

Dateisysteme unter Linux

[Jana Jaeger](#)

Inhaltsverzeichnis

- [Was genau tut ein Dateisystem?](#)
- [Welche Dateisysteme gibt es?](#)
- [Zwei aus X](#)
 - [ext2](#)
 - [ReiserFS - ein Filesystem mit Journaling](#)
- [Mehr Informationen, bitte!](#)

Jeder nutzt sie, keiner sieht sie bei der Arbeit, und ohne sie geht **gar nichts**. Die Rede ist von den Dateisystemen. Dieser Artikel soll einen kleinen Einblick in die Dateiwirtschaft unter Linux vermitteln und zwei der bekannteren, einen Oldie, *ext2*, und einen Newcomer, *ReiserFS*, kurz vorstellen.

Was genau tut ein Dateisystem?

Die Aufgabe eines Dateisystems besteht darin, die einzelnen Datenblöcke einer Datei auf die Festplatte (oder einen anderen Datenträger) abzulegen. Es legt fest, an welcher Stelle des Datenträgers sie gespeichert werden sollen und archiviert diese Informationen gleichzeitig in einer Tabelle. Die Zugriffs- und Lokalisierungsstrategie eines Dateisystems hat direkten Einfluß auf die Durchsatzrate der Schreib- und Lesezugriffe auf das Medium und auf die Zuverlässigkeit des Dateisystems an sich.

Welche Dateisysteme gibt es?

Unter Linux gibt es nicht **das** Dateisystem schlechthin. Linux kommt mit einer ganzen Reihe von Dateisystemen zurecht. Hier eine kurze, beileibe nicht vollständige, Übersicht:

<i>AFS</i>	ein verteiltes Dateisystem, meist im WAN genutzt
<i>autofs</i>	für das automatische Mounten eines Dateisystems bzw. Datenträgers ins Gesamtsystem - ist also kein echtes Dateisystem
<i>devpts</i>	für Pseudoterminals (nach UNIX-98-Spezifikation)
<i>ext2</i>	der Standard unter Linux
<i>ext3</i>	Weiterentwicklung von <i>ext2</i> (noch Beta Status), wird Journaling beherrschen
<i>hpfs</i>	das OS/2-Dateisystem
<i>iso9660</i>	Standardformat für CD-ROMs und DVDs
<i>JFS</i>	Von IBM entwickeltes Journaling Filesystem, ursprünglich für AIX (IBMs UNIX) entwickelt und seit Februar 2000 ist eine Beta-Version unter einer Open Source Lizenz auch als Patch für Linux verfügbar
<i>minix</i>	Dateisystem von Minix, wird oft für Linux-Disketten gebraucht.
<i>msdos</i>	Dateisystem für MS-DOS-Partitionen und -Disketten (kurze Dateinamen)
<i>nfs</i>	Netzwerkdateisystem für UNIX, um Daten von anderen Rechnern zu lesen
<i>ntfs</i>	Windows-NT/2000 Dateisystem (allerdings nur mit Lesezugriff)
<i>nwfs</i>	kann ein NetWarediskfilesystem lesen
<i>proc</i>	Prozeßverwaltung unter <code>/proc</code>
<i>ReiserFS</i>	Reiser-Dateisystem, nach seinem Schöpfer Hans <i>Reiser</i> benannt
<i>smbfs</i>	Samba (Netzwerkdateisystem unter Windows)
<i>swap</i>	Swap-Partitionen oder -Dateien
<i>udf</i>	Universal Disk Format (CD-RWs und DVDs)
<i>usbdevfs</i>	Einbinden und Verwalten von USB-Geräten
<i>vfat</i>	DOS/Windows-9x-Dateisystem (lange Dateinamen)
<i>XFS</i>	Journaling Filesystem unter UNIX. Ursprünglich von SGI für IRIX, das hauseigene UNIX entwickelt, aber mittlerweile unter der GPL in einer Beta-Version als Patch für Linux verfügbar

