

# Das Filesystem

Dieser Teil beschreibt das Unix Dateisystem. Neben dem Aufbau des Dateisystems werden Zugriffsrechte, verschiedenen Arten von Dateien, entfernbare Datenträger und die Realisierung mittels I-Nodes besprochen.

## Inhalt

1.Einleitung.....	3
2.Zugriffsrechte.....	4
3.Einfache Dateien.....	12
3.1.Links.....	13
3.2.Symbolische Links.....	14
4.Verzeichnisse.....	16
4.1.Aufbau des Filesystems.....	17
5.Spezialdateien.....	19
5.1.Blockorientierte Geräte.....	19
5.2.Zeichenorientierte Geräte.....	21

6.Entfernbarer Datenträger.....	23
6.1.Die Datei /etc/fstab.....	24
6.1.1.Die Datei /etc/mstab.....	31
7.Das I-Node.....	33
7.1.Der Superblock.....	36
8.Anlegen von Dateisystemen.....	38
9.Fehlerüberprüfung und Behebung im Dateisystem.....	39
9.1.Die Überprüfung beim Systemstart.....	40
10. Verschlüsselung von Dateisystemen.....	41

## 1. Einleitung

In Unix gibt es drei Arten von Dateien:

- einfache Dateien (ordinary files)
- Verzeichnisse (directories)
- Gerätedateien (special files)

Alle Daten, d.h. Texte, Programme, Bilder usw. werden in einfachen Dateien als Bytefolge abgespeichert.

Verzeichnisse sind Ansammlungen von Verweisen auf Dateien und selbst wieder Dateien.

Special Files sind periphere Geräte, wie Terminals, Drucken oder Plattenlaufwerke (genauer genommen, die Treiber, die die Geräte ansteuern),

Auf Shellebene gibt es keinen Unterschied zwischen diesen Dateiarten. Dies demonstriert das folgende Beispiel.

Drucken kann man mit:

```
$ lpr dat1
$
$ cp dat1 /dev/lp1
$
$ cat dat1 > /dev/lp1
$
```

## 2. Zugriffsrechte

Der Besitzer einer Datei muss in der Lage sein, Dateien vor dem Zugriff anderer zu schützen. Dies wird durch die Zugriffsrechte bewerkstelligt. Dabei wird für jede Datei gespeichert, wer darauf zugreifen darf und welche Rechte er bekommt.

Die Zugriffsrechte werden für drei verschiedene Nutzergruppen getrennt vergeben.

- **Eigentümer** (User/Owner; Symbol: u)  
Das ist der Nutzer, der die Datei erstellt hat. Nur dieser Benutzer kann die Zugriffsrechte ändern. (Zusätzlich darf der Superuser/Root Zugriffsrechte für alle Dateien ändern).
- **Gruppe** (Group; Symbol: g)  
Das ist die Gruppe, dem der Benutzer zugeordnet ist. Man kann so Dateien einer bestimmten Gruppe von Personen zugänglich machen.
- **Alle anderen** (Others; Symbol: o)  
Damit sind alle anderen Nutzer gemeint. Oft spricht man auch von "der ganzen Welt", bezogen auf das Internet. Denn nur die Dateien, die für "Others" freigegeben sind, kann man auch über das Internet aufrufen. Ein Zugriff über das Internet wird wie ein einfacher Nutzer ohne Rechte behandelt. Ähnlich einem anonymen Nutzer eines FTP-Servers.

Es gibt drei verschiedene Rechte:

- die Datei **lesen**,
- die Datei **schreiben** (ändern) und
- die Datei **ausführen** (execute).

Jedes Recht kann jedem der drei Nutzer-Gruppen zugeordnet werden.

Ein Beispiel: der Besitzer darf lesen, schreiben, ausführen, die Gruppe und alle anderen dürfen nur lesen und ausführen. Oder: der Besitzer darf lesen, schreiben, ausführen, die Gruppe und alle anderen dürfen nur lesen.

Aber die Zugriffskontrolle betrifft nicht nur Dateien: Auch für Verzeichnisse lassen sich die Rechte festlegen. Dabei sind ein paar Dinge beachten, denn es gibt Besonderheiten.

Wenn ein Verzeichnis nur das Lesen-Recht oder nur das Ausführen-Recht hat, dann dürfen andere Nutzer die Dateien in diesem Verzeichnis nicht lesen. Denn zum Lesen muss das Verzeichnis geöffnet (gelesen) werden.

Damit Dateien aus einem Verzeichnis generell gelesen werden können, müssen also beide Rechte gesetzt sein (Lesen und Ausführen).

Die Zugriffsrechte einer Datei oder eines Verzeichnisses können durch das Kommando `chmod` geändert werden.. Der Befehl "`chmod`" ist eine Abkürzung von "**change mode**". Die Eingabe des Befehls muss in folgender Syntax erfolgen:

```
chmod modus datei
```

Mit 'modus' sind die Zugriffsrechte gemeint, 'datei' ist der Namen der betreffenden Datei, deren Zugriffsrechte geändert werden sollen. Die Zuordnung kann in zwei verschiedenen Varianten angegeben:

### Erste Variante: **Oktal-Zahl**

```
$ chmod 755 dat  
$
```

Hierbei ist eine 3-stellige Zahl anzugeben, das ist die am häufigsten auftretende Option. Für jede Nutzergruppe (Eigentümer, Gruppe, Andere) gibt es eine Stelle. Um die Rechte zu definieren wird ein spezielles System verwendet.

Jedes Recht hat eine eigene Ziffer. Um mehrere Rechte einer Nutzergruppe zuzuordnen, müssen Sie die Ziffern der einzelnen Rechte addieren, für jeden Nutzer getrennt. In der nachfolgenden Tabelle finden Sie die entsprechenden Werte.

Recht:	Lesen	Schreiben	Ausführen
	r	w	X

Ziffer:	4	2	1
---------	---	---	---

Wollen Sie zum Beispiel einem Benutzer gleichzeitig Lesen und Ausführen zuordnen, addieren Sie "4 + 1 = 5". Insgesamt können bei der Addierung der Rechte Ziffern von 0 bis 7 entstehen, wobei die Zuordnungen immer eindeutig sind:

Die drei Ziffern schreibt man nun in der folgenden Reihenfolge hintereinander. Zuerst die Ziffer für den Eigentümer (USER), dann für die Gruppe (GROUP), zuletzt für die Anderen (OTHERS). Der Befehl "chmod 777 datei.htm" weist alle Rechte allen Nutzergruppen zu.

### Zweite Variante: **Symbol-Modus**

```
$ chmod ugo=rwx datei  
$
```

Hierbei spart man sich das Rechnen, müssen aber mehr eingeben. Außerdem kann man diese Variante nicht in allen FTP-Programmen nutzen.

