programmieren

    path/sh
  - **C-Shell**
    - erweiterter Funktionsumfang, daher einfacher beim interaktiven Gebrauch
    - C-like zu programmieren

    path/csh
  - **Korn-Shell**
    - nahezu kompatible Erweiterung der Bourne-Shell
    - erweitert durch die 'Goodies' der C-Shell
    - erweitert durch zusätzliche Funktionen

    path/ksh

Shell-Überblick III

- weitere Shells sind:
  - **Bourne-Again-Shell**
    - GNU Implementierung der Bourne-Shell (mit Erweiterungen)

    path/bash
  - **tc-Shell**
    - nahezu kompatible Erweiterung der C-Shell
    - erweitert durch zusätzliche Funktionen

    path/tcsh
  - **POSIX Shell**
    - entspricht der Korn Shell

    path/psh
  - **Restricted Shell**
    - eingeschränkte Bourne Shell

    path/Rsh

Shell-Überblick IV

- "Remote Shells":
  - **Remote Shell**
    - Ausführen von Programmen/Kommandos auf entfernten Rechnern

    path/rsh
  - **Secure Shell**
    - sicheres Ausführen von Programmen und Kommandos auf entfernten Rechnern

    path/ssh
- Hinweise zum Aufruf
  - die Pfade *path* sind installationsabhängig
  - die tatsächlichen Pfade bekommt man mit:

    type ***shell***
    which ***shell***
    whereis ***shell***

Beispiel: **`type ksh`** ⇒`/usr/bin/ksh`

Kommandoablauf

- Abgesehen von einigen Shell-internen Kommandos (**`cd`**, **`ls`**, ...) erzeugt die Shell einen eigenständigen Prozeß.
- Der normale Befehlsablauf ist synchron, d.h. die Shell wartet auf das Ende des abgesetzten Kommandos, bevor sie die weitere Eingabe verarbeitet.
- Mit '&' [1] ist ein asynchroner Ablauf (background process) möglich; die Ausgabe erfolgt auf das Terminal, wenn sie nicht umgelenkt wird.

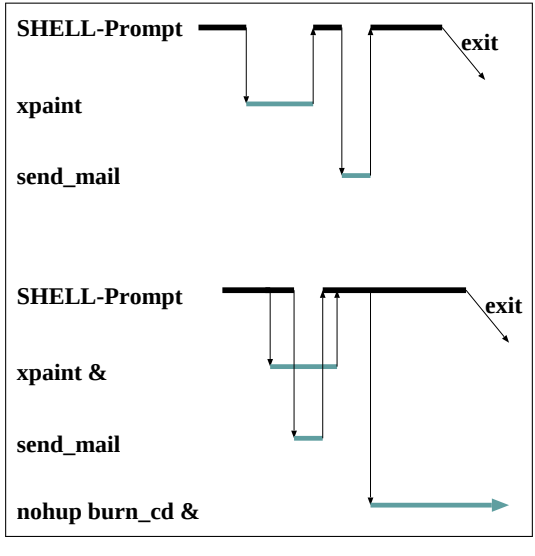
  command **&**

- Mit `nohup` (**noh**ang**up**) läuft das Kommando auch nach Beenden der Session weiter; die Ausgabe erfolgt in die Datei *nohup.out*, wenn sie nicht umgelenkt wird.

  nohup ***command*** **&**

[1] & bewirkt auch ein Newline, d.h. Befehlssequenzen werden abgebrochen, z.B. bei "`date & ; who`" wird "`who`" nicht ausgeführt

Abbildung zum Kommandoablauf



Kommandosyntax I

Ein Kommando ist eine Sequenz von Worten, die durch Leerzeichen getrennt sind:

command **[*options*]** **[*arguments*]**

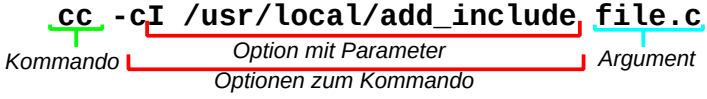
- *command* ist der Name des Kommandos,
  options kontrollieren/ändern die Ausführung und
  arguments sind Argumente für das Kommando
- *options* bestehen in der Regel aus einem Buchstaben als Schlüsselwort und ggf. zusätzlichen Parametern
- Schlüsselworte werden durch ein vorangestelltes Minus '-' ohne Leerzeichen gekennzeichnet (daher Filenamen nicht mit '-' beginnen)
- mehrere Schlüsselworte können ohne jeweiliges '-' hintereinander geschrieben werden, die Reihenfolge ist beliebig

Beispiel:

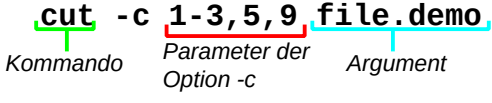
```
uname -s -r
uname -sr
uname -rs
```

Kommandosyntax II

- Schlüsselworte mit Parametern müssen am Ende der Optionsliste stehen, z.B.:



- mehrfache Parameter zu einem Schlüsselwort werden als Liste durch Komma getrennt oder in doppelte Hochkommata eingeschlossen und durch Blank getrennt, z.B.:



- das Ende des Kommandos wird durch den Beginn einer neuen Zeile (**newline** = **Enter**), ein **&** oder durch ein **;** gekennzeichnet: `date; who`
- für Folgezeilen muß das newline durch \ vor der Interpretation der Shell geschützt werden
- Kommentare werden durch ein vorangestelltes **#** gekennzeichnet

Online Help (man)

man [*options*] [*manual*] *command*

- zeigt 'online help' seitenweise an ("more" Format):

<**Space**>	⇒ nächste Seite
b	⇒ vorige Seite
<**Enter**>	⇒ eine Zeile weiter
q	⇒ Beenden

- aufgeteilt in einzelne Manuals:

1	Executable programs or shell commands
2	System calls
	...
8	Administrator commands
9	Kernel Routinen (nicht Standard)
local	/usr/local/... products

Beispiel: `man 1 ls`
liefert Beschreibung aus Manual 1

Online Help (man)

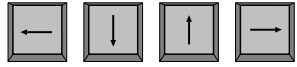
- **man -k *suchmuster***
  sucht das *suchmuster* in den Überschriften der Manuals und gibt die entsprechenden Referenzen und Kommandos aus

  Beispiel: `man -k password`

- weitere Informationen:
  für Linux: `http://www.suse.de`
  Suchmaschinen oder Webseiten anderer Hersteller

Command-Line-Editing I

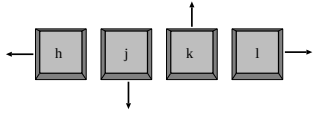
Aktuelle Systeme unterstützen die Pfeil- oder Cursor-Tasten von sich aus:



Über einen Befehl kann man diese Tasten als Umgebung aktivieren:

```
set -o emacs
```

Ansonsten bietet die Korn-Shell die Möglichkeit, Editor-Kommandos auf die Kommandozeile wirken zu lassen. Aktiviert wird dieser Modus durch die <**ESC**>-Taste (`set -o vi` Umgebung). Die Cursor-Tasten werden hierbei nicht unterstützt. Stattdessen müssen zum Bewegen in der Command-History bzw. im Command folgende Tasten benutzt werden:



Command-Line-Editing II

in der vi-Umgebung, wenn die Pfeiltasten nicht funktionieren

Dabei bedeutet:

k	Rückwärtsblättern in der History ↑
j	Vorwärtsblättern in der History ↓
h	Cursor im Kommando nach links ←
l	Cursor im Kommando nach rechts →

Einige Befehle zur Kommando-Modifikation:

i	Einfügen **vor** der aktuellen Cursor-Position (bis zum nächsten <ESC>)
a	Einfügen **hinter** der aktuellen Cursor-Position (bis zum nächsten <ESC>)
A	Anfügen am Zeilenende (bis zum nächsten <ESC>)
x	Löschen des Zeichens unter dem Cursor
r	Einzelnes Zeichen unter dem Cursor ersetzen
R	Mehrere Zeichen ab dem Cursor ersetzen (bis zum nächsten <ESC>)

Übung zum Command-Line-Editing

Benutzen Sie so oft wie möglich die Command History und die Maus.

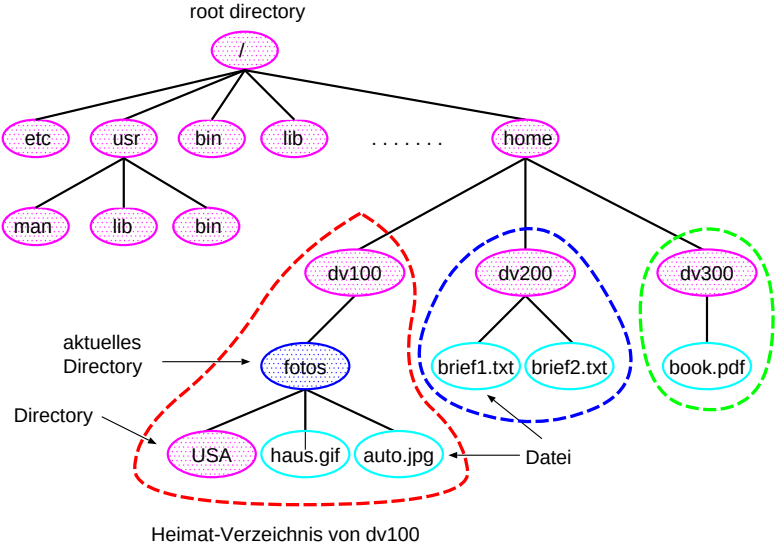
1. Blättern Sie in Ihrer Command History.
2. Ändern Sie einen **`man`** Befehl von vorher.
3. Markieren Sie mit der Maus einen Befehl von vorher und fügen ihn am aktuellen Prompt ein.
   (kopieren→linke Maustaste, einfügen→mittlere Maustaste)
4. Brechen Sie das Blättern durch die Command History mit <Ctrl>-c ab.

Inhalt

Aufbau und Elemente

- Sammlung von Files
- vier Sorten von Files:
  - Daten (ASCII oder binär)
    Dateityp: **normal**
  - besondere Files, die wiederum eine Sammlung von Files beschreiben
    Dateityp: **directory**
  - Zweit-, Dritt-,... - namen von Files
    Dateityp: **link**
  - besondere Files, die Devices beschreiben
    Dateityp: **char** oder **block**
- baumartig strukturiert
- Teilbäume werden auch Subdirectories oder Unterverzeichnisse genannt
- ein Dateibaum kann aus mehreren Filesystemen (eine spezielle Form von Subdirectories) bestehen

Struktur I


root directory
/
etc
usr
bin
lib
. . . . . .
home
man
lib
bin
dv100
dv200
dv300
aktuelles
Directory
fotos
brief1.txt
brief2.txt
book.pdf
Directory
USA
haus.gif
auto.jpg
Datei
Heimat-Verzeichnis von dv100

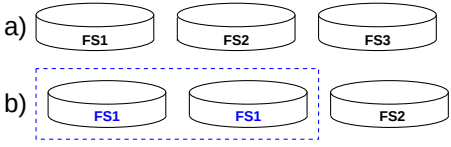
Struktur II

- Directories
  - sind Knoten im Dateibaum, die wiederum Teilbäume enthalten können
  - enthalten Verweise auf Dateien
  - sind streng hierarchisch
  - beginnen bei /
- Jeder Benutzer verfügt über ein **Heimatverzeichnis** im Dateibaum.
  Übliche Notationen sind:
  - /usr/$\boldsymbol{user}$ oder /usr/$\boldsymbol{group}$/$\boldsymbol{user}$ (System V)
  - /u/$\boldsymbol{user}$ oder /u/$\boldsymbol{group}$/$\boldsymbol{user}$ (BSD)
  - /home/$\boldsymbol{group}$/$\boldsymbol{user}$ (Forschungszentrum Jülich)
- Teilbäume, die auf unterschiedlichen Speichermedien oder Speicherbereichen liegen, nennt man Filesysteme.

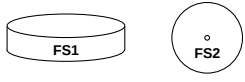
Filesystem

liegt auf einem separaten logischen Device
(Partition)

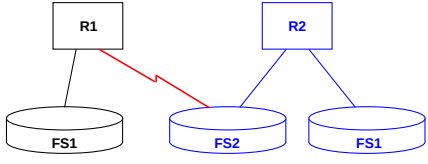
einer oder mehreren
verschiedenen Platten



unterschiedlichen Device Typen



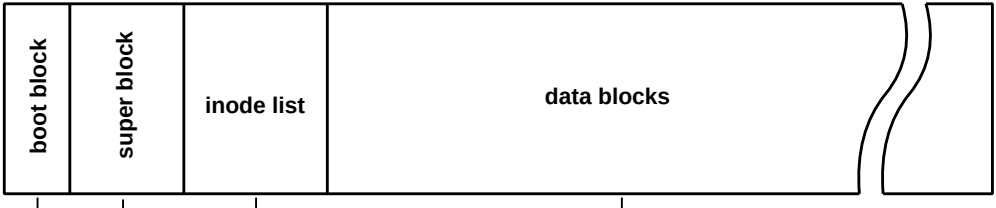
einem anderen Rechner,
der Zugriff erfolgt über das Netz



Filesystem

- wird in den UNIX Dateibaum eingehängt (mount);
  wahlweise statisch oder dynamisch (automount)
- der Einhängepunkt (Mount Point) im Dateibaum muß ein Directory sein;
  dies sollte leer sein (Falls belegt, werden die Daten logisch überlagert.)
- ist intern eine Folge von logischen Blöcken
  (512 Bytes | 1024 Bytes | 2048 Bytes | ...)
- die Blocklänge innerhalb eines Filesystems ist immer gleich;
  zwischen verschiedenen Filesystemen kann die Blocklänge variieren

Aufbau eines Filesystems



dient zum Initialisieren des Betriebssystems

beschreibt die Files: Name, Zugriffsrechte,... und zeigt auf die Datenblöcke

enthalten die Daten der Files; Daten einer Datei müssen nicht aufeinanderfolgend sein

beschreibt das Filesystem: Grösse, max. Anzahl Files, free space

Auflisten der Filesysteme (df)

df [*options*] [*dirname* | *filesystem-name*]

- Akronym für **d**isplay **f**ilesystem
- zeigt den gesamten und freien Speicherplatz des Filesystems an, in dem sich das spezifizierte Directory $dirname$ befindet
- wahlweise in Blöcken von 512 Byte (Standard UNIX) oder 1 KByte (Standard Linux) bei Angabe der **-k** Option
- ohne Angabe von $dirname$ werden alle zum Zeitpunkt vorhandenen Filesysteme aufgelistet

Beispiel:

```
$ df -k
Filesystem           1k-blocks      Used  Available  Use%  Mounted on
/dev/sdb3              5245052   3337392    1907660   64%  /
/dev/sdb1                23302      3940      18159   18%  /boot
zamlin1s:/usr/local   10490084   6700620    3789464   64%  /usr/local
zamlin1s:/home        19004276   5950716   13053560   32%  /home
```

Auflisten des belegten Plattenplatzes (du)

du [*options*] [*dirname*]

- Akronym für **d**isk **u**sage oder **d**irectory **u**sage
- listet belegten Plattenplatz auf Directory-Ebene inklusive Subdirectories
- wahlweise in Blöcken von 512 Byte (Standard UNIX) oder 1 KByte (Standard Linux) bei Angabe der **-k** Option
- die **-s** Option bildet die Gesamtsumme

Beispiel:

```
$ du /home/dv100
16   /home/dv100/app-defaults
120  /home/dv100/vilearn
128  /home/dv100/emacsfiles
1304 /home/dv100/graphics
1688 /home/
$ du -s /home/dv100
1688 /home/dv100
```

Spezifikation von Directories I

- Spezifikation eines Verzeichnisses durch:
  - Angabe eines absoluten Pfadnamens
    von der Wurzel aus alle Knoten (Directories) spezifizieren (1. Zeichen immer /)
    Beispiel: `/home/dv100/fotos/USA`
  - Angabe eines relativen Pfadnamens
    ausgehend von der aktuellen Position im Dateibaum den Pfad beschreiben
    (1. Zeichen **kein** /)
    Beispiel: `USA`
- Symbole mit besonderer Bedeutung:

.	aktuelles Arbeitsverzeichnis
..	übergeordnetes Vaterverzeichnis
~	eigenes Heimatverzeichnis
~user	Heimatverzeichnis des Benutzers *user*
$HOME	eigenes Heimatverzeichnis

Spezifikation von Directories II

Beispiele anhand der Struktur auf Folie 35, aktuelle Position: Verzeichnis **fotos**

Spezifikation von **fotos**:	`.`
	`/home/dv100/fotos`
	`$HOME/fotos`
	`~/fotos`

Spezifikation von **USA**:
(eine Stufe weiter)
	`USA`
	`./USA`
	`/home/dv100/fotos/USA`
	`$HOME/fotos/USA`
	`~/fotos/USA`

Spezifikation von **dv100**:
(eine Stufe zurück)
	`..`
	`/home/dv100`
	`$HOME`
	`~`

Spezifikation von **brief2.txt**:
(quer zu Benutzer dv200)
	`../../dv200/brief2.txt`
	`/home/dv200/brief2.txt`
	`~dv200/brief2.txt`

Aktuelles Directory listen und wechseln (pwd, cd)

pwd

- **p**rint **w**orking **d**irectory
- zeigt das aktuelle Directory mit absolutem Pfadnamen an

Beispiel: **pwd** ⇒`/home/dv100`

cd [*dirname* | -]

