Study-Guide: Gerätedateien, Linux Dateisysteme, Filesystem Hierarchy

In diesem Abschnitt geht es um alles, was mit dem Umgang mit Dateisystemen zu tun hat. Dazu zählen die verschiedenen Befehle zum Anlegen, Reparieren und Überprüfen von Dateisystemen selbst genauso, wie die Befehle zum Modifizieren der Dateien. Viele wichtige Fragen, die Dateisysteme betreffen, können nur vernünftig beantwortet werden, wenn die interne Architektur von Dateisystemen klar ist. Aus diesem Grund habe ich hier noch ein paar Zusatzinformationen zu diesem Thema erstellt, die nicht direkten Bezug auf einzelne Prüfungsziele von LPI nehmen: Das INode Prinzip von Unix Dateisystemen Details zum EXT2 Dateisystem Journaling Dateisysteme Erzeugen von Partitionen und Dateisystemen Erhaltung der Dateisystemintegrität Kontrolle des Ein- und Aushängen von Dateisystemen Verwalten von Diskquotas Zugriffskontrolle auf Dateien mittels Zugriffsrechten Verwaltung von Dateieigentum Erzeugen und Ändern von harten und symbolischen Links Auffinden von Systemdateien und Platzieren von Dateien an den korrekten Ort

Seite:	-= LinuxLernSystem =- (http://www.lpi-test.de)
Kurs:	LPIC-1 [101]
Buch:	Study-Guide: Gerätedateien, Linux Dateisysteme, Filesystem Hierarchy
Gedruckt von:	André Scholz
Datum:	Dienstag, 1 November 2005, 10:24 Uhr

Inhaltsverzeichnis

1.104 - Gerätedateien, Linux Dateisysteme, Filesystem Hierarchy Standard

In diesem Abschnitt geht es um alles, was mit dem Umgang mit Dateisystemen zu tun hat. Dazu zählen die verschiedenen Befehle zum Anlegen, Reparieren und Überprüfen von Dateisystemen selbst genauso, wie die Befehle zum Modifizieren der Dateien.

Viele wichtige Fragen, die Dateisysteme betreffen, können nur vernünftig beantwortet werden, wenn die interne Architektur von Dateisystemen klar ist. Aus diesem Grund habe ich hier noch ein paar Zusatzinformationen zu diesem Thema erstellt, die nicht direkten Bezug auf einzelne Prüfungsziele von LPI nehmen:

    - Das INode Prinzip von Unix Dateisystemen
    - Details zum EXT2 Dateisystem
    - Journaling Dateisysteme Erzeugen von Partitionen und Dateisystemen
- Erhaltung der Dateisystemintegrität
- Kontrolle des Ein- und Aushängen von Dateisystemen
- Verwalten von Diskquotas
- Zugriffskontrolle auf Dateien mittels Zugriffsrechten
- Verwaltung von Dateieigentum
- Erzeugen und Ändern von harten und symbolischen Links
- Auffinden von Systemdateien und Platzieren von Dateien an den korrekten Ort

Das I-Node System

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*1024 oder 10*2048 oder 10*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 ergenen sich dadrch 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.

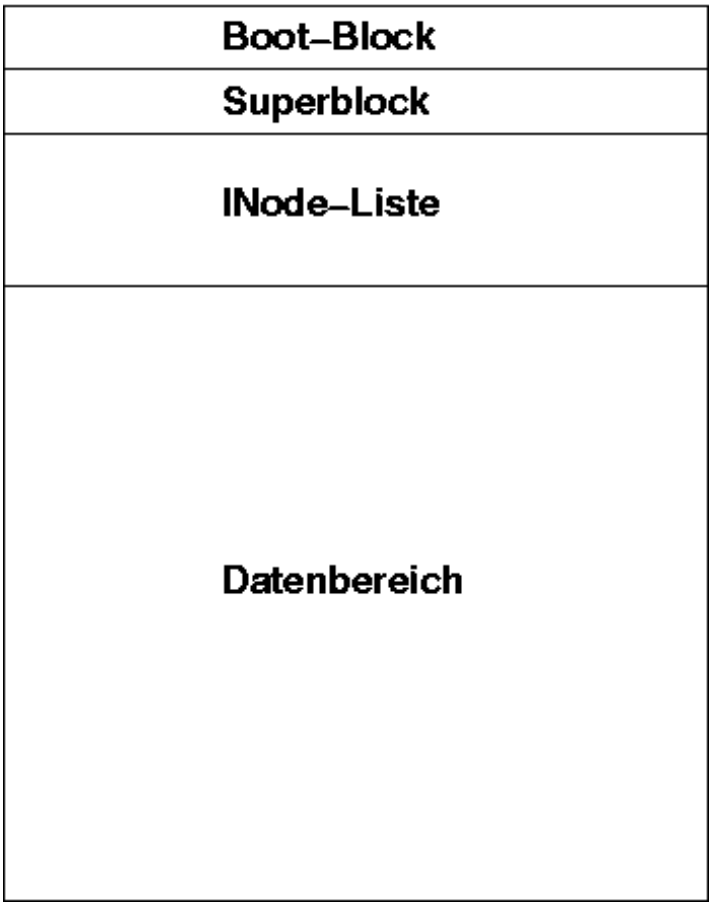
Der Superblock

Ein Unix-Dateisystem besitzt einen sogenannten 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 sogenannte Bootblock auf der Platte, der in etwa dem Bootsektor der DOS-Partitionen entspricht. Zusammengefasst kann man also ein Unix-Dateisystem etwa wie folgt darstellen:



Das EXT2 Dateisystem

Im Wesentlichen entspricht das Second Extended Filesystem (EXT2) von Linux dem Unix-Inode-System, das an anderer Stelle schon beschrieben wurde. Es gibt aber ein paar wesentliche Unterschiede, die hier kurz erläutert werden sollen.

Der Aufbau einer EXT2 Inode

Die wesentlichen Unterschiede des EXT2 Systems zum "normalen" I-Node-System liegen in der Struktur der I-Nodes selbst. Hier sind einige Felder hinzugekommen, die für bestimmte Erweiterungen vorgesehen wurden, die im folgenden genauer erklärt werden. Doch zunächst einmal die Struktur der Inode selbst:

permission	links	owner	group
size		creation time	
modification time		access time	
deletion time		blockcount	
flags		file version (NFS)	
file ACL		dir ACL	
fragment addr.	fr. size	frag. nr	reserved
1. block data		2. block data	
3. block data		4. block data	
5. block data		6. block data	
7. block data		8. block data	
9. block data		10. block data	
11. block data		12. block data	
simple indirect		double indirect	
triple indirect		reserved	
reserved		reserved	

Die wesentlichen Unterschiede zum bisher beschriebenen I-Node-System sind:

- Neben den drei Standard-POSIX-Zeitmarken (ctime, atime, mtime) gibt es hier eine vierte Zeitmarke, die die Löschzeit (*deletion time*) speichert. Das ist gedacht für Spezialprogramme zum Wiederherstellen versehentlich gelöschter Dateien.
- Ein Feld für *flags* ist hinzugekommen. Diese Flags ermöglichen es, daß Dateien bestimmte Eigenschaften bekommen, die weiter unten genauer beschrieben werden.
- Der Eintrag *file version* kann vom NFS-Server zur Unterscheidung verschiedener Versionen einer Datei verwendet werden.
- Zwei Einträge für eine erweiterte Zugriffskontrolle für Dateien (*file ACL*) und Verzeichnisse (*dir ACL*) sind hinzugekommen. ACL steht für Access Control List.
- Die Unterstützung fragmentierter Datenzonen ist vorgesehen.
- Es stehen 12 direkte Adressfelder für Datenblocks zur Verfügung.

Die Flags des EXT2 Filesystems

Jede Datei in einem EXT2 Dateisystem kann neben den normalen Unix-Zugriffsrechten noch einen Satz zusätzlicher Flags bekommen, die hier noch erläutert werden sollen. Um sich die Flags (oder auch Attribute) anzusehen, wird das Programm **lsattr** benutzt, zum Ändern der Attribute wird **chattr** benutzt. Es gibt die Attribute A, a,c,d,i,S,s und u.

- Wenn eine Datei mit gesetztem A-Attribut verändert wird, so wird das *atime* Feld nicht verändert.
- Eine Datei, die das a-Attribut gesetzt hat, kann nur im Append-Mode geöffnet werden. Das heißt, Daten

können an diese Datei angehängt werden, bestehende Daten jedoch nicht gelöscht.
- Eine Datei mit dem c-Attribut wird automatisch vom Kernel komprimiert, bevor sie auf die Platte geschrieben wird. Wenn sie gelesen wird, dann entkomprimiert der Kernel die Datei wieder, ein Anwender merkt also nichts von der Kompression.
- Eine Datei mit gesetztem d-Attribut ist **kein** Kandidat für ein Backup mit dem *dump* Befehl.
- Eine Datei mit gesetztem i-Attribut kann nicht verändert werden, sie kann nicht gelöscht oder umbenannt werden, kein Link kann auf sie erstellt werden und keine Daten können in ihr verändert werden. Dieses Attribut kann nur vom Superuser gesetzt oder entfernt werden.
- Wenn eine Datei mit gesetztem s-Attribut gelöscht wird, dann werden die Bytes der Datei mit Nullen überschrieben. Ein sicheres Löschen, das keine Wiederherstellung erlaubt.
- Wenn eine Datei, deren S-Attribut gesetzt ist, verändert wird, dann wird diese Veränderung synchron auf die Platte geschrieben, ohne lange im Cache zu liegen.
- Wenn eine Datei mit gesetztem u-Attribut gelöscht wird, dann wird ihr Inhalt gesichert, so daß sie wiederhergestellt werden kann.

Erweiterungen des Superblocks

Der Superblock eines EXT2-Dateisystems enthält ein sogenanntes *Valid-Flag*. Sobald dieses Dateisystem gemountet wird, wird das Valid-Flag gelöscht. Erst beim Umount wird es wieder gesetzt. Sollte das System abstürzen, so bleibt das Valid-Flag ungesetzt - beim nächsten Start kann also erkannt werden, daß dieses Dateisystem nicht ordnungsgemäß abgehängt worden ist.

Das Programm zur Überprüfung eines EXT2 Dateisystems (e2fsck) überprüft dieses Valid-Flag und prüft (falls es nicht durch die -f Option gezwungen wurde) nur dann das Dateisystem, wenn das Valid-Flag nicht gesetzt war. So kann beim Systemstart einfach für jede Partition das e2fsck-Programm aufgerufen werden. Es wird nur dann aktiv, wenn es nötig ist.

Der Superblock enthält weiterhin eine Angabe über die maximale Anzahl von Mount-Vorgängen, die zwischen zwei System-Checks ablaufen dürfen. Jedesmal, wenn ein Dateisystem gemountet wird, wird ein Zähler (*mount-count*) um eins erhöht. Wird bei einem Systemstart festgestellt, daß der Wert des Zählers den Wert der maximalen Mountvorgänge überschreitet, so wird ein Systemcheck erzwungen.

Im Superblock steht noch eine Angabe, die festlegt, wieviel Prozent eines Dateisystems für den Superuser root reserviert wird. Standardmäßig sind es 5 Prozent.

Um diese Werte zu manipulieren steht das Programm **tune2fs** zur Verfügung. Dieses Programm sollte aber nur für Dateisysteme verwendet werden, die gerade **nicht** read-write gemountet sind!

Journaling Dateisysteme

Ein großer Nachteil der traditionellen Unix-Dateisysteme ist die interne Verwaltung von Dateisysteminformationen. Der Superblock eines Dateisystems wird beim Mounten des Dateisystems in den Arbeitsspeicher geladen und alle Veränderungen des Superblocks werden dort vorgenommen. Die eigentlichen Veränderungen am Dateisystem (Anlegen, Verändern oder Löschen von Dateien) werden aber tatsächlich minütlich auf der Platte vorgenommen. Erst beim Abhängen des Dateisystems (in der Regel beim Herunterfahren) wird der im Speicher liegende Superblock physikalisch auf die Platte geschrieben.

Dieser Mechanismus spart zwar Zeit, weil Zugriffe auf den Arbeitsspeicher wesentlich schneller erledigt werden können, als Zugriffe auf die Platte, aber er birgt große Risiken. Wenn ein solches Dateisystem nicht korrekt abgehängt wird, etwa bei einem Systemabsturz oder Stromausfall, dann kann der aktuelle Superblock nicht mehr auf die Platte geschrieben werden. Der Zustand des Dateisystems ist dann inkonsistent, das bedeutet, daß der Superblock nicht mehr die tatsächliche aktuelle Situation des Dateisystems beschreibt, sondern die, die das Dateisystem beim vorletzten Mounten aufwies.

