# Linux-Dateisysteme

Seminar »Speicher- und Dateisysteme«

Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg

Autor: Lars Thoms <lars.thoms@spacecafe.org>
Betreuer: Dr. Julian Kunkel

Version: 2016-05-02

## Vorwort

Dateisysteme: Jeder braucht sie – wenige wissen, dass es sie gibt – kaum jemand weiß, wie sie funktionieren.

Diese Ausarbeitung zum Vortrag »Linux-Dateisysteme« im Rahmen des Seminars »Speicher- und Dateisysteme« erläutert Schritt für Schritt den Aufbau und die Funktionsweise eines Dateisystems und deren Komponenten.

Zu Beginn werden Partitionen, deren Verwaltung und Funktionsweise erläutert. Als Beispiel herangezogen werden sowohl das weit verbreitete MBR und das neuere GPT. Abschließend werden spezielle Konfigurationen, wie LVM und RAID betrachtet.

Mit diesen Grundlagen beginnt das eigentliche Thema. Die Definition, Semantik und das Konzept eines Dateisystems, gefolgt von einer Übersicht der bekannten Dateisysteme und deren Einordnung.

Die Funktionsweisen von Dateisystemen werden anhand von zwei bekannteren Beispielen erläutert: EXT4 und BTRFS. Garniert mit Code- und Befehlsschnippseln, welche die Erstellung, Administration und Funktionsweise der beiden Vertreter verdeutlichen sollen.

# Inhaltsverzeichnis

# 1 Partition(stabellen)

Bevor man Dateisysteme betrachtet, muss man zwangsläufig vorher die unteren Schichten betrachten, um ein vollständiges Verständnis von Speichermanagement zu erlangen.

Fast jeder Datenträger heutzutage besteht aus mindestens einer Partition mit einer dazugehörigen Partitionstabelle. Aber was ist eine Partition?

## 1.1 Definition »Partition«

> Unter einer Partition (lat. partitio = „(Ein)teilung“) versteht man einen **zusammenhängenden Teil** des Speicherplatzes eines geeigneten physischen oder logischen Datenträgers. Eine Partition ist ihrerseits ein **logischer Datenträger** [...].
>
> Partitionen sind voneinander weitgehend unabhängig und können von Betriebssystemen wie getrennte Laufwerke behandelt werden. [...] [1]

Eine Partition ist nichts anderes, als ein vorher festgelegter Teil eines Datenträgers. Dieser Bereich lässt sich sogar nachträglich ändern, sofern dies das Dateisystem wiederum zulässt. Diese Informationen müssen aber auf dem Datenträger an einem bekannten Ort hinterlegt werden: der Partitionstabelle.

## 1.2 Partitionstabelle

Die Partitionstabelle befindet sich immer am Beginn des Speicherbereiches des Datenträgers. Bei Festplatten wäre das die Adresse **CHS 0,0,1** (*Cylinder, Head, Sector*), was einen dreidimensionalen Vektor für deren geometrischer Aufbau darstellt. Anderenfalls benutzt man die Adresse LBA 0 (*Logical Block Addressing*).

In der Tabelle befinden sich Informationen über den Datenträger selbst und die erstellten Partitionen. Die Partitionen werden durch deren Startblock, Länge und ggf. mit Flags beschrieben.

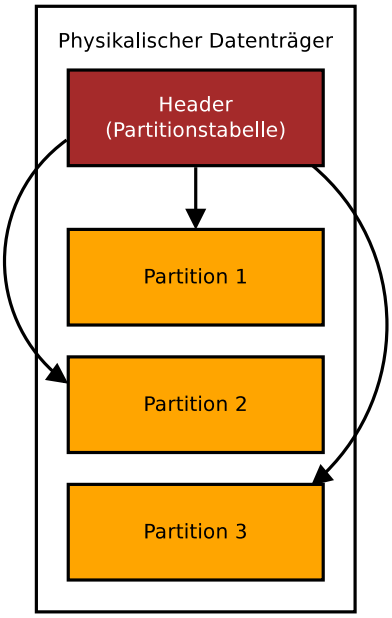


Abbildung 1: Aufteilung eines Datenträgers

## 1.3 MBR (eine Partitionstabelle)

Der **M**aster **B**oot **R**ecord ist eine alte, aber noch sehr häufig verwendete Partitionstabelle. Die ersten Computer mit MBR wurden 1983 durch **IBM-PC XT** und **MS-DOS/PC DOS 2.0** bereitgestellt [2].

[1] Quelle: Wikipedia [13]
[2] Quelle: Wikipedia [12]

Dadurch, dass er aus dieser Zeit stammt, ist der MBR nur 512 B lang (eine Blockgröße) und unterstützt nur vier primäre Partitionen und eine Partitionsgröße von 2 TB.

### 1.3.1 Aufbau des MBR

Den MBR kann in fünf Bereiche unterteilen, wobei der dritte aus einem großen ungenutzten Block (Farbe in der Abbildung 2: weiß) mit properitären Metadaten (grau und hellblau) besteht [3].

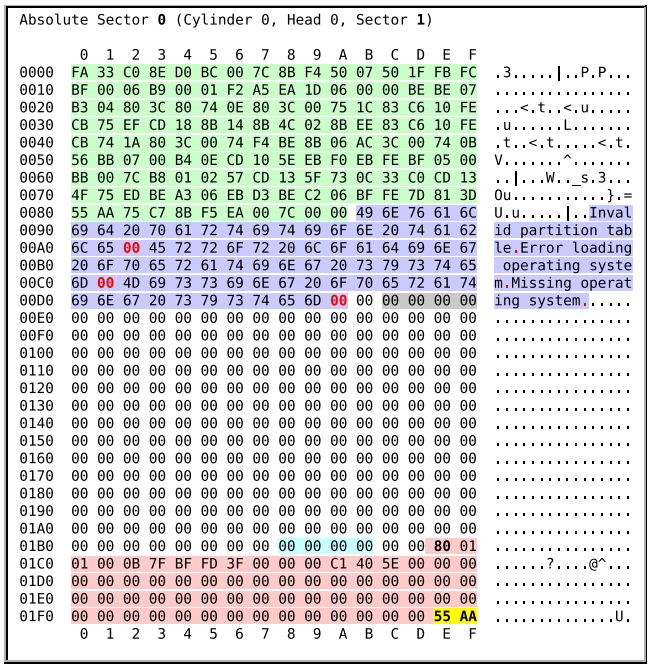


Abbildung 2: Hexdump eines MBR

Der erste Block beinhaltet den Bootloader, der für die Initializierung und das Parsen des restlichen Bereiches zuständig ist. Darauf folgt ein Ressourcenblock mit Fehlermeldungen, die im Falle dem Benutzer angezeigt werden können (z.B. »Invalid partition table«).