- **c**hange **d**irectory
- wechselt das aktuelle Directory
- die Angabe des Directories $dirname$ kann absolut oder relativ erfolgen
- Sonderfälle:

  ohne $dirname$ ⇒ Heimatverzeichnis

  Minus (-) ⇒ vorheriges Verzeichnis

Beispiel:
```
$ cd /home/dv200
$ pwd
/home/dv200
```

Übung zum Wechseln von Directories

Prüfen Sie jede Aktion mit dem Befehl `pwd` *nach.*

1. Positionieren Sie sich auf Ihr Heimatverzeichnis:

   `cd` *oder* `cd ~` *oder* `cd $HOME`

2. Gehen Sie in das Verzeichnis *dirlearn*:

   ```
   cd dirlearn
   ```

3. Gehen Sie wieder in Ihr Heimatverzeichnis:

   ```
   cd ..
   ```

4. Gehen Sie in das Verzeichnis */tmp* und springen Sie zwischen */tmp* und Ihrem Heimatverzeichnis hin und her:

   ```
   cd /tmp
   cd -
   cd -
   ```

5. Gehen Sie in das Heimatverzeichnis des Benutzers *unix*:

   ```
   cd ~unix
   ```

6. Gehen Sie wieder in Ihr Heimatverzeichnis:

   `cd` *oder* `cd ../dv`$_{nnnnnn}$ *oder* `cd $HOME`

Inhalt von Directories auflisten (ls)

ls [*options*] [*filenames* | *directories*]

- Akronym für **l**i**s**t; zeigt Informationen zu Dateien und Directories an
- ohne Parameter wird der Inhalt des aktuellen Directory gelistet
- Art und Umfang der Informationen werden durch Optionen gesteuert; wesentliche sind:
  - **-l** (long) erstellt eine ausführliche Liste; eine Zeile pro Datei
  - **-d** (directory) zeigt das spezifizierte Directory selbst an, nicht den Inhalt
  - **-a** (all) zeigt auch mit Punkt beginnende Dateien an, sogenannte versteckte Dateien
  - **-R** (recursiv) zeigt rekursiv den gesamten Teilbaum an
  - **-F** (format) markiert in der Liste Directories mit /, Binaries mit * und symbolische Links mit @
  - **-t** (time) sortiert Liste nach Änderungsdatum
  - **-i** (inode) zeigt auch die Inode-Nummer an
  - **-s** (size) zeigt Dateigröße in Blöcken an

Directory anlegen und löschen (mkdir, rmdir, rm)

mkdir [*options*] *dirname*

- **m**a**k**e **dir**ectory ⇒ legt ein neues Directory an
  Nicht auf allen Systemen: **-p** Option legt auch Zwischendirectories an (parents)

Beispiel:
```
mkdir test01
mkdir -p test02/test12
```

rmdir [*options*] *dirname*

- **r**e**m**ove **dir**ectory ⇒ löscht ein leeres Directory
  Nicht auf allen Systemen: **-p** Option löscht auch leere Vaterdirectories (parents)

Beispiel:
```
rmdir test01
rmdir -p test02/test12
```

rm -r *dirname*

- **r**e**m**ove **r**ecursive (Beschreibung des `rm` Kommandos: Folie 82)
- löscht auch nichtleere Directories

Benutzersicht von Directories

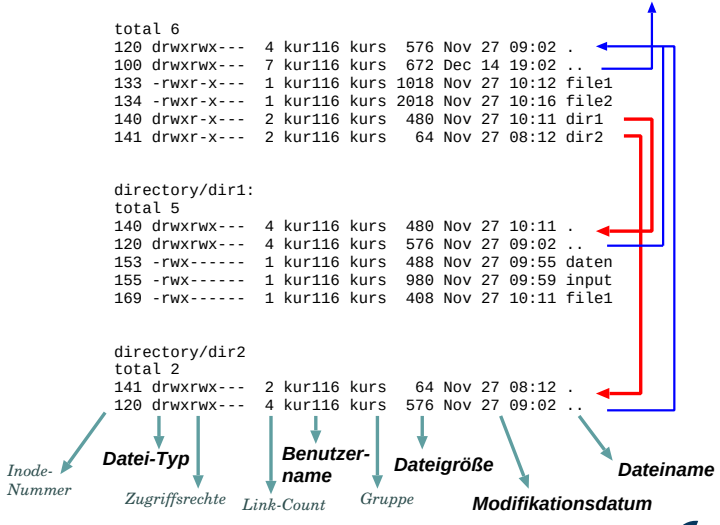
Ausgabeformat des ausführlichen ls Kommandos:

```
kur116@zam511: ls -Rali directory

total 6
120 drwxrwx---  4 kur116 kurs  576 Nov 27 09:02 .
100 drwxrwx---  7 kur116 kurs  672 Dec 14 19:02 ..
133 -rwxr-x---  1 kur116 kurs 1018 Nov 27 10:12 file1
134 -rwxr-x---  1 kur116 kurs 2018 Nov 27 10:16 file2
140 drwxr-x---  2 kur116 kurs  480 Nov 27 10:11 dir1
141 drwxr-x---  2 kur116 kurs   64 Nov 27 08:12 dir2


directory/dir1:
total 5
140 drwxrwx---  4 kur116 kurs  480 Nov 27 10:11 .
120 drwxrwx---  4 kur116 kurs  576 Nov 27 09:02 ..
153 -rwx------  1 kur116 kurs  488 Nov 27 09:55 daten
155 -rwx------  1 kur116 kurs  980 Nov 27 09:59 input
169 -rwx------  1 kur116 kurs  408 Nov 27 10:11 file1


directory/dir2
total 2
141 drwxrwx---  2 kur116 kurs   64 Nov 27 08:12 .
120 drwxrwx---  4 kur116 kurs  576 Nov 27 09:02 ..
```



Übung zum Auflisten von Directories

1. Welche Struktur hat der Teilbaum *dirlearn* in Ihrem Heimatverzeichnis? Fertigen Sie eine Skizze an.

   ```
   ls -R dirlearn
   ```

2. Aus wievielen Directories und Dateien besteht der Teilbaum?

   ```
   ls -lR dirlearn
   ```

   oder

   ```
   ls -FR dirlearn
   ```

   → 6 Directories

   → 7 Dateien

3. Welche Dateien gibt es namentlich mehrfach und wie oft?

   `a22` → `2x`

   `b12` → `2x`

4. Löschen Sie ein Unterverzeichnis des Teilbaumes.

   ```
   rm -r <directory>
   ```

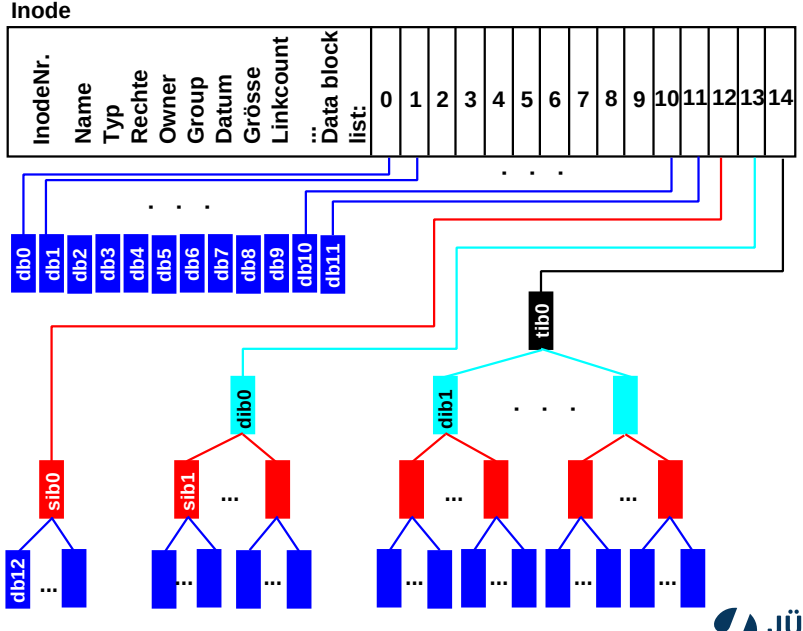
Standard UNIX Directories

/	Root Directory enthält i.allg. nur Unterverzeichnisse.
/etc	Enthält alle Dateien, die zur Systemadministration, Systemkonfiguration und Initialisierung wichtig sind.
/tmp	Wird für temporäre Dateien verwendet.
/usr	Enthält alle Betriebssystemdirectories.
/usr/lib /lib	Enthalten Libraries, die bei den Anwendungen benötigt werden.
/usr/bin /bin	Enthalten häufig benutzte UNIX Befehle sowie alle installationsspezifischen Befehle, die allen Benutzern zugänglich sein sollen.
/usr/local	Sammelpunkt für jegliche optionale Software, die nicht vom Hersteller aus in ein anderes Verzeichnis muSS.
/usr/spool /var/spool	Enthalten Spoolfiles, die z. B. zum Drucken oder Versenden bzw. Einlesen anstehen.
/usr/adm /var/adm	Enthalten Accounting- und Diagnose-Files. Interessant nur für den Systemverwalter.

Dateien (Files)

- Dateien sind eine Aneinanderreihung von Zeichen (Bytes) **ohne** interne Struktur
- die Zeichen werden **sequentiell** in vorformatierte Blöcke auf der Platte geschrieben (Formatierung mit 512, 1024 oder 2048 Bytes)
- eine Datei erhält vom System zur eindeutigen Kennung innerhalb eines Filesystems eine ***Inode***-Nummer (Index Node)
- eine Datei kann nicht größer werden als das Filesystem, in dem sie liegt
- die Dateinamen sind nur innerhalb eines Directory eindeutig
- eine Datei kann beliebig viele Dateinamen haben (Beschreibung des `link` Kommando auf Seite 68)
- die Anzahl der Dateien und der belegte Gesamtplatz können begrenzt (quotiert) sein

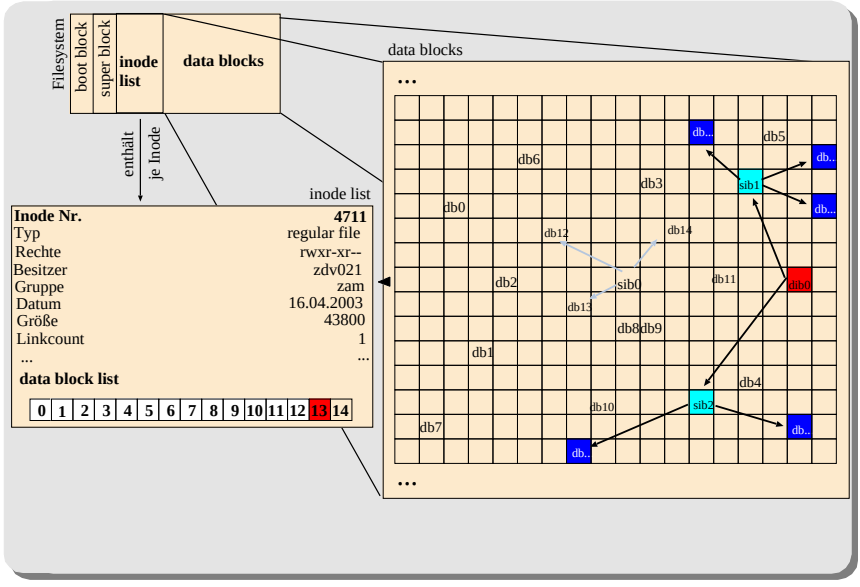
Inode-Struktur



Inode-Struktur

- die Data Block List im Inode soll möglichst kurz sein (13 - 15 Einträge)
- die ersten 10 - 12 sind **direct blocks** und zeigen direkt auf einen Datenblock (db$n$) auf der Platte
- vorvorletzter Block ist ein **single indirect block** und zeigt auf einen Block (sib$n$) mit Zeigern auf direct blocks
- vorletzter Block ist ein **double indirect block** und zeigt auf einen Block (dib$n$) mit Zeigern auf single indirect blocks
- letzter Block ist ein **triple indirect block** und zeigt auf einen Block (tib$n$) mit Zeigern auf double indirect blocks
- die max. Dateigröße und -adressierung ist abhängig von
  - der Blockgröße
  - der maximal darstellbaren Blockadresse

Beispiel für Dateigröße und Adressierung



Beispiel für Dateigröße und Adressierung

- Annahme für Adressierung:
  - Speicherplatz für Blockadresse sei 32 Bit k Integer (4 Bytes)
  - Blockgröße sei 1K (entspricht 1024 Bytes)
  - $\Rightarrow$ 256 Adressen pro Block
- Summierung:
  - 12 (direct)
  - 256 (single)
  - 65536 (double: 256 * 256)
  - 16777216 (triple: 256 * 256 * 256)
  - 16843020 K Blöcke (gesamt)
  - entspricht 16.8 GB

  Theoretisch könnte also ein Volumen von 16.8 GB addressiert werden.
- durch 32 Bit Integer ist aber die Dateigröße auf 2 GB begrenzt ($2^{31} - 1$), weil größere Blockadressen nicht darstellbar sind

Dateinamen I

- Dateinamen sind oft auf 256 Zeichen beschränkt
- zulässig sind im Prinzip alle Zeichen, **aber:**
  - **-** als erstes Zeichen eines Dateinamens vermeiden, da es Konflikte mit Kommando-Optionen geben kann
  - ***** und
  - **?** sollten in Dateinamen nicht vorkommen, da diese Zeichen als Metazeichen vom Kommandointerpreter verarbeitet werden
  - / dient als Trennzeichen zwischen Directory-Ebenen
- es wird zwischen Groß- und Kleinbuchstaben unterschieden
  (die Namen ab, Ab, aB, AB bezeichnen unterschiedliche Dateien)
- Dateinamen, die mit einem Punkt *'.'* beginnen, sind sogenannte versteckte Dateien und meist
  - Profiles zur Initialisierung
    (z.B. .profile, .kshrc, .login, .cshrc, .xinitrc, .Xdefaults)
  - oder Steuerdateien (z.B. .forward, .mailcap, .ssh/config)

Dateinamen II

- Dateinamen können durch Underscores '_' oder Punkte '.' gegliedert werden,

  z.B.:
  ```
  log_01jan94_to_31mar94
  abc.data
  abc.memo
  ```

- den letzten Teil nennt man Endung, Extension oder Suffix; er wird von Anwendungen (Desktop, Mailer) zur richtigen Darstellung genutzt
- Konventionen für Endungen (Auswahl):

.f .f90	FORTRAN Quellprogramme
.c	C Quellprogramme
.o	Object Code
.Z	Dateien mit compress komprimiert
.gz	Dateien mit gzip komprimiert
.gif .jpg .png	Bilddateien
.eps	Bilddateien zum Einziehen in Texte
.ps	Postscript Dateien
.pdf	Portable Documentation Format
.obj	Tgif Dateien (Graphiken)
.tex	Tex-Dateien (Textverarbeitung)
.html	Internet Seiten (WWW)

Dateinamen zerlegen (dirname, basename)

Zum Aufsplitten eines Dateinamens gibt es zwei Kommandos:

dirname ***filename***

- liefert den Pfad des Dateinamens

  Beispiel:
  ```
  $ dirname /home/dv200/test.f
  /home/dv200
  ```

basename ***filename*** **[*suffix*]**

- liefert den Dateinamen ohne Pfad; falls angegeben, wird Endung ***suffix*** ebenfalls weggelassen

  Beispiele:
  ```
  $ basename /home/dv200/test.f
  test.f
  $ basename /home/dv200/test.f .f
  test
  ```

Diese Kommandos werden hauptsächlich in der Programmierung von Shell Scripts verwendet; ein Beispiel hierzu ist auf Seite 81.

Dateien anlegen und ändern

- mittels Kommando:

  touch ***filename1*** **[[*filename2*] ...]**

  Existiert die Datei $filename_n$ nicht, wird eine leere Datei im aktuellen Directory angelegt. Andernfalls wird das Änderungsdatum aktualisiert.

  Beispiel: `touch abc bond.007`

- mittels I/O Redirection (→ Seite 128):

  command $>$ ***filename***
  command $>>$ ***filename***

  Beispiel: `echo "Text in data1" > data1`

- mittels eines Editors, z.B.:

vi	Visual Editor; gehört zum UNIX Betriebssystem, daher auf allen vorhanden
emacs	GNU Emacs; weltweit verbreiteter X-Window fähiger Public Domain Editor
kate	umfangreicher Text Editor

Übung zu Directories und Dateien

Führen Sie der Reihe nach folgende Befehle aus:

```
cd
mkdir dir1
cd dir1
touch file1 file2
cd ..
pwd
mkdir -p dir2/sdir1 dir2/sdir2
ls dir1
ls dir2
```

Skizzieren Sie die Directory-Struktur: $\rightarrow$

Löschen Sie die Directories wieder:

```
rmdir dir2/sdir1 dir2/sdir2
rmdir dir2
rmdir dir1
```

... wie weiter?

vi-Editor (Übersicht)

- Aufruf

  vi ***filename***

- Unterteilt in Command- und Insert-Mode
  - der Command-Mode wird über die <ESC> Taste erreicht
    (mächtige Umgebung zum Steuern des Editiervorgangs)
  - der Insert-Mode bzw. Replace-Mode wird durch die entsprechenden Kommandos erreicht
    (Umgebung für die sequentielle Dateneingabe)

Sichern und Beenden

:w[!] [*fn*]	Sichern der Daten in Datei fn (Write)
:wq [*fn*]	Sichern und Beenden (Write & Quit)
ZZ	Sichern und Beenden (Save & Exit)
:q[!]	Beenden ohne Sichern (Quit Always)

vi Tasten-Befehle

Ctrl-f	Blättere einen Bildschirm weiter (forward)
Ctrl-d	Blättere halben Bildschirm weiter (down)
Ctrl-b	Blättere einen Bildschirm zurück (backward)
Ctrl-u	Blättere halben Bildschirm zurück (up)
i	Füge vor Cursor-Position ein (Insert-Mode)
a	Füge hinter Cursor-Position ein (Insert-Mode)
I	Füge am Zeilenanfang ein (Insert-Mode)
A	Füge am Zeilenende ein (Insert-Mode)
o	Füge Zeile hinter Cursor-Zeile ein (Insert-Mode)
O	Füge Zeile vor Cursor-Zeile ein (Insert-Mode)
r	Ändere das Zeichen unter dem Cursor
R	Ändere Text ab Cursor-Position (Replace-Mode)
x	Lösche das Zeichen unter dem Cursor
D	Lösche Rest der Zeile
n**dd**	Lösche n Zeilen ab der Cursor-Zeile
J	Verkette zwei Zeilen
n**yy**	Kopiere n Zeilen ab Cursor-Zeile in Buffer
p	Kopiere Zeilen aus Buffer hinter Cursor-Zeile
P	Kopiere Zeilen aus Buffer vor Cursor-Zeile
u	Letzten Befehl rückgängig machen (Undo)
.	Letzten Befehl wiederholen

Übung zum vi-Editor

1. Editieren Sie die Datei vi.example in Ihrem Heimatverzeichnis: `vi vi.example`
2. Gehen Sie in den Input Mode:

   `i`

   Sie sind im Input Mode, sehen dies aber nicht. Beenden Sie den Input Mode mit:

   <ESC>
3. Geben Sie folgende vi-Optionen ein (Hinweis: Diese Befehle können auch in einen vi-Profile ~/.exrc eingetragen werden, siehe .exrc.smp.):

   ```
   :set showmode
   :set number
   ```
4. Gehen Sie wieder in den Input Mode:

   `i`

   Sie sind im Input Mode und sehen dies nun links unten. Beenden Sie den Input Mode mit:

   <ESC>
5. Folgen Sie nun den Anweisungen in der Datei.
6. Legen Sie eine Datei ~/.exrc an (siehe Punkt 3).