Linux reagiert auf diese Inkonsistenz mit einem erzwungenen Dateisystemcheck beim Booten, wenn das System merkt, daß eine Platte nicht korrekt abgehängt wurde. Dieser Systemchek kann - je nach Partitionsgröße - sehr lange dauern und im schlimmsten Fall einen manuellen Eingriff erfordert. Das bedeutet, daß ein abgestürzter Server lange nicht mehr zur Verfügung steht und eventuell sogar nicht mehr automatisch wieder hochfährt.

Um dieses Problem zu lösen, wurden moderne Dateisysteme mit einem neuen Mechanismus ausgestattet, dem Journaling. Dabei wird jede Transaktion zwischen dem System und der Platte in einem Journal mitprotokolliert, so daß nach einem Absturz in sehr kurzer Zeit wieder ein konsistenter Zustand hergestellt werden kann. Das bedeutet nicht, daß alle Daten wiederhergestellt werden können, die durch den Absturz womöglich verlorengingen, sondern nur, daß es zu keiner Diskrepanz zwischen Superblock und Dateisystem kommen kann.

Dazu wird beim Systemstart - falls es zu einem Absturz gekommen war - das Journal gelesen und die darin enthaltenen Transaktionen werden quasi Stück für Stück rückgängig gemacht, bis es wieder zu einem konsistenten Zustand kommt. Dieser Vorgang kann wesentlich schneller ausgeführt werden, als ein Dateisystemcheck eines herkömmlichen Dateisystems, weil nicht die ganze Platte überprüft werden muß. Die Dauer ist auch nicht abhängig von der Größe der Partition.

Linux bietet heute mehrere Dateisystemtypen an, die das Journaling beherrschen. Bemerkenswert sind dabei die folgenden:

Ext3fs

Ext3fs ist nicht wie der Name vermuten ließe ein echter Nachfolger von Ext2fs, dem Standard-Dateisystem unter Linux. Ext3fs ist vielmehr eine Erweiterung von Ext2fs, da sich am Dateisystem so gut wie nichts entscheidendes zum "Vorgänger" verändert hat. Die Erweiterung beschränkt sich - etwas übertrieben gesagt - im wesentlichen auf das Hinzufügen einer Datei - dem Logfile. Das Ziel bei der Entwicklung von Ext3fs war es mit minimalen Änderungen eine komplette Lösung zu finden, die das Journaling unterstützt. Hierfür wurde dann eine Kopie des Ext2fs-Quellcodes gemacht und eine textuelle Ersetzung durchgeführt mit der "ext2" durch "ext3" ersetzt wurde. Hinzugefügt wurde dann eine vom Dateisystem selbst völlig unabhängige Schicht die das Journaling übernimmt. Das Dateisystem selber hat nichts mit dem Journaling zu tun. Die einzige Änderung die das System hinter sich hat ist, dass es jede Aktion die Schreibend auf die Festplatte zugreift in eine Transaktion packt. D.h. es wird der Journaling-Schicht mitgeteilt, welche Blöcke innerhalb einer Transaktion geändert werden sollen. Den Rest übernimmt dann die neue Journaling-Schicht.

ReiserFS

Beim Dateisystem des Hans Reiser werden im Journal alle Metadatenänderungen protokolliert, die mehr als einen Block betreffen. Bei einem Systemabsturz ist also lediglich sichergestellt, dass das System sehr schnell wieder in einen konsistenten zustand gesetzt wird. Datenverluste kann es weiterhin geben, da ja "nur" die Metadaten im Journal gesichert werden.

Die Datenstruktur wird hier in einem B*-Baum gespeichert. In den Blättern werden vier verschiedene Arten von

Datensätzen gespeichert:

- direkte Datensätze: Alle kleinen Dateien (< 1 Block) werden direkt im Baum gespeichert. (Eine 10 Zeichen große Datei landet bei Ext2fs in einem Block von z.B. 1024 Bytes) Hierdurch wird Plattenplatz und ein zusätzlicher Plattenzugriff gespart
- indirekte Datensätze: Hier werden Zeiger auf größere Dateien, die dann außerhalb des Baumes liegen, gespeichert
- Verzeichnisse: Enthält die einzelnen Einträge eines Verzeichnisses
- stat data: Was Ext2fs in Inodes speichert wird hier direkt im Baum gespeichert.

Jeder Datensatz besitzt einen eindeutigen Schlüssel (Hashfunktion), der zum Sortieren und Wiederfinden benutzt wird. Durch die Verwaltung des Baumes (ausbalancieren beim Hinzufügen bzw. Löschen von Einträgen) nimmt das Speichern etwas mehr Rechenzeit in Anspruch als der einfache Mechanismus beim ext2.

XFS

XFS ist eine Portierung des Dateisystems von IRIX, das bereits seit langem auf den bekannten Grafik-Workstations und Servern von Silicon-Graphics läuft. Die Version 1.0 erschien am 1.Mai 2001. Da es in diesem Bereich besonders auf Sicherheit und Geschwindigkeit ankommt ist XFS besonders für große Dateien ausgelegt. Wenn man bedenkt, dass Dateien bis zu 9.000 PByte groß sein können (zur Information die Reihenfolge: Kilo-, Mega-, Giga-, Terra-, Peta-, Exa-Byte).

Die technischen Details füllen ganze Seiten. Zu erwähnen wäre z.B. dass mit dem plattformübergreifenden xfsdump/ xfsrestore sogar Dumps von IRIX nach Linux transferiert werden können.

1.104.1 - Erzeugen von Partitionen und Dateisystemen

Beschreibung: Prüfungskandidaten sollten in der Lage sein, Plattenpartitionen zu konfigurieren und Dateisysteme auf Medien wie Festplatten zu erzeugen. Dieses Lernziel beinhaltet verschiedene **mkfs** Kommandos zur Erzeugung von verschiedenen Dateisystemen auf Partitionen, einschließlich *ext2, ext3, reiserfs, vfat* und *xfs*.

Die wichtigsten Dateien, Bezeichnungen und Anwendungen:

- **fdisk**
- **mkfs**

Wie in jedem Betriebssystem, so müssen Festplatten auch unter Linux vorbereitet werden, um vom System benutzt werden zu können. Die beiden Schritte, die nötig sind, sind

- Partitionieren einer Festplatte
- Dateisystem auf den Partitionen erstellen (unter DOS wird das "formatieren" genannt)

Für jede dieser beiden Aufgaben gibt es ein (bzw. mehrere) Programm(e) um sie zu bewältigen.

Partitionieren einer Festplatte mit fdisk

Eine neue Festplatte ist komplett leer. Damit ein beliebiges Betriebssystem auf die Platte zugreifen kann, müssen zunächst einmal bestimmte Bereiche auf der Platte angelegt werden, die das System nutzen kann - die sogenannten Partitionen. Selbst wenn eine Platte nicht unterteilt werden soll, also als ein Bereich genutzt werden soll, muß eine Partition erstellt werden.

Festplatten bestehen aus einer bis mehreren runden übereinanderliegenden Magnetscheiben, die in konzentrische Kreise (Spuren) eingeteilt werden. Alle übereinanderliegenden Spuren werden Zylinder genannt und gemeinsam angesprochen. Das liegt an der Tatsache, daß für jede Scheibenoberfläche ein Schreib/Lesekopf existiert. Alle diese Schreib/Leseköpfe sind starr miteinander verbunden. Wenn einer davon auf der Spur 5 seiner Plattenoberfläche ist, so sind alle anderen auch auf der Spur 5 jeweils ihrer Oberfläche. Daher werden Schreib- und Lesevorgänge immer auf allen untereinanderliegenden Spuren gleichzeitig durchgeführt. So entsteht der Begriff Zylinder, eben alle übereinanderliegenden Spuren.

Partitionen sind jetzt ringförmige Bereiche auf einer Platte. Eine Partition wird durch einen Start- und einen Endzylinder begrenzt. Der erste Zylinder (Zylinder 0) ist nie Teil einer Partition, sondern enthält Verwaltungsinformationen, wie den Master-Boot-Sektor und die Tabelle mit den Partitionsinformationen.

Wenn eine Platte z.B. 1024 Zylinder besitzt und in zwei Partitionen aufgeteilt werden soll, so entstünde etwa eine Aufteilung wie die folgende:

Partition	Startzylinder	Endzylinder
1	1	512
2	513	1023

Jede physikalische Festplatte kann bis zu vier sogenannte primäre Partitionen (primary partitions) aufnehmen. Das liegt daran, daß auf dem Zylinder 0 nur Platz gelassen wurde für vier Einträge in der Partitionstabelle (partition-table).

Eine dieser vier primären Partitionen darf eine sogenannte erweiterte Partition (extended partition) sein, die ihrerseits wieder behandelt wird, wie eine physikalische Festplatte und selbst wieder Unterpartitionen (logical partitions) aufnehmen kann. Linux kann bis zu 11 solcher logischer Partitionen pro erweiterter Partition verwalten.

Linux hat eine eindeutige Namenskonvention für Festplatten und Partitionen. Dabei wird unterschieden zwischen

IDE- und SCSI-Festplatten. Die Namen für die IDE-Festplatten sind:

/dev/hda
: Der Master des ersten IDE-Kanals

/dev/hdb
: Der Slave des ersten IDE-Kanals

/dev/hdc
: Der Master des zweiten IDE-Kanals

/dev/hdd
: Der Slave des zweiten IDE-Kanals

/dev/hde
: Der Master des dritten IDE-Kanals

/dev/hdf
: Der Slave des dritten IDE-Kanals

...

SCSI-Festplatten werden der Reihe nach mit Buchstaben von a bis z durchnummeriert, zuerst alle Platten des ersten SCSI-Hostadapters, dann die des zweiten usw. Für jeden Hostadapter gilt, daß die Platten der Reihe nach durchnummeriert werden, die Platte mit der niedrigsten SCSI-ID zuerst, bis zur Platte mit der höchsten SCSI-ID. Der Name der SCSI-Platten beginnt immer mit **/dev/sd**. Also wäre die erste SCSI-Platte **/dev/sda**, die nächsten **/dev/sdb**, **/dev/sdc** usw.

Die Partitionen jeder Platte sind jetzt einfach als Nummern an diese Namen angehängt. Die primären Partitionen tragen die Nummern 1 bis 4, wobei eine dieser primären Partitionen - unabhängig von ihrer Nummer - eine erweiterte Partition sein kann. Die logischen Partitionen (Unterpartitionen) innerhalb der erweiterten Partition tragen die Nummern 5 bis 15. Daraus lassen sich also aus den Namen der Partitionen eindeutige Informationen gewinnen. Ein paar Beispiele:

/dev/hda1
: Die erste primäre Partition des Masters des ersten IDE-Kanals

/dev/hdb5
: Die erste logische Partition des Slaves des ersten IDE-Kanals

Wenn eine solche Bezeichnung also eine Nummer hat, so ist die jeweilige Partition angesprochen, wenn sie keine Nummer hat, so ist die ganze physikalische Platte gemeint.

Um jetzt eine bestimmte Platte zu partitionieren, müssen wir also zunächst einmal wissen, wie diese Platte heißt. Das Programm zum Partitionieren von Festplatten heißt fdisk und wird zusammen mit dem Namen der zu partitionierenden Platte aufgerufen. Wollen wir also auf der ersten SCSI-Festplatte im System (/dev/sda) Partitionen einrichten, so schreiben wir:

```
fdisk /dev/sda
```

Es erscheint zunächst mal die wenig aussagekräftige Zeile

```
Command (m for help):
```

Also drücken wir doch mal das m:

```
Command action
   a   toggle a bootable flag
   b   edit bsd disklabel
   c   toggle the dos compatibility flag
   d   delete a partition
   l   list known partition types
   m   print this menu
   n   add a new partition
   o   create a new empty DOS partition table
   p   print the partition table
```

```
   q   quit without saving changes
   s   create a new empty Sun disklabel
   t   change a partition's system id
   u   change display/entry units
   v   verify the partition table
   w   write table to disk and exit
   x   extra functionality (experts only)

Command (m for help):
```

Für uns sind nur ein paar wenige dieser Befehle notwendig, insbesondere die folgenden:

a
: Bootflag ein- und ausschalten. Manche Systeme, wie etwa DOS/Windows können nur von Partitionen booten, die den sogenannten Bootflag gesetzt haben oder - im DOS-Jargon - aktiviert sind. Linux benötigt diesen Flag überhaupt nicht.

d
: Delete - Löschen einer Partition. Hier muß die jeweilige Partitionsnummer angegeben werden. Mit dem p-Befehl kann die Partitionstabelle angezeigt werden, die uns die gewünschte Nummer zeigt. Erweiterte Partitionen, die noch logische Partitionen enthalten können nicht gelöscht werden.

l
: Liste aller Partitionstypen. Jede Partition hat eine hexadezimale Kenn-Nummer, die vom jeweiligen System gesetzt wird und dieser Partition einen bestimmten Typ zuweist. Linux benutzt dabei drei Kenn-Nummern, 82 für eine Swap-Partition (eine Erweiterung des Arbeitsspeichers), 83 für eine normale Linux-Plattenpartition und 8e für eine Partition des Logical Volume Managers.

n
: Neue Partition anlegen. Wenn die Platte bisher keine erweiterte Partition besitzt, dann wird zunächst gefragt, ob eine primäre oder eine erweiterte Partition erstellt werden soll. Gibt es schon eine erweiterte Partition, so wird statt dessen nach primärer und logischer Partition gefragt. Existieren schon vier primäre Partitionen und eine davon ist eine erweiterte, dann fällt diese Frage weg, denn es können dann ja nur logische Partitionen angelegt werden.