Für die Nutzergruppen werden folgenden Buchstaben u=USER, g=GROUP und o=OTHERS verwendet. Die Rechte stellt man auch durch Buchstaben dar. Dabei steht "r" für Lesen (read), "w" für Schreiben (write) und "x" für Ausführen (execute).

Dieser Befehl in unserem Beispiel weist der Datei wieder alle Zugriffsrechte für alle Nutzer zu. Sie können auch einzelne Nutzergruppen und Rechte weglassen.

Um verschiedenen Nutzergruppen unterschiedliche Rechte zuzuweisen, kann man mehrere Angaben machen, die mit einem Komma getrennt werden.

```
$ chmod u=rwx,g=rx,o=r datei
$
```

Dieser Befehl ist gleichbedeutend mit "chmod 754". Sie können auch anstatt des "ist gleich" ein Plus- oder Minus-Zeichen verwenden. Mit Plus weisen Sie Rechte zu, und mit dem Minus-Zeichen entziehen Sie die Rechte wieder.

```
$ chmod u+w,g-w,o-wx datei
$
```

Dieser Befehl erteilt USER zusätzlich das Recht "Schreiben", Group entzieht er das Recht "Schreiben" und allen Anderen entzieht er die Rechte "Schreiben" und "Ausführen".

Beispiele der Verwendung:

```
$ ls -l dat
-rw----- 1 as      users      53 Dec 23 13:45 dat
$
```

Die Datei ist also nur für den Besitzer lesbar.

```

$ chmod 640 dat
$ ls -l dat
-rw-r-----  1 as      users      53 Dec 23 13:45 dat
$
$ chmod 000 dat
$ ls -l dat
-----  1 as      users      53 Dec 23 13:45 dat
$
$ chmod u+rw,g+r dat
$ ls -l dat
-rw-r-----  1 as      users      53 Dec 23 13:45 dat
$

```

Besitzt ein Benutzer Schreibrecht für ein Verzeichnis, kann er auch Dateien in diesem Verzeichnis löschen, für die er über keinerlei Rechte verfügt (*es wird ja nicht die Datei selbst gelesen oder verändert, sondern der Verweis auf diese Datei im Verzeichnis wird gelöscht*).

Um dies zu verhindern, kann das Verzeichnis mit dem Attribut `t` (dem **sticky bit**) versehen werden:

```

$ chmod +t verzeichnis
$

```

In diesem Verzeichnis können dann nur Dateien gelöscht werden, falls

- der Benutzer Eigentümer der Datei ist oder

- der Benutzer Eigentümer des Verzeichnisses ist oder
- der Benutzer Schreibrecht für die Datei besitzt

Dies ist die Voreinstellung des Verzeichnisses „/tmp“.

Beispiel:

```
$ mkdir tmp
$ ls -ld tmp
drwxrwxr-x  2 as      users      4096 Dec 23 13:55 tmp
$ chmod +t tmp
$ ls -ld tmp
drwxrwxr-t  2 as      users      4096 Dec 23 13:54 tmp
$
```

Neben diesen Schutzbits gibt es noch ein so genanntes **Set-User-ID-Bit** (s-Bit). Normalerweise wird ein Programm beim Aufruf mit den Rechten des Aufrufers ausgeführt. Damit kann das cp-Kommando z.B. auf die Dateien zugreifen, die der Benutzer des Kommandos hat. Das cp-Kommando gehört dem Systemverwalter. Ist für eine Datei das s-Bit gesetzt, so wird das Programm mit den Rechten des Eigentümers der Datei ausgeführt. Somit erhält der Aufrufer für die Zeit der Programmausführung die Rechte des Eigentümers der Datei.

Dies wird vom **passwd**-Kommando verwendet, damit der Aufrufer sein Passwort in der ansonsten geschützten Datei /etc/passwd ändern kann.

```
$ ls -l /usr/bin/passwd
-rwsr-xr-x  1 root      shadow      27456 Nov 13  1999 /usr/bin/passwd
$
$ chmod 000 dat
$ chmod 4755 dat
$ ls -l dat
-rwsr-xr-x  1 as       users      53 Dec 24 13:45 dat
$
```

### 3. Einfache Dateien

Einfache Files (regular files) enthalten Daten, die irgendwann von Benutzern definiert wurden. Dies können Texte, Programmquellen oder ausführbare Programme sein.

In Unix werden keine komplizierten Dateiformate oder Zugriffsmethoden verwaltet:

- eine Datei ist einfach eine Folge von Zeichen,
- ein File kann in beliebig großen Abschnitten byteweise von vorn nach hinter gelesen und geschrieben werden,
- gezielt kann auf ein bestimmtes Byte positioniert werden,
- Speicherplatz braucht für eine Datei nicht explizit reserviert zu werden, wenn hinter das letzte Zeichen der Datei geschrieben wird, so wird die Datei automatisch vergrößert.

Durch „>>“ kann man an eine Datei am Ende einfügen:

```
$ cat dat
Dies ist ein Text mit 3 Zeilen
2. Zeile
Letzte Zeile
$
$ echo neue letzte Zeile >> dat
$
$ cat dat
Dies ist ein Text mit 3 Zeilen
2. Zeile
Letzte Zeile
neue letzte Zeile
$
```

### 3.1.Links

Unter Unix existieren für Dateien so genannte Links. Unter diesen Links ist etwas Ähnliches zu verstehen, wie unter den Verknüpfungen von Windows 95/98. Die Tatsache, dass Dateien mehrere Namen besitzen können wurde unter Unix schon sehr früh eingeführt, es handelt sich hierbei eigentlich um mehrere Dateinamen im Inhaltsverzeichnis einer Platte, die auf die selben Blöcke zeigen. Unter Unix spricht man hier von so genannten Hard-Links.

Darunter ist tatsächlich nichts anderes zu verstehen, als ein **zweiter Name** für ein und dieselbe Datei. Wenn eine Datei zwei Namen hat, so kann nicht zwischen Original und Link unterschieden werden. Es ist möglich, eine der beiden Dateien zu löschen und trotzdem ist die andere noch vorhanden. Physikalisch handelt es sich aber immer um die selbe Datei, d.h., sie nimmt nur einmal den Speicherplatz auf der Platte in Anspruch.

Diese Tatsache erklärt aber auch die Einschränkung, der ein Hardlink unterworfen ist. Der Link muss sich auf der selben Partition befinden, auf der sich die Originaldatei befindet. Das liegt in der Natur des Links, weil er ja nur ein weiterer Name für die Datei ist, der im Inhaltsverzeichnis der Platte bzw. der Partition gespeichert ist.