Die wichtige Partitionstabelle befindet sich im vierten Block und ist nur 64 B lang, dadurch bleibt pro Partition nur 16 B zum Adressieren des Bereiches übrig. Am Ende steht eine 2 B lange Signatur und schließt den MBR ab.

| Adresse | Farbe (Abbildung 2) | Funktion |
|---|---|---|
| 0x000 (139B) | grün | Bootloader Programmcode |
| 0x08B (80B) | blau | Fehlermeldung (String) |
| 0x1BE (64B) | rot | Partitionstabelle |
| 0x1FE (2B) | gelb | Bootsektor-Signatur |

[3] (Bild-)Quelle: An Examination of the Standard MBR [8]

## 1.4 Primäre/Logische Partition

Dadurch, dass der MBR nur vier primäre Partitionen ermöglicht, jedoch der Anspruch nach mehr Partitionen bestand, wurde als »Workaround« die logische Partition definiert.

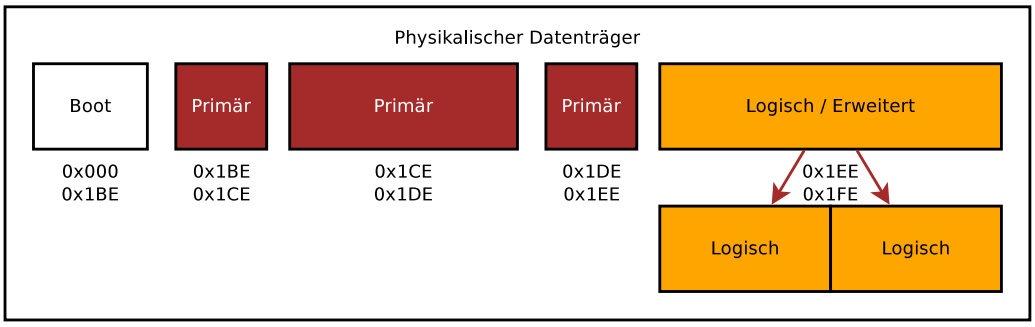


Abbildung 3: Aufteilung des MBR

Dabei wird im MBR anstatt einer Partitionsdefinition eine Adresse im Datenblock des Datenträgers hinterlegt, wo eine weitere Partitionstabelle mit Partitionsadressen zu finden ist.

In der Abbildung 3 sind die entsprechenden Offsets des MBR eingezeichnet und bei der vierten Partition handelt es sich um einen Container, der zwei weitere logische Partitionen enthält.

## 1.5 GPT (eine bessere Partitionstabelle)

Durch die massiven Einschränkungen des MBR wurde die **GUID Partition Table** entwickelt (*GUID: Globally Unique Identifier, engl.: global eindeutige Zahl*) und ist Teil des UEFI-Standards. Sie bietet dem Benutzer 128 Partitionen und eine gespiegelte, sekundäre GPT.

Auch die Partitionsgröße vergrößert sich bei einer Blockgröße von 512 B auf 8 ZiB, da 64 bit für die Adressierung der Blöcke zur Verfügung stehen.

$$512\,\mathrm{B}\ \text{(Blockgröße)} \cdot 2^{64}\ \text{(Adresslänge)} \approx 7.556 \times 10^{22}\,\mathrm{B} = 8\,\mathrm{ZiB}\ \text{(max. Partitionsgröße)}$$

Standardmäßig ist heutzutage eine Blockgröße von 4 KiB, was einer Partitionsgröße von 64 ZiB entspricht.

### 1.5.1 Aufbau der GPT

Die GUID Partition Table besteht aus einem **Protective MBR**, einem primären und einem sekundären Bereich [4].

An der Adresse **LBA 0** befindet sich aus Kompatibilitätsgründen immer noch ein MBR, welches normalerweise von den Partitionierungswerkzeugen mit angelegt und verwaltet wird, aber nicht Bestandteil der GPT ist.

[4] Bildquelle: Wikimedia-Commons [11]

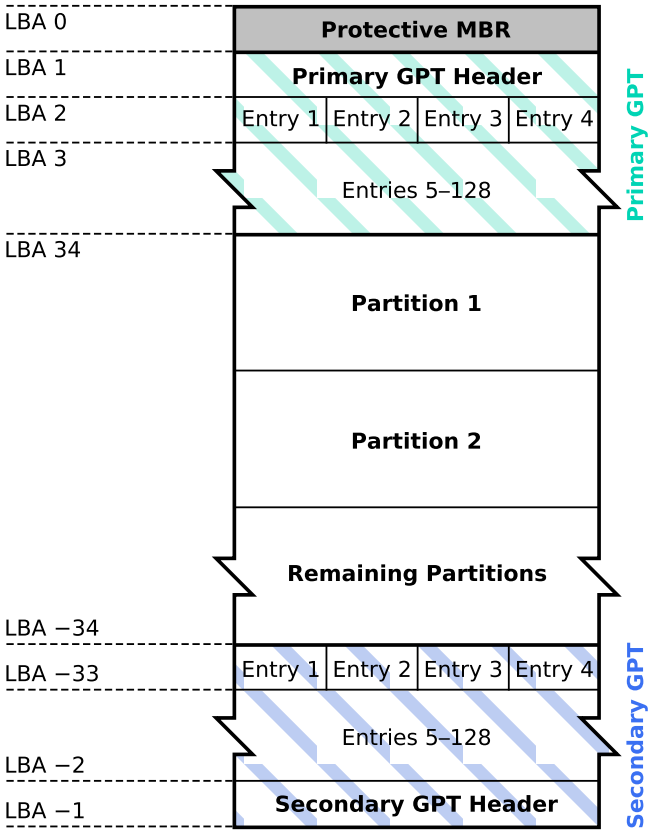


Abbildung 4: Aufbau der GPT

Die primäre GPT beginnt ab **LBA 1** mit den eigenen Headern, welcher unter anderem Metadaten über den physikalischen Datenträger, als auch beispielsweise die UUID (*Universally Unique Identifier*, engl.: universeller, eindeutiger Bezeichner), enthält.

Darauf folgt die Partitionstabelle (**LBA 2** bis **LBA 34**) mit 128 Datensätzen (á 128 B bei einer Blockgröße von 512 B). Danach beginnt der Datenblock mit den Partitionen, dem Dateisystem und den Daten der Benutzer.

Am Ende des Datenträgers befindet sich die **sekundäre** GPT. Dies ist ein Backup-Bereich und enthält, sofern keine Bitflips o.Ä. vorliegen, die gleichen Informationen. Er beginnt ab **LBA -34** und geht bis **LBA -1**. Ein weiteres Protective MBR existiert nicht.

### 1.5.2 GPT mit zwei Partitionen erstellen

Eine GPT kann unter Linux sehr einfach mit dem Kommandozeilenprogramm gdisk erstellt werden. Dafür muss mit mit Hilfe von lsblk der das richtige **Blockgerät** herausgesucht werden. In dem folgenden Beispiel handelt es sich um **/dev/sdc**.