Zusätzliche Namen für Dateien (link)

- Eine Datei bzw. ein Directory kann beliebig viele Namen haben
- Ein Link dient zur Vergabe eines zusätzlichen Namens für eine Datei oder auch ein Directory
- Es gibt zwei unterschiedliche Formen:
  - Soft Links
  - Hard Links
- Links können
  - im gleichen Directory
  - an einer anderen Stelle im gleichen Filesystem
  - an beliebiger Stelle im Dateibaum (nur Soft Links)

  stehen

Soft Link (ln)

ln -s *orig_name link_name*

- ein **s**oft **lin**k, auch ***s**ymbolic **lin**k* genannt, ist ein Zeiger auf die Originaldatei
- geht auch für Directories
- hat einen eigenen Inode, der Dateityp ist *link*
- geht daher über Filesystem-Grenzen hinweg
- Änderungen sind unter beliebigem Namen möglich
- wird der Soft Link gelöscht, ist der Zeiger weg, nicht die Originaldatei
- wird die Originaldatei gelöscht, weist der Zeiger ins Leere
- wird wieder eine Datei mit dem Namen der Originaldatei angelegt, ist sie mit dem $link\_name$ ansprechbar
- **Bei relativen Pfadangaben immer vom Zieldirectory aus arbeiten!!**

Beispiel

`cd ~/Beispiel`	wechsel in das Verzeichnis
`ls -li`	liste Verzeichnis
`ln -s brief1 FZJ`	erstelle Link mit dem Namen FZJ auf brief1
`ls -li`	liste Verzeichnis
`cd ~/Beispiel/fotos`	wechsel in das Verzeichnis
`ls -Rli`	liste Verzeichnis
`ln -s USA Urlaub`	erstelle Link mit dem Namen ’Urlaub’ auf ’USA’
`ls -Rli`	liste Verzeichnis
`ls -li Urlaub/*`	liste Verzeichnis
`touch Urlaub/seattle`	erstelle Datei
`ls -Rli`	liste Verzeichnis

Beispiel

`cd ~/Beispiel`	wechsel in das Verzeichnis
`ls -Rli`	liste Verzeichnis
`ln -s fotos/auto.jpg Eigentum/PKW_bild`	
	1. Versuch: Link vom Vaterdirectory aus
`ls -Rli`	liste Verzeichnis $\Rightarrow$ Link nicht brauchbar
`rm Eigentum/*`	lösche alle Dateien in *Eigentum*
`cd fotos`	wechsel in das Verzeichnis *fotos*
`ln -s auto.jpg ../Eigentum/PKW_bild`	
	2. Versuch: Link vom Original Directory aus
`cd ..; ls -Rl`	wechsel in Vaterverzeichnis und liste es auf
	$\Rightarrow$ nicht brauchbar
`cd Eigentum; rm *`	wechsel ins Verzeichnis *Eigentum* und lösche alles
`ln -s ../fotos/auto.jpg PKW_bild`	
	3. Versuch: Link vom Zieldirectory aus
`cd ..;ls -Rli`	wechsel in Vaterverzeichnis und liste es auf
	$\Rightarrow$ ist OK

Hard Link (ln)

ln ***orig_name link_name***

- ein Hard **lin**k geht nur für Dateien, nicht für Directories
- der $link\_name$ ist eine Verknüpfung mit dem Inode der Originaldatei $orig\_name$
- daher nur innerhalb eines Filesystems (Eindeutigkeit der Inodes)
- Anzahl der Namen wird im Link-Count des Inode festgehalten
- Originaldatei und Hard-Links sind nicht unterscheidbar
- wird einer der Namen gelöscht, bleibt der Inhalt in den anderen Dateien erhalten, bis letzter Name gelöscht wird (Link-Count=0)
- Änderungen können unter beliebigem Namen erfolgen
- Ersetzen der Datei unter beliebigem Namen muß durch `mv` erfolgen, damit der Inode erhalten bleibt (nicht `cp` und `rm` !)
- ist $link\_name$ ein existierendes Directory, wird der Link mit dem Namen der Originaldatei unter das Directory gelegt (siehe Beispiel 2)

Hard Link (ln) - Beispiel 1

```
$ ls -li
679 -rw-r--r-- 1 zdv045 zam  5 Jun 18 18:04 prog.conf
$ ln prog.conf PROG.CONF
$ ls -li
679 -rw-r--r-- 2 zdv045 zam  5 Jun 18 18:04 PROG.CONF
679 -rw-r--r-- 2 zdv045 zam  5 Jun 18 18:04 prog.conf
$ ln PROG.CONF Prog.Conf
$ ls -li
679 -rw-r--r-- 3 zdv045 zam  5 Jun 18 18:04 PROG.CONF
679 -rw-r--r-- 3 zdv045 zam  5 Jun 18 18:04 Prog.Conf
679 -rw-r--r-- 3 zdv045 zam  5 Jun 18 18:04 prog.conf
$ echo test >> PROG.CONF
$ ls -li
679 -rw-r--r-- 3 zdv045 zam 10 Jun 18 18:07 PROG.CONF
679 -rw-r--r-- 3 zdv045 zam 10 Jun 18 18:07 Prog.Conf
679 -rw-r--r-- 3 zdv045 zam 10 Jun 18 18:07 prog.conf
$ rm Prog.Conf prog.conf
$ ls -li
679 -rw-r--r-- 1 zdv045 zam 10 Jun 18 18:07 PROG.CONF
```

Hard Link (ln) - Beispiel 2

```
$ ls -Rli
825 drwx------  2 zdv060 zam 512 Feb 1 15:29 dir1
935 drwx------  2 zdv060 zam 512 Feb 1 15:29 dir2
./dir1:
827 -rw-------  1 zdv060 zam   0 Feb 1 15:29 Dez
826 -rw-------  1 zdv060 zam  18 Feb 1 12:20 Nov
./dir2:

$ ln dir1/Dez dir2
$ ls -Rli
825 drwx------  2 zdv060 zam 512 Feb 1 15:29 dir1
935 drwx------  2 zdv060 zam 512 Feb 1 15:30 dir2
./dir1:
827 -rw-------  2 zdv060 zam   0 Feb 1 15:29 Dez
826 -rw-------  1 zdv060 zam  18 Feb 1 12:20 Nov
./dir2:
827 -rw------- 2 zdv060 zam 0 Feb 1 15:29 Dez
```

Aufgabenblock (1)

1. Wie heißt Ihr aktuelles Verzeichnis?
   Listen Sie seinen Inhalt auf. Probieren Sie verschiedene Optionen von `ls` aus.
   (Tipp: `man ls`)
2. Verzweigen Sie in das Verzeichnis */usr/local/unixkurs* und vergleichen Sie den Inhalt der Verzeichnisse *aufg.1a* und *aufg.1b*. Was für ein Unterschied besteht zwischen den Dateien in diesen Verzeichnissen?
3. Verzweigen Sie wieder in Ihr Heimatverzeichnis.
   Legen Sie dort ein Verzeichnis *testdir1* an.
   Legen Sie dann darunter mit `vi` die Datei *abc* an mit dem Inhalt
   ```
   Dies ist eine Testdatei
   ```
   Löschen Sie anschließend das Verzeichnis *testdir1*.
4. Wieso haben Directories immer einen *link-count* ≥ 2?

Shell-Expansion von Dateinamen

- Da die Shell die Kommandoeingabe interpretiert, ist eine kompaktere Schreibweise von Dateinamen mit Hilfe von Wildcard-Zeichen möglich:

*****	beliebig viele Zeichen, inklusive dem leeren String
?	genau ein beliebiges Zeichen
[...]	jedes der in [ ] eingeschlossenen Zeichen; Bereiche erlaubt, z.B. [a-z]
[!...]	jedes nicht in [ ] eingeschlossene Zeichen; Bereiche erlaubt (gilt nicht in jeder Shell)

- Mit \ versteckt man das folgende Zeichen vor der Interpretation durch die Shell, d.h. das Zeichen bleibt als solches bestehen
- Jede Übereinstimmung wird in die Parameterliste aufgenommen
  (bei manchen Systemen max. Parameterlänge: 2048)
- Einige Anwendungen machen die Auflösung selbst; dann ist die Maske für die Dateinamen in einfache Hochkommata einzuschließen

Übung zur Expansion von Dateinamen

Die Dateinamen seien:

```
abc1  abc2  abc3  ab1  ac?  bd1  bc2
```

Maske	**Ergebnis**
*	abc1 abc2 abc3 ab1 ac? bd1 bc2
a*1	abc1 ab1
a?1	ab1
***[c-f]1**	abc1 bd1
***?**	abc1 abc2 abc3 ab1 ac? bd1 bc2
***\?**	ac?
d*	d* (falls kein Dateiname der Maske entspricht, findet keine Erweiterung statt)

1. Vollziehen Sie dies nach:

```
cd
mkdir testdir2
cd testdir2
touch abc1 abc2 abc3 ab1 ac\? bd1 bc2
```

   Veranschaulichen Sie mit `echo` ... die Expansion.

2. Wie zeigt man alle .<**Files**> im Heimatverzeichnis auf?

Kopieren von Dateien (cp)

cp [*options*] *filename copyname*

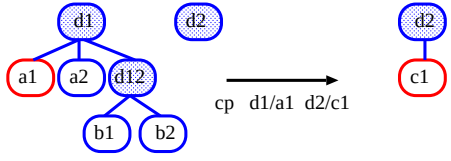
- Akronym für **co**p**y**
- legt Kopien von ein oder mehreren Dateien an:
  filename ist der Name der zu kopierenden Datei
  copyname ist der Name der Kopie
- ist *copyname* ein Directory, können mehrere Dateien in dieses Directory unter Beibehaltung ihres Namens kopiert werden
- existiert *copyname* bereits, wird die Datei ***ohne Information des Benutzers überschrieben***, wenn nicht die "**-i**" Option spezifiziert wurde
- wichtige Optionen:
  -p die neue Datei erhält Zugriffsrechte und Modifikationsdatum der alten Datei (sonst: alte Zugriffsrechte verknüpft mit *umask* (s.u.))
  -r falls *filename* ein Directory ist, werden rekursiv alle Dateien und Unterverzeichnisse kopiert

Kopieren von Dateien (cp) - Beispiel 1

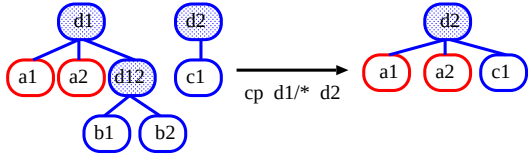
```
# 1. Kopieren einer einfachen Datei
$ ls -li
473 -rw-r--r-- 1 zdv045 zam 8 Jul 19 10:34 telefon
$ cp telefon tel.1; ls -li
474 -rw-r--r-- 1 zdv045 zam 8 Jul 19 10:38 tel.1
473 -rw-r--r-- 1 zdv045 zam 8 Jul 19 10:34 telefon
$ cp -p telefon tel.2; ls -li
474 -rw-r--r-- 1 zdv045 zam 8 Jul 19 10:38 tel.1
475 -rw-r--r-- 1 zdv045 zam 8 Jul 19 10:34 tel.2
473 -rw-r--r-- 1 zdv045 zam 8 Jul 19 10:34 telefon
# 2. Kopieren eines Directory
$ ls -lRi
73 drwxr-sr-x 2 zdv045 zam 512 Jul 19 10:45 adresse
./adresse:
75 -rw-r--r-- 1 zdv045 zam   5 Jul 19 10:45 anna
74 -rw-r--r-- 1 zdv045 zam  10 Jul 19 10:45 otto
$ cp -pr adresse adr.old; ls -lRi
76 drwxr-sr-x 2 zdv045 zam 512 Jul 19 10:45 adr.old
73 drwxr-sr-x 2 zdv045 zam 512 Jul 19 10:45 adresse
./adr.old:
78 -rw-r--r-- 1 zdv045 zam   5 Jul 19 10:45 anna
77 -rw-r--r-- 1 zdv045 zam  10 Jul 19 10:45 otto
./adresse:
75 -rw-r--r-- 1 zdv045 zam   5 Jul 19 10:45 anna
74 -rw-r--r-- 1 zdv045 zam  10 Jul 19 10:45 otto
```

Kopieren von Dateien (cp) - Beispiel 2

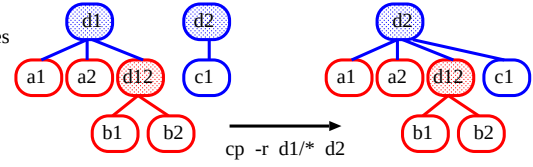
Kopieren eines Files
in gleiches oder anderes Directory



Kopieren mehrerer Files
in anderes Directory

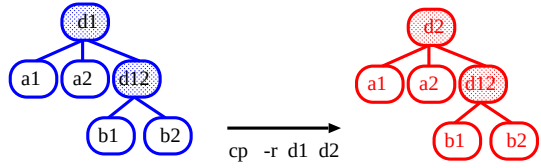


Kopieren von Files incl. Subdirectories
in anderes Directory

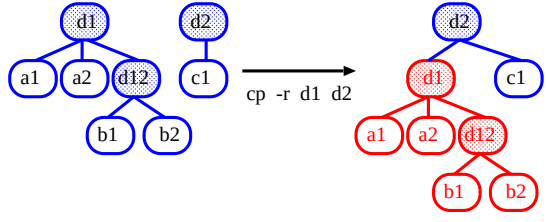


Kopieren von Dateien (cp) - Beispiel 3

Kopieren eines Directory:
Zieldirectory existiert nicht



Kopieren eines Directory:
Zieldirectory existiert bereits



Umbenennen von Dateien (mv)

mv [*options*] *filename newname*

- Akronym für **m**o**v**e
- ein `mv` innerhalb eines Filesystems bedeutet lediglich ein Umbenennen von Directories oder Dateien; es werden keine Daten bewegt, der Inode bleibt erhalten
- zwischen unterschiedlichen Filesystemen verhält sich `mv` wie `cp` mit anschließendem `rm`
- falls *newname* ein Directory ist, können mehrere Dateien logisch von einem Directory in ein anderes verlagert werden; dabei ist allerdings keine Namensänderung möglich
- die Zugriffsrechte bleiben erhalten
- *filename* darf auch ein Directory sein; *newname* darf dann nicht existieren oder muß ein Directory sein
- existiert *newname*, so geht der alte Inhalt ohne Benutzerinformation verloren, wenn nicht **-i** Option spezifiziert wurde

Umbenennen von Dateien (mv) - Beispiel

```
# 1. Umbenennen einer einfachen Datei
$ ls -li
437 -rw-r--r-- 1 zdv045  zam    5 Jul 18 18:20 abc
$ mv abc xyz
$ ls -li
437 -rw-r--r-- 1 zdv045  zam    5 Jul 18 18:20 xyz
# 2. Umbenennen eines Directory
$ ls -lR
drwxr-sr-x 2 zdv045  zam  512 Jul 19 10:22 wohnung
./wohnung:
total 16
-rw-r--r-- 1 zdv045  zam    6 Jul 19 10:18 miete
-rw-r--r-- 1 zdv045  zam    7 Jul 19 10:19 strom
$ mv wohnung haus
$ ls -lR
drwxr-sr-x 2 zdv045  zam  512 Jul 19 10:22 haus
./haus:
total 16
-rw-r--r-- 1 zdv045  zam    6 Jul 19 10:18 miete
-rw-r--r-- 1 zdv045  zam    7 Jul 19 10:19 strom
```

Übung zu cp und mv

```
cd
mkdir kurs
cp /usr/local/unixkurs/fortran/submm.f kurs
cd kurs
ls -lisa

mv submm.f abc.f
ls -lisa
cp abc.f neu.f
ls -lisa
mv -i abc.f neu.f

mkdir testdir1
mv testdir1 testdir2
cp neu.f testdir2
ls -Rlisa
mv testdir2 testdir1
ls -Rlisa
```

Ausführliches Beispiel zu mv

Es sollen alle *.f77 Dateien in *.f90 umbenannt werden.
Der Befehl `mv *.f77 *.f90` geht nicht (Fehlermeldung).

```
# Liste der existierenden Dateien:
$ ls *.f77 *.f90
a.f77 b.f77 c.f77
# ksh-Programmschleife:
$ for F in `ls *.f77`
> do
>   mv $F `basename $F .f77`.f90
> done

# Liste der Dateien nach Umbenennung:
$ ls *.f77 *.f90
a.f90 b.f90 c.f90
```

Löschen von Dateien (rm)

rm [*options*] *filename*

- Akronym für **re**m**o**ve; es werden ein oder mehrere Dateien bzw. Directories gelöscht, die durch $filename$ spezifiziert werden
- wichtige Optionen:
  - **-i** Vor dem Löschen wird nach einer Bestätigung gefragt
  - **-r** Ist $filename$ ein Directory, wird rekursiv der gesamte Teilbaum gelöscht inklusive Subdirectories
  - **-f** Erzwingt das Löschen von Dateien, obwohl Zugriffsrechte es untersagen (überschreibt **-i**)
- ***das Löschen ist nur von den Zugriffsrechten des übergeordneten Directory abhängig (write Recht), nicht vom Recht der Datei selbst***
- verfügt die Datei über zwei oder mehrere Namen (link), wird nur der entsprechende Directory-Eintrag weggenommen; die Daten bleiben unter den anderen Namen verfügbar (→ Hard / Soft Link auf Seite 68)

Löschen von Dateien (rm) - Beispiel

```
# 1. Loeschen von Dateien
$ ls -l
-rw-r--r-- 1 zdv045  zam  4 Jul 19 14:58 abc
-rw-r--r-- 1 zdv045  zam  6 Jul 19 14:59 abc.1
-rw-r--r-- 1 zdv045  zam  3 Jul 19 14:58 elf

$ rm elf; ls -l
-rw-r--r-- 1 zdv045  zam  4 Jul 19 14:58 abc
-rw-r--r-- 1 zdv045  zam  6 Jul 19 14:59 abc.1

# 2. Loeschen von Directories und Dateien
$ ls -lR
drwxr-sr-x 3 zdv045  zam 512 Jul 19 15:10 haus
./haus:
drwxr-sr-x 2 zdv045  zam 512 Jul 19 15:13 wand
./haus/wand:
-rw-r--r-- 1 zdv045  zam   8 Jul 19 15:13 fenster
-rw-r--r-- 1 zdv045  zam   7 Jul 19 15:13 ziegel

$ rm -r ha*; ls -lR
total 0
```

VORSICHT! Ein Blank vor dem * hat verheerende Folgen. Der gesamte Teilbaum ab dem aktuellen Directory wird gelöscht! Besonders peinlich, wenn man auf / steht und root-Rechte hat.