Hardlinks werden mit dem Befehl `ln` angelegt.

```
$ ls -l dat*
-rwxr-xr-x  1 as      users      71 Dec 23 14:17 dat
-rw-rw-r--  1 as      users      53 Dec 23 11:28 dat2
$
$ ln dat dat_hl
$ ls -l dat*
-rwxr-xr-x  2 as      users      71 Dec 23 14:17 dat
-rw-rw-r--  1 as      users      53 Dec 23 11:28 dat2
-rwxr-xr-x  2 as      users      71 Dec 23 14:17 dat_hl
$
```

### 3.2.Symbolische Links

Im Gegensatz zu Hardlinks sind symbolische Links wesentlich flexibler, aber auch fehleranfälliger. Hier handelt es sich tatsächlich um Verweise auf eine bestehende Datei, d.h., der symbolische Link ist in Wahrheit eine Datei, die nur den Dateinamen einer anderen Datei enthält. Damit das System diese Datei nicht als Textdatei mißinterpretiert wird sie als SymLink gekennzeichnet, als Dateityp steht ein `l` statt eines Bindestrichs.

Durch diese Tatsache kann man beim SymLink immer zwischen Original und Link unterscheiden, im Gegensatz zum Hardlink, bei dem das nicht möglich ist. Symbolische Links erlauben es auch, Verweise über die Partitions Grenzen hinweg zu erstellen. Allerdings ist nicht sichergestellt, dass der Link auch auf etwas zeigt, was existiert. Wird z.B. ein SymLink auf eine Datei angelegt und anschließend die Datei gelöscht, so zeigt der Link ins Leere.

Als weiterer Vorteil von SymLinks ist noch die Tatsache zu nennen, dass sie auch auf Verzeichnisse zeigen können. Damit ist es also etwa möglich, ein Verzeichnis, das nicht mehr auf die Partition passt, in einer anderen Partition abzuspeichern und in der ursprünglichen Partition einfach einen SymLink auf die neue Position zu setzen.

Symbolische Links werden wie Hardlinks mit dem Befehl `ln` angelegt. Zur Unterscheidung wird dort einfach der Parameter `-s` (für symbolischen Link) benutzt.

```
$ ln -s dat dat_sl
$ ls -l dat*
-rwxr-xr-x  2 as      users      71 Dec 23 14:17 dat
-rwxr-xr-x  2 as      users      71 Dec 23 14:17 dat_l
lrwxrwxrwx  1 as      users      3 Dec 23 14:49 dat_sl -> dat
$
```

## 4. Verzeichnisse

Verzeichnisse sind Inhaltsverzeichnisse über eine Menge von Dateien. Diese Dateien können selbst wieder Verzeichnisse enthalten, wodurch ein hierarchisches Filesystem entsteht.

Mit dem Kommando `mkdir` kann ein neues Verzeichnis angelegt werden. `pwd` zeigt das aktuelle Verzeichnis an.

```
$ $ pwd
/home/as/Vorlesungen/Unix
$ mkdir tmpDir
$ ls -ld tmpDir
drwxr-xr-x  2 as      staff      512 Dec 23 14:44 tmpDir
$
```

Das Kommando `cd` ist zum Wechseln des aktuellen Verzeichnisses da.

```
$ cd tmpDir
$ ls -al
drwxr-xr-x  2 as      staff      512 Dec 23 14:44 .
drwxr-xr-x  6 as      staff      512 Dec 23 14:44 ..
$
```

Wird ein neues Verzeichnis angelegt, so werden automatisch zwei Dateien erzeugt:

die Datei mit Namen „.“ ist ein Verweis auf das Verzeichnis selbst

die Datei mit Namen „..“ ist ein Verweis auf das Vater-Verzeichnis.

## 4.1.Aufbau des Filesystems

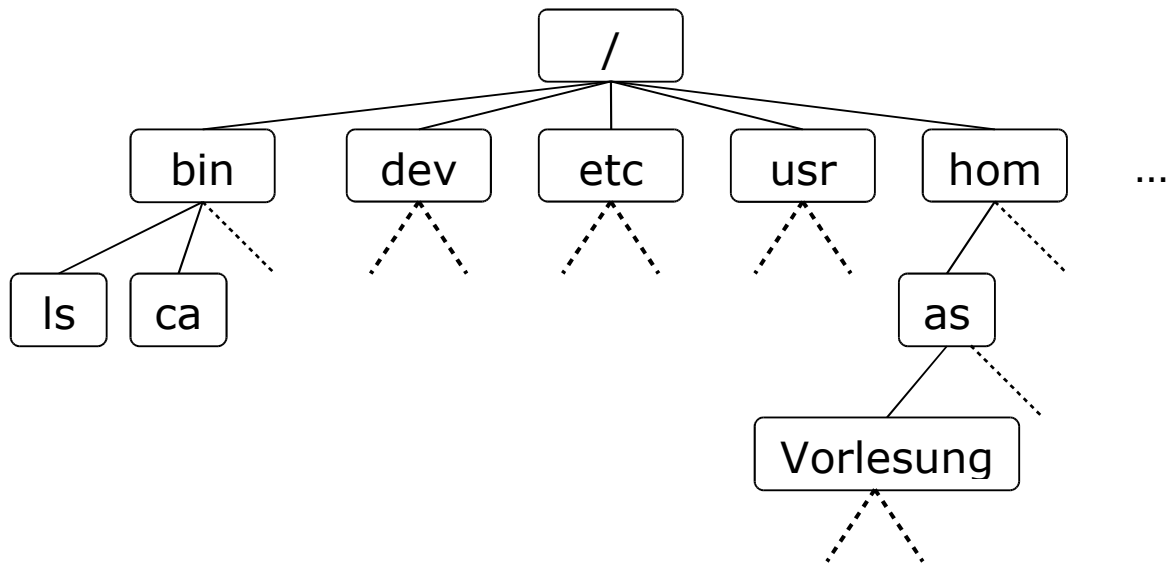
Die Wurzel des Dateisystems nennt man root, sie wird mit „/“ bezeichnet. Im Verzeichnis / existieren mehrere Verzeichnisse, z.B. home für die Homeverzeichnisse der Benutzer, bin für ausführbare Dateien, die zum Lieferumfang des Unix Systems gehören.

Um Verzeichnisnamen und enthaltenen Dateinamen zu trennen wird das Zeichen „/“ verwendet.

Dateien im aktuellen Verzeichnis kann man mit ihrem Namen ansprechen, Dateien außerhalb des aktuellen Verzeichnisses sind über ihren **Pfadnamen** anzusprechen. Dies ist der Weg von der Wurzel über alle Verzeichnisse hinweg bis zum aktuellen Verzeichnis, in dem die Datei steht inklusive des Dateinamens.

```
$ cd /  
$ cat /home/as/Vorlesungen/Unix/dat  
Test  
$
```

Die folgende Übersicht zeigt ein typische Unix Dateisystem:



## 5. Spezialdateien

Unix betrachtet auch jedes Stück Hardware als Datei. Dabei wird zwischen block- und zeichenorientierten Geräten unterschieden.