Zu Beginn muss die Partitionstabelle erstellt werden:

```
$ sudo gdisk /dev/sdc
Partition table scan:
  MBR: not present
  BSD: not present
  APM: not present
  GPT: not present

> o
This option deletes all partitions and creates a new protective MBR.
```

Darauf folgend können eine oder mehere Partitionen angelegt werden. Dabei ist zu beachten, dass auf die Eingabe des ersten Sektors verzichtet werden kann, da gdisk immer lückenlos die nächste Partition anlegt. Der letzte Sektor wird relativ und nicht absolut zum ersten Sektor gewählt, z.B. 1 GiB.

```
> n
Partition number (1-128, default 1):
First sector (34-3940445, default = 2048) or {+-}size{KMGTP}:
Last sector (2048-3940445, default = 3940445) or {+-}size{KMGTP}: 1G
Current type is 'Linux filesystem'
Hex code or GUID (L to show codes, Enter = 8300):
Changed type of partition to 'Linux filesystem'
```

Abschließend müssen die Änderungen auf dem Datenträger gespeichert werden.

```
> w
Final checks complete. About to write GPT data. THIS WILL OVERWRITE
EXISTING PARTITIONS!!
OK; writing new GUID partition table (GPT) to /dev/sdc.
The operation has completed successfully.
```

Nun wurde eine Partition auf dem Datenträger erstellt.

```
> p
Disk /dev/sdc: 3940479 sectors, 1.9 GiB
Logical sector size: 512 bytes
Disk identifier (GUID): 00000000-0000-0000-0000-000000000000
Partition table holds up to 128 entries
First usable sector is 34, last usable sector is 3940445
Partitions will be aligned on 2048-sector boundaries
Total free space is 1845307 sectors (901.0 MiB)

Number  Start (sector)    End (sector)  Size       Code  Name
   1            2048         2097152   1023.0 MiB  8300  Linux filesystem
```

## 1.6 RAID

Es gibt noch weitere Konzepte um eine logische Partition zu erstellen oder zu verwalten. Darunter fällt das **R**edundant **A**rray of **I**ndependent **D**isks. Hierbei handelt es nur um eine Verknüpfung von mehreren physikalischen Blockgeräten zu einem logischen.

Dies kann sowohl via Hardware mit einem RAID-Controller oder mit Hilfe von Software geschehen. Dabei unterstützen moderne Dateisysteme (wie ZFS und BTRFS) dem Benutzer bei dieser Aufgabe.

Es gibt für die Verknüpfung von Datenträgern verschiedene Modi [5], zwischen den der Anwender wählen muss. Die verbreitesten Modi sind **RAID 0**, **RAID 1** und **RAID 5**.

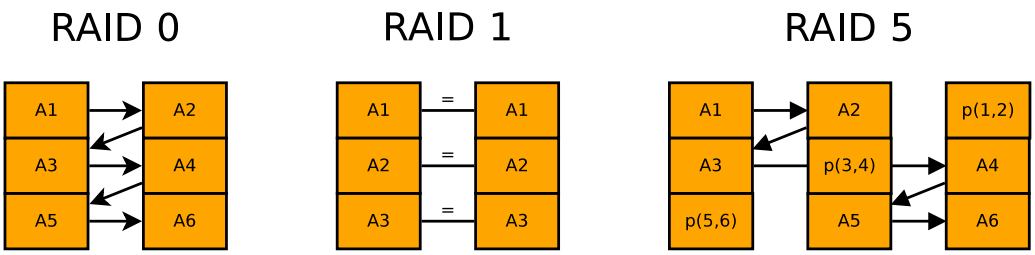


Abbildung 5: RAID-Modi

**RAID 0** Bei diesem Modus handelt es sich um das sogenannte **Striping**-Verfahren. Dabei werden Datenblöcke von einer Datei abwechselnd auf den Datenträgern verteilt. Dadurch erhöht man die Performance beim Lese-/Schreibvorgang, jedoch ist die Konsistenz der Daten bei einem Ausfall eines Datenträgers nicht mehr gewährleistet.

**RAID 1** Hierbei werden alle Daten 1:1 gespiegelt (**Mirroring**). Dies erhöht nicht Performance – bei softwareseitigem RAID verringert sich möglicherweise auch die gesammte Schreibgeschwindigkeit – jedoch sind die Daten relativ sicher gespeichert, da die komplette Datei mindestens doppelt vorliegt.

**RAID 5** Dies ist eine Mischung aus **Striping** und **Paritätsbildung**. Beim Schreiben wird wie beim RAID 0 die Datenblöcke gleichmäßig über die physikalischen Datenträger verteilt, jedoch werden Paritäten gebildet. In der Abbildung 5 werden abwechselnd auf zwei von drei Datenträgern die Datenblöcke und die Parität zwischen ihnen geschrieben.

In diesem Beispiel kann ein Datenträger ausfallen und komplett rekonstruiert werden. Dieses RAID ist auch um mehr Datenträger erweiterbar, was die Anzahl der gleichzeitig wiederherzustellender Datenträger erhöht.

**RAID 10** Dieser Modus ist eine Mischung aus RAID 0 und RAID 1. Dabei werden zwei RAID 1-Verbunde zu einem RAID 0-Verbund kombiniert.

[5] Quelle: RAID-Modi [6]

## 1.7 LVM

Das **L**ogical **V**olume **M**anager ist nicht zu verwechseln mit dem RAID. Es handelt sich hierbei lediglich um eine Fusion von mehreren Partitionen gleicher oder unterschiedlicher physikalischer Datenträger zu einer neuen virtuellen Partition.

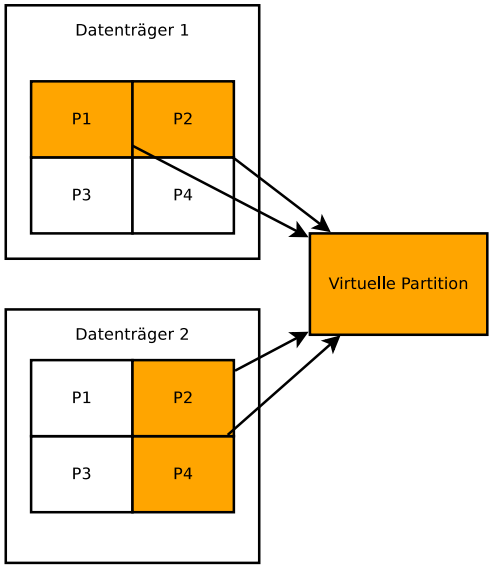


Abbildung 6: Virtuelle Partition mit LVM

# 2 Konzept eines Dateisystems

## 2.1 Definition »Dateisystem«

> Das Dateisystem (FS) ist Bestandteil des Betriebssystems und bildet die Schnittstelle zwischen diesem und den Laufwerken. Es legt fest, wie der Computer Dateien auf den Datenträgern benennt, speichert, organisiert und verwaltet. Ein Dateisystem besteht aus Dateien, Verzeichnissen und Adressen, über die die Dateien lokalisiert werden [6].