  Die nächste Frage ist die nach der Nummer der anzulegenden Partition und dann müssen Start- und Endzylinder angegeben werden. Statt einem End-Zylinder kann auch eine Größenangabe gemacht werden, die mit einem Pluszeichen (+) eingeleitet werden muß und einen Größenpostfix M oder K für Mega- oder Kilobyte besitzen darf. So bedeutet `+256M` eben eine Partitionsgröße von 256 Megabyte.

p
: Print Partition-Table - Gibt die Partitionstabelle der Platte aus.

t
: Wechselt den Typ der Partition. Hier kann dann einer der Typen angegeben werden, die mit dem l-Befehl angezeigt wurden.

q
: Quit - Beendet das Programm ohne die vorgenommenen Änderungen tatsächlich vorzunehmen. Es bleibt also alles beim Alten.

w
: Write Partition-Table - Schreibt die vorgenommenen Änderungen in die Partitionstabelle und beendet das Programm.

Mit diesen Befehlen sind alle denkbaren Aktionen durchzuführen, die wir für den Umgang mit Linux normalerweise brauchen. Wenn Sie allerdings nur die Partitionstabelle einer physikalischen Festplatte anzeigen wollen, so können Sie fdisk auch einfach mit dem Parameter `-l` aufrufen, dann zeigt es nur die Partitionstabelle des anfgegebenen Laufwerks (oder aller Laufwerke, wenn keines angegeben wurde) und beendet sich dann sofort wieder.

Erstellen eines Dateisystems mit mkfs und seinen Verwandten

Bevor eine Partition (oder eine Diskette) von Linux benutzt werden kann, muß zunächst ein Dateisystem erstellt werden. Das ist die gleiche Prozedur, die unter DOS/Windows "Formatieren" genannt wird. Das Prinzip eines Unix/Linux Dateisystems unterscheidet sich aber grundlegend von dem des DOS/Windows Dateisystems. So finden Sie

hier zunächst einmal ein kurzen Überblick über den Aufbau von Unix-Inode-Systemen, dessen Verständnis im Folgenden vorausgesetzt wird, wenn wir im Einzelnen das Erstellen des Dateisystems besprechen. Dieses Verständnis wird insbesondere auch im nächsten Kapitel wichtig sein.

Das I-Node System

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*1024 oder 10*2048 oder 10*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 ergenen sich dadrch 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.

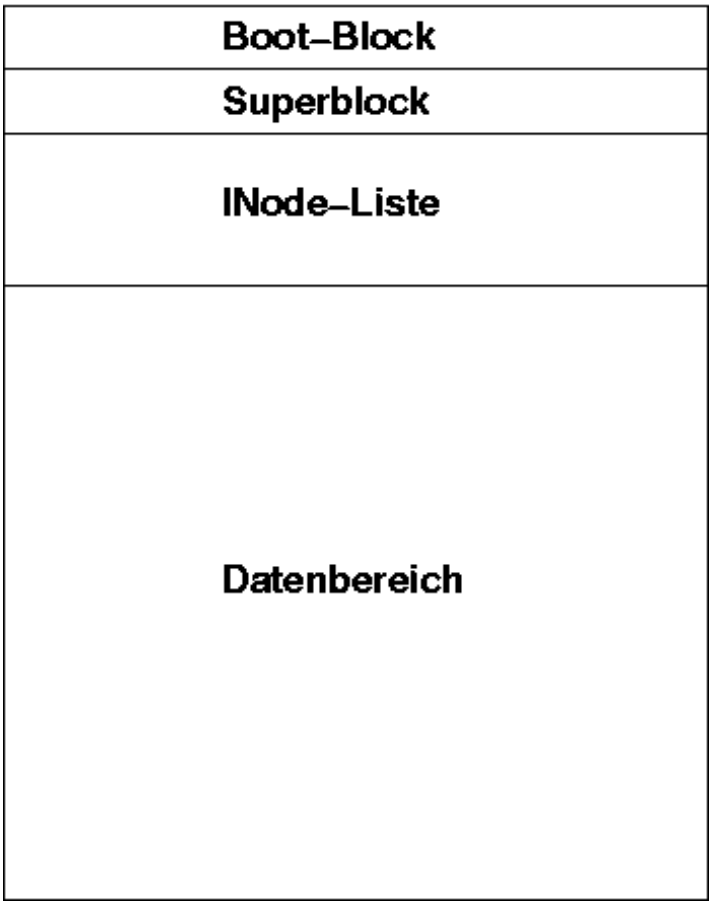
Der Superblock

Ein Unix-Dateisystem besitzt einen sogenannten 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 sogenannte Bootblock auf der Platte, der in etwa dem Bootsektor der DOS-Partitionen entspricht. Zusammengefasst kann man also ein Unix-Dateisystem etwa wie folgt darstellen:



Der häufigste Dateisystemtyp unter Linux ist das *Second Extended Filesystem* oder kurz EXT2. Es unterscheidet sich in einzelnen Details vom Standard-Unix-Dateisystem, benutzt aber die selben Techniken, was I-Nodes angeht. Für das Verständnis der LPIC-Level1 Prüfung reicht das Wissen um diesen Standard. Für die speziell Interessierten findet sich hier noch Informationen über das EXT2 Filesystem:

Das EXT2 Dateisystem

Im Wesentlichen entspricht das Second Extended Filesystem (EXT2) von Linux dem Unix-Inode-System, das an anderer Stelle schon beschrieben wurde. Es gibt aber ein paar wesentliche Unterschiede, die hier kurz erläutert werden sollen.

Der Aufbau einer EXT2 Inode

Die wesentlichen Unterschiede des EXT2 Systems zum "normalen" I-Node-System liegen in der Struktur der I-Nodes selbst. Hier sind einige Felder hinzugekommen, die für bestimmte Erweiterungen vorgesehen wurden, die im folgenden genauer erklärt werden. Doch zunächst einmal die Struktur der Inode selbst:

permission	**links**	**owner**	**group**
size		**creation time**	
modification time		**access time**	
deletion time		**blockcount**	
flags		**file version (NFS)**	
file ACL		**dir ACL**	
fragment addr.	**fr. size**	**frag. nr**	**reserved**
1. block data		**2. block data**	

3. block data	**4. block data**
5. block data	**6. block data**
7. block data	**8. block data**
9. block data	**10. block data**
11. block data	**12. block data**
simple indirect	**double indirect**
triple indirect	**reserved**
reserved	**reserved**

Die wesentlichen Unterschiede zum bisher beschriebenen I-Node-System sind:

- Neben den drei Standard-POSIX-Zeitmarken (ctime, atime, mtime) gibt es hier eine vierte Zeitmarke, die die Löschzeit (*deletion time*) speichert. Das ist gedacht für Spezialprogramme zum Wiederherstellen versehentlich gelöschter Dateien.
- Ein Feld für *flags* ist hinzugekommen. Diese Flags ermöglichen es, daß Dateien bestimmte Eigenschaften bekommen, die weiter unten genauer beschrieben werden.
- Der Eintrag *file version* kann vom NFS-Server zur Unterscheidung verschiedener Versionen einer Datei verwendet werden.
- Zwei Einträge für eine erweiterte Zugriffskontrolle für Dateien (*file ACL*) und Verzeichnisse (*dir ACL*) sind hinzugekommen. ACL steht für Access Control List.
- Die Unterstützung fragmentierter Datenzonen ist vorgesehen.
- Es stehen 12 direkte Adressfelder für Datenblocks zur Verfügung.

Die Flags des EXT2 Filesystems

Jede Datei in einem EXT2 Dateisystem kann neben den normalen Unix-Zugriffsrechten noch einen Satz zusätzlicher Flags bekommen, die hier noch erläutert werden sollen. Um sich die Flags (oder auch Attribute) anzusehen, wird das Programm **lsattr** benutzt, zum Ändern der Attribute wird **chattr** benutzt. Es gibt die Attribute A, a,c,d,i,S,s und u.

- Wenn eine Datei mit gesetztem A-Attribut verändert wird, so wird das *atime* Feld nicht verändert.
- Eine Datei, die das a-Attribut gesetzt hat, kann nur im Append-Mode geöffnet werden. Das heißt, Daten können an diese Datei angehängt werden, bestehende Daten jedoch nicht gelöscht.
- Eine Datei mit dem c-Attribut wird automatisch vom Kernel komprimiert, bevor sie auf die Platte geschrieben wird. Wenn sie gelesen wird, dann entkomprimiert der Kernel die Datei wieder, ein Anwender merkt also nichts von der Kompression.
- Eine Datei mit gesetztem d-Attribut ist **kein** Kandidat für ein Backup mit dem *dump* Befehl.
- Eine Datei mit gesetztem i-Attribut kann nicht verändert werden, sie kann nicht gelöscht oder umbenannt werden, kein Link kann auf sie erstellt werden und keine Daten können in ihr verändert werden. Dieses Attribut kann nur vom Superuser gesetzt oder entfernt werden.
- Wenn eine Datei mit gesetztem s-Attribut gelöscht wird, dann werden die Bytes der Datei mit Nullen überschrieben. Ein sicheres Löschen, das keine Wiederherstellung erlaubt.
- Wenn eine Datei, deren S-Attribut gesetzt ist, verändert wird, dann wird diese Veränderung synchron auf die Platte geschrieben, ohne lange im Cache zu liegen.
- Wenn eine Datei mit gesetztem u-Attribut gelöscht wird, dann wird ihr Inhalt gesichert, so daß sie wiederhergestellt werden kann.

Erweiterungen des Superblocks

Der Superblock eines EXT2-Dateisystems enthält ein sogenanntes *Valid-Flag*. Sobald dieses Dateisystem gemountet wird, wird das Valid-Flag gelöscht. Erst beim Umount wird es wieder gesetzt. Sollte das System abstürzen, so bleibt das Valid-Flag ungesetzt - beim nächsten Start kann also erkannt werden, daß dieses Dateisystem nicht ordnungsgemäß abgehängt worden ist.

Das Programm zur Überprüfung eines EXT2 Dateisystems (e2fsck) überprüft dieses Valid-Flag und prüft (falls es nicht durch die -f Option gezwungen wurde) nur dann das Dateisystem, wenn das Valid-Flag nicht gesetzt war. So kann beim Systemstart einfach für jede Partition das e2fsck-Programm aufgerufen werden. Es wird nur dann aktiv,

wenn es nötig ist.

Der Superblock enthält weiterhin eine Angabe über die maximale Anzahl von Mount-Vorgängen, die zwischen zwei System-Checks ablaufen dürfen. Jedesmal, wenn ein Dateisystem gemountet wird, wird ein Zähler (*mount-count*) um eins erhöht. Wird bei einem Systemstart festgestellt, daß der Wert des Zählers den Wert der maximalen Mountvorgänge überschreitet, so wird ein Systemcheck erzwungen.

Im Superblock steht noch eine Angabe, die festlegt, wieviel Prozent eines Dateisystems für den Superuser root reserviert wird. Standardmäßig sind es 5 Prozent.

Um diese Werte zu manipulieren steht das Programm **tune2fs** zur Verfügung. Dieses Programm sollte aber nur für Dateisysteme verwendet werden, die gerade **nicht** read-write gemountet sind!

Das Programm, mit dem wir Dateisysteme erstellen heißt `mkfs` (Make FileSystem) und ist eigentlich nur ein Frontend für weitere Programme, die dann für jedes Linux-Dateisystem extra zur Verfügung stehen, wie etwa `mkfs.ext2, mkfs.minix, mkfs.msdos` oder `mkfs.xiafs`.