### 5.1. Blockorientierte Geräte

Als blockorientiertes Gerät gilt jedes Gerät, das nicht einzelne Zeichen verarbeitet, sondern ganze Blocks. Dabei handelt es sich z.B. typischerweise um Festplatten, Disketten und andere Laufwerke. Diese Laufwerke werden dann als Dateien verwaltet, für den Anwender bleibt der Unterschied zwischen Gerät und Datei nahezu transparent.

Die Gerätedateien (engl. special files), sowohl die block- wie auch die zeichenorientierten befinden sich im Verzeichnis **/dev**. Sie belegen physikalisch keinen Platz auf der Festplatte, sind also selbst nicht die Gerätetreiber, wie oft fälschlicherweise gedacht wird. Diese Dateien haben eigentlich nur zwei Nummern, die die Adresse des Gerätetreibers im Kernel symbolisieren. Die erste Nummer, auch major number genannt zeigt, um welchen Gerätetreiber es sich handelt, die zweite (minor number) beschreibt, das wievielte Gerät es ist, das diesen Treiber benutzt.

Als Beispiel folgt hier die Ausgabe des `ls -l` Befehls auf einige Gerätedateien des Blocktyps:

```

                Major Minor
                V      V
brw-rw----  1 root   disk    1,    0 Apr  5 01:12 ram0
brw-rw----  1 root   disk    1,    1 Apr  5 01:12 ram1
brw-rw----  1 root   disk    1,    2 Apr  5 01:12 ram2
```

```

brw-rw---- 1 root   disk    1,   3 Apr  5 01:12 ram3
...
brw-rw-rw- 1 root   disk    2,   0 Apr  5 01:12 fd0
brw-rw-rw- 1 root   disk    2,   1 Apr  5 01:12 fd1
...
brw-rw---- 1 root   disk    3,   0 Apr  5 01:12 hda
brw-rw---- 1 root   disk    3,   1 Apr  5 01:12 hda1
brw-rw---- 1 root   disk    3,   2 Apr  5 01:12 hda2
brw-rw---- 1 root   disk    3,   3 Apr  5 01:12 hda3
brw-rw---- 1 root   disk    3,   4 Apr  5 01:12 hda4
brw-rw---- 1 root   disk    3,   5 Apr  5 01:12 hda5
...
brw-rw---- 1 root   disk    3,  64 Apr  5 01:12 hdb
brw-rw---- 1 root   disk    3,  65 Apr  5 01:12 hdb1
brw-rw---- 1 root   disk    3,  66 Apr  5 01:12 hdb2
brw-rw---- 1 root   disk    3,  67 Apr  5 01:12 hdb3
...

```

Typische blockorientierte Geräte sind hier also der Arbeitsspeicher (ram1-4 Major-Number 1), die Diskettenlaufwerke (fd0, fd1 Major Number 2) und die Festplatten. (Major Number 3 für IDE-Platten) Bei den Festplatten wird unterschieden zwischen den ganzen Platten (hda, hdb, hdc, hdd) und den darauf befindlichen Partitionen (hda1, hda2, hdb1, hdb2,...)

Auf diese Gerätedateien kann zugegriffen werden, wie auf jede normale Datei, sie haben einen Zugriffsmodus (rwx...) wie normale Dateien und Eigentümer und Gruppenzugehörigkeit.

Damit ist es z.B. möglich den Diskcopy-Befehl mit dem Befehl [cat](#) wie folgt zu realisieren:

```
$ cat /dev/fd0 > Datei
$ cat Datei > /dev/fd0
```

Zunächst wird also alles aus der Datei `/dev/fd0` (erstes Floppy-Laufwerk) in die Datei `Datei` kopiert, im zweiten Befehl wird - nachdem eine Leerdiskette eingelegt wurde - der Inhalt der Datei wieder auf die Diskette geschrieben. Es wird hier nicht dateiorientiert gearbeitet, sondern wie bei DOS-diskcopy clusterorientiert.

## 5.2. Zeichenorientierte Geräte

Zeichenorientierte Geräte werden nicht blockweise angesteuert sondern durch einzelne Bytes. Typischerweise handelt es sich hier um serielle oder parallele Schnittstellen, die Terminalleitungen (auch die der virtuellen Terminals) oder den Soundanschluss.

Auch hier wieder ein paar Beispiele:

ISDN Anschlüsse

```
crw----- 1 root    root    45,    0 Apr   5 01:13 isdn0
crw----- 1 root    root    45,    1 Apr   5 01:13 isdn1
```

BIOS-ROM

```
crw----- 1 root    root    31,    0 Apr   5 01:13 rom0
crw----- 1 root    root    31,    1 Apr   5 01:13 rom1
```

Terminalleitungen für die virtuellen Terminals

```
crw-rw-rw- 1 root    root    5,     0 Apr   5 01:12 tty
crw--w--w- 1 root    tty     4,     0 Apr   5 01:12 tty0
crw----- 1 root    root    4,     1 Aug   8 11:39 tty1
```

Serielle Leitungen (COM1 und COM2)

```
crw-rw----  1 root      uucp          4,  64 Apr  5 01:12 ttyS0
crw-rw----  1 root      uucp          4,  65 Apr  5 01:12 ttyS1
```

Sowohl die block- als auch die zeichenorientierten Gerätedateien werden mittels dem Befehl **mknod** angelegt. Die einzige Information, die man dazu braucht ist die der Major- und Minornummer und die Frage, ob es sich um ein block- oder zeichenorientiertes Gerät handelt.

## 6. Entfernbare Datenträger

Im Gegensatz zu anderen Betriebssystemen fasst ein Unix-System grundsätzlich alle **Partitionen** und **angeschlossenen Laufwerke** zu einem **Dateibaum** zusammen. Dadurch entfallen die lästigen Laufwerksbuchstaben, die in größeren Netzwerken schnell an die Grenze der Übersichtlichkeit führen.

Physikalisch bleiben die Daten auf den entsprechenden Partitionen bzw. Laufwerken, logisch wird aber aus all den verschiedenen Partitionen ein einziger Dateibaum zusammengebaut. Das Prinzip ist dabei ganz einfach das, dass mit dem Befehl **mount** (montiere) die Partitionen an einem bestimmten Punkt des Dateisystems eingehängt werden. Als ein solcher **Mountpoint** dient jeweils ein leeres Verzeichnis. (Falls das Verzeichnis nicht leer ist, so wird der Inhalt während des Zustands, in dem ein anderes Laufwerk in das Verzeichnis gemountet ist versteckt. Nach dem Wiederabhängen des Laufwerks mit dem Befehl **umount** wäre der Inhalt des Verzeichnisses wieder unverändert vorhanden).