## 2.2 Aufbau eines Dateisystems

Grundsätzlich kann man ein Dateisystem in vier Blöcke unterteilen: **Boot**, **Superblock**, **Inode-Liste** und **Datenbereich**.

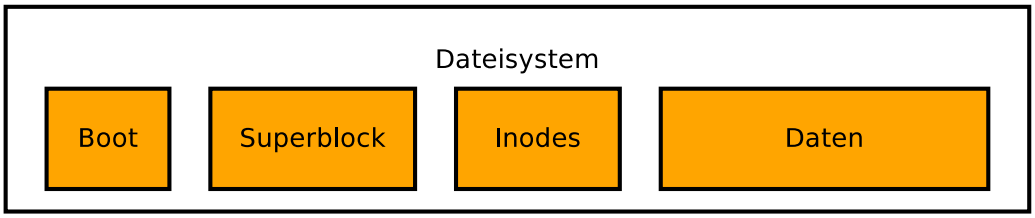


Abbildung 7: Grundlegende Struktur eines Dateisystems

Der Boot-Block ist ein reservierter, vom Dateisystem ungenutzter, Bereich. Hier befinden sich je nach Betriebssystem und Konfiguration Bootloader. Beim Dateisystem **EXT4** ist dieser beispielsweise 1024 B lang.

Darauf folgt der Superblock. Er beinhaltet die Metadaten des Dateisystems. Unter anderem die physikalische/logische Größe des zur Verfügung stehenden Speichers, die Länge der Inode-Liste, die Anzahl freier Blöcke u.s.w.) [7].

Ein Inode ist eine Eintrag für eine Datei oder ein Verzeichnis mit deren Metadaten und Adressen. Diese werden durchnummeriert in der Inode-Liste hinterlegt.

Der vierte Block wird mit Hilfe der Inodes adressiert und beinhaltet die Daten (sowohl Dateien als auch Verzeichnisse) der Benutzer.

## 2.3 Semantik eines Dateisystems

### 2.3.1 POSIX

Damit ein Betriebssystem zu verschiedenen Dateisystemen eine gemeinsame Schnittstelle bereitstellen kann, gibt es das **P**ortable **O**perating **S**ystem **I**nterface, welches von der IEEE Computer Society in mehreren Versionen verfasst worden ist.

[6] Quelle: ITWissen [4]
[7] Mehr über Superblocks: Wikipedia [14]

Es beschreibt die Mindestanforderungen für Schnittstellen, wie die Präsentation der Daten in einem hierarchischem Baum. Weiterhin standardisiert es die Rechteverwaltung (welche mit chmod und chown zu ändern sind) und die Bereitstellung von drei Zeitstempeln (atime, ctime, und mtime) [8].

### 2.3.2 »Durability«

> Durability is when you write something or change the filesystem and it's still there after the system crashes or loses power unexpectedly. Durability is what you need at a high level to say 'your email has been received' or 'your file has been saved' [9].

Diese Eigenschaft kann man nicht wirklich ins Deutsche übersetzen. Aber es beschreibt die Qualität des Dateisystems, dass die Daten geschrieben worden sind und auch später an diesem Ort ohne Fehler gelesen werden können. Dies ist vor allem bei Archivsystemen eine sehr wichtige Anforderung.

## 2.4 Linux PageCache

Der PageCache ist ein Algorithmus im Linux-Kernel, der den freien, ungenutzten Arbeitsspeicher zum Zwischenspeichern benutzt. Wird eine Datei zum ersten Mal aufgerufen, wird sie in den Arbeitsspeicher geladen. Aber anstatt diesen Speicher zu überschreiben, wird er freigegeben, falls dieser Speicher unbedingt benötigt wird [10].

Dadurch beschleunigt man die nächsten n-Aufrufe dieser Datei, da der Transfer vom physikalischen Datenträger zum Arbeitsspeicher immens Zeit kostet.

Unter Linux lässt sich mit Hilfe von free dieser »genutzte« Speicherbereich auslesen:

```
1 $ free -mh
2
3                total        used        free      shared  buff/cache
                    ↪ available
4 Mem:           7,6G        3,3G         53M        568M        4,2G
    ↪          3,6G
5 Swap:            0B          0B          0B
```

Dabei ist die Spalte **buff/cache** Ausschlag gebend. In diesem Beispiel wird von den 7.6 GB Arbeitsspeicher rund 4.2 GB für den PageCache genutzt. Die Uptime von diesem Computer lag indes bei 5 Tagen, 3 Stunden und 58 Minuten.

[8] Quelle: IEEE [3]
[9] Quelle: Chris Siebenmann, University of Toronto [10]
[10] Quelle: Linux Page Cache [2]

## 2.5 VFS

Der **V**irtual **F**ile System **S**witch ist eine Abstraktionsebene zwischen dem Benutzer und dem Betriebssystem. Er konstruiert die lokal sichtbare, nahtlose Verzeichnisstruktur, egal welches Dateisystem oder Netzwerkprotokoll hinter einem Verzeichnis steckt.

Dafür läd er auch automatisch Kernelmodule, z.B. xfs, btrfs oder vfat, oder hält eine Schnittstelle für SSHFS oder NFS bereit.

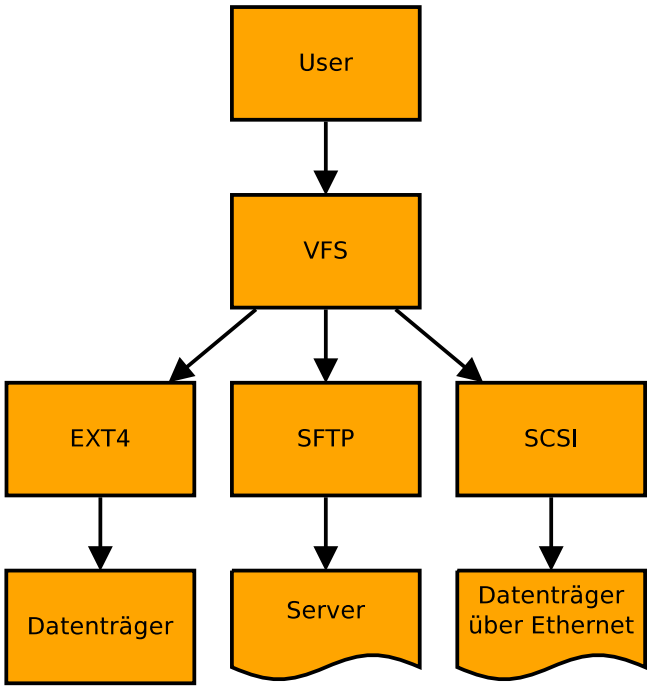


Abbildung 8: Schema von VFS

# 3 Übersicht über Dateisysteme in Linux

Es gibt sehr viele Dateisysteme aus verschiedenen Jahrzehnten. Einige sind properitär für Workstations gebaut worden, andere dagegen sind OpenSource und werden stetig weiter entwickelt.