Zwei aus X

Aus der Fülle der oben genannten Beispiele werden nun zwei herausgegriffen, um die unterschiedlichen Lösungsansätze für das Speicherproblem unter Linux zu diskutieren.

ext2

ext2 bewältigt seine Verwaltungsaufgabe, indem es den vorhandenen Speicherplatz in Blöcke gleicher Größe (z.B. je 1024 Byte) einteilt und diese Blöcke durchnummeriert. Die Menge der Blöcke wird anschließend in verschiedene Gruppen aufgeteilt, die jeweils zur Speicherung unterschiedlicher Datentypen genutzt werden.

Von "Gruppe" kann man bei der ersten Gruppe in der Reihe eigentlich nicht sprechen. Es handelt sich hierbei um einen einzelnen Block, den Bootblock. Er enthält eine Menge von Minimaldaten, die zum Start des Betriebssystems notwendig sind. Die zweite "Gruppe" umfaßt ebenfalls nur einen einzigen Block, den Superblock genannt wird. Im Superblock wird festgelegt, wie groß die nun folgenden Gruppen sind. Er enthält darüber hinaus Informationen über das gesamte Filesystem. Aus Sicherheitsgründen wird der Superblock repliziert. Jetzt endlich folgen wirkliche "Gruppen" von Blöcken. Die dritte und die vierte Gruppe fungieren als eine Art von Landkarten (Bitmaps) über die beiden letzten Gruppen. Die Blöcke der dritten Gruppe sind als Inode Bitmap organisiert. Hier wird protokolliert, welche Blöcke für Inodes frei sind. In der vierten Gruppe findet sich eine entsprechende Bitmap über die verfügbaren Blöcke für Daten. In der fünften Gruppe liegt der eigentliche Speicherplatz für Inodes (je acht Inodes können bei einer Blockgröße von 1024 Bytes in einem Block gespeichert werden). Die sechste Gruppe dient als Speicherplatz für Daten (1024 Bytes pro Block). Hier und nur hier werden die eigentlichen Daten gespeichert, die anderen fünf Gruppen sind der Verwaltungsüberbau ;-)

Wichtigste Größe in einem UNIX-Filesystem und damit auch in *ext2* sind die Inodes (oder auch Informationsknoten), die mit Ausnahme des Dateinamens alle für eine Datei relevanten Daten enthalten. So werden hier gespeichert:

- Benutzer- und Gruppen-ID
- Zugriffsrechte
- Größe der Datei
- Anzahl der Links, die auf diese Datei verweisen
- Daten der Erstellung, letzten Änderung oder Löschung dieser Datei
- Verweise auf die ersten zwölf Blöcke dieser Datei
- wenn nötig: Ein-, zwei- oder gar dreifach indirekte Verweise auf weitere Datenblöcke (dazu später mehr ...)

Eine Datei besteht also aus ihrem entsprechenden Inode und aus mehreren Datenblöcken, die die eigentlichen Daten enthalten. Die Verwaltungsinformationen im Inode und der eigentliche Dateninhalt der Datei sind vollkommen getrennt und können auch unabhängig voneinander bearbeitet werden. Wer eine Datei in ein neues Verzeichnis verschiebt, ändert den Inode und die entsprechenden Verzeichniseinträge, aber niemals wird er die Daten direkt ändern.

Umfaßt eine Datei mehr als 12 kByte (12 mal 1024 Byte oder zwölf Datenblöcke), birgt ein weiterer Inode im Inode einen Verweis auf einen Block, der selbst keine Daten, dafür aber Verweise auf bis zu 256 weitere Datenblöcke enthält. Hier spricht man vom einfach indirekten Verweis. Bei Dateien einer über einer Größe von 268 kByte (12 Blöcke direkt referenziert, 256 einfach indirekt) verweist ein weiterer Inode auf einen Block, der auch selbst wieder auf 256 Datenblöcke zeigt, die wiederum selbst auf 256 Datenblöcke zeigen können. In diesem Fall redet man vom zweifach indirekten Verweis. Natürlich läßt sich auch ein dreifach indirekter Verweis realisieren.