Dieses Prinzip erfordert eine besondere Partition, nämlich die, die die eigentliche Wurzel des ganzen großen Dateibaums enthält. Diese Partition wird als Wurzelpartition (**root-partition**) bezeichnet und beim booten ins Verzeichnis `/` gemountet. Alle weiteren Partitionen werden dann in Verzeichnisse gemountet, die ihrerseits entweder auf der Wurzelpartition oder auf schon gemounteten anderen Partitionen liegen. Die Wurzelpartition muss zwingend einige Verzeichnisse enthalten, die nicht auf anderen Partitionen liegen dürfen. Ein simples Beispiel ist das Verzeichnis, das die Bauanleitung enthält, wohin die anderen Partitionen gemountet werden sollen (`/etc`) oder das Verzeichnis, das den `mount`-Befehl selber enthält (`/bin`).

In der Regel werden die ganzen Partitionen, die verwendet werden sollen bereits während des Bootvorgangs automatisch gemountet. Dazu muss das System aber eine entsprechende **Bauanleitung** besitzen, die klärt, in welche Verzeichnisse welche Partitionen gehängt werden sollen. Diese Bauanleitung liegt in der Datei **/etc/fstab**. Diese Datei kann zweckmäßigerweise auch Angaben zu den Partitionen oder Laufwerken beinhalten, die nicht automatisch gemountet werden sollen, wie etwa Wechselplatten oder CD-Rom-Laufwerke.

Durch das Kommando **mount** ohne Argumente sieht man die gerade gemounteten Dateisysteme:

```
$ mount
/proc on /proc read/write/setuid on Fri Dec 21 09:09:47 2001
/ on /dev/dsk/c0t0d0s0 read/write/setuid/largefiles on Fri Dec 21 09:09:47 2001
/usr on /dev/dsk/c0t0d0s6 read/write/setuid/largefiles on Fri Dec 21 09:09:47 2001
/dev/fd on fd read/write/setuid on Fri Dec 21 09:09:47 2001
/var on /dev/dsk/c0t0d0s4 read/write/setuid/largefiles on Fri Dec 21 09:09:47 2001
/opt on /dev/dsk/c0t8d0s6 read/write/setuid/largefiles on Fri Dec 21 09:09:49 2001
/opt1 on /dev/dsk/c0t8d0s7 read/write/setuid/largefiles on Fri Dec 21 09:09:49 2001
/tmp on swap read/write/setuid on Fri Dec 21 09:09:49 2001
/home/as on /export/home/users/as read/write/setuid on Tue Dec 25 12:17:47 2001
$
```

## 6.1. Die Datei **/etc/fstab**

Diese Datei hat einen recht einfachen Aufbau, wie immer bei Unix ist sie eine einfache Textdatei, jede Zeile beschreibt eine Partition bzw. ein Laufwerk. Die einzelnen Felder der Zeile sind durch Tabs oder Leerzeichen voneinander getrennt.

```
$ cat /etc/fstab
/dev/hda1      /boot        ext2         defaults    1 1
/dev/hda2      swap         swap         defaults    0 1
/dev/hda3      /            ext2         defaults    1 2
/dev/cdrom     /cdrom       iso9660      ro,noauto,user 0 0
/dev/fd0       /floppy      auto         noauto,user 0 0
/dev/hdd       /S.u.S.E.   9660        ro          0 0
$
```

Die Zeilen haben folgendes Format:

Gerätedatei Mountpoint Dateisystemtyp Optionen Dump Reihenfolge

## Gerätedatei

Hier steht die Gerätedatei des blockorientierten Geräts, das die Partition oder das Laufwerk bezeichnet. Es ist notwendig, den gesamten Pfad zu dieser Datei anzugeben also z.B. /dev/hda3.

Wenn es sich bei dem zu mountenden Dateisystem um ein Netzlaufwerk (NFS) handelt, dann steht in diesem Feld der Rechnername des Dateiservers, gefolgt von einem Doppelpunkt und dem Pfad zu dem freigegebenen Verzeichnis, z.B. hal.mydomain.de:usr/public

## Mountpoint

Hier steht das Verzeichnis, in das die Partition oder das Laufwerk eingehängt werden soll. Notwendig ist der absolute Pfad (mit führendem /)

## Dateisystemtyp

Hier wird der Typ des Dateisystems angegeben, das diese Partition bzw. dieses Laufwerk enthält. Der Kernel muss das jeweilige Dateisystem unterstützen, sonst kann es nicht gemountet werden. Typische Angaben sind hier:

### *minix*

Das Dateisystem des Minix-Systems. Wird manchmal noch für Disketten benutzt, für Festplatten ist es nicht mehr üblich.

### *ext*

Das erweiterte Minix-System, das von frühen Linux-Versionen benutzt wurde. Hat keine Bedeutung mehr und wird wahrscheinlich bald komplett aus dem Kernel entfernt.

### *ext2*

Das heute **übliche Linux-Dateisystem**. Wird für alle Dateisysteme von Festplatten und Wechselmedien (Zip o.ä.) benutzt. ext2 ist das Dateisystem mit der besten Performance, das für Linux zur Verfügung steht.

### *xiafs*

Ein weiteres Linux-Dateisystem, wird nicht mehr benutzt, seit ext2 sich durchgesetzt hat. Aus Kompatibilitätsgründen noch im System enthalten.

### *msdos*

Das MSDOS Dateisystem. Damit können auch DOS-Platten unter Linux angesprochen werden. Achtung, es handelt sich hier um das alte DOS-Dateisystem mit der Dateinamenbeschränkung auf das 8.3 Prinzip.

### *umsdos*

Das Dateisystem von Linux, wenn Linux auf DOS-Partitionen betrieben werden soll. Nur nötig, wenn DOS und Linux auf einer Partition zusammen laufen sollen.

### *vfat*

Das moderne Windows Dateisystem, das schon lange Dateinamen kennt.

### *proc*

Ein Pseudo Dateisystem, das als Schnittstelle zum Kernel dient. Die Dateien, die hier zu finden sind benutzen physikalisch keinen Platz auf irgendwelchen Platten sondern sind direkte Ausgaben des Kernels.

### *iso9660*

Das Dateisystem von CD-ROM Laufwerken. Es werden neben dem ursprünglichen ISO System auch die sogenannten Rock Ridge Extensions unterstützt, die ein Dateisystem mit langen Dateinamen, Eigentümer und Gruppenzugehörigkeit wie unter Unix üblich ermöglichen.