Übung zu rm

Vollziehen Sie folgende Befehle nach. Überprüfen Sie mit `pwd` oder `ls -lisa`, ob der Befehl den erwarteten Effekt erzielt hat.

```
cd
cd kurs
ls -lisa
rm testdir1
rm testdir1/*
ls -lisa
cd ..
ls -lisa kurs
rm -ri kurs
```

Vergleich von Dateien (diff)

diff [*options*] *file_1 file_2*

- vergleicht textuell und zeigt an, welche Zeilen unterschiedlich sind
  (nicht bei Binär-Dateien)
- Synchronisation beim Vergleich bzgl. Leerzeilen erfolgt automatisch
- wichtige Optionen:
  -b Anzahl der Blanks spielen keine Rolle
  -i ignoriert Groß-/Kleinschreibung
- der Return Code (**$?**) ist:
  0 bei Gleichheit
  1 bei Ungleichheit
  >1 bei Fehler
- Alternativ gibt es den **cmp** Befehl, der bei Binär-Daten und Script-Programmierung Sinn macht, wenn nur das Ergebnis gleich oder ungleich von Bedeutung ist, nicht die Unterschiede selbst. Der Befehl ist evtl. schneller, da er bei erster Ungleichheit abbricht.

Vergleich von Dateien (diff) – Beispiel

```
$ cat Datei1
Hallo,
dies ist der File Datei1.

$ cat Datei2
Hallo,
dies ist der File Datei2.

$ diff Datei1 Datei2
2c2
< dies ist der File Datei1.
---
> dies ist der File Datei2.
```

(wobei $<$ für die 1. Datei, $>$ für die 2. Datei steht und 2c2 bedeutet: ersetze (c) Zeile 2 in 1. Datei durch Zeile 2 aus 2. Datei, um aus 1. Datei die 2. Datei zu erhalten)

Komprimieren von Dateien (gzip, gunzip)

Aus der GNU-Suite sind oft die entsprechenden Kompressionsalgorithmen vorhanden:

gzip [*options*] *filename*

- komprimiert Dateien mit einer Variante des Lempel-Ziv Algorithmus
- abhängig vom Input 50 - 60 % Reduktion; die **-v** Option zeigt die Reduktionsrate an
- Endung ist **.gz**

gunzip [*options*] *filename*

- erkennt und verarbeitet `gzip` (**.gz**,**.tgz**), `pack` (**.z**) und `compress` (**.Z**)

Beispiel:

```
$ ls -l
-rw-r--r-- 1 zdv045 zam 425266 Sep 13 15:58 host
$ gzip host; ls -l
-rw-r--r-- 1 zdv045 zam 84235 Sep 13 16:00 host.gz
```

Aufgabenblock (2)

1. Verzweigen Sie in das Verzeichnis:
   /usr/local/unixkurs/aufg.2/dir1
   Kopieren Sie den gesamten Inhalt in Ihr Heimatverzeichnis unter: *ueb2/dir1*
2. Löschen Sie in *~/ueb2/dir1* alle Dateien, deren Namen aus 3 Zeichen bestehen.
3. Alle Dateien, die unter *~/ueb2/dir1* stehen, sollen unter gleichem Namen nur noch in *~/ueb2/dir2* stehen.
4. Welche Filesysteme sind auf Ihrem System konfiguriert?
5. Wieviel Plattenplatz haben Sie in Ihrem Heimatverzeichnis verbraucht?

Inhalt

Zugriffsrechte

- die Zugangskontrolle erfolgt über zugeordnete Zugriffsrechte
- Files und Directories besitzen diese Attribute
- Alias Namen (links) haben die gleichen Rechte wie das Original
- unabhängige Vergabe der Rechte an:

Eigentümer (user)		Gruppe (group)		Rest der Welt (others)	
Leseberechtigung	r	Leseberechtigung	r	Leseberechtigung	r
Schreibberechtigung	w	Schreibberechtigung	w	Schreibberechtigung	w
Ausführberechtigung	x	Ausführberechtigung	x	Ausführberechtigung	x

- kleinste Zugehörigkeit des Benutzers wird zur Prüfung genutzt
  (user $<$ group $<$ others)
- Vergabe von Rechten an Einzelpersonen ist nur über Access Control Lists (**ACL**) möglich (Implementierung und Aufruf ist systemabhängig)

Zugriffsrechte bei Dateien

r Die Datei darf gelesen werden.

w Die Datei darf beschrieben/geändert werden.

x Die Datei darf ausgeführt werden.

Normalerweise nur für compilierte Programme und Shell Scripts (Prozeduren).

Sonderfall (anstelle von x):

s Die Datei nimmt zur Ausführungszeit die Rechte des Eigentümers (set-uid bit) bzw. der Gruppe (set-gid bit) an.

Beispiel: `passwd`

```
-rwsr-xr-x 1 root ... /usr/bin/passwd
```

Zum Ändern des Passworts muß eine Datei gelesen und geschrieben werden, die nur root ändern darf:

Linux: `-rw-r----- root.../etc/shadow`

AIX: `-rw------- root.../etc/security/passwd`

Daher wird der Benutzer während des Ausführung des Befehls auf Grund des set-uid bit 's' kurzzeitig root.

Zugriffsrechte bei Directories

r Das Directory darf gelesen werden. Notwendig zum Auflisten des Directory-Inhalts.

w In das Directory darf geschrieben werden. Notwendig zum Anlegen und **Löschen** einer Datei innerhalb des Directories. Das Löschen ist im allgemeinen unabhängig von den Zugriffsrechten der Datei selbst möglich.

x In das Directory darf verzweigt werden. Es berechtigt nicht zum Auflisten des Directory.

Sonderfall (anstelle von x):

t In Verzeichnissen mit diesem Attribut kann nur der Eigentümer einer Datei diese löschen (sticky bit).

Beispiel: `/tmp` Directory

Niemals Schreibzugriff auf Directories an andere vergeben!

Darstellung der Zugriffsrechte

- Die Zugriffsrechte werden in Tripeln codiert.
- Es gibt einen Übergang von der symbolischen zur binären Schreibweise und letztlich zur Darstellung als Oktalzahl.
- Eine 1 bedeutet, daß die der Position entsprechende Berechtigung gesetzt ist.

symbolisch	binär	oktal
`- - -`	`000`	`0`
`- - x`	`001`	`1`
`- w -`	`010`	`2`
`- w x`	`011`	`3`
`r - -`	`100`	`4`
`r - x`	`101`	`5`
`r w -`	`110`	`6`
`r w x`	`111`	`7`

- Angewandt auf jede Klasse (user, group, others) ergibt dies 3-stellige Oktalzahlen von 000 - 777.
- Für die Sonderfälle wird eine führende Stelle dazugenommen
  (4xxx = set-uid, 2xxx = set-gid, 1xxx = sticky).

Default Zugriffsrechte beim Anlegen (umask)

- Die UNIX Standardberechtigung ist:
  - ***666*** bei normalen Dateien
  - ***777*** bei ausführbaren Dateien und Directories
  - → 'group' und 'others' haben **alle** Rechte!

umask [*nnn*]

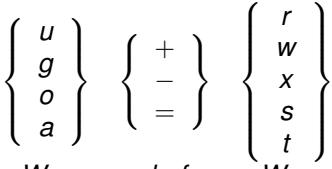
- Das Ausblenden nicht gewünschter Rechte geschieht über **`umask`** - eine binär codierte Maske, die Positionen der Zugriffsrechte der Standardberechtigung **löscht**:
  - **`umask`** Position = 0 → übernehme aus Standard
  - **`umask`** Position = 1 → lösche Berechtigung
- Default Zugriffsrechte sind die Kombination von Standardberechtigung und `umask`.

  Beispiel:
  ```
  standard: 666 = 110110110 = rw-rw-rw-
  umask:    077 = 000111111
  default:  600 = 110000000 = rw-------
  ```
- `umask` kann individuell gesetzt werden.
  Ohne Parameter wird der aktuelle Wert gelistet.

Zugriffsrechte ändern (chmod)

chmod [*options*] *mode filename*

- Zugriffsrechte können nur vom Eigentümer oder Superuser (root) geändert werden
- Die **-R** Option bewirkt eine rekursive Ausführung
- ***mode*** kann als Oktalzahl oder in einer symbolischen Form angegeben werden:



Wer: u → user, g → group, o → others, a → all
darf: + → zufügen, – → wegnehmen , = → soll sein
Was: r → read, w → write, x → execute, s → Set uid/gid, t → Sticky bit

Beispiele:

1) `chmod u=x,g=x,o= game`
2) `chmod 110 game` *(entspricht Beispiel 1)*
3) `chmod ug+x game`
4) `chmod g-r,a+x game`

Zugriffsrechte ändern (chmod)

- Symbolische Notation wird verwendet, wenn relative Rechte vergeben werden sollen.
- Oktalzahlen verwendet man meist, wenn Zugriffsrechte einen definierten Status erhalten sollen.

Beispiele:

```
# 1. Datei hat keine Rechte
$ ls -l
----------  1 demo   190 Mar 26 08:52 file1

# 2. Explizites Setzen von Rechten
$ chmod 711 file1; ls -l
-rwx--x--x  1 demo   190 Mar 26 08:52 file1

# 3. Relatives Zufuegen eines Rechtes
$ chmod  g+r  file1; ls -l
-rwxr-x--x  1 demo   190 Mar 26 08:52 file1

# 4. Setzen des Sticky bits fuer Directory
$ mkdir testdir; chmod 1777 testdir; ls -l
-rwxr-x--x  1 demo   190 Mar 26 08:52 file1
drwxrwxrwt  2 demo   512 Mar 26 08:52 testdir
```

Aufgabenblock (3)

1. Legen Sie mit `touch` eine Datei *a1* an. Welche Zugriffsrechte hat diese? (Begründung!).
2. Wie können Sie eine Datei *a2* anlegen, die die Zugriffsrechte
   `r w - - - - - - - - -` hat? (drei Möglichkeiten)
3. Geben Sie Ihrer Gruppe *Schreib-* und *Leserecht* auf *a2*.
4. Setzen Sie Ihre `umask` auf 077. Kopieren Sie jetzt *a2* auf *a3*. Welche Zugriffsrechte hat *a3* (Begründung)? Was hätte man machen können, um die Zugriffsrechte nicht zu ändern?
5. Legen Sie ein Verzeichnis *testdir* an. Bewegen Sie die Datei *a2* mit `mv` in dieses Verzeichnis. Nehmen Sie alle Zugriffsrechte der Datei weg und löschen Sie die Datei. Begründen Sie das Resultat.
6. Welche Zugriffsrechte hat */tmp* (Begründung)?
7. Verbieten Sie der Welt, Ihr Heimatverzeichnis aufzulisten.

Inhalt

Prozesse allgemein

- Ein Prozeß ist die kleinste ausführbare Einheit.
- Jeder Prozeß besitzt einen eigenen Adreßraum.
- Jeder Prozeß hat eine eindeutige Nummer, die **Process Identification** (PID).
- Jeder Prozeß hat einen eindeutigen Vaterprozeß (Parent PID = PPID).
- Der Ahnvater aller Prozesse hat die Nummer **1**.

Beispiel:

```
UID        PID   PPID   STIME   TIME CMD
root         1      0   Jul 19  0:11 /etc/init
zdv045   30458      1   Jul 19  0:36 dtterm
zdv045   22456  30458   Jul 19  0:01 /bin/ksh
zdv045   42476  22456 16:49:11  0:00 ps -ef
```

Prozesse allgemein

- Jedem Prozeß werden zwei Benutzernummern zugeordnet:
  real userid identifiziert den Benutzer, unter dem der Prozeß läuft
  effective userid wird zum Überprüfen jeglicher Zugriffsrechte verwendet
- Im Normalfall sind diese identisch; Dateien mit dem Setuid Bit (**`rws`**...) setzen die *effective userid* auf die des Eigentümers.
  Beispiel:

```
  PID    RUSER     USER      COMMAND
38188    zdv045    zdv045    /bin/ksh
31548    zdv045    zdv045    cp dump /tmp/dump
41764    zdv045    root      passwd
44738    zdv045    zdv045    ps -o %p%u%U%a
```

- Prozesse laufen im Vorder- oder Hintergrund (& Parameter).

Prozeßumgebung

- Es gibt spezielle Filedescriptoren, die für **jeden** Prozeß automatisch angelegt werden:

0	stdin	Standardeingabe
1	stdout	Standardausgabe
2	stderr	Standardfehlerausgabe

- Die normalen Einstellungen sind:

  0 die Tastatur

  1,2 der Bildschirm bzw. das Fenster auf dem Bildschirm

- Jeder Prozeß verwaltet eigene Dateipositionen

  → gleichzeitig ablaufende Prozesse können die Ausgabe auf stdout vermengen

  → kaum noch lesbarer Bildschirm

- Das Ende eines Prozesses kann sein:

  normal (exit status 0)

  fehlerhaft (exit status n)

  Abbruch (exit status 128 + signal code)

Der Exit Status kann durch eine Shell Variable **$?** abgefragt werden.

Auflisten und Abbrechen von Prozessen (ps, kill)

ps [*options*]

- listet die aktiven Prozesse auf
- ohne Optionen werden nur eigene Prozesse in Kurzform aufgelistet (PID, TTY, TIME, CMD)
- wichtige Optionen:
  - **-e** listet jeden Prozeß im System
  - **-f** gibt eine ausführliche Form aus
  - **-u** listet Prozesse des Benutzers uid

kill [*signal*] *pid*

- schickt dem Prozeß mit der Prozeßnummer pid das angegebene Signal
- es dürfen nur Signale zu eigenen Prozessen verschickt werden
- standardmäßig wird Signal **15** (Terminate) verschickt; der Prozeß kann noch eine Art Checkpointing durchführen, bevor er endet
- **kill -9** ist nicht abfangbar und bewirkt einen Abbruch des Prozesses, ohne dass Daten oder Zustände gesichert werden

Beispiel & Übung zu ps und kill

Beispiel zu ps und kill:

```
$ ps
  PID TTY      TIME CMD
31548 pts/40   0:15 a.out
38188 pts/40   0:00 /bin/ksh
40080 pts/40   0:00 ps
41764 pts/40   0:56 /usr/local/bin/firefox
$ kill 31548
$ ps
  PID TTY      TIME CMD
38188 pts/40   0:00 /bin/ksh
40090 pts/40   0:00 ps
41764 pts/40   0:56 /usr/local/bin/firefox
```

Übung zu ps und kill

Führen Sie folgende Befehle aus und überprüfen Sie den Effekt jeweils mit `ps`:

`xterm &`
`ps`
`kill` ***pid*** *von xterm*
`ps`

Prozeßverwaltung (jobs, bg, fg)

jobs [*options*]

- listet die aktiven Jobs auf, die unter job Kontrolle laufen (%jobid, status, command)
  Beispiel:
  ```
  $ xterm &
  $ jobs
  [2]+ Running   xterm
  ```
- **-l** Option listet zusätzlich die pid

bg [*pid*] [*%jobid*]

- Akronym für **b**ack**g**round
- schickt den Prozeß, der zuvor mit <**Ctrl**>**-z** gestoppt wurde, zur weiteren Ausführung in den Hintergrund (ShellPrompt → Befehlseingabe wieder möglich)

fg [*pid*] [*%jobid*]

- Akronym für **f**ore**g**round
- läßt den angegebenen Prozeß, der zuvor mit <**Ctrl**>**-z** gestoppt wurde, im Vordergrund weiterlaufen

Übung zu bg, fg und jobs

```
xterm
```

```
<Ctrl>-z
```

... 1. Versuch im neuen xterm Befehle einzugeben

```
bg
```

... 2. Versuch im neuen xterm Befehle einzugeben

```
date
```

... wieder im rufenden xterm

```
jobs
```

```
fg %1
```

... neuen xterm beenden

Automatisches Ausführen von Kommandos

UNIX Systeme besitzen einen System-Prozeß (daemon) namens ***cron***, der zu benutzerorientiert bestimmten Zeitpunkten Kommandos ausführt. Dem Benutzer stehen drei Möglichkeiten zur Verfügung, den Zeitpunkt und die Kommandos zu spezifizieren:

at zum einmaligen Starten zu einem bestimmten Zeitpunkt:

at [*-f file*] *time* [*date*]

batch zum einmaligen Starten irgendwann abhängig von der Systemlast:

batch [*file*]

crontab zum regelmäßigen Starten; minimal kann ein einminütiges Wiederholungsintervall angegeben werden

Die Ausgabe des Kommandos wird per `mail` zugeschickt ($userid@hostname$), kann bzw. sollte aber bei Bedarf auf Dateien umgelenkt werden.