Die Verwaltung von Dateien und Verzeichnissen ist sich sehr ähnlich. Ein Verzeichnis ist im Prinzip wie eine Textdatei mit mehreren Zeilen gegliedert. In jeder Zeile dieser Textdatei steht der Name eines Unterverzeichnisses oder einer Datei und die Inode-Nummer der Datei. Zusätzlich findet sich hier ein Verweis auf sich selbst (.) und das übergeordnete Verzeichnis (..).

Die Dateiverwaltung läuft nun, wie an der folgenden Ausgabe des Kommandos `ls -lai` zu sehen ist, ab:

```
jj@gromit:/tmp/tmp/jj > ls -lai
insgesamt 4
223244 drwxr-xr-x  4 jj users      1024 Jan 21 17:00 .
180247 drwxr-xr-x  3 jj users      1024 Jan 21 17:03 ..
 84002 drwxr-xr-x  2 jj users      1024 Jan 21 17:00 dir1
223245 drwxr-xr-x  2 jj users      1024 Jan 21 17:01 dir2
```

Von links nach rechts erklärt: Die linke Spalte gibt die Inode-Nummern an, unter der die Verweise, Unterverzeichnisse oder gegebenenfalls auch Dateien abgelegt sind. In der nächstenn Spalte werden die Zugriffsrechte aufgelistet. Darauf folgt die Spalte mit der Anzahl fester Links, die auf diese Dateien oder Verzeichnisse zeigen (Hardlinks). Eine Datei hat zwar nur einen Inode, aber in diesem können mehrere Links auf diese Datei gespeichert werden. Beispielsweise ist für das Unterverzeichnis `dir1` die Zahl von zwei Hardlinks angegeben. Konkret besagt dies: `dir1` verweist einmal auf sich selbst und enthält einen Verweis auf das Verzeichnis eine Ebene darüber, hier `jj`. Die nächsten Spalten geben Benutzer- und Gruppen-ID, Verzeichnisgröße, Änderungsdatum und Namen an.

Jetzt zum Mounten und Unmounten der entsprechenden Platte oder Partition und den Geschehnissen nach einem Absturz bzw. einem "sauberen" Herunterfahren des Systems. Im Superblock (Gruppe 2) wird beim Mounten des Dateisystems ein sogenanntes Valid Bit gelöscht, das nach einem sauberen Unmount wieder gesetzt wird. Fehlt dieses Bit beim nächsten Start des Systems, bedeutet dies, daß das System nicht ordnungsgemäß heruntergefahren wurde. Jetzt wird ein Programm gestartet, *e2fsck*, das die Daten auf Konsistenz überprüft und wenn nötig zu reparieren versucht. Dies ist nicht immer erfolgreich, Dateien (oder Datenblöcke), die nicht ordnungsgemäß wiederhergestellt werden konnten oder nicht richtig referenziert werden können, landen in einem extra dafür vorgesehenen `/lost+found`-Verzeichnis der Partition.

Da *fsck* über die gesamte Partition (oder Platte) laufen muß, kann diese Prozedur unter Umständen je nach Größe des Dateisystems einige Minuten kostbarer Uptime in Anspruch nehmen. Ist man auf eine effizientere (schnellere) Art der Dateiverwaltung angewiesen, sollte man eine andere Art der Dateiverwaltung in Betracht ziehen, z. B.:

ReiserFS - ein Filesystem mit Journaling

ReiserFS ist ein hybrides Filesystem, d.h. es wird ein "normales", nicht log-orientiertes Filesystem erstellt, für das es noch zusätzlich ein Journal gibt. In diesem Journal werden alle Metadatenänderungen, die mehr als einen Block betreffen, vorgemerkt, und erst wenn sie im Journal stehen in das echte Filesystem geschrieben. Daten werden direkt in das Dateisystem geschrieben. Bei *ReiserFS* kann es also weiterhin zu einem Datenverlust bei einem Systemabsturz kommen, nur die Metadaten sind in Ordnung.