Der grundsätzliche Aufruf von mkfs ist

```
mkfs  [ -t Dateisystemtyp ] [ Optionen ] Gerätedatei [ Blocks ]
```

Dabei sind die Optionen abhängig vom verwendeten Dateisystemtyp. Wird der Dateisystemtyp weggelassen wird heutzutage EXT2 angenommen, aber darauf gibt es keine Garantie. Werden die Blocks weggelassen, so werden alle zur Verfügung stehenden Blocks verwendet.

Das Programm zur Erstellung von EXT2-Dateisystemen kennt die folgenden wichtigen Parameter:

-b *Blockgröße*
: Spezifiziert die Größe der Blocks in Byte. Gültige Größen sind 1024, 2048 und 4096.

-c
: Checkt das Gerät nach schlechten Blocks ab, bevor das Dateisystem erstellt wird.

-i *Bytes-Per_Inode*
: Stellt das Verhältnis von Plattenplatz zur Menge der Inodes ein. Je größer der angegebene Wert ist, um so weniger Inodes werden erstellt. Dieser Wert sollte grundsätzlich kleiner als die Blockgröße des Dateisystems sein.

-N *Anzahl der Inodes*
: Gibt die absolute Anzahl der zu erstellenden Inodes an. Wird anstatt des -i Parameters verwendet bzw. überschreibt dessen Einstellungen.

In der Regel ist es ausreichend, ein Dateisystem mit den Standard-Parametern anzulegen, also einfach gar keine Parameter anzugeben. ein einfaches

```
mkfs /dev/hdb5
```

legt ein solches Dateisystem auf der Partition 5 der zweiten IDE-Platte an. In Ausnahmefällen ist es aber notwendig, bestimmte Parameter zu verändern. So werden beispielsweise beim Anlegen eines EXT2-Dateisystems auf einer Diskette nur 184 Inodes angelegt. Normalerweise würde das sicher auch reichen, um die 1,44 MByte Diskettenplatz aufzuteilen. Wenn wir aber etwa eine Bootdiskette erstellen wollen, die schon alleine hunderte von Gerätedateien im `/dev`-Verzeichnis braucht, ist diese Einstellung definitiv nicht brauchbar. Hier müssten wir also mit dem -N Parameter manuell die Anzahl der zu erstellenden Inodes hochsetzen.

Explizites Anlegen eines EXT2- oder EXT3-Dateisystems

Um sicherzugehen, daß tatsächlich ein Dateisystem des Typs EXT2 (second extended) oder seine Erweiterung zum Journaling Dateisystem (ext3) erzeugt wird, muß entweder der Dateisystemtyp mit der Option `-t ext2` angegeben werden, oder es wird gleich das entsprechende Programm mke2fs aufgerufen.

Mit diesem Programm können beide Dateisystemtypen erzeugt werden. Um ein Journaling-Dateisystem vom Typ ext3 zu erzeugen muß der Parameter `-j` (journaling) mit angegeben werden.

Anstatt **mke2fs** kann auch **mkfs.ext2** oder **mkfs.ext3** angegeben werden.

Explizites Anlegen eines Reiser-Dateisystems

Wenn ein Dateisystem des Typs REISERFS angelegt werden soll, so muß dem **mkfs**-Programm der Parameter `-t reiserfs` angegeben werden, oder es wird das Programm mkreiserfs direkt aufgerufen.

Anstatt **mkreiserfs** kann auch **mkfs.reiserfs** angegeben werden.

Eine explizite Angabe der anzulegenden Inodes entfällt bei dieser Dateisystemart, weil das Reiserfs die Inodes dynamisch anlegt, je nach Bedarf.

Explizites Anlegen eines XFS-Dateisystems

Wenn ein Dateisystem des Typs XFS angelegt werden soll, so muß dem **mkfs**-Programm der Parameter `-t xfs` angegeben werden, oder es wird das Programm **mkfs.xfs** direkt aufgerufen.

Explizites Anlegen eines VFAT-Dateisystems

Wenn ein Dateisystem des Typs VFAT (Windows Dateisystem) angelegt werden soll, so muß dem **mkfs**-Programm der Parameter `-t vfat` angegeben werden, oder es wird das Programm **mkdosfs** direkt aufgerufen.

Anstatt **mkdosfs** kann auch **mkfs.vfat** angegeben werden.

1.104.2 - Erhaltung der Dateisystemintegrität

Beschreibung: Prüfungskandidaten sollten in der Lage sein, die Integrität von Dateisystemen zu prüfen, freien Speicherplatz und Inodes zu überwachen und einfache Dateisystemprobleme zu beheben. Dieses Lernziel beinhaltet die Kommandos, die für die Verwaltung eines Standard-Dateisystems notwendig sind sowie die zusätzlichen Notwendigkeiten eines *Journaling* Dateisystems.

Die wichtigsten Dateien, Bezeichnungen und Anwendungen:

- **du**
- **df**
- **fsck**
- **e2fsck**
- **mke2fs**
- **debugfs**
- **dumpe2fs**
- **tune2fs**

Dieser Bereich hat eine verhältnismäßig hohe Bewertung und es kommen in der LPI 101 Prüfung wirklich einige Fragen zu diesem Thema an die Reihe. Zunächst einmal ist es notwendig, die Architektur der Unix-Dateisysteme zu verstehen, das wurde bereits auf den letzten Seiten erklärt. Ein großer Schwerpunkt liegt im Verständnis des I-Node-Systems. So ist z.B. eine häufige Fehlerursache das Fehlen von freien I-Nodes. Obwohl auf einer Platte noch haufenweise Megabytes frei wären kann keine Datei mehr angelegt werden. Daher sind Techniken notwendig, die das Erkennen solcher Probleme ermöglichen.

Das Programm fsck und seine Verwandten

Wie schon beim Anlegen von Dateisystemen, so ist auch beim Reparieren bzw. Prüfen der Systeme für jedes Dateisystem ein spezielles Programm vorhanden, das genau das jeweilige System kennt. Wie beim Anlegen gibt es aber eben auch wieder ein sogenanntes Frontend, das dann die jeweiligen Programme aufruft. Dieses Frontend heißt **fsck** (FileSystemCheck). Dieses Frontend ruft dann die einzelnen Filesystem-Checker für die jeweiligen Dateisysteme auf, als da wären:

- e2fsck bzw. fsck.ext2 für das Second Extended Filesystem
- reiserfsck für das ReiserFS
- fsck.minix für Minix Dateisysteme
- fsck.msdos für DOS-FAT-Systeme
- fsck.vfat für Windows-VFAT-Systeme
- fsck.xfs für XFS-Dateisysteme

Grundsätzlich ist jedes dieser Programme dafür gedacht, die Konsistenz eines Dateisystems zu überprüfen und gegebenenfalls zu reparieren.

Die Anwendung des fsck-Programms sollte immer nur auf nicht gemounteten Dateisystemen stattfinden, da sonst die Gefahr droht, daß ein Schreibzugriff eines anderen Prozesses (wie etwa die ständige Synchronisation) Veränderungen vornimmt, die den Check bzw. die Reparatur durcheinanderbringen und so den Schaden nur vergrößern. Das ist allerdings ein Problem beim Überprüfen des Wurzel-Dateisystems, da es nicht so einfach möglich ist, es zu checken, ohne es zu mounten. In diesem Fall sollte grundsätzlich in den Single-User-Mode gewechselt werden und das Wurzeldateisystem sollte Read-Only gemountet sein!

Beim Systemstart wird das Programm fsck mit der Option -A aufgerufen, was das Programm veranlasst, alle Dateisysteme zu überprüfen, die in /etc/fstab aufgelistet sind. Die Reihenfolge ist dabei grundsätzlich durch die Angabe des sechsten Feldes innerhalb der /etc/fstab Datei geklärt. Das Wurzeldateisystem wird zuerst überprüft und dann werden entsprechend den Nummern in diesem sechsten Feld (fs_passno) die anderen Dateisysteme der Reihe nach abgearbeitet. Wenn mehrere solcher Systeme die gleiche Nummer haben, dann wird versucht, sie gleichzeitig zu bearbeiten.

Soll ein Dateisystem manuell (nicht beim Systemstart) überprüft werden, so gibt es ein paar Dinge zu bedenken. Neben der Tatsache, die oben schon erwähnt wurde, daß das System nicht gemountet sein sollte, gibt es ein paar zu bemerkende Optionsschalter, die bekannt sein müssen.

Die grundsätzliche Aufrufform ist

```
fsck Optionen  Dateisystem
```

Um das Dateisystem jetzt manuell zu überprüfen, sollten zumindestens die wichtigsten Optionen bekannt sein, sonst kann es schlimmstenfalls dazu kommen, daß es gar nicht überprüft wird. Die folgenden Optionen beziehen sich hauptsächlich auf das EXT2 Dateisystem, das im Augenblick sicherlich der Standard unter Linux ist.

-f
: force - die Überprüfung wird erzwungen, auch wenn das Dateisystem ein gesetztes Valid-Flag hat. Das ist im Handbetrieb fast immer der Fall, daher ist das -f ein sehr wichtiger Parameter.

-p
: preen - Automatische Reparatur ohne jede Nachfrage.

-n
: no - Das Dateisystem wird ReadOnly geöffnet und alle Fragen, ob eine bestimmte Aktion durchgeführt werden soll, werden automatisch mit n(ein) beantwortet. Es werden also keine Veränderungen durchgeführt, aber man kann sehen, was passieren würde...

-y
: yes - Das genaue Gegenteil von -n. Alle gestellten Fragen werden mit y(es) beantwortet.

Die Angabe des Dateisystems erfolgt in Form der entsprechenden Gerätedatei. Ein manueller Aufruf könnte also z. B. so aussehen:

```
e2fsck -f /dev/hda7
```

Damit würde das Dateisystem auf der siebten Partition (der dritten logischen Partition innerhalb der erweiterten Partition) des Masters des ersten IDE-Kanals zwingend (-f) überprüft.

Die Überprüfung und Reparatur von Journaling-Dateisystemen läuft grundsätzlich anders ab, als die von herkömmlichen Dateisystemen. Hier wird - im Falle einer Inkonsistenz - der Transaction-Log - eben das Journal - zurückverfolgt und alle Transaktionen rückgängig gemacht, bis das System wieder konsistent ist. Dieser Vorgang ist wesentlich schneller, als der bei einem traditionellen Dateisystem, da nicht die ganze Platte überprüft werden muß.

Das Programm df

Das Programm df (Disk Free) dient dazu, die Belegung einzelner Dateisysteme (oder aller gemounteten Dateisysteme) zu ermitteln. Die Anwendung ist sehr einfach, wird df ohne Parameter angewandt, so zeigt es alle gemounteten Dateisysteme etwa in der folgenden Form:

```
Filesystem           1k-blocks      Used Available Use% Mounted on
/dev/hda2              2071328   1051656    914448  53% /
/dev/hda5              3099108   1737096   1204580  59% /usr
/dev/hda6              2071296    767708   1198364  39% /opt
/dev/hda7              2071296    215212   1750860  11% /home
```

Aus dieser Ausgabe ist also zu entnehmen, welche Dateisysteme gerade gemountet sind, wieviel 1 Kilobyte-Blocks insgesammt zur Verfügung stehen (1k-blocks), wieviel davon belegt sind (Used), wieviel also noch frei sind (Available), die prozentuale Auslastung - also wieviel Prozent sind belegt (Use%) und schließlich der Mountpoint, an dem das Dateisystem eingehängt ist.

Wird stattdessen der Befehl df mit einem bestimmten Dateisystem aufgerufen, entweder durch die Nennung des Mountpoints oder durch die Angabe der entsprechenden Gerätedatei, so werden nur die Angaben über dieses Dateisystem ausgegeben. Hätten wir also entweder

```
df /usr
```

oder

```
df /dev/hda5
```

eingegeben, so wäre es zur folgenden Ausgabe gekommen:

```
Filesystem           1k-blocks      Used Available Use% Mounted on
/dev/hda5              3099108   1737096   1204580  59% /usr
```

Ein wichtiger Parameter für df ist noch die Angabe -i oder --inodes. Wird **df** mit dieser Option aufgerufen, so werden statt den Angaben über die Kilobyte-Blöcke jetzt Angaben über die I-Nodes gemacht. die Ausgabe sähe jetzt also folgendermaßen aus:

```
Filesystem            Inodes   IUsed   IFree IUse% Mounted on
/dev/hda2             263296   16769  246527    6% /
/dev/hda5             393600  100095  293505   25% /usr
/dev/hda6             263296   32595  230701   12% /opt
/dev/hda7             263296   11088  252208    4% /home
```

Jetzt sehen wir also die Anzahl aller Inodes, die Anzahl der benutzten Inodes (IUsed), die Anzahl der freien Inodes (IFree) und wieder die prozentuale Auslastung (IUse%).