Jedoch kann man die Linux-Dateisysteme in drei Kategorien unterteilen: **»Normale« Dateisysteme**, **Pseudo-Dateisysteme** und **Verteile Dateisysteme**.

| | |
|---|---|
| **»Normale« Dateisysteme** | EXT, EXT2, EXT3, EXT4, XFS, BTRFS, ZFS, ReiserFS, Reiser4, JFS, NEXT3, Tux3, ... |
| **Pseudo-Dateisysteme** | ramfs, tmpfs, procfs, sysfs, swapfs, ... |
| **Verteile Dateisysteme** | OrangeFS, Lustre, BeeGFS, HDFS, Ceph, ... |

**»Normale« Dateisysteme** Bei dem »normalen« Dateisystem handelt es sich um die Dateisysteme, die auf physikalischen Datenträgern, wie Festplatten, SSDs oder USB-Sticks geschrieben werden. Darunter fallen die oben genannten, aber auch beispielsweise vFAT.

**Pseudo-Dateisysteme** Sie werden hauptsächlich vom Betriebssystem simuliert. Das **tmpfs** ist ein Dateisystem, was vollständig im Arbeitsspeicher liegt und größendynamisch angepasst wird. **procfs** dagegen wird vom Linux-Kernel unter /**proc** eingebunden und stellt dem Benutzer verschiedene Informationen und Funktionen über die Prozesse des Systems zur Verfügung.

**Verteile Dateisysteme** Sie werden in Rechenzentren und Hochleistungsrechenzentrum zwecks der großen Datenmengen benötigt. Sie bündeln mehrere Knoten, welche aus mehreren physikalischen Datenträgern bestehen, zu einem großen Dateisystem. Dabei verwenden sie zur Kommunikation sowohl Ethernet, als auch InfiniBand und ermöglichen die Trennung zwischen Daten und Metadaten. Dadurch werden die Metadaten auf hochverfügbaren Knoten mit sehr schnellen Speicher hinterlegt, was ein schnelleres Auffinden und Verwalten ermöglicht

# 4 Dateisystem: EXT4

Das **fourth extended filesystem** wurde 2008 vom Linux-Kernel Entwicklerteam vorgestellt und ist der Nachfolger von **EXT3**. Es wird heutzutage als Standarddateisystem, in Bezug auf »normales« Dateisystem, bei sehr vielen Linux-Distributionen (u.A. Debian, Ubuntu und Fedora) genutzt.

Im Gegensatz zum Vorgänger unterstützt diese Version 48 bit-Blockadressen. Das bedeutet, dass sich die maximale Partitionsgröße bei einer Blockgröße von 4 KiB auf 1 EiB erhöht.

$$4\,\text{KiB (Blockgröße)} \cdot 2^{48}\ \text{(Adresslänge)} \approx 1.153 \times 10^{18}\,\text{B} = 1\,\text{EiB (Partitionsgröße)}$$

Des Weiteren werden nun **Extents** genutzt. Dabei handelt es sich um die Adressierung von großen zusammenhängenden Blöcken, anstatt jeden Block einzeln zu adressieren (siehe Abschnitt 4.2) [11].

## 4.1 Erzeugen von EXT4 auf Partition 1

Um eine Partition mit EXT4 zu formatieren, kann man das Kommandozeilenprogramm mke2fs benutzen. Dabei wird eine Partition genutzt, welche in diesem Beispiel unter **/dev/sdc1** zu finden ist.

```
1  $ sudo mkfs.ext4 /dev/sdc1
2  Creating filesystem with 261888 4k blocks and 65536 inodes
3  Filesystem UUID: 00000000-0000-0000-0000-000000000000
4  Superblock backups stored on blocks:
5      32768, 98304, 163840, 229376
6
7  Allocating group tables: done
8  Writing inode tables: done
9  Creating journal (4096 blocks): done
10 Writing superblocks and filesystem accounting information: done
```

Die Ausgabe von mke2fs erläutert, dass das Dateisystem eine Blockgröße von 4 KiB, 261 888 Blöcke und 65536 Inodes besitzt. Außerdem legt das Werkzeug vier Backups vom Superblock und ein Journal (siehe Abschnitt 4.3) mit einer Länge von 4096 Blocks an.

## 4.2 Inodes

Ein Inode ist, wie in Abschnitt 2.2 angeschnitten, ein Eintrag für eine Datei oder ein Verzeichnis. Wobei ein Verzeichnis nichts anderes ist, als eine Liste mit den enthaltenen Verzeichnissen und Dateien, welche sich aber nicht im Inode, sondern im referenzierten Datenspeicher befinden.

In einem Inode stehen auch keine Verzeichnis-/Dateinamen oder dessen Datentyp (Video-, Audio- oder Textdatei), sondern exakt nur folgende Daten [12]:

[11] Quelle: heise Open Source [7]
[12] Quelle: Linux-Praxis [5]

- 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 0-11
- Adresse des ersten Indirektionsblocks
- Adresse des Zweifach-Indirektionsblocks
- Adresse des Dreifach-Indirektionsblocks

Die Adressierung bei EXT4 ist historisch gewachsen. In der vorherigen Version wurden Datenblöcke, in dieser Extents adressiert. Sie sind ein logisch zusammenhängender Speicherbereich mit hintereinander gelisteten Datenblöcken. Dadurch beugt man der Fragmentierung des Datenbereich vor, indem man dynamisch verschieden große Lücken vollständig nutzen kann, anstatt alles hintereinander zu schreiben oder alle Datenblöcke stark zu fragmentieren.

In Abbildung 9 wird die Adressierungsmodalität des Inodes visualisiert. Die ersten 12 Einträge enthalten Adressen zum Datenbereich. Der nächste Eintrag ist der sogenannte »erste Indirektionsblock«. Er verweist auf einen Block im Datenbereich, welcher weitere Adressen enthält. Bei dem Zweifach- und Dreifach-Indirektionsblock ergänzt man diese Methode um eine weitere Dimension.