### *hpfs*

Das High Performance Filesystem von OS/2. Weil es dazu keine ausreichenden Dokumentationen von IBM gibt, kann dieser Dateisystemtyp nur als Read-Only System gemountet werden.

### *sysv*

Das Dateisystem der Unix System V Version. Für Platten, die aus solchen Systemen ausgebaut wurden...

### *nfs*

Das **Network File System**, die unter Unix übliche Art, Dateisysteme übers Netz freizugeben.

Welche Dateisysteme vom Kernel aktuell unterstützt werden kann in der Datei `/proc/filesystems` nachgesehen werden. Sollten die benötigten Systeme hier nicht zu finden sein, so können sie als Modul nachgeladen werden oder es muß ein neuer Kernel kompiliert werden. Es ist mit speziellen Mount-Befehlen auch möglich, smb und ncpfs Dateisysteme zu mounten, experimentell existieren auch schon Treiber für ntfs.

Wenn in diesem Feld statt einem Dateisystemtyp *ignore* angegeben ist, wird der Eintrag ignoriert. Dies ist nützlich, wenn unbenutzte Partitionen angezeigt werden sollen.

## **Optionen**

Hier können verschiedene Optionen gesetzt werden, die von Dateisystem zu Dateisystem unterschiedlich sein können. Dieses Feld enthält eine durch Kommas (ohne Leerzeichen) getrennte Liste von solchen Optionen. Welche Optionen zur Verfügung stehen ist für nicht

NFS-Dateisysteme auf der Handbuchseite von mount(8) nachzulesen, für NFS-Dateisysteme unter nfs(5)

Ein paar wichtige Optionen werden hier genauer dargestellt:

### *defaults*

Benutzt die voreingestellten Optionen *rw*, *suid*, *dev*, *exec*, *auto*, *nouser*, und *async*.

### *dev*

Erlaubt es, Gerätedateien auf der Partition zu nutzen

### *nodev*

Erlaubt es nicht, Gerätedateien auf der Partition zu nutzen

### *exec*

Erlaubt es, binäre Programme auf der Partition auszuführen

### *noexec*

Erlaubt es nicht, binäre Programme auf der Partition auszuführen. Das kann praktisch sein, wenn zwei unterschiedliche Unix-Systeme installiert sind, deren Binärdateien nicht kompatibel sind. So kann es zu keinen Verwechslungen kommen.

### ***auto***

Partition wird automatisch beim Booten gemountet

*noauto*

Partition wird nicht automatisch beim Booten gemountet

*user*

Erlaubt es einem Normaluser das Dateisystem zu mounten. Normalerweise darf nur der Systemverwalter Dateisysteme mounten, für Wechselplatten, CD-ROM- und Diskettenlaufwerke ist es aber häufig nötig, dass auch Normaluser sie mounten können.

*rw*

Mountet das Dateisystem zum Lesen und Schreiben

*ro*

Mountet das Dateisystem ReadOnly

Jedes Dateisystem hat selbst noch einige spezielle Optionen für individuelle Einstellungen. Diese speziellen Optionen können in den jeweiligen Handbuchseiten und den Kernelquellen nachgelesen werden.

## **Dump**

Hier steht eine Ziffer, entweder 0 oder 1. Das Feld beschreibt, ob der Unix-Befehl dump das Dateisystem sichern soll oder nicht. Wird heute selten gebraucht, als Anhaltspunkt kann man sagen, dass alle Partitionen mit Unix-Dateisystemen (minix, ext, ext2, xiafs, sysv) hier eine 1 stehen haben sollten, alle anderen eine 0.

## **Reihenfolge**

Auch hier stehen Ziffern, entweder 0, 1 oder größer. Es handelt sich um die Angabe, in welcher Reihenfolge der Dateisystemcheck beim Booten vor sich gehen muss. Eine **0** bedeutet grundsätzlich **keinen Dateisystemcheck**, die 1 steht für die Wurzelpartition, die immer als erstes überprüft werden sollte, die anderen Ziffern legen die Reihenfolge der weiteren Überprüfung fest. Meist haben alle anderen Dateisysteme hier die 2 stehen, weil sie dann parallel überprüft werden.

Auch hier kann man vereinfacht sagen, dass für alle Dateisysteme, bei denen im vorherigen Feld eine 0 steht, auch hier eine 0 stehen sollte, die Partition, die die Wurzel des Dateisystems enthält bekommt eine 1, alle anderen eine 2.

### 6.1.1. Die Datei `/etc/mtab`

Jedesmal, wenn ein Dateisystem mit **mount** in den Dateibaum gehängt wurde, trägt der mount-Befehl dieses Dateisystem in die Datei `/etc/mtab` ein. Auch hier handelt es sich um eine reine Textdatei, die aber nicht geeignet ist, manuell verändert zu werden.

Diese Datei enthält also immer eine Liste der Dateisysteme, die gemountet sind. Wird der Befehl `mount` ohne jeden Parameter eingegeben, so liest er diese Datei und gibt den Inhalt in etwas aufbereiteter Form auf dem Bildschirm (genauer der Standard-Ausgabe) aus.

Wichtig ist diese Datei auch für das `umount`-Kommando, wenn es mit der Option `-a` angewandt wird, wenn also alle gemounteten Dateisysteme abgehängt werden sollen. Meist ist das nur beim Herunterfahren des Systems notwendig und sinnvoll. Hier findet `umount` die Informationen, welche Dateisysteme gemountet sind, welche also abgehängt werden müssen.

Natürlich werden die entsprechenden Einträge in dieser Datei bei jedem umount auch modifiziert, d.h., wenn ein Dateisystem per umount abgehängt wird, so löscht umount den entsprechenden Eintrag in der Datei /etc/mtab.

## 7. Das I-Nodes

**Unix-Dateisysteme** sind nach einem anderen Prinzip aufgebaut, als etwa DOS-FAT-Systeme. Es gibt zwar viele verschiedene Dateisysteme unter Unix, gemeinsam haben sie jedoch eben das Prinzip der Funktionsweise. Und dieses **Prinzip** beruht auf den sogenannten **I-Nodes**.

Jede Partition enthält ein Dateisystem, dieses Dateisystem wiederum enthält eine Art **Inhaltsverzeichnis**, die **I-Node-Liste**. Die einzelnen Elemente der *I-Node-Liste* sind die Dateiköpfe, also die Orte wo Dateiattribute, Größe usw. gespeichert sind. Diese Dateiköpfe werden **I-Nodes** genannt.

Wie unter DOS auch, so verwalten Unix-Dateisysteme nicht zwangsläufig die Sektoren einer Festplattenpartition sondern Zuordnungseinheiten, die hier aber nicht Cluster sondern **Block** heißen. Beim Anlegen eines Dateisystems kann die Blockgröße angegeben werden, die auf dieser Partition verwendet werden soll. Typische Blockgrößen sind 512, 1024 oder 2048 Byte. Voreingestellt sind meist 1024 Byte pro Block.