Diese Angaben sind auserordentlich wichtig, weil - wie oben schon erwähnt - es dazu kommen kann, daß zwar noch reichlich Platz in Kilobyte auf einer Partition sein kann, jedoch keine Inodes mehr frei sind, weil sehr viele, sehr kleine Dateien darauf gespeichert sind.

Auch hier gibt es einen wichtigen Hinweis für moderne Journaling-Dateisysteme. Diese Systeme speichern ihre Daten in einer völlig anderen Struktur ab, in sogenannten B-Bäumen. Das hat zur Folge, daß solche Systeme keinen reservierten Platz für die Inodes aufweisen, also Inodes dynamisch anlegen können, wenn sie benötigt werden. Aus diesem Grund zeigt **df -i** bei solchen Systemen keine realen Werte für die Inodes an.

Das Programm du

Das Programm du (Disk Usage) zeigt den Platzbedarf einzelner Dateien bzw. Verzeichnisse an. Das ist z.B. wichtig, wenn es darum geht, herauszufinden, welches Verzeichnis besonders viel Platz benötigt auf einer Partition, von der der df-Befehl gezeigt hatte, daß der Platz langsam knapp wird.

Das Programm ist in der Regel nur sinnvoll mit Kommandozeilenparametern anwendbar, weil es sonst alle Dateien und Verzeichnisse zeigt und die Ausgabe so etwas unübersichtlich wird. Der wichtigste Optionsschalter ist **-s**, der dafür sorgt, daß nur die Summe der verwendeten Bytes aller übergebenen Verzeichnisse ausgibt. So ist schnell feststellbar, wieviel Platz ein Verzeichnis mit allen darin enthaltenen Dateien und Unterverzeichnissen benötigt. Der Aufruf

```
du -s /opt
```

führt dann nur noch zu einer Ausgabe, die etwa folgendermaßen aussehen könnte:

```
767708 /opt
```

Die Angaben erfolgen normalerweise in Kilobyte, die obige Ausgabe besagt also, daß 760 Megabyte im /opt-Verzeichnis belegt sind. Wenn wir jetzt wissen wollen, wie sich diese Summe zusammensetzt, dann können wir einfach alle Verzeichnisse mitangeben, indem wir schreiben:

```
du -s /opt/*
```

und bekommen jetzt eine Auflistung wie folgt:

```
20      /opt/Office51
20      /opt/fsuite
88556   /opt/gnome
157568  /opt/kde
184220  /opt/kde2
16      /opt/lost+found
31372   /opt/netscape
20924   /opt/netscape6
4       /opt/nps
258728  /opt/office52
4       /opt/oracle
8       /opt/skyrix
9476    /opt/slab
11764   /opt/tfd
8       /opt/tngfw
4       /opt/virtuoso-lite
5012    /opt/www
```

Jetzt lässt sich also schon ziemlich genau sagen, wer hier den vielen Platz braucht...

Werkzeuge für den Umgang mit Ext2-Dateisystemen

In dieser Stufe der Linux-Zertifizierung werden für die hier genannten Werkzeuge noch keine umfassenden Kenntnisse verlangt. Wichtig ist, zu wissen, daß sie existieren und welche Aktionen damit vorgenommen werden können. An dieser Stelle also nur eine kurze Beschreibung der entsprechenden Tools mit jeweils einem Hinweis auf die Handbuchseiten. Ich habe die entsprechenden Handbuchseiten übersetzt, ein Studium dieser Information kann sicherlich nicht schaden.

- mke2fs
  Das Werkzeug zum Anlegen des Dateisystems wurde in Abschnitt 1.104.1 - Erzeugen von Partitionen und Dateisystemen bereits ausführlich besprochen.
- debugfs
  Ein mächtiges Werkzeug, um ein EXT2-Dateisystem zu bearbeiten. Mit diesem Programm ist es möglich, manuell alle möglichen Einstellungen des Dateisystems zu verändern, aber auch es auf einen Schlag unbrauchbar zu machen.
- dumpe2fs
  Gibt den Superblock und Block-Gruppeninformationen eines EXT2-Dateisystems auf die Standard-Ausgabe aus. So kann diese wichtige Information zwischengespeichert werden um damit ein Dateisystem manuell wiederherzustellen.
- tune2fs
  Ermöglicht wichtige Einstellungen von Dateisystem-Parametern des EXT2-Dateisystems.

1.104.3 - Kontrolle des Ein- und Aushängen von Dateisystemen

Beschreibung: Prüfungskandidaten sollten in der Lage sein, das Einhängen (Mounten) eines Dateisystems zu konfigurieren. Dieses Lernziel beinhaltet die Fähigkeit, Dateisysteme manuell ein- und auszuhängen, die Konfiguration des Mountens von Dateisystemen bei Systemstart und das Konfigurieren von wechselbaren Datenträgern, die von Benutzern gemountet werden können, wie z.B. Bänder, Disketten und CDs.

Die wichtigsten Dateien, Bezeichnungen und Anwendungen:

- `/etc/fstab`
- **mount**
- **umount**

Unter Linux/Unix werden die verschiedenen Laufwerke nicht wie etwa unter Windows als einzelne Dateisysteme mit eigenem Laufwerksbuchstaben verwaltet, sondern sie werden zu einem einzigen Baum zusammengebaut. Der englische Begriff *to mount* (anbringen, montieren) steht für das Einhängen der verschiedenen Dateisysteme in den Dateibaum. Dieses Einhängen geschieht entweder automatisch beim Systemstart, oder einzelne Dateisysteme werden manuell während des Betriebs *gemountet*. Dieses manuelle Mounten bezieht sich in der Regel auf Wechsemedien, wie etwa Disketten, CD-ROMs oder auch Netzverbindungen (im Windows-Jargon: *Netzlaufwerke*)

In diesem Zusammenhang sind für uns zwei Programme und zwei Dateien wichtig. Die Programme sind diejenigen, die wir zum Einhängen (*mount*) und Abhängen (*umount*) benötigen, die Dateien zeigen, was wohin eingehängt werden soll (/etc/fstab) und was gerade eingehängt ist (/etc/mtab). Diese vier Elemente sollen hier dargestellt werden.

Das Programm mount

Mit dem Programm mount werden Dateisysteme an bestimmte Plätze, also in bestimmte Verzeichnisse eingehängt, neudeutsch gemountet. Die grundsätzliche Form des Programms ist simpel:

```
mount [-t Dateisystemtyp] [ -o Optionen] [Gerätedatei] [Mountpoint]
```

Die meisten Dateisystemtypen werden heute von mount selbstständig erkannt, die Angabe des Dateisystemtyps kann daher meist weggelassen werden. Die Optionen sind abhängig vom verwendeten Dateisystemtyp, sie werden später noch etwas genauer dargestellt. Wichtig ist also zunächst einmal die Angabe, welches Gerät (Gerätedatei in /dev) soll wohin eingehängt werden.

Um z.B. eine Diskette (/dev/fd0) ins Verzeichnis /floppy einzuhängen würde der Befehl

```
mount /dev/fd0 /floppy
```

genügen. Nachdem dieser Befehl ausgeführt wurde ist der Inhalt der Diskette im Verzeichnis /floppy zu finden. Eventuelle Inhalte, die vorher in diesem Verzeichnis waren sind jetzt unsichtbar, sobald das Diskettenlaufwerk aber wieder abgehängt ist, sind sie wieder vorhanden.

Sobald ein Dateisystem mit dem mount-Befehl eingehängt wurde (und dabei nicht die Option -n gesetzt wurde) wird ein Eintrag in die Datei /etc/mtab geschrieben, der das Dateisystem und den Mountpoint beschreibt.

Existiert bereits ein Eintrag in /etc/fstab, der das zu mountende Dateisystem beschreibt, so genügt dem mount-Befehl die Angabe entweder der Gerätedatei oder des Mountpoints, um das Dateisystem einzuhängen.

Soll statt eines lokalen Dateisystems ein NFS-Verzeichnis gemountet werden, so muß statt der Gerätedatei der Rechnername des NFS-Servers, gefolgt von einem Doppelpunkt ":" und dem freigegebenen Verzeichnispfad angegeben werden. So würde die Zeile

```
mount einstein.my.domain:/usr/public /mnt
```

das Verzeichnis `/usr/public` des Rechners `einstein.my.domain` in das lokale Verzeichnis `/mnt` einhängen.

Der mount-Befehl kann auch einfach mit dem Parameter -a aufgerufen werden, ohne Nennung eines Dateisystems. Dann werden alle Dateisysteme eingehängt, die in der Datei /etc/fstab stehen und dort nicht die Option noauto gesetzt haben. Das passiert gewöhnlich beim Hochfahren des Systems.

Die Optionen zum Einhängen der Dateisysteme

Das Programm mount kann mit dem -o Kommandozeilenparameter verschiedene Optionen setzen, wie ein Dateisystem gemountet werden soll. Die selben Optionen können auch in der Datei /etc/fstab angegeben werden (siehe unten).

Folgende Optionen werden von allen Dateisystemen verstanden:

async
: Alle Ein-/Ausgabe Operationen des Dateisystems werden asynchron vorgenommen.

atime
: Die Zugriffszeit der Inodes (atime) werden bei jedem Zugriff gesetzt. (Voreinstellung)

auto
: Wenn diese Option in /etc/fstab steht, wird das Dateisystem gemountet, wenn mount mit der Option -a aufgerufen wird.

defaults
: Entspricht den voreingestellten Optionen: rw, suid, dev, exec, auto, nouser und async

dev
: Gerätedateien auf dem Dateisystem sind gültig und werden interpretiert.

exec
: Erlaubt die Ausführung von Binärdateien auf dem Dateisystem.

noatime
: Die Inode-Zugriffszeiten (atime) werden nicht bei jedem Zugriff gesetzt.

noauto
: Wenn ein Eintrag in /etc/fstab diese Option gesetzt hat, so wird dieses Dateisystem nicht durch den Befehl mount -a eingehängt. Also wird es auch nicht beim Systemstart automatisch gemountet.

nodev
: Gerätedateien auf diesem Dateisystem werden nicht interpretiert.

noexec
: Verbietet die Ausführung von Binärdateien auf diesem Dateisystem.

nosuid
: Die SUID und SGID Bits von Programmen auf diesem Dateisystem werden ignoriert.

nouser
: Ein Normaluser (nicht root) darf dieses Dateisystem nicht mounten.

remount
: Ein bereits gemountetes Dateisystem soll neu gemountet werden. Das wird meist benutzt, um die Optionen eines bereits gemounteten Dateisystems neu zu setzen, insbesondere um ein ReadOnly Dateisystem wieder beschreibbar zu mounten.

ro
: Read Only - Das Dateisystem wird ReadOnly (nur lesbar) gemountet.

rw
: ReadWrite - Das Dateisystem wird ReadWrite (les- und schreibbar) gemountet.

suid
: SUID und SGID Bits werden interpretiert.

sync
: Alle Ein- und Ausgabeoperationen werden synchron durchgeführt.

user
: Erlaubt einem Normaluser, das Dateisystem zu mounten.

Neben diesen Optionen, die für alle verwendeten Dateisystemtypen gelten, existieren noch viele verschiedene Optionen für jeden einzelnen Dateisystemtyp. Diese Optionen hier darzustellen würde einerseits den Rahmen dieser Darstellung sprengen und andererseits sind sie auch nicht notwendiger Bestandteil der LPIC 101 Prüfung.

Wer sie nachlesen will, kann das auf der Handbuchseite des mount-Befehls tun.

Das Programm umount

Das Programm umount (**nicht** unmount !!) hängt ein oder mehrere eingehängte Dateisysteme wieder ab. Das abzuhängende Dateisystem kann entweder durch die Nennung der entsprechenden Gerätedatei oder durch die Angabe des Mountpoints spezifiziert werden.

Ein Dateisystem kann nicht abgehängt werden, wenn es in Benutzung (busy) ist. Das ist in der Regel dann der Fall, wenn ein Prozess ein Verzeichnis dieses Dateisystems als aktuelles Arbeitsverzeichnis hat.

Wird umount mit dem Parameter -a aufgerufen, so werden alle Dateisysteme abgehängt, die in der Datei /etc/mtab aufgelistet sind. Das passiert gewöhnlich während des Shutdowns.