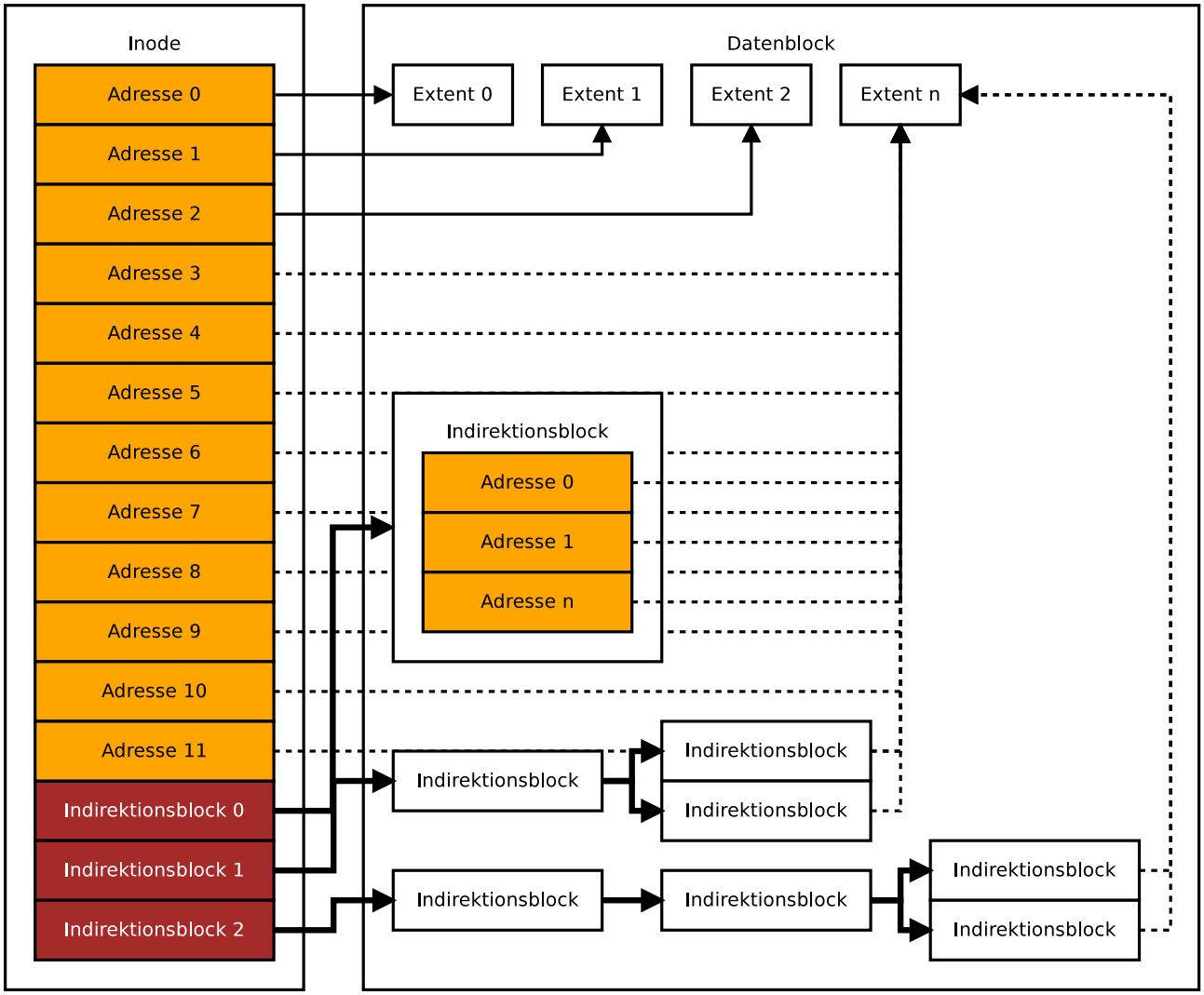


Abbildung 9: Die Adressierung von Inodes zu Extents und die Indirektionsblöcke

### 4.2.1 Inode auslesen

Mit dem Kommandozeilenprogramm stat kann man den entsprechenden Inode einer Datei oder eines Verzeichnisses auslesen. In diesem Beispiel wird die leere Datei **Seminar.md** ausgelesen:

```
1  $ stat Seminar.md
2    File: 'Seminar.md'
3    Size: 0       Blocks: 0          IO Block: 4096
4  Device: 821h/2081d  Inode: 12          Links: 1
5  Access: (0644/-rw-r--r--)  Uid: (1000)   Gid: (1000)
6  Context: unconfined_u:object_r:unlabeled_t:s0
7  Access: 2016-04-24 12:00:16.129690659 +0200
8  Modify: 2016-04-24 11:58:58.902414338 +0200
9  Change: 2016-04-24 11:58:58.902414338 +0200
10  Birth: -
```

## 4.3 Journal

Um die Robustheit des Dateisystems auch bei extremen Randbedingungen, wie z.B. ein Stromausfall, zu gewährleisten, schreiben einige Dateisysteme, darunter EXT4, ein Journal. Dabei unterscheidet man zwischen einem Metadaten- und einem Full-Journal.

In einem Metadaten-Journal werden alle anstehenden Änderungen, die sich im Bereich der Inodes (u.A. Adressenänderung), des Superblocks oder ähnlich kritisch ereignen, protokolliert und nach der sehr kurzen, erfolgreichen Schreibphase mit einem »Commit« versehen.

Dadurch lassen sich nach einem Neustart die vorgenommenen Änderungen nachvollziehen und das Dateisystem wird nicht beschädigt. Im Fall des Falles können die letzten Änderungen der Nutzerdaten verloren gehen, aber dafür ist das Dateisystem konsistent.

Der Unterschied zu einem Full-Journal besteht darin, dass auch die Nutzdaten mit protokolliert werden [13]

## 4.4 Rechteverwaltung

Laut POSIX muss jeder Inode einem Benutzer und einer Gruppe angehören (änderbar via chown). Daraus resultieren drei Rechtekategorien: Besitzer, Gruppe und Andere. Zusammen mit dem vorangehenden Sticky-Bit werden diese Zugriffsregeln mit 4 Oktets beschrieben.

In Abbildung 10 wird die Zusammensetzung der Oktets beschrieben. Dabei gibt es drei Arten von Zugriffsregeln: Lesen, Schreiben und Ausführen (bei Dateien) oder Auslisten (bei Verzeichnissen). Durch die Summe von Zweierpotenzen und die Konkatination der Oktets, ergibt sich eine dreistellige Zahl (das erste Oktet ist vernachlässigbar).