Verwaltung von Jobs (at, atq, atrm, crontab)

```
at  -l
atq
```

listet sowohl mit `at` als auch mit `batch` abgeschickte Jobs in der Warteschlange

```
at  -r  jobid          (nicht Linux)
atrm  jobid
```

löscht einen Job aus der Warteschlange

```
crontab  -l
crontab  -e
crontab  -r
```

- **-e** lädt die crontab-Datei in den vi-Editor zum Zufügen, Löschen oder Ändern von Einträgen (für emacs-Nutzer: `export VISUAL=emacs`)
- **-r** löscht die crontab-Datei

Verwaltung von Jobs (at, atq, atrm, crontab)

-l listet die crontab Einträge in der Form:

min	Minute (0 - 59, *)
hour	Stunde (0 - 23, *)
mday	Tag des Monats (1 - 31, *)
month	Monat (1 - 12, *)
wday	Tag der Woche (0 - 6, 0=Sonntag, *)
cmd	Kommando

Beispiele:

```
# 1. Startet am naechsten Freitag um 20:00 Uhr
#    die Prozedur namens Shell_Script
$ at -f $HOME/Shell_Script 8 pm Friday

# 2. Jeden Sonntag (0) vor Mitternacht (23:59)
#    wird im Unterverzeichnis group_work
#    des Benutzers ($HOME) fuer alle Dateien das
#    Lese- und Execute-Recht gesetzt (chmod).
$ crontab -l
59 23 * * 0 chmod -R g+rx $HOME/group_work
```

Übung zu cron

Erstellen Sie einen Job, mit dem Sie an eine Datei *DATUM* minütlich das aktuelle Datum angehängen. Die Datei *DATUM* soll in Ihrem Heimatverzeichnis liegen.
Der Job soll ab jetzt bis zur nächsten vollen Stunde laufen.
Tipp: Benutzen Sie einen crontab-Eintrag der Form:

```
* hh DD MM * command >> filename
```

Vergewissern Sie sich, dass Ihr Eintrag läuft, indem Sie den Inhalt der Datei prüfen.
Löschen Sie dann Ihren crontab-Eintrag wieder.

Inhalt

Allgemeine Befehle

passwd	Ändert das Benutzer-Password
id	Listet aktuelle User- und Group-Id
groups	Listet die zugeordneten Gruppen
env	Listet die exportierten Variablen; kann auch verwendet werden, um Prozesse mit genau dem angegebenen Environment zu starten
hostname	Listet den Namen des Rechners
script [*fn*]	Protokolliert Terminalsession in einer Datei fn (Beenden mit `exit`, Defaultdatei *typescript*)
alias [...]	Listet bzw. definiert Kürzel für längere Kommandos (Beispiel: `alias ll='ls -lisa'`)
echo *text*	Gibt den angegebenen Text $text$ und/oder Inhalt der Shell-Variablen $\$VAR$ auf dem Bildschirm aus

Dateien seitenweise ausgeben (more)

more [*filename*]

- gibt den Inhalt einer Datei seitenweise aus
- fehlt die Dateiangabe, wird die Eingabe über Ausgabeumlenkung (Seite 128) oder Pipe (Seite 133) erwartet
- Navigation in der more-Umgebung:

<**Enter**>	blättert eine Zeile weiter
<**Space-Bar**>	blättert eine Seite weiter
b	blättert eine Seite zurück (back)
q	bricht die Ausgabe ab (quit)
/***string***	sucht nach Zeichenketten

- mit **v** schaltet man auf den vi-Editor um (mit ":q!“ verläßt man diesen) bzw. auf den Editor, der mit **`VISUAL`** angegeben ist

Dateien ausgeben bzw. verketten (cat)

cat [*options*] [*filename1* [...]]

- Akronym für con**cat**enate
- gibt den Inhalt von ein oder mehreren Dateien in der angegebenen Reihenfolge aus
- fehlt die Dateiangabe, wird die Eingabe über Pipe (Seite 133) oder die Tastatur erwartet
- durch Ausgabeumlenkung (Seite 128) können Dateien kopiert und zusammengefügt werden
- es werden jegliche Arten von Dateien unterstützt

Beispiele:

```
# 1. Inhalt einer Datei ausgeben
$ cat zeile_1
Die Lerche in die Luefte steigt,
# 2. Inhalt von 2 Dateien ausgeben
$ cat zeile_1 zeile_2
Die Lerche in die Luefte steigt,
Der Loewe bruellt, wenn er nicht schweigt.
# 3. Inhalt von 2 Dateien verketten und
#    auf 3. Datei ausgeben
$ cat zeile_1 zeile_2 > reim
```

Dateien umgekehrt ausgeben/verketten (tac)

tac [*options*] [*filename1* [...]]

- gibt den Inhalt von ein oder mehreren Dateien in der angegebenen Reihenfolge aus **beginnend mit der letzten Zeile jeder Datei**
- Pipe (Seite 133) und Ausgabeumlenkung (Seite 128) wie bei `cat`
- hilfreich bei Log-Dateien, zuerst die aktuellen Einträge

Beispiel:

```
$ tac alert.log | more
Mon Aug 07 17:11:35 2017
Archived Log entry 104646 added for thread 1 sequence 77437 ID 0x3fd8264f dest 1:
Mon Aug 07 17:10:13 2017
TT00: Standby redo logfile selected for thread 1 sequence 77438 for LOG_ARCHIVE_DEST_2
Mon Aug 07 17:10:12 2017
Current log# 4 seq# 77438 mem# 2: /a02/oradata/admin3/FZJA/redog04m03.log
Mon Aug 07 17:10:11 2017
Archived Log entry 104644 added for thread 1 sequence 77436 ID 0x3fd8264f dest 1:
Mon Aug 07 16:55:11 2017
```

Dateien horizontal verketten (paste)

paste [*options*] *filename1* *filename2* [...]

- fügt Zeilen von zwei oder mehr Dateien horizontal zusammen
- Zeilen werden standardmäßig durch tab getrennt
- die Ausgabe einer kompletten Zeile (Ausgabe der entsprechenden Zeilen aller Dateien) wird mit einem Zeilenende beendet
- mögliche Optionen:
  -d *list* benutzt die Zeichen aus *list* zur Trennung der Zeilen aus den einzelnen Dateien
  -s fügt alle Zeilen einer ganzen Datei zu einer Zeile zusammen

Beispiel:

```
$ paste -d"-\t" obst alter telnr
Birnen-Fritz    Meier   25      6736734
Kiwis-Erika     Silber  42      783722
Avocados-Toni   Meier   53      12934737
Bananen-Angela  Gold    25      8423
Orangen-Franz   Gold    12      7342
Aepfel-Tim      Meyer   45      83747
Stachelbeeren-Emil      Gold    66      783467
Kirschen-Manfred        Silber  29      8347834
Brombeeren-
```

Dateien aufteilen (split)

split [*options*] [*filename* [*outname*]]

- zerlegt eine Datei in mehrere Teile
- Größe der Teile ist durch Option steuerbar:
  - **-l** (lines) Anzahl Zeilen (Default 1000)
  - **-b** (bytes) Anzahl Zeichen
- die Teildateien heißen $outname$, an die eine Folge von Buchstaben angehängt wird:
  - aa an die 1. Datei
  - ab an die 2. Datei ...
  - zz an die 676. Datei
- fehlt $outname$, wird **x** genommen
- die Option **-a** gibt die Länge der Buchstaben-Endung und somit die Anzahl maximal möglicher Teildateien an

Beispiel:

```
#  Zerlegt die Datei in Bloecke zu 4096 Zeichen
$ split -b 4k README
$ ls -l
-rw-r--r-- 1 zdv045 zam 8903 Jul 27 10:24 README
-rw-r--r-- 1 zdv045 zam 4096 Jul 27 10:25 xaa
-rw-r--r-- 1 zdv045 zam 4096 Jul 27 10:25 xab
-rw-r--r-- 1 zdv045 zam  711 Jul 27 10:25 xac
```

Dateien drucken im Netz (lpr)

lpr [*options*] [*filename* [...]]

- nutzt einen Spooling-Daemon, um die angegebenen Dateien zu drucken
- fehlt die Dateiangabe, wird die Eingabe über Pipe (Seite 133) oder die Tastatur erwartet
- mit **-P druckername** wird der Drucker im Netz angegeben, auf dem gedruckt werden soll; der Name ist von der lokalen Konfiguration abhängig
  (/etc/printcap oder /etc/qconfig)
  Default: Shell-Variable $PRINTER

Beispiele:

```
lpr -Pzam23 .profile → A4 s/w ASCII
lpr -Ptia00pd kurs.ps → A4 s/w PS duplex
lpr -Ptia00cf demo.ps → A4 Farbfolien
lpr -Ptia00c2 bild.ps → A2 Farbposter
```

Verwaltung der Druckjobs (lpq, lprm)

bei Standard-UNIX

lpq [-P *druckername*]

- listet Jobs in der Print-Queue, bei Linux zusätzlich Option **-l**
- listet Status der lokalen und remote Print-Queue

mit CUPS

lpstat [-P *druckername*]

- listet Jobs in der Print-Queue
- listet Status der lokalen und remote Print-Queue

lprm [-P *druckername*] *spoolid*

- löscht den Druckauftrag mit Nr. $spoolid$
- man kann nur eigene Druckaufträge löschen

Einige UNIX-Derivate sind bei **-P** blankkritisch.

Beispiele & Aufgabenblock (4)

Beispiele:

```
$ lpq -Pzam23
Rank    Owner   Job  Files     TotalSize
active  sander  680  dataset   100000bytes

$ lprm -Pzam23 680
$  lpq -Pzam23
Queue   Dev   Status    Job Files   ...
------- ----- --------- --- ------- ...
zam23 dzam2 READY
zam23: Tue May  8 16:43:51 2001:
zam23: no entries
```

Aufgabenblock (4)

1. Geben Sie den Inhalt der Datei */etc/profile* auf dem Bildschirm aus. Nun geben Sie obige Datei seitenweise aus.
2. Drucken Sie die Datei *$HOME/.profile* auf dem Drucker *zam23* und fragen Sie den Status ab. Löschen Sie den Druckauftrag wieder.

Dateien suchen (find)

find ***pathname search_criteria action***

- mit dem `find` Kommando können ganze Teilbäume nach Dateien durchsucht werden, die gewisse Kriterien erfüllen
- *pathname* ist ein absolut oder relativ angegebenes Start-Directory
- die so gefundenen Dateien werden mit einer im `find` angegebenen *action* versehen; z.B. Ausgabe des Dateinamens oder auch Ausführen eines Kommandos
- es müssen `r` und `x` Zugriffsrechte auf den zu durchsuchenden Directories vorhanden sein
- wichtige Suchkriterien:

  -name Dateiname, nach dem gesucht werden soll; Wildcard-Zeichen sind erlaubt und müssen in '...' oder "..." eingeschlossen werden

  -mtime letztes Änderungsdatum

Dateien suchen (find)

- weitere Suchkriterien:
  - **-atime** letztes Zugriffsdatum
  - **-inum** Inode Nummer
- wichtige Aktionen:
  - **-print** gibt die gefundenen Dateien samt Pfad aus (Default)
  - **-ls** gibt die gefundenen Dateien analog zu `ls -l` aus
  - **-exec** führt für jede gefundene Datei das spezifizierte Kommando (bis \;) aus

Beispiele:

```
# 1. Listet core-Dateien im Heimatverzeichnis
$ find $HOME -name core -ls

# 2. Alle Dateien im Verzeichnis /tmp, die
#    laenger als 7 Tage nicht mehr benutzt
#    wurden, werden geloescht
$ find /tmp -atime +7 -exec rm {} \;
```

Dateien archivieren und zurückholen (tar)

tar [*options*] [*filenames*]

- Akronym für **t**ape **ar**chiver
- `tar` faßt die angegebenen Dateien bzw. Directories zu einer tar-Datei zusammen
- ursprüngliche Pfadnamen, Eigentümer und andere Dateiattribute werden mit abgelegt
- komplettes und selektives Zurückholen einzelner Dateien aus einer tar-Datei möglich
  Vorsicht: Dateipfad beim Erstellen wird beim Extrahieren ungeprüft übernommen!
- wichtige Optionen:
  - **-c** erstellt eine neue tar-Datei (**c**reate)
  - **-x** extrahiert aus einer tar-Datei (e**x**tract)
  - **-t** listet den Inhalt einer tar-Datei auf (lis**t**)
  - **-v** gibt die Operationen aus (**v**erbose)

Dateien archivieren und zurückholen (tar)

-f spezifiziert den Namen der tar-Datei (auch Gerätedatei möglich)
- anstelle des Dateinamens bewirkt Lesen/Schreiben von/auf stdin/stdout (siehe Pipes Seite 133)

-p behält die Zugriffsrechte der Dateien bei

Hinweise zum Gebrauch von tar:

- Eine tar-Datei sollte durch die Endung **.tar** als solche gekennzeichnet werden (**.tar** wird nicht automatisch angehängt!).
- Werden Directories mit **`tar`** archiviert, sollte die tar-Datei in einem anderen Directory angelegt werden.
- Die Angabe *filenames* sollte immer **relativ** erfolgen, da mit absolutem Pfadnamen abgespeicherte Dateien nur mit diesem Namen wieder ausgelesen werden können.
- Um keine vorhandenen Daten zu überschreiben, sollten tar-Dateien in einem neuen Verzeichnis zurückgeschrieben werden.
- **Fremde tar-Dateien immer vorher mit -tvf auflisten und Pfadnamen prüfen!**

Beispiele zur Archivierung mit tar

```
# Erstelle tar-Datei vom aktuellen Directory
$ cd $HOME/vilearn
$ tar -cvf $HOME/vilearn.tar .
./
./3temp
./5tricks
./1basics
./3cutpaste
./2moving
./4inserting
```

```
# Liste Inhalt der tar-Datei
$ tar -tvf $HOME/vilearn.tar
drwx------ kur224/kurs       0 2003-09-01 14:11:08 ./
-rw------- kur224/kurs     531 2003-09-01 14:11:08 ./3temp
-rw------- kur224/kurs    9326 2003-09-01 14:11:08 ./5tricks
-rw------- kur224/kurs    5941 2003-09-01 14:11:08 ./1basics
-rw------- kur224/kurs   11082 2003-09-01 14:11:08 ./3cutpaste
-rw------- kur224/kurs    9001 2003-09-01 14:11:08 ./2moving
-rw------- kur224/kurs    5442 2003-09-01 14:11:08 ./4inserting
```

Beispiele zur Archivierung mit tar

```
# Extrahiere eine Datei aus der tar-Datei
$ mkdir $HOME/newdir
$ cd $HOME/newdir
$ tar -xvf $HOME/vilearn.tar ./5tricks
./5tricks
# nie absolute Pfadnamen verwenden!
$ tar -cvf $HOME/vilearn.v2.tar $HOME/vilearn
/home/kur224/vilearn/
/home/kur224/vilearn/3temp
/home/kur224/vilearn/5tricks
/home/kur224/vilearn/1basics
/home/kur224/vilearn/3cutpaste
/home/kur224/vilearn/2moving
/home/kur224/vilearn/4inserting
# Extrahiere eine gepackte tar-Datei
$ gzip $HOME/vilearn.tar
$ gunzip -c $HOME/vilearn.tar.gz | tar -xvf -
./
./3temp
./5tricks
./1basics
./3cutpaste
./2moving
./4inserting
```

Aufgabenblock (5)

1. Wieviel Speicherplatz belegt Ihr nach */tmp* archiviertes und komprimiertes Heimatverzeichnis?
   Tipp: Verwenden Sie Ihre Userid für den Dateinamen.
2. Suchen Sie Dateien mit dem Namen *fonts.dir* .
   Tipp: Starten Sie die Suche im Unterverzeichnis */usr/share/fonts*
3. Finden Sie in dem Teilbaum */usr/share/fonts* alle Dateien, die mit **helvB14** beginnen. Starten Sie die Suche aus Ihrem Heimatverzeichnis.
4. Extrahieren Sie aus der tar-Datei (Archiv) */usr/local/unixkurs/aufg.5.tar* die Datei(en) *./testdir* in ein neues Directory *tar_dir* unterhalb Ihres Heimatverzeichnisses.