Die Datei /etc/mtab

Die Datei /etc/mtab enthält immer eine Liste aller gerade gemounteten Dateisysteme. Diese Datei wird niemals von Hand editiert, sie ist ausschließlich von den Programmen mount und umount zu beschreiben. Es existieren zwar für beide Befehle jeweils die Option -n, die verhindert, daß dieser Eintrag gemacht wird, das dient aber nur dazu, daß auch auf Systemen, deren /etc-Verzeichnis ReadOnly gemountet ist, beide Befehle funktionieren.

Die Datei /etc/fstab

Die Datei /etc/fstab ist sozusagen die Bauanleitung des Systems, in der genau steht, welches Dateisystem wohin gemountet werden soll. Es obliegt dem Systemadministrator, diese Datei zu erstellen und zu pflegen. Jedes Dateisystem wird durch eine separate Zeile in der fstab repräsentiert; innerhalb einer Zeile werden die Felder durch Tabs oder Leerzeichen getrennt. Die Reihenfolge der Zeilen in der fstab ist wichtig, da fsck(8), mount(8), und umount(8) diese Datei sequentiell abarbeiten.

Das erste Feld, (fs_spec), beschreibt das zu mountende blockorientierte Device oder remote filesystem.

Das zweite Feld, (fs_file), gibt den Mountpunkt für das Dateisystem an. Bei Swap-Partitionen sollte hier `none'' stehen.

Das dritte Feld, (fs_vfstype), beschreibt den Typ des Dateisystems. Hier steht entweder (bei festen Partitionen) das jeweilige Kürzel für das Dateisystem, das auf der Partition angelegt ist (ext2, minix, vfat, swap, ...) oder (bei Wechselplattenlaufwerken wie Zip- oder Diskettenlaufwerken) auto, damit der mountbefehl den jeweiligen Typ selbst erkennt. Bei NFS-Verbindungen steht hier entsprechend der Begriff nfs.

Das vierte Feld, (fs_mntops), beschreibt die zum Dateisystem gehörenden Mountoptionen. Hier werden die Optionen, die oben näher beschrieben wurden, als eine durch Kommas getrennte Liste angegeben. Falls keine speziellen Optionen gewünscht sind, wird hier der Begriff defaults eingegeben.

Das fünfte Feld, (fs_freq), wird von dump(8) benutzt um zu entscheiden welche Dateisysteme gedumpt werden müssen. Eine 1 bedeutet, daß das Dateisystem mit dump bearbeitet werden soll, eine 0 bedeutet, daß es nicht gedumpt werden muß. Ist das fünfte Feld nicht vorhanden, wird für diesen Wert Null angenommen und dump geht davon aus, daß das Dateisystem nicht gedumpt werden muß.

Das sechste Feld, (fs_passno), wird von fsck(8) benutzt um die Reihenfolge, in der die Dateisysteme während des Reboots geprüft werden, festzulegen. Das root Dateisystem sollte mit einer fs_passno von 1 versehen sein, andere Dateisysteme mit einer fs_passno von 2. Dateisysteme innerhalb eines Laufwerks werden sequentiell geprüft, Dateisysteme auf verschiedenen Laufwerken jedoch gleichzeitig, um parallel arbeitende Hardware zu unterstützen. Ist das sechste Feld nicht vorhanden oder Null, wird sinnigerweise eine Null zurückgegeben und fsck geht davon aus, daß das Dateisystem keiner Prüfung bedarf.

Normalerweise darf nur root Dateisysteme an- und abhängen. Das macht bei Festplattenpartitionen durchaus Sinn, ist aber für PC-Hardware mit Wechselmedien wie Disketten, CD-Laufwerke oder ZIP-Disks sehr unpraktisch. Wenn also gewünscht wird, daß ein Normaluser bestimmte Laufwerke auch mounten darf, so sollte bei den Optionen nach dem "defaults" noch ein "user" stehen. Damit dieses Laufwerk dann aber beim Start nicht automatisch gemountet

wird (und so eine Fehlermeldung provoziert, wenn z.B. keine Diskette eingelegt ist) sollte auch noch die Option "noauto" benutzt werden.

Ein typischer Eintrag in einem sehr einfachen Linux-System könnte also folgendermaßen aussehen:

```
/dev/hda2       /       ext2    defaults 1 1
/dev/hda3       swap    swap    defaults 0 2
/dev/hda5       /usr    ext2    defaults 1 2
/dev/hdb        /cdrom  auto    ro,noauto,user 0 0
/dev/fd0        /floppy auto    defaults,noauto,user 0 0
```

1.104.4 - Verwalten von Diskquotas

Beschreibung: Prüfungskandidaten sollten in der Lage sein, Diskquotas für Benutzer zu verwalten. Dieses Lernziel beinhaltet das Einrichten von Diskquotas für ein Dateisystem, das Bearbeiten, Prüfen und Erstellen von Berichten über Userquotas.

Die wichtigsten Dateien, Bezeichnungen und Anwendungen:

- **quota**
- **edquota**
- **repquota**
- **quotaon**

Das Erstellen von Diskquotas hatte in der ersten Version der LPI101 Prüfung nur eine Bewertung von 1, es ist also fragwürdig, ob überhaupt nur eine Frage zu diesem Thema in der Prüfung vorkommt. Trotzdem ist es ein offizielles Thema und wird daher hier behandelt. Abgesehen von der niedrigen Bewertung ist es aber doch so, daß es ein sehr praktischer Mechanismus ist, den man in der Systemverwaltung häufig brauchen kann.

Worum geht es? Quotas sind Mechanismen, die es erlauben, bestimmten Usern oder Gruppen einen eingeschränkten Platz auf einem bestimmten Dateisystem zu gewähren. Das heißt, es ist z.B. möglich, einem User fest vorzugeben, wieviel Platz er in seinem Homeverzeichnis nutzen darf. Damit kann verhindert werden, daß ein User übermäßig viel Platz in Anspruch nimmt und so den anderen Usern Platz wegnimmt. In der Praxis findet man diesen Mechanismus häufig bei Webservern, die bestimmten Usern eine eingeschränkte Menge Platz anbieten, um Webseiten darauf abzulegen.

Das Prinzip der Quotas läuft darauf hinaus, daß der Kernel bei jedem schreibenden Zugriff eines Users auf ein Dateisystem überprüft, ob der User noch Platz hat, oder ob er seine Quote schon erreicht hat. Um das zu gewährleisten, muß schon beim Mounten des Dateisystems festgelegt werden, daß dieses Dateisystem eine Quotierung des Platzes haben soll. Das wird in der Regel in der Datei /etc/fstab festgelegt, wo bei den Optionen eines Dateisystems die Begriffe **`usrquota`** bzw. **`grpquota`** angefügt werden. Userquotas sind Einschränkungen für einzelne User, Gruppenquotas entsprechend Einschränkungen für bestimmte Usergruppen.

Voraussetzungen

Die erste Voraussetzung zur Verwendung von Disk-Quotas ist die Verwendung eines Kernels, der quotas unterstützt. Das sollte heute standardmäßig jeder Kernel anbieten.

Als nächstes muß in /etc/fstab angegeben werden, welche Dateisysteme Disk-Quota benutzen sollen. Jedes Dateisystem, das dieses Feature anbieten soll muß hier bei den Mount-Optionen den Begriff `usrquota` für Userquotas und/oder `grpquota` für Gruppenquotas enthalten. Eine /etc/fstab-Datei könnte also dann folgendermaßen aussehen:

```
/dev/hda2       /       ext2    defaults,usrquota 1 1
/dev/hda3       swap    swap    defaults 0 2
/dev/hda5       /usr    ext2    defaults 1 2
/dev/hda6       /opt    ext2    defaults 1 2
/dev/hda7       /home   ext2    defaults,usrquota 1 2
```

Wir haben also sowohl für das Wurzeldateisystem (hier /dev/hda2), als auch für das Dateisystem, auf dem die Homeverzeichnisse liegen (/home auf /dev/hda7) Userquotas angegeben.

Anlegen der Quotafiles mit quotacheck

Die Angabe, welche User auf welchem Dateisystem wieviel Platz bzw. wieviel Inodes benutzen dürfen, steht auf der Wurzel des jeweiligen Dateisystems in den Dateien `quota.user` bzw. `quota.group`. Diese Dateien sind

Binärdateien, die zunächst einmal angelegt sein müssen. Dazu dient das Programm **quotacheck**. Um dieses Programm zu benutzen sind root-Privilegien nötig.

Das Programm quotacheck kann entweder für jedes Dateisystem einzeln aufgerufen werden, indem ihm die entsprechende Gerätedatei als Parameter mit angegeben wird, oder es wird mit dem Parameter -a aufgerufen und arbeitet so alle Dateisysteme ab, die in der Datei /etc/fstab eine Quotaangabe gesetzt haben.

Im einfachsten Fall schreiben wir also (als root) die Zeile

```
quotacheck -avug
```

was bedeutet, daß alle Dateisysteme bearbeitet werden, die Quotas unterstützen (-a), daß dort sowohl Userquotas (-u), als auch Gruppenquotas (-g) berücksichtigt werden und daß das Programm uns auch mitteilt, was es gerade tut (-v).

Dieser Befehl sollte immer dann angewandt werden, wenn ein neues Dateisystem mit quotas erstellt wurde oder wenn Dateisysteme nicht sauber heruntergefahren wurden, also typischerweise nach einem Systemabsturz, wenn auch fsck ausgeführt wird. Die meisten Distributionen bieten bereits fertige Startdateien an, die diese Aufgabe übernehmen.

Nach der Abarbeitung dieses Befehls existieren auf allen Dateisystemen, die die Mountoption `usrquota` gesetzt hatten die Datei `quota.user` und auf allen Dateisystemen, die die `grpquota`-Option aktiviert hatten die Datei `quota.group`. Diese Dateien enthalten binär codiert alle wichtigen Angaben über die festgelegten User- bzw. Gruppenquotas, insbesondere auch die Angaben, welche User wieviel Platz bzw. wieviele Dateien auf diesem Dateisystem im Augenblick in Anspruch nehmen.

Quotas definieren mit edquota

Nachdem die Quotadateien jetzt angelegt sind, müssen wir als nächstes die Beschränkungen definieren, die für den jeweiligen User bzw. die Gruppe gewünscht sind. Auch das darf natürlich nur root vornehmen. Für diese Aufgabe existiert das Programm **edquota**.

1.104.5 - Zugriffskontrolle auf Dateien mittels Zugriffsrechten

Beschreibung: Prüfungskandidaten sollten in der Lage sein, Dateizugriff mittels Zugriffsberechtigungen zu steuern. Dieses Lernziel beinhaltet Zugriffsrechte auf reguläre und spezielle Dateien sowie Verzeichnisse. Ebenfalls enthalten sind Zugriffsmodi wie suid, sgid und *sticky bit*, die Verwendung des Gruppenfeldes für die Vergabe von Zugriffsrechten an Arbeitsgruppen, das *immutable flag* und der voreingestellte Dateierstellungsmodus.

Die wichtigsten Dateien, Bezeichnungen und Anwendungen:

- **chmod**
- **umask**
- **chattr**

Die Zugriffsrechte unter Linux/Unix sind im Prinzip sehr einfach aufgebaut, jedoch aber auch sehr wirkungsvoll, wenn man weiß, wie sie im Einzelnen angewandt werden. Das Wissen um diese Rechte ist absolutes Grundwissen für alle, die mit Systemverwaltung unter Linux beschäftigt sind und sollte daher im Schlaf beherrscht werden.

Grundsätzlicher Aufbau von Zugriffsrechten

Jede Datei in einem Linux-System - also auch Gerätedateien, Sockets, Pipes, Verzeichnisse, usw. - besitzt in ihrer Inode eine Angabe über die Zugriffsrechte. Außerdem hat jede Datei genau einen Eigentümer und genau eine Gruppe, der sie zugehört. Die Zugriffsrechte beziehen sich immer auf eben diesen Eigentümer der Datei, auf Gruppenmitglieder der Gruppe, der die Datei angehört und auf den Rest der Welt.

Für diese drei Kategorien (Eigentümer, Gruppe, Rest) existiert jeweils eine Angabe, die beschreibt, ob die Datei für die jeweilige Kategorie lesbar (r), beschreibbar (w) und ausführbar (x) ist.

Der Befehl `ls -l` *Dateiname* zeigt uns für jede Datei eben diese Angaben. So bedeutet die Ausgabe:

```
-rw-r-----   1 hans     autoren         1519 Jul 29  2000 Testdatei
```

Die Datei **Testdatei** ist eine reguläre Datei (-). Sie gehört dem User **hans**, der sie lesen und verändern darf (**rw-**). Die Datei gehört zur Gruppe **autoren**. Mitglieder dieser Gruppe dürfen die Datei lesen (**r--**). Der Rest der Welt hat keinerlei Rechte auf diese Datei (**---**).