Anders als unter DOS werden diese Blöcke aber nicht in einer Tabelle (FAT) zusammengefasst, sondern die I-Nodes enthalten selbst die Verweise auf diese Blöcke. Das Format eines typischen I-Nodes sieht etwa so aus:

Typ und Zugriffsrechte
Anzahl der Hardlinks
Benutzernummer (UID)

Gruppennummer (GID)
Größe der Datei in Bytes
Datum der letzten Veränderung (mtime)
Datum der letzten Statusänderung (ctime)
Datum des letzten Zugriffs (atime)
Adresse von Datenblock 0
...
Adresse von Datenblock 9
Adresse des ersten Indirektionsblocks
Adresse des Zweifach-Indirektionsblocks
Adresse des Dreifach Indirektionsblocks

Nach den Datumsfeldern stehen **zehn Felder**, die direkt die **Adressen der Datenblöcke** beinhalten können. Benutzt die Datei weniger Platz sind die Felder einfach leer.

Ist die **Datei größer als 10 Blöcke** (also größer als  $10 \cdot 1024$  oder  $10 \cdot 2048$  oder  $10 \cdot 512$ ), so enthält das nächste Feld der I-Node eine **Adresse eines Blockes, der wiederum bis zu 128 Adressen anderer Blöcke enthalten kann**. Sollte das auch noch nicht ausreichen, so enthält der nächste I-Node-Eintrag eine Adresse eines **Zweifach-Indirektionsblocks**, eines Blocks, der bis zu 128 Adressen auf Blöcke mit wiederum 128 Adressen enthält. Und falls auch das noch zu wenig sein sollte, so enthält der nächste Eintrag die Adresse eines Blockes, der wiederum 128 Adressen von Zweifach-Indirektionsblöcken enthalten kann. Damit sind dann **Dateigrößen von 1, 2 oder 4 Gigabyte** (je nach Blockgröße von 512, 1024 oder 2048 Byte) möglich.

Zu beachten ist, dass der Dateiname nicht in der I-Node auftaucht. Die I-Node ist sozusagen nur die Referenz auf die Datenblöcke, die eine Datei benutzt und der Ort, an dem die Attribute wie Eigentümer, Gruppe, Größe und Zugriffsdaten gespeichert sind.

Je nach verwendetem Dateisystem liegt das Wurzelverzeichnis auf einer festgelegten I-Node, meist 1 oder 2. Grundsätzlich ist es aber dem Dateisystem bekannt, welche I-Node das Wurzelverzeichnis enthält.

Jedes **Verzeichnis** (Ordner, Directory) - auch das Wurzelverzeichnis ist unter Unix nichts anderes als eine Datei, deren Inhalt einfach die **Dateinamen** der **enthaltenen Dateien samt ihren I-Node-Nummern** enthält. Damit wird auch klar, warum Unix kein Problem mit Hard-Links hat, also mit Dateien mit mehreren Namen. Es handelt sich ja nur um verschiedene Namenseinträge, die eben die selbe I-Node-Nummer besitzen.

Das Standard Linux-Dateisystem ext2 hat zusätzlich zu den gezeigten I-Node Einträgen noch verschiedene andere, die das System in mancherlei Hinsicht noch leistungsfähiger macht. Für den Anwender ergeben sich dadurch keinerlei Veränderungen, in der Praxis sind aber einige positive Effekte feststellbar.

So benutzt das ext2 System beispielsweise bis zu 12 direkte Datenblockadressen, es hat noch ein zusätzliches Datumsfeld für das Datum des Löschens der Datei (für später zu entwickelnde Undelete-Funktion) und es bietet weitere Attribute, die hier nicht genauer dargestellt werden sollen weil das den allgemeinen Unix-Rahmen dieses Kurses sprengen würde.

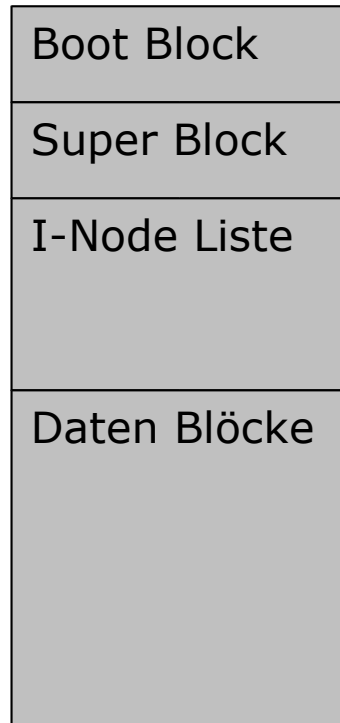
## 7.1. Der Superblock

Ein **Unix-Dateisystem** besitzt einen so genannten **Superblock**, einen Block, der die grundlegenden Informationen zum Dateisystem selbst enthält. Einige wichtige Daten des Superblocks sind:

- Die Größe des Dateisystems in Blöcken
- Die Größe der Blöcke in Bytes
- Zeiger auf den ersten freien Datenblock
- Zeiger auf erste freie I-Node
- Verschiedene Statusbits (Flags)

Auch hier unterscheiden sich die verschiedenen Unix-Dateisysteme voneinander, was an zusätzlichen Informationen im Superblock gespeichert ist. Das wesentliche an dieser Struktur ist, dass der **Superblock beim Mounten** eines Dateisystems in den **Speicher** gelesen wird und **alle Veränderungen dort vorgenommen** werden. Erst beim **Wiederabhängen** des Dateisystems werden diese **Veränderungen physikalisch auf der Platte** gespeichert. Das erklärt auch, dass es nach einem Systemabsturz zu Inkonsistenzen in einem Dateisystem kommen kann.

Noch vor dem Superblock steht der so genannte **Bootblock** auf der Platte, der in etwa dem Bootsektor der DOS-Partitionen entspricht. Zusammengefasst kann man also ein Unix-Dateisystem etwa wie folgt darstellen:



## 8. Anlegen von Dateisystemen

Bevor eine Partition benutzt werden kann, muss auf ihr zunächst ein **Dateisystem angelegt** werden. Unter DOS/Windows wird dieser Vorgang **formatieren** genannt, unter Unix heißt er korrekterweise Anlegen des Dateisystems. Der Befehl, mit dem dieser Vorgang gestartet wird ist mkfs (Make Filesystem).

Folgendes Kommando legt ein Dateisystem auf einer Diskette an.

```
$ /etc/mkfs /dev/dsk/f0q15dt 2200
$
```

Der letzte Parameter des mkfs Kommandos legt die Anzahl der Blöcke fest, aus denen das Filesystem besteht.