Inhalt

Ein-/Ausgabeumlenkung

- In der Regel gilt für Kommandos:
  - lesen von Standardeingabe (0)
  - schreiben die normale Ausgabe auf Standardausgabe (1)
  - schreiben Fehlermeldungen auf Standardfehlerausgabe (2)
- Diese befinden sich in einer interaktiven Umgebung alle auf dem zugeordneten Terminal.
- Soll die E/A von bzw. auf Dateien erfolgen, so kann die E/A umgelenkt (I/O Redirection) werden (z.T. blankkritisch).
- Operatoren der Ein-/Ausgabeumlenkung:

  `>` Schreiben auf eine Datei (von vorne)

  `>>` Schreiben auf eine Datei (anhängen)

  `<` Lesen von einer Datei

  `<<` Erstellen einer Datei (Here Document)
- nicht benötigte Ausgabe nach `/dev/null` (Mülleimer)

Formate der Ein-/Ausgabeumlenkung

command **[*options*]** $<$***infile***

- $\mathit{command}$ liest die Eingabe von der Datei infile
  Beispiel: `mail m.mustermannfz-juelich.de < fruits`

command **[*options*]** $>$***outfile***

- $\mathit{command}$ schreibt die normale Ausgabe in die Datei $\mathit{outfile}$.
  Falls die Datei bereits existiert, geht der alte Inhalt verloren.
  `set -o noclobber` verhindert das Überschreiben schon existierender Dateien.
  (`set +o noclobber` schaltet dies wieder aus.)
  Beispiel: `date >Datum`

command **[*options*]** $>>$***outfile***

- $\mathit{command}$ hängt die Ausgabe an das Dateiende von $\mathit{outfile}$, die ggf. angelegt wird.
  Beispiel: `lpq >>PRT_stat`

Ein-/Ausgabeumlenkung

command **[*options*]** **2>*errfile***

- ***command*** schreibt Fehlermeldungen in die Datei ***errfile*** (Filedescriptor 2)
  Beispiel: `rm abc 2>Errors`

command **[*options*]** **<*infile*** **>*outfile***

- ***command*** liest die Eingabe aus Datei ***infile*** und schreibt die Ausgabe in Datei ***outfile***
  Beispiel: `cat <fruits >fruechte`

command **[*options*]** **>*outfile*** **2>&1**

- ***command*** schreibt normale Ausgabe und Fehlermeldungen in Datei ***outfile***
  Beispiel: `ls -l *.gz >ls.out 2>&1`

command **[*options*]** **<<*identifier***
...Eingabe...
identifier

- Here Document; was folgt, gilt als Eingabe bis zu der Zeile, die ab *Spalte 1* nur ***identifier*** enthält. (Anwendung: Shell Programmierung)

Übung zu E/A Umlenkung

```
cd
rmdir testdir1
rmdir testdir1 >rm_error
ls -l rm_error
rmdir testdir1 2>rm_error
ls -l rm_error
cat rm_error

rmdir testdir1 2>>rm_error
ls -l rm_error
cat rm_error
```

```
set -o noclobber
rmdir testdir1 2>rm_error
set +o noclobber
cat rm_error

ssh zam1362 2>/dev/null <<EOF
  echo "** Rechner:"
  hostname
  echo "** Datum/Uhrzeit:"
  date
  echo "** Verzeichnis:"
  pwd
  EOF
```

Ausgabe verdoppeln

tee **[*options*]** ***outfile***

- erwartet Daten aus der Standardeingabe und reicht sie unverändert an die Standardausgabe durch kopiert sie aber gleichzeitig in die angegebene Datei $outfile$
- existiert $outfile$, wird sie überschrieben, anderenfalls angelegt
- Verwendung in komplexen Pipes (Seite 133) zum Aufspüren und Beheben von Fehlern
- Optionen:
  - **-a** Ausgabe wird an $outfile$ angehängt (append)
  - **-i** ignoriert Interrupt Signale (<Ctrl>-c)

Beispiel:

```
$ cd $HOME/vilearn
$ tar -cvf ~/vilearn1.tar . | tee ~/backup.log
```

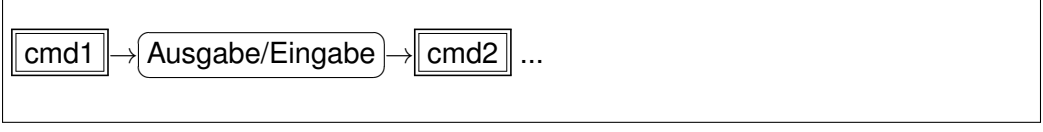
Logfile `backup.log` wird zu Kontrollzwecken angelegt.
Ausgabeumleitung mit `> backup.log` unterdrückt die Ausgabe, die mit `-v` extra eingeschaltet wurde. Fortschritt von `tar` müsste dann mit `tail -f backup.log` verfolgt werden.

Pipes und Filter

- Die Shell bietet auf Kommandoebene eine Möglichkeit, die Ausgabe eines Kommandos direkt als Eingabe an das nächste Kommando weiterzureichen. Diese Vorgehensweise wird durch eine sogenannte **Pipe** oder **Pipeline** realisiert. Format:

  cmd1 | ***cmd2*** **[** | ***cmd3*** **[...]]**

- Die Pipe hält die Daten zwischenzeitlich in einem Datenpuffer:



- Beispiel: `ls -Ral | more`

  Der Effekt ist analog zur Befehlsfolge:
  ```
  ls -Ral >outls
  more outls
  ```

Pipes und Filter

- Pipe arbeitet nur in eine Richtung: $cmd_1 \rightarrow cmd_2 \rightarrow ... \rightarrow cmd_n$
- **Vorsicht beim Gebrauch von E/A Umlenkung bei Pipes!**
  Die Befehle der Pipe werden:
  1. in umgekehrter Reihenfolge in einen Stack geschrieben, wobei die Filedescriptoren (0,1,2) bereits entsprechend geöffnet werden:

     1. ↓ | cmd_n | ... | cmd_2 | cmd_1 | ↑ 2.

  2. von unten nach oben abgearbeitet.

  Beispiel: `cmd1 <data | cmd2 >data`

  Die Datei `data` wird gelöscht (weil Open für output von `cmd2`) bevor `cmd1` sie lesen kann!
- Ein **Filter** bezeichnet ein Kommando, das seine Eingabe liest, transformiert und wieder ausgibt.
- Typische Filter Kommandos sind: **`wc grep sort cut awk sed`**

Muster suchen (grep, egrep) I

grep [*options*] *Muster* [*filename*]
egrep [*options*] *Muster* [*filename*]

- Akronym für **g**lobal **r**egular **e**xpression **p**rint
- die Datei wird nach $Muster$ durchsucht
- alle Zeilen, die $Muster$ enthalten, werden ausgegeben

Beispiel:

```
$ cat telefon
Mueller Anna   4711
Mueller Berta  4804
Meier   Hugo   8976
Melk    Klara  8083
Meyer   Egon   3099
Schmitz Melia  8011
```

```
$ grep Mueller telefon
Mueller Anna   4711
Mueller Berta  4804

$ grep 4711 telefon
Mueller Anna   4711
```

Muster suchen (grep, egrep) II

- *Muster* kann hierbei ein ***regulärer Ausdruck*** sein
  (siehe Seite 137 oder `man grep` bzw. `man ed`)
- **`egrep`** ist eine erweiterte Form und ermöglicht das Suchen nach mehreren Mustern
  (z.B. Oder-Verknüpfung der Muster durch |).

  Beispiel:
  ```
  $ egrep "Meier|Meyer" telefon
  Meier   Hugo   8976
  Meyer   Egon   3099
  ```
- wichtige Optionen:
  - **-i** ignoriert Groß- und Kleinschreibung
  - **-n** gibt vor jeder Zeile die Zeilennummer aus
  - **-c** gibt nur die Anzahl der Zeilen aus, die das Muster enthalten
  - **-v** gibt die Zeilen aus, die das Muster **nicht** enthalten
  - **-q** liefert nur exit code (0 bei gefundenem Muster, sonst 1)

  Beispiel:
  ```
  $ grep -iv mueller telefon
  Meier   Hugo   8976
  Melk    Klara  8083
  Meyer   Egon   3099
  Schmitz Melia  8011
  ```

Reguläre Ausdrücke

Ein regulärer Ausdruck definiert ein Textmuster, welches eine Klasse von Wörtern (innerhalb einer Zeile) beschreibt. Die folgenden Metazeichen helfen bei der Definition eines Textmusters:

\	verhindert die Interpretation des folgenden Zeichens
.	**ein** beliebiges Zeichen an dieser Stelle
*****	unmittelbar vorangestellter reguläre Ausdruck beliebig oft (auch keinmal); **.*** ist eine beliebige, auch leere Zeichenkette
^	Zeilenanfang
$	am Ende des Ausdrucks das Zeilenende, sonst sich selbst
\<	Wortanfang
\>	Wortende
[...]	alle in den Klammern aufgeführten Zeichen können auftreten
[^...]	alle in den Klammern aufgeführten Zeichen dürfen **nicht** auftreten
[abc], [x-z], [0-9]	Zeichenklasse

Reguläre Ausdrücke - Beispiele

```
# Suchen nach bestimmten Mustern
$ grep Me telefon
Meier   Hugo   8976
Melk    Klara  8083
Meyer   Egon   3099
Schmitz Melia  8011

$ grep "^Me" telefon
Meier   Hugo   8976
Melk    Klara  8083
Meyer   Egon   3099

$ grep "^Me.er" telefon
Meier   Hugo   8976
Meyer   Egon   3099
```

```
$ grep "11" telefon
Mueller Anna   4711
Schmitz Melia  8011
Busch   Anton  1158

$ grep "11$" telefon
Mueller Anna   4711
Schmitz Melia  8011

$ grep 'go\>' telefon
Meier   Hugo   8976

grep '\<.go' telefon
Meyer   Egon   3099
```

Komplexe Beispiele zu grep und Pipes

```
# Suche include in C-Programmen
$ grep include *.c
rexec.c:#include <sys/types.h>
rexec.c:#include <stdio.h>
# Suche nach Muster mit Leerzeichen
$ grep "Sicherung erfolgreich" logfile | more
# Suche nach C-Programmen in Teilbaum
$ ls -Ral . | grep "\.c$"
```

Hierbei werden alle Dateien des Teilbaums (inkl. seiner Subdirectories), die mit **.c** enden, aufgelistet.

Das **$** Symbol kennzeichnet das Zeilenende. Mit dem \ wird der **.** vor der Shell versteckt.

Entspricht dem Aufruf: `ls -Ral . | grep \\.c$`
Hierbei versteckt das erste \ das zweite vor der Shell.
Erste Interpretation durch die Shell: → `grep \.c$`
dann erst Auswertung des Musters durch `grep` mit **.** als Metazeichen

```
# Liste aktiver Prozesse eines Scripts (<...> ersetzen)
$ ps -ef | grep <pgrun> | grep -v grep
$ ps -ef | grep <pid_pgrun>
```

Muster suchen (zgrep)

zgrep [*options*] *Muster* [*filename.gz*]

- sucht Muster in komprimierten (gezippten) Dateien
- Arbeitsweise und Optionen analog zu `grep`
- die komprimierte Datei muss nicht zuvor ausgepackt werden

Beispiel:

```
# Suchen in gz-Datei nach einem Muster
$ grep -i schmitz telefon.bckp.gz

$ zgrep -i schmitz telefon.bckp.gz
Schmitz Melia  8011
```

Extrahieren von Zeichenketten (cut)

cut [*options*] [*filename*]

- bedeutet Herausschneiden von Informationen
- was gewünscht wird, gibt man mit *options* an:
  - **-c *list*** Ausgabe von bestimmten Spalten (column)
  - **-f *list*** Ausgabe von "Wortbereichen" (field)
- *list* kann folgende Formen haben:

n1,n2,...	Kommaliste
n1-n2	Bereich von - bis
-n2	vom Zeilenanfang bis n2
n1-	von n1 bis zum Zeilenende

- mit **-d*char*** wird das Trennzeichen zur "Worterkennung" angegeben (delimiter); Default ist Tabulator

  Achtung: Wenn "Blank" der Delimiter ist, dann zählt jedes Blank im Text für sich!
  Besser dann awk-Benutzung: `awk '{print $`n`[,$`m`[,...]]}'`
- einige UNIX-Systeme dürfen kein Blank zwischen **-f**, **-c** und deren Werten haben

Extrahieren von Zeichenketten (cut) - Beispiele

```
# Erstellen einer Datei
# mit Ausgabe des date Befehls
$ date > date.out
# Auflisten des Inhalts der Datei
$ cat date.out
Mon Jun 29 09:50:37 MET DST 1992
# Herausholen des 2. Wortes
$ cut -f 2 -d' ' date.out
Jun
# Herausholen des 2. Wortes
# aber mit : als Wortbegrenzung
$ cut -f 2 -d":" date.out
50
# Herausholen des 9. und 10. Zeichens
# aus aktueller date Ausgabe
$ date | cut -c9-10
29
```

Beispiele für Filter-Kommandos

`ps -ef | grep emacs`
liefert Informationen über alle Prozesse und Benutzer, die zur Zeit den emacs-Editor benutzen

`ls -l | grep ^d`
listet alle Directories des aktuellen Verzeichnisses auf

`tr "[a-z]" "[A-Z]" < file_1 >file_2`
übersetzt alle in ***file_1*** klein geschriebenen Buchstaben in Großbuchstaben und schreibt das Ergebnis nach ***file_2***

`cut -c2- file_1 > file_2`
eliminiert die 1. Spalte der Datei ***file_1*** und schreibt das Ergebnis nach ***file_2***

`cat file | col -b > file_2`
konvertiert gewisse Kontroll-Zeichen, z.B. von PC-Dateien oder *script*- Dateien (CR/LF $\Rightarrow$ UNIX ^M) und schreibt das Ergebnis nach ***file_2***

Übung zu Filter-Kommandos

1. Erstellen Sie eine Datei

```
echo '#Dies ist ein Test' > tf1
```

und erzeugen eine Datei *upper*, die den Text in Großbuchstaben und eine Datei *lower*, die den Text in Kleinbuchstaben enthält:

```
tr "[a-z]" "[A-Z]" < tf1  > upper
tr "[A-Z]" "[a-z]" < tf1  > lower
```

2. Erstellen Sie eine Datei

```
cat > tf2 <<"EOF"
# Oh, wie gut
# dass keiner weiss,
# dass ich
# Rumpelstilzchen heiss
EOF
```

und entfernen Sie die Kommentarzeichen:

```
cut -c2- tf2 > reim
```

Sortieren und Mischen von Dateien (sort)

sort [*options*] [*filename*]

- sortiert die Zeilen einer Datei und schreibt das Resultat auf Standardausgabe
- *options* sind Sortierkriterien:
  - **-r** kehrt die Sortierreihenfolge um; normal ist aufsteigend
  - **-n** für numerische Sortierung; normal ist nach ASCII-Tabelle
  - **-k** *pos1[,pos2]* gibt die Wort-Positionen innerhalb einer Zeile an, nach der sortiert werden soll. Fehlt pos_2, so gilt bis Zeilenende.
    Die Option **-k** sollte mit der Option **-b** verwendet werden.
  - **-b** ignoriert führende Sonderzeichen incl. Leerzeichen
- fehlt Position *pos1*, wird nach dem ersten Wort jeder Zeile sortiert
- bei Angabe mehrerer Positionen pos_n wird das nächste Wort genommen, wenn Daten des vorherigen Wortes gleich sind

Sortieren & Mischen von Dateien (sort) - Beispiele

```
# Inhalt der Datei
cat fruits                      --> 3 apple
                                    1 pineapple
                                    10 banana
                                    5 peach

# ASCII Sortierung
sort fruits                     --> 1 pineapple
                                    10 banana
                                    3 apple
                                    5 peach

# Numerische Sortierung
sort -n fruits                  --> 1 pineapple
                                    3 apple
                                    5 peach
                                    10 banana

# Sortierung nach 2. Wert
sort -k2 fruits                 --> 3 apple
                                    10 banana
                                    5 peach
                                    1 pineapple
```

Komplexe Filter Kommandos (awk, sed)

Die Verwendung von Filtern kann durch komplexe Kommandos sehr vereinfacht werden. Hier sind zwei Kommandos von wesentlicher Bedeutung:

awk A.**A**ho, P.**W**einberger, B.**K**erninghan

C-ähnliche Kommandosprache mit den Möglichkeiten einer Programmiersprache

Syntax: awk '[BEGIN] *{ action }*
[pattern] { action }
...
[END] *{ action }'*

Beispiele:
```
# Letztes Wort der Eingabe ausgeben
cat fruits | awk '{ print $NF}'
# Erste 2 Wörter umgekehrt ausgeben
cat fruits | awk '{ print $2, $1}'
```

sed **S**tream **Ed**itor, gleiche Kommandosyntax wie der vi

Beispiel:
```
# Ersetze 1.Muster durch 2.Muster
sed 's/KFA/Forschungszentrum/' list>fzj_list
```

Aufgabenblock (6)

1. Schreiben Sie die ersten 8 Zeilen der Datei */usr/local/unixkurs/aufg.6/limmerick* auf die Datei *$HOME/firstlines*.
   Tipp: `man head`
   Schreiben Sie außerdem die letzten 8 Zeichen auf die Datei *$HOME/lastbytes*.
   Tipp: `man tail`
2. Listen Sie alle FORTRAN-Programme (Endung .f) des Verzeichnisses */usr/local/unixkurs/fortran* auf. Verwenden Sie hierzu `grep`.
3. Listen Sie mit Hilfe des `cut` Kommandos obige Dateinamen **ohne** Endung auf.
4. Welche Zeilen der Datei */usr/local/unixkurs/.profile* bestehen nur aus Kommentar?
   Tipp: Wie lautet das Kommentarzeichen und wo muß es stehen?
   Wie viele Zeilen sind es?
   Tipp: `man wc`
5. Listen Sie die Dateien unter */usr/bin* der Dateigröße nach auf.

Shell-Variablen I

- Die Benutzerumgebung ist wesentlich davon mitbestimmt, welche Variablen welche Werte innerhalb der entsprechenden Shell besitzen.
- Shell-Variablen sind grundsätzlich Zeichenketten und werden von der Shell und von einigen UNIX-Kommandos bei der Verarbeitung berücksichtigt.

  Beispiel: Suchpfad PATH der Shell für Kommandos
- Definition und Zuweisung durch:

  ***NAME*=[*wert*]**

  NAME ist der Variablenname; beginnt mit einem Buchstaben und kann Ziffern sowie Underscores/Unterstriche (_) enthalten

  wert ist der Inhalt der Variablen; kann auch leer sein (" ", ' ')

  Leerzeichen vor und hinter dem = sind verboten!

  Beispiel: `FEIERTAG=Himmelfahrt`

Shell-Variablen II

- Inhalt der Shell-Variablen (Parameterersetzung) erhält man mit:

  $*NAME* oder **${*NAME*}**

  Beispiel:
  ```
  FARBE1=blau; echo $FARBE1
  FARBE2="hell gelb"; echo ${FARBE2}
  ```

- Der Gültigkeitsbereich der Shell-Variablen ist nur innerhalb der ausführenden Shell. Soll auch in aufgerufenen Programmen auf die Variable zugegriffen werden, muß diese exportiert werden:

  export *NAME* oder **export *NAME=wert***

  `set -a` bewirkt eine automatische Exportierung aller neu definierten Variablen
  `set +a` schaltet diese Funktion wieder ab

- Die Übergabe der Shell-Variablen geht nur in eine Richtung:

  Rufender (parent) $\rightarrow$ Aufgerufener (child)

  Die Rückgabe von einem **child process** an einen **parent process** ist nicht möglich.