Das erste Zeichen der Ausgabe zeigt uns also, um was für eine Art Datei es sich handelt. Folgende Dateiarten sind unter Linux definiert:

Zeichen	Dateiart
-	Reguläre (normale) Datei
d	Verzeichnis (directory)
l	Symbolischer Link (symlink)
b	Blockorientierte Gerätedatei (block device)
c	Zeichenorientierte Gerätedatei (character device)
p	Feste Programmverbindung (named pipe)
s	Netzwerk Kommunikationsendpunkt (socket)

Das dem ersten Zeichen folgende Konstrukt ist die Beschreibung des Zugriffsmodus. Die ersten drei Zeichen beschreiben die Rechte des Eigentümers der Datei, die nächsten drei die Rechte eines Gruppenmitglieds der Gruppe, der auch die Datei zugehört und die letzten drei die Rechte aller anderen User. Ein `r` bedeutet Leserecht (read), ein `w` Schreibrecht (write) und ein `x` Ausführungsrecht (execute). Diese Rechte können numerisch dargestellt werden. Dazu werden die Rechte wie folgt bezeichnet:

Eigentümer			Gruppenmitglied			Rest der Welt		
r	w	x	r	w	x	r	w	x
4	2	1	4	2	1	4	2	1

Die Nummern werden für jede der drei Kategorien einzeln addiert. Ein Zugriffsrecht von `rw-r-----` wäre so also numerisch darstellbar als 640. Die 6 errechnet sich aus dem r (4) plus w (2) des Eigentümerrechtes, die 4 ist einfach das Leserecht (r) des Gruppenmitglieds und die 0 entspricht keinem gesetzten Recht.

Für reguläre Dateien sind diese Rechte einfach zu durchschauen. Das Leserecht bedeutet, daß der Inhalt der Datei gelesen werden darf. Das Schreibrecht bedeutet, daß der Inhalt der Datei verändert werden darf (und somit die Datei auch gelöscht werden darf) und das Ausführungsrecht bedeutet, daß die Datei ein Programm ist, das ausgeführt werden darf.

Etwas anders sieht es mit Verzeichnissen aus. Hier bedeutet das Leserecht, daß der Inhalt eines Verzeichnisses aufgelistet werden darf, das Schreibrecht, daß Dateien im Verzeichnis angelegt und gelöscht werden dürfen und das Ausführungsrecht, daß in das Verzeichnis gewechselt werden darf. Das hat einen Haken, zu dem wir später noch kommen werden. Hat nämlich ein User Schreibrecht auf ein Verzeichnis, so darf er darin auch Dateien löschen, auf die er selbst keinerlei Rechte besitzt. Das liegt an der Tatsache, daß ein Verzeichnis genau genommen nur eine Datei ist, die Dateinamen und die dazu passenden Inode-Nummern gespeichert hat. Eine Datei in einem Verzeichnis ist also letztlich nichts anderes als eine Zeile Text in einer Textdatei. Und wer Schreibrechte auf eine Textdatei hat, kann Zeilen daraus löschen!

Spezielle Rechte

Neben diesen sichtbaren Rechten existieren noch drei weitere, die man sich als weitere (führende) Nummer vorstellen kann. Dabei handelt es sich um das *Substitute UserID Bit* (4), das *Substitute GroupID Bit* (2) und das *Sticky Bit* (1).

Substitute UserID Bit (SUID)

Dieses Recht gilt ausschließlich für ausführbare Dateien. Hat eine ausführbare Datei dieses Recht gesetzt, so erscheint in der Darstellung durch das ls -l Kommando statt dem x beim Eigentümerrecht ein s. Jeder User, der dieses Programm ausführt, tut dies unter der effektiven UserID des Users, dem die Datei gehört.

So hat z.B. das Programm /usr/bin/passwd die Aufgabe, daß auch normale User damit ihr eigenes Passwort ändern können. Dieses Passwort wird aber gespeichert in einer Datei, die nur von root beschrieben werden darf. Das Programm /usr/bin/passwd hat als Eigentümer root und hat das Substitute UserID Bit gesetzt. Jeder User, der dieses Programm ausführt tut dies also unter der effektiven UserID von root und hat daher die Rechte von root während der Ausführung. Das ermöglicht es dem Programm, das veränderte Passwort zu speichern.

Das ist beim Passwort-Programm nicht weiter problematisch, eine richtige Sicherheitslücke entsteht, wenn eine Shell mit diesem Bit ausgestattet ist und als Eigentümer root hat. Dann könnte jeder User, der diese Shell ausführt, root-Rechte benutzen!

Substitute GroupID Bit (SGID)

Dieses Recht gilt einerseits für ausführbare Dateien und andererseits für Verzeichnisse. Hat ein ausführbares Programm dieses Recht gesetzt, so gilt der gleiche Mechanismus, wie beim Substitute UserID Bit, nur diesmal eben die Gruppenmitgliedschaft betreffend. Ein User, der dieses Programm ausführt, tut dies als Gruppenmitglied der Gruppe, der das Programm gehört, statt seiner eigentlichen Gruppenkennung. Er hat also die Rechte eines Gruppenmitglieds dieser Gruppe, auch wenn er selbst nicht Mitglied dieser Gruppe ist. Statt dem x beim Gruppenrecht stellt das ls -l Kommando hier ein s dar.

Hat ein Verzeichnis dieses Recht gesetzt, dann liegt der Fall etwas anders. Legt ein User, der Schreibrecht auf ein Verzeichnis hat, in diesem Verzeichnis eine Datei an, so erhält diese Datei normalerweise die Gruppenmitgliedschaft der primären Gruppe des Users, der sie eben angelegt hat. Das führt schnell zum Chaos, wenn z.B. mehrere User zusammen an einem Projekt arbeiten. Alle diese User sind Gruppenmitglieder einer bestimmten Gruppe, aber sie haben diese Gruppe nicht als primäre Gruppe. Das

heißt, es kann sein, daß die einzelnen User die Dateien der jeweils anderen Projektmitarbeiter nicht lesen können. Wenn dieses Verzeichnis aber der gemeinsamen Gruppe gehört und eben das Substitute GroupID Bit darauf gesetzt ist, dann werden Dateien in diesem Verzeichnis grundsätzlich unter der GruppenID abgespeichert, der das Verzeichnis gehört. Mit diesem Mechanismus sind Arbeitsverzeichnisse für bestimmte Projektgruppen realisierbar.

Sticky Bit

Das Sticky Bit hat nur eine Bedeutung für Verzeichnisse. Oben wurde schon das Problem erwähnt, daß ein User, der Schreibrecht auf ein Verzeichnis hat, innerhalb dieses Verzeichnisses alle Dateien löschen kann, auch wenn sie ihm nicht gehören und er keinerlei Rechte auf diese Dateien besitzt. Das wird durch das Sticky Bit verhindert.

Ein Verzeichnis in dem jeder User Schreibrecht haben muß, wie etwa das /tmp-Verzeichnis, sollte unbedingt dieses Bit gesetzt haben. Ansonsten kann ein böswilliger User dort die Dateien anderer User löschen.

Das gesetzte Sticky-Bit wird vom ls -l Kommando durch ein t statt des x in der Kategorie "Rechte des Restes der Welt" dargestellt.

Um diese speziellen Rechte auch numerisch darzustellen, wird vor den oben genannten "normalen" Rechten noch eine Oktalziffer angefügt, so daß sich das folgende Bild ergibt:

Sonderrechte			Eigentümer			Gruppenmitglied			Rest der Welt		
SUID	SGID	Sticky	r	w	x	r	w	x	r	w	x
4	2	1	4	2	1	4	2	1	4	2	1

Ein Recht von 4755 bedeutet also, daß das Substitute UserID Bit gesetzt ist (4), der Eigentümer Lese-, Schreib- und Ausführungsrechte (1+2+4=7) hat während sowohl Gruppenmitglieder, als auch der Rest der Welt nur Lese- und Ausführungsrecht (1+4=5) besitzen.

Ändern von Zugriffsrechten mit chmod

Um die oben beschriebenen Rechte vergeben bzw. ändern zu können existiert das Programm chmod (change mode). Es erlaubt, die Rechte von Dateien und Verzeichnisse zu verändern. Die prinzipielle Anwendung ist einfach:

`chmod [`*`Optionen`*`] `*`Modus Datei(en)`*

Die wichtigste Option ist dabei `-R` oder `--recursive`, mit dem ganze Unterverzeichnisse mit allen Dateien darin auf einmal bearbeitet werden können.

Als Modus kann entweder ein numerischer, oder ein symbolischer Modus angegeben werden. Der numerische Modus entspricht genau dem, was im letzten Abschnitt dieser Seite beschrieben wurde, er wird hier nicht nochmal erklärt. Nur ein kurzes Beispiel noch, der Aufruf

```
chmod 644 foo.txt
```

würde der Datei `foo.txt` ein Zugriffsrecht von `rw-r--r--` geben. Mit numerischem Modus ist grundsätzlich immer ein absoluter Wert gesetzt, egal, welche Rechte vorher gesetzt waren.

Der symbolische Modus besteht aus einer Angabe der Rechte durch Buchstaben. Die Grundsätzliche Form ist:

```
[ugoa] +|-|= [rwxstugo], ...
```

Das führende Zeichen steht für die zu verändernde Kategorie, u steht für User (Eigentümer), g für group (Gruppenmitglied), o für other (Andere) und das a steht für all (Alle). Wird dieses führende Zeichen weggelassen, dann wird standardmäßig a (Alle) angenommen.

Vorsicht, oft wird diese Angabe verwechselt und das o mit dem Begriff Owner (Eigentümer) verwechselt.

Das kann fatale Folgen haben, weil man dann die Rechte, die man eigentlich dem Eigentümer geben will, dem Rest der Welt verleiht!

Das folgende Rechenzeichen +, - oder = beschreibt, ob ein Recht den bestehenden Rechten hinzugefügt (+) werden soll, davon abgezogen (-) werden soll oder absolut (=) gesetzt werden soll.

Dem Rechenzeichen folgt die Angabe der zu setzenden (oder addierenden oder subtrahierenden) Rechte in der Form einer Zeichenkette, die aus den Buchstaben r,w,x,s,t,u,g,o besteht. Dabei meinen r, w und x wie üblich Read, Write und Execute. Wird dem User das s-Recht gesetzt, so meint das das Substitute UserID Bit (beispielsweise durch `u+s`), bekommt hingegen die Gruppe dieses Recht, so ist das Subtitute GroupID Bit gemeint (`g+s`). Das t steht für das Sticky-Bit und kann nur dem Rest der Welt vergeben werden (`o+t`).

Folgen dieser Angabe noch ein u, g oder o, so ist dies ein Ausschlußkriterium in Verbindung mit dem a am Anfang. Das nachgestellte u, g oder a schützt also die Rechte der User, Gruppenmitglieder bzw. Anderen vor Veränderung.

Das ganze kann jetzt mehrmals hintereinander angewandt werden, indem mehrere solcher Modi durch Kommas getrennt angegeben werden. So ist es etwa möglich zu schreiben:

```
chmod u=rwx,g=rx,o-rwx foo
```

um dem Programm foo das Recht `rwxr-x---` zu setzen. Gewöhnlich tut man sich aber in diesem Fall leichter mit einer numerischen Angabe (hier 750).

Das praktische an der symbolischen Angabe von Modi ist die Fähigkeit, bestimmte Rechte auf die bestehenden Rechte aufzuaddieren bzw. sie von den bestehenden Rechten abzuziehen. Ein einfaches `+x` bedeutet z.B., daß allen drei Kategorien User, Group und Other ein Ausführungsrecht zu den schon vorhandenen Rechten gegeben wird.

Wird als Option ein -R oder ein --recursive angegeben, so verändert chmod die Rechte eines ganzen Verzeichnisbaums, inclusive aller enthaltenen Dateien und Unterverzeichnisse.

Voreingestellten Zugriffsmodus mit umask bestimmen

Es bleibt jetzt natürlich die Frage, welche Zugriffsberechtigungen beim Anlegen einer Datei verwendet werden. Um das festzulegen, kennt Unix das Kommando umask, dessen Anwendung aber etwas merkwürdig ist.

Der Befehl umask erwartet eine Maske als Parameter, die sich auf den Zugriffsmodus bezieht, den neu zu erstellende Dateien bekommen sollen. Das Wort Maske bedeutet, daß nicht die Werte eingegeben werden, die gesetzt werden sollen, sondern umgekehrt, die Werte, die nicht gesetzt (maskiert) werden sollen. Die einfachste Möglichkeit besteht darin, die gewünschten oktalen Werte jeweils von 7 abzuziehen:

Wollen wir z.B. dafür sorgen, daß alle unsere Dateien die Zugriffsberechtigung `rw-r-----` bekommen, also Lese- und Schreibrecht für den Eigentümer, Leserecht für Gruppenmitglieder und keine Rechte für den Rest der Welt, dann entspräche das der oktalen Darstellung 640.