## 9. Fehlerüberprüfung und Behebung im Dateisystem

Dateisysteme sind aus komplexen Zusammenhängen aufgebaut und haben für jede Veränderung eine ganze Reihe von Aktionen notwendig, bis die Veränderung tatsächlich fertig gestellt ist. Falls es während einer solchen Veränderung zu einem Systemabsturz, einem ungesicherten Stromausfall o.ä. kommt, so kann das System die anfallenden Arbeiten nicht mehr korrekt ausführen - es entstehen Inkonsistenzen im Dateisystem.

Unix-Dateisysteme sind verhältnismäßig stabil im Gegensatz zu anderen Dateisystemen, aber trotzdem nicht gefeit gegen solche Fehler. Aus diesem Grund gibt es seit dem Anfang der Unix-Dateisysteme ein Programm, das die Systeme überprüft und gegebenenfalls repariert - *file system check* kurz **fsck**

In der Regel werden die **Dateisysteme beim Systemstart** geprüft, **bevor sie gemountet** sind. Das ist wichtig, weil das Überprüfen bereits gemounteter Systeme immer die Gefahr birgt, dass ein anderer User oder Prozess das System benutzt und verändert, während es überprüft wird. Dadurch können wieder neue Inkonsistenzen entstehen. Auch für eine manuelle Anwendung von fsck wird in den meisten Fällen das Dateisystem abgehängt und erst dann überprüft.

Eine Sonderrolle spielt in diesem Zusammenhang die Überprüfung des **Wurzeldateisystems**. Dieses System kann ja nicht abgehängt werden, schon aus dem Grund, dass das Programm fsck auf der Wurzelpartition gespeichert ist. Die Überprüfung und Reparatur des Wurzeldateisystems sollte daher grundsätzlich nur im **Single-User Modus** durchgeführt werden und evt. sollte das Dateisystem mit dem Befehl

```
mount -o ro,remount Gerätedatei
```

als Read-Only neu gemountet werden. Damit sollten alle denkbaren Fehler ausgeschlossen sein. Beim Systemstart wird daher auch zuerst das Wurzelsystem Read-Only gemountet, überprüft und erst dann wieder als Read-Write gemountet.

### 9.1. Die Überprüfung beim Systemstart

Jedesmal, wenn das System neu gebootet wird, werden alle Dateisysteme überprüft. In der Regel wird aber nur überprüft, ob sie "sauber" sind - nur wenn nicht wird die eigentliche Überprüfung gestartet. Das funktioniert aber nur bei der Verwendung des Ext2-Dateisystemtyps.

Dieser Typ hat im Superblock ein so genanntes **Valid-Flag** (manchmal auch Clean-Flag genannt). Jedesmal, wenn ein Dateisystem **gemountet** wird, wird dieses Flag auf **0** gesetzt, erst beim ordentlichen Abhängen durch **umount** wird es wieder auf **1** gesetzt. Beim Systemstart überprüft das fsck-Programm also zunächst, ob dieses Flag auf 1 steht, wenn ja, so gilt das Dateisystem als sauber und wird nicht weiter überprüft (es sei denn die -f Option ist dem fsck.ext2 Programm mitgegeben).

Falls das System vorher nicht sauber heruntergefahren wurde, die Dateisysteme also das Clean-Flag nicht setzen konnten, erkennt das fsck Programm dies und überprüft die Dateisysteme.

Zusätzlich enthält der Superblock einer ext2-Partition noch einen Zähler, der beschreibt, wie oft das Dateisystem gemountet wurde, seit es das letzte Mal überprüft wurde. Nach 20 erfolgten Mounts wird das Dateisystem überprüft, auch wenn es das Clean-Flag gesetzt hat.

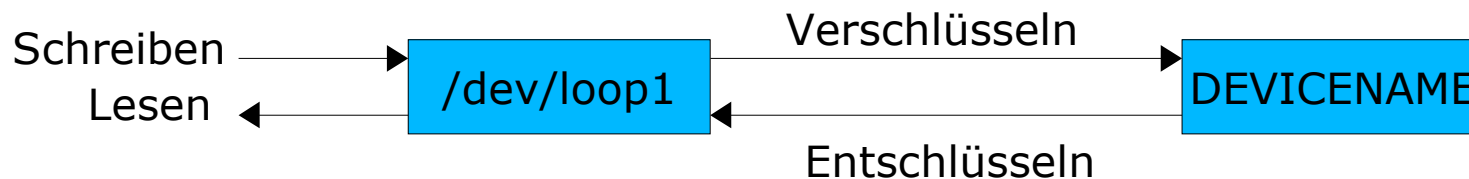
## 10. Verschlüsselung von Dateisystemen

Es existieren Dateisystemtypen, die die Informationen auf der Festplatte verschlüsseln. Seit dem **Linux Kernel 2.6** liefert ein Standard-API die notwendige Funktionalität: das **Crypto-API**. Es werden mehrere symmetrische Verfahren unterstützt mit 128 Bit- 256 Bit Schlüssellänge, z.B. DES, 3DES, Twofish.

Verwenden kann man das Crypto-API über ein Loopback-Device mittels des Kommandos `losetup`.

```
# losetup -e twofish-256 /dev/loop1 DEVICENAME
```

Dabei muss das als Passwort der zu verwendende Schlüssel angegeben werden. Danach wird jeder Zugriff auf das Device über Ver-/Entschlüsselung vollzogen.



Normalerweise ist das Device eine Festplattenpartition. Aber auch eine mittels `dd` angelegte Datei kann dazu verwendet werden.

Im folgenden Beispiel wird eine Datei angelegt und als verschlüsselte „Platte“ verwendet, auf die ein Filesystem erzeugt wird. Dieses Filesystem kann dann „gemounted“ werden und normal benutzt werden. Die Dateien mit ihren Inhalten sind darauf dann verschlüsselt abgelegt.

### 1. Anlegen einer Datei (/file), die als Partition dienen soll:

```
# dd if=/dev/zero of=/file bs=1k count=100
```

### 2. Initialisierung des Loopback-Devices:

```
# losetup -e des /dev/loop0 /file  
Password:  
Init (up to 16 hex digits):
```

Anstelle des Namens des Algorithmus ann eine Nummer angegeben werden, "-e 1" für XOR

### 3. Erstellen eines ext2-Dateisystems auf dem Loopback-Device:

```
# mkfs -t ext2 /dev/loop0 100
```

### 4. Das Dateisystem einhängen und verwenden:

```
# mount -t ext2 /dev/loop0 /mnt  
# echo a > a  
...  
# umount /dev/loop0  
# losetup -d /dev/loop0
```

Die Daten in der Datei /file sind verschlüsselt und nur entschlüsselt über das Loopback-Device lesbar.