Shell-interne Kommandos

echo [*args*]
gibt angegebene Argumente auf Standardausgabe aus

exit [*rc*]
verläßt die Shell

export [*varlist*]
ermöglicht auch einer aufgerufenen Shell den Zugriff auf die angegebenen Shell-Variablen

set [*options*]
setzen von Shell-Parametern

. dateiname
- führt die Shell-Kommandos aus der Datei innerhalb der gleichen Shell aus
- ohne "." würde eine neue Shell gestartet
- execute- Recht für die Datei ist nicht erforderlich

Übung zu Shell-Variablen

1. Anlegen und Auflisten einer Variablen in aktueller Shell:

```
VAR="bin in der alten Shell"
echo $VAR
```

2. Anlegen eines Shell-Scripts:

```
cat > var.ksh <<"EOF"
echo "vorher: VAR = $VAR"
VAR="Bin in var.ksh"
echo "nachher: VAR = $VAR"
EOF
chmod 700 var.ksh
```

3. Aufruf des Shell-Scripts ($\Rightarrow$ Eröffnen einer neuen Shell):

```
var.ksh
```

   Auflisten der Variablen wieder in der aktuellen Shell:

```
echo $VAR
```

4. Exportieren der Variablen an nachfolgende Shells:

```
export VAR
```

Übung zu Shell-Variablen

5. Erneutes Aufrufen des Shell-Scrips *var.ksh*:
   `var.ksh`
   (VAR behält innerhalb des Shell-Scripts den Wert der aktuellen Shell!)
6. Überprüfen der Variablen wieder in der aktuellen Shell:
   `echo $VAR`
   (Ein Ändern von VAR innerhalb der neuen Shell hat keinen Einfluß auf VAR der aktuellen Shell!)
7. Aufruf des Shell-Scripts mit **.** ($\Rightarrow$ *var.ksh* läuft in der aktuellen Shell):
   `. var.ksh`
8. Überprüfen der Variablen wieder in der aktuellen Shell:
   `echo $VAR`
   (Ein Ändern von VAR innerhalb des Shell-Scripts verändert nun auch VAR der aktuellen Shell)

Shell-Environment

Einige Shell-Variablen werden dem Benutzer für seine Umgebung mitgeliefert und sollten daher nur gezielt genutzt werden:

HOME	Heimatverzeichnis
PATH	Suchpfad für Kommandos
PS1	Prompt-String
PS2	Folgeprompt-String
PWD	aktuelles Verzeichnis
SHELL	Login-Shell
USER	Login-Username
PRINTER	Default-Drucker
DISPLAY	X11- Server-Bildschirm
VISUAL	Standard-Editor
LANG	Spracheinstellung

Profiles

Der Benutzer kann seine Umgebung in Profiles selbst definieren.

- bei jedem neuen Login (ksh- und bash-Shell):

  /etc/profile als Systemprofile
  $HOME/.profile oder **$HOME/.bash_profile**
  als Benutzerprofile; zum Setzen von Objekten, die an Kind-Prozesse vererbt werden (z.B. exportierte Variablen, ...); möglichst keine X-Kommandos

- bei jedem Eröffnen einer neuen Shell (nur für zentral verwaltete Systeme im JSC):

  /etc/bashrc als Systemprofile für die Bash
  $HOME/.bashrc
  als Benutzerprofile für die Bash; zum Setzen von Objekten, die nicht an Kind-Prozesse vererbt werden (z.B. alias, set, ...)

Shell-Ersetzungsmechanismen I

Da jegliche Eingabe zunächst von der Shell interpretiert wird, gibt es Möglichkeiten, die Ersetzungsmechanismen zu nutzen und zu steuern.

- Parameterersetzung durch $

  $*VAR*

  An Stelle von $*VAR* wird der **Inhalt** von *VAR* eingesetzt; dieser Vorgang ist einstufig.

  Beispiele:
  ```
  $ echo $HOME   → /home/kur
  $ echo $FARBE2 → hell gelb
  $ FARBE3=${FARBE2}gruen
  ```

Shell-Ersetzungsmechanismen II

- Kommandoersetzung durch Backquotes

  `` `command` ``

  An Stelle des in Backquotes `` `...` `` eingeschlossenen Kommandos wird **das Ergebnis** des Kommandos als Zeichenkette eingesetzt.

  Beispiele:
  ```
  $ echo `pwd`              → /home/kur
  $ echo `ls`               → haus auto
  $ FLIST=`ls`
  $ echo $FLIST             → haus auto
  ```

- Kommandoersetzung durch `$(...)`

  `$(command)`

  An Stelle des in `$(...)` eingeschlossenen Kommandos wird **das Ergebnis** des Kommandos als Zeichenkette eingesetzt.

  Beispiele:
  ```
  $ echo "Dir: "$(pwd)            → Dir: /home/kur
  $ echo "List: "$(ls)           → List: haus auto
  $ FLIST=$(ls)
  $ echo $FLIST                   → haus auto
  ```

Shell-Ersetzungsmechanismen III

- Doublequotes: **"*string*"**

  Bei Zeichenketten in doppelten Hochkommata `"..."` findet **sowohl** Kommando- **als auch** Parameterersetzung statt; Beispiele:

```
$ echo "$HOME"
/home/kur
$ echo "List: `ls`"
List: haus auto
$ echo "$HOME: $(ls)"
/home/kur: haus auto
```

- Quotes: **'*string*'**

  Bei Zeichenketten in einfachen Hochkommata `'...'` findet **weder** Kommando- **noch** Parameterersetzung statt; Beispiele:

```
$ echo '$HOME'
$HOME
$ echo 'List: `ls`'
List: `ls`
$ echo '$HOME: $(ls)'
$HOME: $(ls)
```

Shell-Ersetzungsmechanismen - Beispiel

```
$ cat belegt.ksh →  BELEGT=`du -sk $HOME | cut -f1`
                    echo '$HOME = '"$HOME"
                    echo "belegt heute (`date +%d.%m.%y`)"
                    echo "$BELEGT KBytes"
$ belegt.ksh →      $HOME = /home/kur
                    belegt heute (19.10.01)
                    1700 KBytes
```

Das Kommando **eval** ermöglicht eine zweistufige Ersetzung:
(Verwendung in der Shell-Programmierung)

eval ***command***

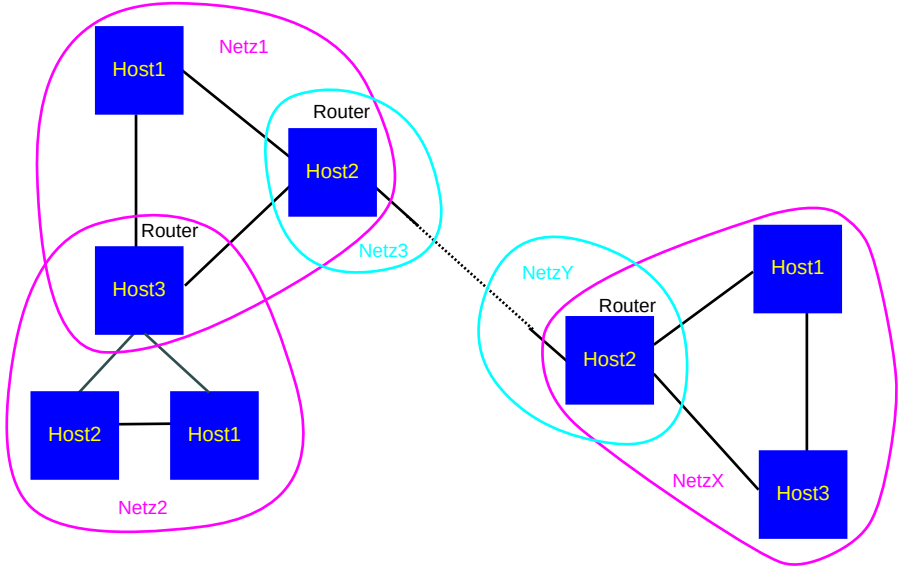
Beispiel:
```
$ FORMAT='+%A, %d %B %Y, %T'
$ DATUM='date "$FORMAT"'
$ echo $DATUM
date "$FORMAT"
$ eval $DATUM
Tuesday, 21 February 2012, 12:27:06
```

Aufgabenblock (7)

1. Listen Sie in Ihrem Heimatverzeichnis die Namen aller Unterverzeichnisse auf. (Tipp: `man grep`)
2. Weisen Sie der SHELL-Variablen *FILES* die Namen aller Einträge Ihres Heimatverzeichnisses zu (nur oberste Schicht).
3. Setzen Sie Ihren Prompt-String auf
   '*userid@host:current_dir* > '
   Das Verzeichnis soll sich mit jedem `cd` ändern. Testen Sie dies im aktuellen Fenster. Tragen Sie die Änderung für den Prompt in die richtige Datei ein, damit er bei jedem neuen Fenster aktiv ist.
4. Geben Sie drei Möglichkeiten an, wie die Shell einen * auf den Bildschirm schreiben kann.
5. Legen Sie ein Verzeichnis *$HOME/tmp* an und kopieren Sie alle Dateien aus */usr/local/unixkurs/aufg.7* dorthin. Gehen Sie nach *$HOME/tmp* und legen dort eine Datei mit dem Namen *A** an. Löschen Sie diese, nachdem Sie sich vergewissert haben, daß sie existiert.

Inhalt

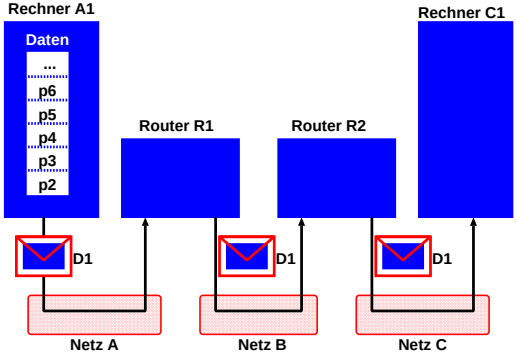
Aufbau eines Netzes



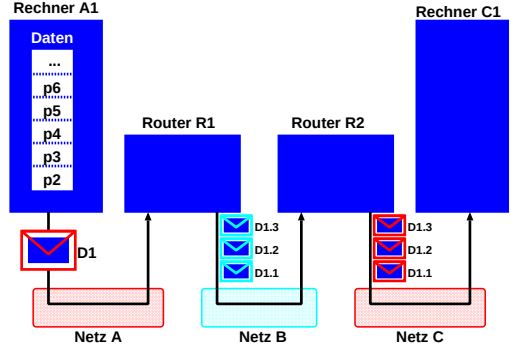
Internet I

- großes Wide Area Network (WAN)
- DoD 1969 (Department of Defense)
- weiterentwickelt an Universitäten
- einheitlicher Adressraum mit eindeutigen Internet-Adressen
- benutzt Transmission Control Protocol/Internet Protocol (TCP/IP)
  - zerlegt Daten in Pakete (TCP oder UDP)
  - versieht Pakete mit einem Header analog einem Briefumschlag mit Adresse und Absender (Datagram / IP-Packet)
  - versieht Datagrams mit Netzwerkbeschreibung (Frame)
  - paßt Paketgröße der Netzwerktopologie an (Fragmentierung bzw. Maximum Transmission Unit Path Discovery)
- Status 1996:
  - 5 Mill. Rechner
  - 50 Mill. Nutzer
- Status 2008:
  - $>$500 Mill. Rechner
  - über 1,4 Milliarde Nutzer

Internet II



TCP/IP: Transport der Internet-Pakete durch gleichartige Netze



TCP/IP: Transport der Internet-Pakete mit Fragmentierung

Internet-Adressen

Format der Internet-Adressen im Internet Protokoll Version 4 (IP V4):

iii.mmm.nnn.ooo

wobei jedes der Felder *iii, mmm, nnn, ooo* Werte zwischen 0 und 255 (1 Byte) annehmen kann, z.B. `134.94.5.116`

- Class-A Netzwerk :

1. Byte	*2. Byte*	*3. Byte*	*4.Byte*
Network	Host		
1 - 127	0 - 1677215		

- Class-B Netzwerk :
  (Forschungszentrum)

1. Byte	*2. Byte*	*3. Byte*	*4.Byte*
Network		Host	
128 - 191	2 - 254	0 - 65535	

- Class-C Netzwerk :

1. Byte	*2. Byte*	*3. Byte*	*4.Byte*
Network			Host
192 - 223	0 - 255	1 - 254	0 - 255

Internet-Namen & Mailadressen

- Zur einfacheren Schreib- und Merkweise gibt es zu den Internet-Adressen sogenannte Internet-Namen:

  host.domain

  Beispiele:
  ```
  zam1362.zam.kfa-juelich.de 134.94.12.49
  www.fz-juelich.de          134.94.99.20
  ```
  Die Namensauflösung geschieht über:
  - einen Nameserver (/etc/resolv.conf)
  - die lokale Hosttabelle (/etc/hosts)
- Neben den Internet-Namen der Rechner gibt es noch die Mailadressen der Benutzer:

  userid@host.domain

  Beispiele:
  ```
  unix724@zam1362.zam.kfa-juelich.de  (intern)
  m.mustermann@fz-juelich.de         (weltweit)
  ```

Secure Shell (ssh, scp) I

- ist Bestandteil jeder modernen Linux/UNIX Installation
- erlaubt eine verschlüsselte Kommunikation zu entfernten Rechnern sowohl bzgl. des Passwords als auch der Daten
  (Datenschutz/Datensicherheit wird gesetzlich gefordert)
- besitzt eine verschlüsselte X-Verbindungs-Funktionalität
- dient als Ersatz veralteter Internet-Dienste (telnet/ftp/R-Dienste):
  - "Telnet"

    ssh [*username*@]*hostname*

    Beispiel:
    ```
    $ ssh zam1362.zam.kfa-juelich.de
    Password: <password>
    ```
  - "rexec"

    ssh [*username*@]*hostname command*

    Beispiel:
    ```
    $ ssh kur224@zam1362.zam date
    Password: <password>
    Mon Jul 31 17:05:43 DFT 2000
    ```

Secure Shell (ssh, scp) II

scp als Ersatz für ftp:

1. Holen einer Datei:

   scp [*username*@]*hostname*:*infile* *outfile*

2. Wegschreiben einer Datei:

   scp *infile* [*username*@]*hostname*:[*outfile*]

3. Holen eines Teilbaumes:

   scp -r [*username*@]*hostname*:*indir* *outdir*

4. Wegschreiben eines Teilbaumes:

   scp -r *indir* [*username*@]*hostname*:[*outdir*]

Es gelten die gleichen Regeln wie beim UNIX `cp` Command. Es können mehrere Dateien durch Spezifikation von $infile/outfile$ mittels Wildcard-Zeichen weggeschrieben bzw. geholt werden. Mit der **-p** Option werden die Zugriffsrechte und das Modifikationsdatum übernommen.
Vorsicht! “Links” werden allerdings aufgelöst!

Verschicken von Mail (mail)

Mittels des Kommandos **mail** können Nachrichten oder Dateien verschickt werden.

- Verschicken einer Nachricht:

  mail [-s '*subject*'] *adresse*
  Texteingabe
  <Ctrl>-d

  Die Eingabe wird mit <Ctrl>-d oder **.** beendet und an die angegebene Mailadresse $adresse$ geschickt.

- Verschicken einer Datei:

  mail [-s '*subject*'] *adresse* <*filename*

  Bei Binärdaten sollte uuencode/uudecode verwendet werden!

Die Mailadresse $adresse$ hat als offizielle Email die Form: $V.Nachname$@fz-juelich.de

Internet-Namen auflösen (nslookup)

nslookup [*hostname* | *adresse*]

- Befehl zum Auflösen der Internet Namensgebung beim Name-Server in die numerische Internet-Adresse oder Ausgabe des zugehörigen Hostnamen zur Internet-Adresse
- falls *hostname* ein Alias-Name ist, wird der richtige Hostname ebenfalls ausgegeben
- nützlich zum Überprüfen, ob der eigene Rechner den anderen Rechner überhaupt kennt und falls Alias-Namen benutzt werden, ob dieser auf den gewünschten Rechner zeigt
- unter Linux wird auch `host` verwendet

Beispiele:

```
$ nslookup zam1326.zam.kfa-juelich.de
Name:   zam1326.zam.kfa-juelich.de
Address: 134.94.12.89
$ nslookup oraproxy
oraproxy.zam.kfa-juelich.de canonical name = oraclient.zam.kfa-juelich.de
Name:   oraclient.zam.kfa-juelich.de
Address: 134.94.105.34
```

Erreichbarkeit eines Rechners (ping)

ping [*hostname* | *adresse*]

- testet die Erreichbarkeit des entfernten Rechners, der mit $hostname$ oder $adresse$ angegeben wird
- die Auflösung des Rechnernamens wird implizit vorgenommen
- <Ctrl>-c beendet das Kommando

Beispiele:

```
$ ping oraproxy
PING oraclient (134.94.105.34) 56(84) bytes of data.
64 bytes from oraclient (134.94.105.34): icmp_seq=1 ttl=64 time=11.3 ms
64 bytes from oraclient (134.94.105.34): icmp_seq=2 ttl=64 time=0.153 ms
--- oraclient ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1003ms
rtt min/avg/max/mdev = 0.153/5.769/11.385/5.616 ms
```

```
$ ping www.google.de
PING www.l.google.com (209.85.135.104) 56(84) bytes of data.
64 bytes from mu-in-f104.google.com (209.85.135.104): icmp_seq=1 ttl=246 time=12.2 ms
64 bytes from mu-in-f104.google.com (209.85.135.104): icmp_seq=2 ttl=246 time=12.5 ms
--- www.l.google.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1004ms
rtt min/avg/max/mdev = 12.279/12.421/12.564/0.180 ms
```

Aufgabenblock (8)

1. Loggen Sie sich mit `ssh` auf dem Rechner zam1362 mit Ihrer Userid und Ihrem Passwort ein. Sie beenden die Verbindung mit `exit`.
2. Holen Sie mit `scp` vom entfernten Rechner zam1362 die Datei *smiley.obj*:

Rechner:	`zam1362`
Benutzer:	***Ihre Userid***
Password:	***Ihr Passwort***
Pfad:	`/tmp`

3. Kopieren Sie mit `scp` Ihre Datei *fruits* mit Ihrer Userid als Endung auf den Rechner zam1362 in das Verzeichnis */tmp*:

Rechner:	`zam1362`
Benutzer:	***Ihre Userid***
Password:	***Ihr Passwort***
Pfad:	`/tmp`
Quelle:	`fruits`
Ziel:	`fruits.`$<$***Userid***$>$

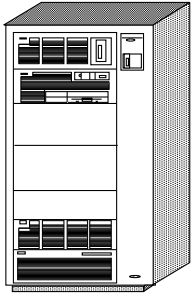
4. Ermitteln Sie die numerische Adresse Ihres eigenen Rechners.