[13] Quelle: Journaling Dateisysteme [9]

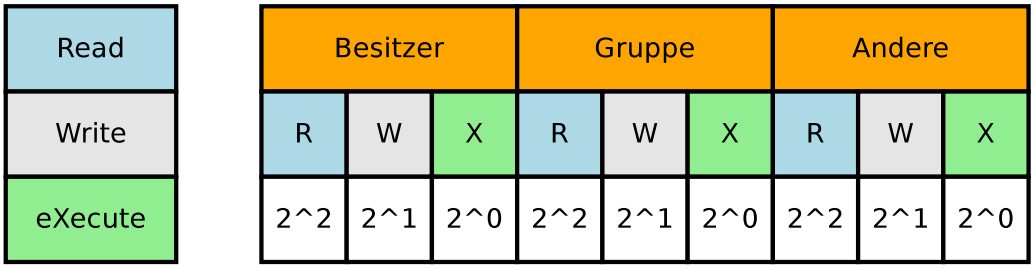


Abbildung 10: Rechteverwaltung via chmod

Im folgenden Beispiel wird nur dem Benutzer das Recht zum Lesen und Schreiben gegeben:

1. Oktet: $(1 \cdot 2^2) + (1 \cdot 2^1) + (0 \cdot 2^0) = 6$
2. Oktet: $(0 \cdot 2^2) + (0 \cdot 2^1) + (0 \cdot 2^0) = 0$
3. Oktet: $(0 \cdot 2^2) + (0 \cdot 2^1) + (0 \cdot 2^0) = 0$

Im folgenden Beispiel wechselt der Besitz der Datei »Seminar.md« zum Benutzer »user1« mit der Gruppe »group5«:

```
1 $ chown user1:group5 Seminar.md
2
3 // Auslesen der Rechte
4 $ ls -l Seminar.md
5 -rw-r--r--. 1 user1   group5  0    24. Apr 11:58 Seminar.md
```

Um nun die Zugriffsrechte für eine Datei oder ein Verzeichnis zu ändern, wird das Kommandozeilenprogramm chmod benötigt:

```
1 $ chmod 600 Seminar.md
2
3 // Auslesen der Rechte
4 $ ls -l Seminar.md
5 -rw-------. 1 user1   group5  0    24. Apr 11:58 Seminar.md
```

## 4.5 Linking

Werden Dateien oder Verzeichnisse an verschiedenen Orten benötigt, muss man sie nicht kopieren, sondern man erstellt einen Link. Unterschieden wird zwischen einem Hardlink und einem Symbolic-Link.

Der Hardlink funktioniert nur partitionsintern, da beide Dateien/Verzeichnisse den selben Inode teilen. Dieser verfügt, wie in Abschnitt 4.2 erwähnt, über einen Hardlink-Zähler, welcher de-/inkrementiert wird. Eine Datei ist somit erst »gelöscht«, wenn der Zähler unter 1 dekrementiert wird.

Im Gegensatz dazu gibt es den Symbolic-Link. Er funktioniert übergreifend und enthält als Dateiinformation den absoluten oder relativen Pfad zur verlinkten Datei oder zum verlinkten Verzeichnis. Er wird

auch nicht gelöscht, falls die verlinkte Datei oder das verlinkte Verzeichnis temporär nicht existieren sollten.

In den folgenden Beispielen wird zuerst ein Hardlink, dann ein Symbolic-Link erstellt:

```
$ ln Seminar.md Proseminar.md

// Inode-IDs auslesen
$ ls -i Seminar.md Proseminar.md
12 Seminar.md  12 Proseminar.md
```

```
$ ln -s Seminar.md Oberseminar.md

// Link auslesen
$ ls -l Oberseminar.md
lrwxrwxrwx. 1 user1 group5 8 24. April 17:15 Oberseminar.md -> Seminar.md
```

# 5 Dateisystem: BTRFS

Die Entwicklung des **B-tr**ee **f**ile **s**ystems begann 2007 und wird stetig weiterentwickelt. Auf lange Sicht soll es EXT4 als Standard-Dateisystem bei einigen Linux-Distributionen ablösen. Der Grund hierfür ist die Unterstützung von vielen neuen, gewünschten Features. Einige davon sind:

- Copy-On-Write
- Kompression
- Subvolumes/Snapshots
- Integriertes RAID

Eine recht vollständige Liste hält der Wikipedia-Artikel zu BTRFS vor.

## 5.1 Erzeugen von BTRFS auf Partition 2

Das Formatieren funktioniert ähnlich wie bei Abschnitt 4.1, jedoch mit dem Kommandozeilenprogramm btrfs-progs.

```
1  $ sudo mkfs.btrfs /dev/sdc2
2  Label:              (null)
3  UUID:               00000000-0000-0000-0000-000000000000
4  Node size:          16384
5  Sector size:        4096
6  Filesystem size:    899.04MiB
7  Block group profiles:
8    Data:             single            8.00MiB
9    Metadata:         DUP              52.94MiB
10   System:           DUP              12.00MiB
11 SSD detected:       no
12 Incompat features:  extref, skinny-metadata
13 Number of devices:  1
14 Devices:
15    ID        SIZE  PATH
16     1   899.04MiB  /dev/sdc2
```

## 5.2 Copy-on-Write

Dies ist ein essentielles Feature von BTRFS, welches die Datenkonsistenz erhöhen soll. In der Theorie wird eine Datei vollständig mit seinen Änderungen in einen anderen Speicherbereich kopiert und am Ende der Transaktion die Adresse im Inode neu verlinkt. Dadurch stellt man sicher, dass eine Datei immer einen definierten Zustand hat: alte Version, falls sich zwischenzeitlich ein Stromausfall ereignet, oder neue Version, wenn die Transaktion erfolgreich abgewickelt worden ist.

Da dies in der Praxis mit großen Dateien recht unhandlich ist, werden statt der ganzen Datei nur die Extents kopiert, aufgeteilt und neu verlinkt.

Der wesentliche Nachteil bleibt aber: die Performanceeinbußen durch den erhöhten Schreibbedarf.

## 5.3 Kompression

Weiterhin bietet BTRFS dem Benutzer an, transparent und automatisch alle Daten zu komprimieren. Dabei unterscheidet jedoch, ob die vorliegenden Daten komprimiert werde können. Viele Formate, wie beispielsweise Videos, sind zumeist schon durch speziell angepasste Verfahren optimiert worden.

Standardmäßig wird der LZO-Algorithmus benutzt, jedoch unterstützt BTRFS auch den GZIP-Algorithmus, was aber nicht empfohlen wird.

## 5.4 Subvolumes

> A btrfs subvolume is not a block device (and cannot be treated as one) instead, a btrfs subvolume can be thought of as a POSIX file namespace. This namespace can be accessed via the top-level subvolume of the filesystem, or it can be mounted in its own right [14].

Durch diese Funktion erhält man Vorteile gegenüber Partitionen, denn bei Partitionen wird die Speicherverteilung konkret festgelegt und kann mit recht viel Aufwand nachträglich geändert werden. Subvolumes benutzen alle zusammen den kompletten zur Verfügung stehenden Speicherplatz.