Beim Mounten einer *ReiserFS* Partition wird die Log-Datei ausgewertet und die Änderungen ausgeführt, die aufgrund eines Systemabsturzes nicht durchgeführt werden konnten. Die Partition ist dann wieder in einem konsistenten Zustand. Diese einfache Rekonstruktion des Status Quo hat einen großen Vorteil: Der Filesystem-Check beim Booten eines Systems dauert nur wenige Sekunden, auch bei großen Platten wo *ext2* mehrere Minuten benötigen würde.

Natürlich wird man die Vorteile des "Journaling" nicht genießen können, ohne auch ein paar Nachteile in Kauf zu nehmen. Die Schreibzugriffe auf die Platte sind häufiger, da Änderungen erst angemeldet

und dann durchgeführt werden.

Das *ReiserFS* besitzt aber eine Reihe von Features, die es vor allem bei kleinen Dateien und vielen Dateien in einem Verzeichnis schneller macht als *ext2*. Die zentrale Datenstruktur bei *ReiserFS* ist ein Baum, Informatiker bezeichnen die implementierte Variante als einen B*-Baum. In den Blättern des Baumes werden vier verschiedene Arten von Datensätzen gespeichert:

direkte Datensätze:

Diejenigen Teile von Dateien, die nicht mehr in einen Block reinpassen. Der Plattenplatz wird dadurch besser ausgenutzt (ein 10 Zeichen große Datei landet auf einem traditionellen System wie *ext2* in einem Block von z.B. 1024 Bytes) und kleine Dateien sind direkt im Baum, es ist kein zusätzlicher Plattenzugriff notwendig.

indirekte Datensätze:

Dies sind Zeiger auf große Dateien, die außerhalb des Baumes liegen. Alle Daten, bis auf den letzten Teil (siehe direkte Datensätze) sind in solchen Dateien gespeichert.

Verzeichnisse:

Enthält die einzelnen Einträge eines Verzeichnisses.

"stat data":

ext2 benutzt Inodes, *ReiserFS* speichert dagegen diese Daten im Baum ab und belegt somit nicht unnötige Inodes (bei *ext2* wird die Anzahl und Lage der Inodes bei der Formatierung festgelegt und kann später nicht geändert werden).

Jeder Datensatz besitzt einen eindeutigen Schlüssel (hier werden Hashfunktionen benutzt), der zum Sortieren und Wiederfinden benutzt wird.

Das Verwalten des B*-Baumes kostet etwas mehr Zeit als die simplen Techniken von *ext2*, aber dafür sind Zugriffe effizienter durchführbar. Je nach Benchmark kann *ext2* oder *ReiserFS* schneller sein, aber im normalen Einsatz überwiegen die Vorteile von *ReiserFS* deutlich.

Mehr Informationen, bitte!

Benutzer einer [SuSE](#) Linux Version 6.4 oder höher kommen schon seit jeher in den Genuß sowohl von *ReiserFS* als auch von *ext2*. Da sämtliche Kernelversionen kleiner als 2.4.1 noch keine Unterstützung für *ReiserFS* enthalten, müssen Benutzer einer anderen Distribution als SuSE die entsprechenden Kernelpatches nachinstallieren. SuSE hat die entsprechenden Patches bereits in den ausgelieferten Kernels auf 6.4, 7.0 und integriert. Andernfalls laden Sie sich bitte die zu ihrer Kernelversion passenden Kernelpatches von [Nemesys](#) herunter und folgen Sie den dort gegebenen Anweisungen zur Installation und Konfiguration. 