Inhalt

Eigenschaften des X-Window Systems

- Fenstersystem für grafische Bildschirme
- Basis für eine grafische und objektorientierte Benutzeroberfläche
- netzwerkfähig (Client-Server-Prinzip)
- portabel

Client-Server Prinzip des X-Window System



Remote Computer

X-Client-Programm

enthält die Anwendungs-logik

verantwortlich für den Fensterinhalt

schickt Grafikbefehle ab

fordert

Dienste

liefert

Dienste

X-Server-Programm

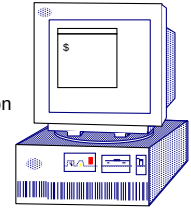
verwaltet lokale Resourcen

stellt die angeforderten Grafiken dar

leitet Benutzerinteraktion weiter

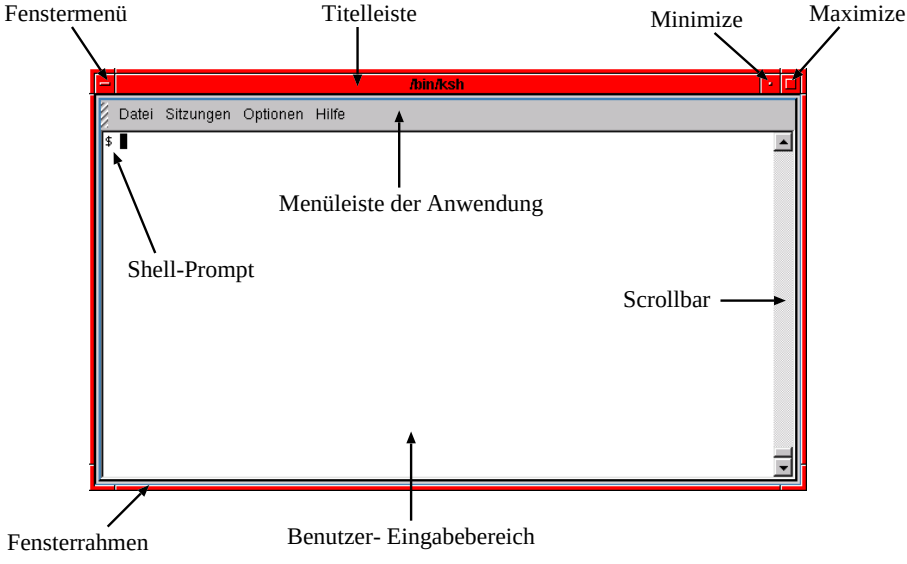
Netzwerk

X-Pakete



Arbeitsplatz

Fenster-Layout



Cut & Paste

Cut & Paste im *xterm*, *dtterm* (CDE) bzw. *konsole* (KDE):

1. Mauszeiger dort positionieren (hinschieben), wo der Text genommen werden soll
2. linke Maustaste drücken, gedrückt halten und über den gewünschten Bereich ziehen $\Rightarrow$ der markierte Bereich wird mit einer anderen Farbe hinterlegt oder umgekehrt dargestellt
3. linke Maustaste loslassen $\Rightarrow$ der markierte Bereich ist nun im Cut & Paste-Buffer
4. Mauszeiger dort positionieren, wo der Text hin soll
   (kann auch innerhalb eines anderen Fensters sein)
5. mittlere Maustaste drücken $\Rightarrow$ der Inhalt des Cut & Paste-Buffers wird an die aktuelle Position kopiert

Viele andere X-Anwendungen unterstützen ebenfalls Cut & Paste (so oder in ähnlicher Weise).

Elementares (1)

- Beispiele für Window-Manager:

  `dtwm` CDE Window Manager (AIX, SUN, Compaq: Common Desktop Environment)
  `kwm` oder `kdeinit` KDE Window Manager (Linux: eigentlich Cool Desktop Environment; da es C für CDE schon gab, blieb nur das K)
- die Aufgaben des Window Managers sind:
  - er verwaltet die Fenster
    Verschieben (Titelleiste)
    Vergrößern, Verkleinern (Rahmen)
    Minimieren (⊡), Maximieren (□), Schließen (⊠, ⊟)
  - bietet *Pop-Up Menus* für mausgesteuerte Aktionen (rechte Maustaste oder auch jede Maustaste über dem Bildschirmhintergrund)
  - bietet eine Funktion zum Verlassen der X-Umgebung (i.allg. über *Pop-Up Menu* (quit))

Elementares (2)

```
DISPLAY=X-server-address
export   DISPLAY
```

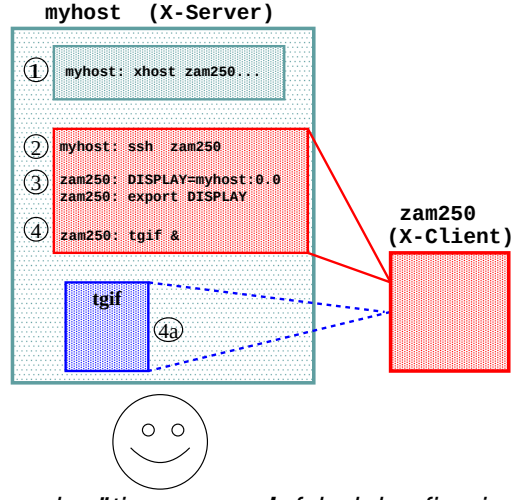
- die Shell-Environment Variable **DISPLAY** spezifiziert den X-Server für die Ausgabe der X-Anwendungen
- $X\text{-}server\text{-}address$ hat folgendes Format:
  :0.0 der lokale X-Server
  ***hostname*:0.0** der X-Server des Rechners $hostname$

```
xhost   [+]hostname
xhost   –hostname
xhost
```

- **+** erlaubt dem Rechner $hostname$, ein Fenster auf dem aktuellen Rechner aufzumachen
- **–** entsprechend verbietet dem Rechner $hostname$ Fenster auf dem aktuellen Rechner aufzumachen
- wird kein Parameter angegeben, so werden die vergebenen Berechtigungen aufgelistet

Typische Anwendung für xhost und DISPLAY

Es soll vom eigenen Desktop-System (myhost) eine X-Windows-Anwendung auf einem entfernten Rechner (zam859) gestartet werden.
Der X-Server läuft auf dem lokalen und der X-Client auf dem entfernten System.



*Dies ist bei aktuellen Betriebssystemen nur noch nötig, wenn **ssh** falsch konfiguriert ist und keine X-Weiterleitung macht.*

Typische Fehlermeldungen

Typische Fehlermeldungen einer X-Anwendung:

- `Error: Can't open display`
  Die DISPLAY-Variable auf dem Rechner, auf dem die Anwendung läuft (*X-Client*), ist falsch gesetzt.

  Prüfen mit: `echo $DISPLAY`

- ```
  Xlib:  connection to "hostname:0.0" refused by server
  Xlib:  Client is not authorized to connect to server
  Error: Can't open display: hostname:0.0
  ```

  Der *X-Client* darf auf dem lokalen Rechner (*X-Server*) kein Fenster eröffnen. Der *X-Server* muß dem *X-Client* mittels `xhost` oder `xauth` (Tki-0246) dieses erst erlauben.

  Prüfen mit: `xhost`
  ```

Customizing X11-Clients

Jede X-Anwendung verwendet zur Fenstergestaltung gewisse Attribute (X-Ressourcen):

Vordergrundfarbe, Hintergrundfarbe, Schrifttyp, Größe, Titel

Durch die Suchreihenfolge des X-Window-Systems werden sukzessive Konfigurationsdateien nach Attributen eines X-Clients durchsucht (Überschreiben möglich):

1. Dateien im Verzeichnis **$XAPPLRESDIR**
   Vereinbarungsgemäß: $HOME/app-defaults
   (wird leider nicht von allen X-Clients beachtet)
2. Attribute in **$HOME/.Xdefaults**
3. Falls über **$XENVIRONMENT** eine Datei spezifiziert wurde, die darin enthaltenen Attribute, sonst die in der Datei $HOME/.Xdefaults des Clients
4. Attribute, die über die **Kommandozeile** spezifiziert wurden

Customizing X11- Clients

Ein Benutzer konfiguriert über:

- **$XAPPLRESDIR** (=$HOME/app-defaults) wann immer möglich
- **$HOME/.Xdefaults** sonst

Die Dateinamen für die X-Ressourcen der X-Clients unter $HOME/app-defaults lauten u.a.: Emacs, XClock, Mwm, XLoad, Tgif, XLock, XTerm, Dtterm ...

Beispiele:

Setzt rote Hintergrundfarbe:

```
cd $HOME/app-defaults
echo "XTerm*background: red" >>XTerm
```

Belegt <F3>-Taste mit `ls -lisa` :

```
$ cat >>XTerm <<'EOF'
>  XTerm*VT100.translations:     #override  \
>  <Key>F3   : string("ls -lisa") string(0x0d)
>  EOF
```

X11-Clients Beispiele

xterm	öffnet eine Terminal-Emulation
xlock	sperrt den Bildschirm; weitere Nutzung nur nach Eingabe des Passwords möglich
xman	X-basierte Version des man-Kommandos
firefox	WWW Browser
thunderbird	Mail-Interface

spezielle Clients im Forschungszentrum Jülich:

dsm	
adsmback	Datensicherung mittels TSM/ADSM
adsmarch	Datenarchivierung mittels TSM/ADSM

Inhalt

Editor vi – Grundsätzliches

vi ***filename***

- unterteilt in Command- und Insert-Mode
  - der Command-Mode wird über <ESC> erreicht.
    Mächtige Umgebung zum Steuern des Editiervorgangs
  - Der Insert-Mode wird durch die entsprechenden Kommandos erreicht.
    Umgebung für die sequentielle Dateneingabe
- den entsprechenden Terminaltyp erhält man über die Shell-Variable ***TERM***
- nicht von allen Terminals aus nutzbar (systemspezifisch)

Editor vi – Grundsätzliches

vi ***fn1 fn2 ...***

editiert die angegebenen Dateien in der Reihenfolge

vi +18 ***fn***

editiert die Datei *fn* und positioniert den Cursor auf Zeile 18

vi +/"*muster*"' ***fn***

editiert der Datei *fn* und positioniert den Cursor beim ersten Auftreten der Zeichenkette *muster*

vi -r ***fn***

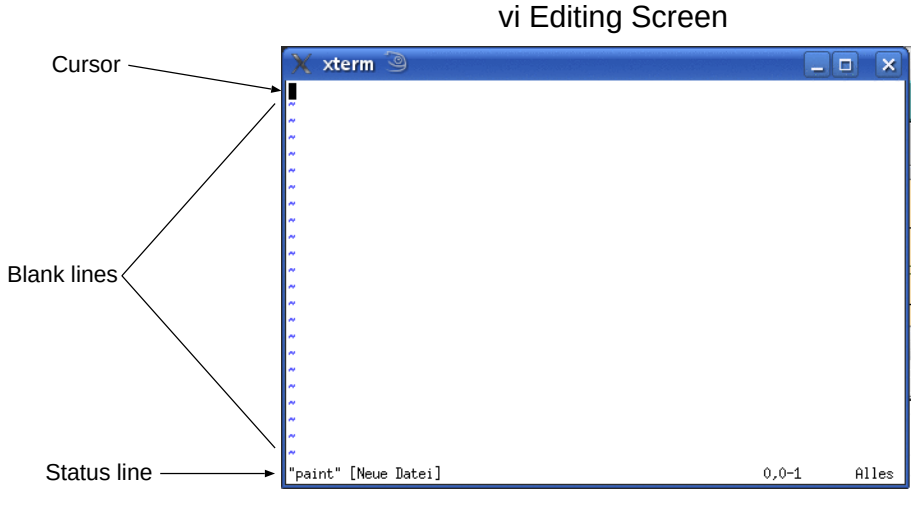
Wiederherstellen einer editierten Datei nach einem Systemabsturz

view ***fn***

editiert die Datei *fn* read-only

Editor vi – Grundsätzliches

Befehl: vi paint



Editor vi – Grundsätzliches

Beim Aufruf des Editors wird die Existenz der SHELL-Variable **EXINIT** geprüft, auf der eine durch | getrennte Liste von initial ausgeführten vi-Kommandos abgelegt ist. Existiert diese Variable nicht, so wird die Datei ~/**.exrc** für die initialen vi-Kommandos benutzt.
Einstellen der vi-Umgebung über

:set *option*

Beispiel:

:set number	zeigt die Zeilennummern an
:set autowrite	führt ein automatisches write durch, bevor einige Kommandos ausgeführt werden
:set ignorecase	ignoriert Klein/Großschreibung beim Suchen (Kurzform :set ic)
:set all	gibt alle Einstellungen aus

Command-Mode

Ctrl-F	blättere einen Bildschirm weiter
Ctrl-D	blättere einen halben Bildschirm weiter
Ctrl-B	blättere einen Bildschirm zurück
Ctrl-U	blättere einen halben Bildchirm zurück
h	positioniert ein Feld nach links $\leftarrow$
j	positioniert ein Feld nach unten $\downarrow$
k	positioniert ein Feld nach oben $\uparrow$
l	positioniert ein Feld nach rechts $\rightarrow$
$	Cursor ans Zeilenende
0	Cursor an den Zeilenanfang
***n*|**	Cursor auf Spalte *n*
+	Cursor an den Anfang der nächsten Zeile
:*n*	Cursor in Zeile *n*
%	falls der Cursor auf einer öffnenden Klammer steht, wird die entsprechende schließende Klammer gesucht
G	Cursor an das Ende der Datei

File Manipulation

:w[!] *[fn]*	sichert die aktuelle Datei unter dem Filenamen *fn*
ZZ	Save & Exit
:wq	Write & Quit
:q!	Quit Always
:r *filename*	fügt die Datei *filename* hinter der aktuellen Cursorposition ein
:e *filename*	editiert einen weiteren File
:e#	springt zum ursprünglich editierten File zurück
:n	editiert den nächsten File in der aktuellen Fileliste (z.B. vi *)
:file *fn*	ändert den Filenamen beim nächsten Sichern

Text-Operationen

i	füge vor der aktuellen Cursor-Position ein. Tippfehler sind mit <Del> oder <Backspace> zu korrigieren (Insert-Mode)
***n*i*string*ESC**	fügt n-mal *string* vor dem aktuellen Cursor ein
o	füge eine neue Zeile ein (Insert-Mode)
a	füge hinter der aktuellen Cursor-Position ein (Insert-Mode)
A	füge am Zeilenende ein (Insert-Mode)
r	ändere das Zeichen unter dem Cursor
R	ändere den Text ab der Cursor-Position bis zum nächsten <ESC>
x	lösche das Zeichen unter dem Cursor
***n*dd**	*n* Zeilen ab der Cursor-Zeile löschen (wird im Puffer gespeichert)
:*i*,*j*d	Bereich von Zeile *i* bis *j* löschen und puffern
u	Undo

Text-Operationen

.	wiederhole das letzte Änderungskommando an der aktuellen Cursorposition
***n*yy**	kopiere die nächsten *n* Zeilen in den Puffer
p	kopiere den Puffer hinter die aktuelle Zeile
J	verknüpfe die aktuelle Zeile mit der folgenden
/*string*	suche den angegebenen String. Mit n bzw. <Space> kann man erneut vorwärts und mit N bzw. ? rückwärts suchen
:*i*,*j*s/*str1*/*str2*/g	ersetze *str1* durch *str2* und zwar von Zeile *i* an bis zur Zeile *j*. Wird g angegeben, so wird *str1* auch mehrfach in einer Zeile ersetzt
:g/^*str*/d	lösche alle Zeilen, die mit *str* anfangen
!!*cmd*	führt den Befehl *cmd* aus und ersetzt die aktuelle Zeile durch die Ausgabe von *cmd*

Inhalt

Literatur

Literatur

- S. Hetze, D. Hohndel, M.Müller u.a.
  LinuX Anwenderhandbuch und Leitfaden für die Systemverwaltung
  1995, LunetIX Softfair, ISBN 3-929764-04-0
  *** für Benutzer, deutsch, Lern- und Nachschlagewerk, auf SuSE Linux CDs bis Version 7.1 als dvi-Datei (5. Auflage) und im Netz als html-Dateien (7. Auflage) verfügbar,
  URL: `http://www.wss-stuttgart.de/linux/`
- M.G. Sobell
  A Practical Guide to UNIX System
  1994, Benjamin-Cumnings, ISBN 0-305-97565-1
- Paul W. Abrahams, B.R. Larson
  UNIX for the Impatient
  1992, Addison-Wesley, ISBN 0-201-82376-4
  *** für Benutzer, Lern- und Nachschlagewerk

Literatur

- S.G. Kochan, P.H. Wood
  UNIX Shell Programming
  1989, ISBN 0-672-48448-X
- Jürgen Gulbins, Karl Obermayr
  UNIX
  1995, Springer, ISBN 3-540-58864-7
  ** für Benutzer, deutsch
- M.J. Bach
  The Design of the UNIX Operating System
  1986, Prentice Hall, ISBN 0-13-201757-1
  *** für Administratoren, Grundlagenwissen
- B.W. Kernighan, R. Pike
  The UNIX Programming Environment
  1984, Prentice Hall, ISBN 0-13-937681-x