Weiterhin ermöglichen sie es recht einfach sogenannte »Snapshots« anzulegen. Dabei wird, ähnlich wie bei einem Versionierungsprogramm wie git, der aktuelle Stand eingefroren und als Subvolume gespeichert. Bei jeder nachfolgenden Änderung werden nur noch die Blockunterschiede in einem neuen Subvolume hinterlegt.

### 5.4.1 Subvolume erstellen

Fast alle Änderungen an BTRFS werden mit dem Kommandozeilenprogramm btrfs-progs durchgeführt. Es ist zu empfehlen, dass man das Programm in einem Verzeichnis benutzt, wo das entsprechende BTRFS eingebunden worden ist.

```
1 $ sudo btrfs subvolume create ./Subvolume1
2 Create subvolume './Subvolume1'
3
4 $ sudo btrfs subvolume list ./
5 ID 256 gen 7 top level 5 path Subvolume1
6
7 $ ls -l
8 total 0
9 drwxr-xr-x. 1 root root 0 24. Apr 09:27 Subvolume1/
```

### 5.4.2 Quota

Ein weiterer Aspekt von Subvolumes sind Quota. Standardmäßig ist der Speicherplatz der Subvolumes nur durch die physikalische Größe beschränkt – also im Bezug auf die Quota unlimitiert. Dabei wird bei

[14] Quelle: Btrfs-Wiki [1]

BTRFS-Quota zwischen komprimierter und unkomprimierter Quota unterschieden.

Ein sinnvolles Szenario hierfür wäre beispielsweise die Homeverzeichnisse von Benutzerkonten auf einem Server mit einer Quota zu beschränken.

```
1  // Quota-Feature aktivieren
2  $ sudo btrfs quota enable ./
3
4  // Quota anzeigen
5  $ btrfs qgroup show -r ./
6  qgroupid         rfer         excl     max_rfer
7  --------         ----         ----     --------
8  0/5          16.00KiB     16.00KiB         none
9  0/256        16.00KiB     16.00KiB         none
10
11 // Quota setzen
12 $ sudo btrfs qgroup limit 10M 0/256 ./
13
14 // Quota anzeigen
15 $ btrfs qgroup show -r ./
16 qgroupid         rfer         excl     max_rfer
17 --------         ----         ----     --------
18 0/5          16.00KiB     16.00KiB         none
19 0/256        16.00KiB     16.00KiB     10.00MiB
```

## 5.5 Deduplikation

Neben Komprimierung gibt es eine weitere Möglichkeit Speicherplatz zu optimieren: Deduplikation. Dabei werden alle mehrfach vorkommenden Blöcke auf ein Exemplar reduziert und entsprechend verlinkt.

Wird dies während des Schreibvorganges durchgeführt, benötigt man hierfür sehr viel Arbeitsspeicher, da eine große Lookup-Table mit allen Block-Hashes erforderlich ist.

Alternativ gibt es die Möglichkeit manuell mit einem Werkzeug alle oder eine Auswahl von Dateien gegenseitig auf identische Blöcke vergleichen zu lassen.

## Zusammenfassung

Dateisysteme sind komplex und übernehmen heutzutage viel mehr Aufgaben, als *nur* das einfache Abspeichern von Daten. Sie sichern die Integrität der Daten mittels Prüfsummen ab und bieten Deduplikation auf Blockebene oder mit Hilfe von Links auf Dateiebene.

Einige bieten auch die Möglichkeit RAIDs zu erstellen und zu verwalten. Auch die Funktion des Namespacing (Subvolumes bei BTRFS) und der Snapshots liefern verschiedene Dateisysteme mit, was für Serversysteme sehr praktisch ist. Ebenfalls komprimieren sie auf Wunsch die Daten transparent auf den Datenträger, ohne zusätzlichen Aufwand für den Benutzer.

Auch die Wahl der geeigneten Partitionstabelle ist nicht zu unterschätzen. Ist ein UEFI-Boot gewünscht, ist GPT erforderlich. Möchte man mehr als 4 Partitionen erstellen oder hat man Festplatten mit hoher Speicherkapazität, ist ebenfalls etwas anderes als MBR erforderlich.

Abschließend gilt auf jeden Fall der Hinweis, dass jedes Dateisystem für bestimmte Einsatzszenarien besser oder schlechter geeignet ist.

# Literatur


[1] Btrfs-Wiki. *SysadminGuide*. 2015. URL: https://btrfs.wiki.kernel.org/index.php/SysadminGuide#Subvolumes (besucht am 24. 04. 2016).

[2] Werner Fischer. *Linux Page Cache*. 2013. URL: https://www.thomas-krenn.com/de/wiki/Linux_Page_Cache (besucht am 21. 04. 2016).

[3] IEEE. *IEEE Std 1003.1*. 2004. URL: http://pubs.opengroup.org/onlinepubs/009695399/utilities/xcu_chap01.html#tag_01_07_01_03 (besucht am 26. 04. 2016).

[4] ITWissen. *Dateisystem*. 2016. URL: http://www.itwissen.info/definition/lexikon/Dateisystem-file-system.html (besucht am 21. 04. 2016).

[5] Linux-Praxis. *Das I-Node System*. 2015. URL: http://www.linux-praxis.de/lpic1/lpi101/inode.html (besucht am 24. 04. 2016).

[6] Samara Lynn. *RAID Levels Explained*. 2014. URL: http://www.pcmag.com/article2/0,2817,2370235,00.asp (besucht am 20. 04. 2016).

[7] heise Open Source. *Extents*. 2009. URL: http://www.heise.de/open/artikel/Extents-221268.html (besucht am 24. 04. 2016).

[8] Daniel B. Sedory. *An Examination of the Standard MBR*. 2012. URL: http://thestarman.pcministry.com/asm/mbr/STDMBR.htm (besucht am 20. 04. 2016).

[9] SelfLinux. *Journaling Dateisysteme*. URL: http://www.selflinux.org/selflinux/html/dateisysteme_journaling02.html (besucht am 24. 04. 2016).

[10] Chris Siebenmann. *Consistency and durability in the context of filesystems*. 2015. URL: https://utcc.utoronto.ca/~cks/space/blog/tech/FSConsistencyAndDurability (besucht am 26. 04. 2016).

[11] Wikimedia-Commons. *GUID Partition Table Scheme*. 2007. URL: https://en.wikipedia.org/wiki/File:GUID_Partition_Table_Scheme.svg (besucht am 20. 04. 2016).

[12] Wikipedia. *Master Boot Record*. 2016. URL: https://de.wikipedia.org/wiki/Master_Boot_Record (besucht am 20. 04. 2016).

[13] Wikipedia. *Partition (Datenträger)*. 2015. URL: https://de.wikipedia.org/wiki/Partition_(Datentr%C3%A4ger) (besucht am 20. 04. 2016).

[14] Wikipedia. *Superblock*. 2015. URL: https://de.wikipedia.org/wiki/Superblock (besucht am 26. 04. 2016).