Die dafür notwendige umask wäre dann:

```
7-6=1
7-4=3
7-0=7
```

Mit dem Befehl

```
umask 137
```

würden also alle Dateien, die wir anlegen die gewünschte Zugriffsberechtigung 640 bekommen. Das hat noch einen kleinen Haken, weil die Verzeichnisse, die wir erstellen würden auch diesen Modus bekämen. Damit wäre für uns selbst das Durchsuchungsrecht (x) nicht gesetzt. Linux hat aus diesem Grund dafür den Mechanismus entwickelt, daß selbst wenn im umask-Kommando das x-Recht gesetzt ist, beim Erzeugen von normalen Dateien dieses Recht

nicht gesetzt wird, beim Erzeugen von Verzeichnissen hingegen schon. Ein vernünftiges umask-Kommando setzt also zumindestens für den Eigentümer auch das x-Recht. Damit wäre ein typischer Wert für umask z.B. 022 (`rwxr-xr-x`) oder 027 (`rwxr-x---`).

Neben dieser umständlichen Methode gibt es aber auch die symbolische Form, die die Rechte direkt bezeichnet. Sie wird in der Form `u=...,g=...,o=...` eingegeben, wobei für ... immer die entsprechenden Rechte eingesetzt werden. Also würde der Befehl

```
umask u=rwx,g=rx,o=
```

die gleiche Wirkung haben wie

```
umask 027
```

Typischerweise steht eine umask-Anweisung in einer der Shell-Startdateien wie z.B. `/etc/profiles` oder `~/.profile`. Jeder User kann seine Voreinstellung also selbst einstellen, eine der wenigen Möglichkeiten, mit denen er dem Systemverwalter ins Handwerk pfuschen kann, wenn der versucht, sein System sicher zu machen. Allerdings bezieht sich diese Einstellung ja nur auf die neu anzulegenden Dateien des jeweiligen Users...

Erweiterte Dateiattribute im EXT2-Dateisystem

Dateien auf EXT2-Dateisystemen besitzen neben den oben genannten Attributen noch weitere, die einen zusätzlichen Schutz bieten können. Diese Attribute können mit dem Befehl lsattr angezeigt und mit chattr verändert (gesetzt) werden.

Die wichtigsten dieser Attribute sind:

a (append)
: Eine Datei mit a-Attribut kann schreibend nur im Anhängen-Modus geöffnet werden. Das bedeutet, daß ein User, der Schreibrecht auf diese Datei hat, zwar Daten an das Ende der Datei anhängen kann (etwa durch die Verwendung der >>-Umleitung), jedoch keine Veränderungen am bestehenden Inhalt der Datei vornehmen darf. Nur der Superuser kann dieses Attribut setzen oder entfernen.

i (immutable)
: Eine Datei mit gesetztem i-Attribut kann nicht modifiziert werden. Sie kann weder gelöscht, noch umbenannt werden, es kann kein Hardlink auf sie angelegt werden und keine Daten können angehängt werden. Nur der Superuser kann dieses Attribut setzen oder entfernen.

s (save-delete)
: Wenn eine Datei das s-Attribut gesetzt hat, werden alle Blöcke dieser Datei beim Löschen der Datei mit Nullzeichen überschrieben. Dadurch ist die Datei auch durch Methoden wie dem Dateisystemdebuger nicht wieder herstellbar oder ihr Inhalt ist nicht mehr einsehbar, wenn sie einmal gelöscht wurde.

Eine Liste aller möglichen Attribute entnehmen Sie der Handbuchseite von chattr.

1.104.6 - Verwaltung von Dateieigentum

Beschreibung: Prüfungskandidaten sollten in der Lage sein, den Benutzer- und Gruppenbesitz von Dateien zu steuern. Dieses Lernziel beinhaltet das Ändern des Besitzers und der Gruppenzugehörigkeit einer Datei sowie des Standardeigentümers von neuen Dateien.

Die wichtigsten Dateien, Bezeichnungen und Anwendungen:

- **chmod**
- **chown**
- **chgrp**

Im letzten Abschnitt haben wir gesehen, wie Zugriffsrechte gesetzt bzw. verändert werden. All diese Zugriffsrechte beziehen sich ja auf Eigentümer bzw. Gruppen von Usern. Von daher benötigen wir noch die Technik, Eigentümer und Gruppenzugehörigkeit einer Datei zu verändern. Dazu gibt es die Programme chown und chgrp. Diese Programme sollen hier vorgestellt werden.

Ändern des Dateieigentümers mit chown

Um den Eigentümer einer Datei zu wechseln bzw. besser ausgedrückt, eine Datei einem anderen User zu übereignen, existiert das Programm chown. Dieses Programm darf nur vom Systemverwalter ausgeführt werden, ein normaler User kann also nicht seine Dateien anderen Usern übereignen.

Das Programm chown hat eine einfache Aufrufform:

```
chown [Optionen] Username Datei(en)
chown [Optionen] [Username][.][Gruppenname] Datei(en)
chown [Optionen] --reference=Referenzdatei Datei(en)
```

Eine oder mehrere Dateien bekommen so als Eigentümer den genannten Usernamen zugewiesen. Der Username kann hier entweder als Name oder als Nummer (UserID - UID) angegeben werden.

Wenn nur ein Username oder eine UserID angegeben wurde, so wird dieser User zum Eigentümer jeder angegebenen Datei und die Gruppenmitgliedschaft der Dateien wird nicht verändert. Wenn dem Username ein Doppelpunkt oder Punkt und ein Gruppenname (oder eine GruppenID) ohne Leerzeichen folgt, dann wird auch die Gruppenzugehörigkeit der Dateien geändert. Wenn dem Usernamen ein Punkt oder Doppelpunkt folgt, aber kein Gruppenname angegeben wird, so wird der angegebene User zum Eigentümer der angegebenen Dateien und die Dateien bekommen die Gruppenzugehörigkeit zu der Gruppe, die die Login-Gruppe dieses Users ist (die Gruppe, die im Gruppenfeld in /etc/passwd für diesen User angegeben ist). Wenn aber der Username weggelassen wurd, aber ein Punkt oder Doppelpunkt, gefolgt von einem Gruppennamen angegeben wurde, so wird nur die Gruppenzugehörigkeit der Dateien verändert, der Eigentümer bleibt unverändert. In diesem Fall arbeitet chown genau wie chgrp.

Wenn statt einem Usernamen und/oder Gruppennamen die Option `--reference=`*`Referenzdatei`* angegeben wurde, so werden Eigentümer und Gruppenzugehörigkeit der angegebenen Dateien so gesetzt, wie die der Referenzdatei.

Wird als Option ein `-R` oder ein `--recursive` angegeben, so verändert chown die Eigentümer eines ganzen Verzeichnisbaums, inclusive aller enthaltenen Dateien und Unterverzeichnisse.

Ändern der Gruppenzugehörigkeit einer Datei mit chgrp

Die Gruppenzugehörigkeit einer Datei kann auch mit dem Befehl **chgrp** gewechselt werden. Die Aufrufform ist ähnlich der von chown:

```
chgrp [Optionen] Gruppe Datei(en)
```

Auch hier kann die Gruppe wieder entweder als Gruppenname oder als GruppenID angegeben werden. Die Benutzung von chgrp ist nur dem Eigentümer und dem Superuser (root) erlaubt. Der Eigentümer kann eine Datei nur den Gruppen zuordnen, denen er selbst auch angehört.

Wie schon bei chown und chmod kann auch chgrp mit der Option `-R` oder `--recursive` ein ganzer Verzeichnsast rekursiv bearbeitet werden, d.h., daß alle Verzeichnisse und enthaltene Dateien bearbeitet werden.

Sicherstellen der Gruppenzugehörigkeit von Dateien in Verzeichnissen

Wenn ein Verzeichnis für eine bestimmte Gruppe von Usern beschreibbar ist und diese User darin eine gemeinsame Arbeit leisten sollen, so wäre es ja praktisch, wenn alle User innerhalb dieses Verzeichnisses alle Dateien, die sie anlegen, eben der Gruppe zuweisen, der das Verzeichnis angehört. Das ist aber nicht automatisch der Fall.

Jeder User kann beliebig vielen Gruppen angehören. er ist aber immer Mitglied einer sogenannten Login-Gruppe (manchmal auch Initialgruppe genannt). Wenn ein User eine Datei anlegt, so wird diese Datei normalerweise die Gruppenzugehörigkeit der Login-Gruppe dieses Users zugewiesen bekommen.

Um sicherzustellen, daß alle Dateien in einem bestimmten Verzeichnis beim Anlegen immer die Gruppenzugehörigkeit zu der Gruppe bekommen, der das Verzeichnis gehört, muß das Verzeichnis das SGID-Bit gesetzt bekommen. Das wird mit dem Befehl chmod eingestellt, wie auf der letzten Seite beschrieben.

Wenn ein User allerdings nur vorübergehend die Standard-Gruppe wechseln will, so steht im dazu das Kommando newgrp zur Verfügung. Das würde allerdings das Mitdenken aller User voraussetzen, was immer eine Schwachstelle bedeutet...

Wenn also der Systemverwalter ein Verzeichnis für eine bestimmte Projektarbeit anlegt, tut er gut daran, diesem Verzeichnis das SGID-Bit zu setzen, um die Gruppenmitgliedschaft der Dateien in diesem Verzeichnis von vorneherein festzulegen.

1.104.7 - Erzeugen und Ändern von harten und symbolischen Links

Beschreibung: Prüfungskandidaten sollten in der Lage sein, harte und symbolische *Links* auf eine Datei zu verwalten. Dieses Lernziel beinhaltet das Erzeugen und Bestimmen von Links, das Kopieren von Dateien über Links und das Verwenden von verknüpften Dateien zur Unterstützung von Tätigkeiten der Systemadministration.

Die wichtigsten Dateien, Bezeichnungen und Anwendungen:

- **ln**

Links sind eine Spezialität von Unix-Dateisystemen, die gerne und häufig verwendet werden. Der Umgang mit ihnen ist eine wichtige und manchmal etwas verkopfte Angelegenheit, die ein Systemverwalter sicher beherrschen muß.

Unix und Linux unterscheiden zwischen Hardlinks (direkte Links) und Symlinks (symbolische Links). Die Eigenschaften dieser beiden Typen von Links sowie die Techniken, um sie anzulegen bzw. mit ihnen umzugehen sind Inhalt dieses Kapitels.

Hardlinks

In einem Unix-Dateisystem ist ein Dateiname nur ein Verzeichniseintrag, der in einem Verzeichnis gespeichert ist. Neben diesem Namen ist immer auch ein Verweis auf die Inode gespeichert, die dann die eigentlichen Eigenschaften (Eigentümer, Gruppenmitgliedschaft, Zugriffsrechte, usw.) der Datei und ihre physikalischen Speicheradressen auf der Platte enthält. Dieses Prinzip wurde auf den Seiten über Inodes und über das EXT2 Dateisystem bereits umfassend dargestellt.

Ein Hardlink ist nichts anderes, als ein neuer Verzeichniseintrag auf eine schon bestehende Inode. Also genau genommen ein zweiter Dateiname für eine Datei. Weil die Zugriffsrechte, Eigentümer usw. ja in der Inode stehen, haben alle Hardlinks einer Datei die selben solchen Attribute.

Wenn ein oder mehrere Hardlinks auf eine Datei zeigen, so ist nicht mehr zu unterscheiden, welcher davon das Original und welche die Links sind. Es handelt sich ja einfach nur um Namenseinträge, die auf die selbe Inode zeigen. Das heißt auch, daß Hardlinks immer noch gültig sind, wenn die Datei, auf die sie zeigen gelöscht wurde. Es wurde ja eben nicht die Datei gelöscht, sondern nur einer ihrer Namen. Solange noch weitere Namen existieren, wird die Datei nicht physikalisch gelöscht.

Die Ausgabe des `ls -l` Kommandos zeigt für jede Datei gleich nach dem Zugriffsmodus die Anzahl der Hardlinks (also der Namenseinträge), die diese Datei besitzt.

```
-rw-r--r--   1 root     root         4326 Apr  8 15:02 datei1.txt
-rw-r--r--   5 root     root         1578 Apr  8 15:02 datei2.txt
```

Auch hier wird nicht zwischen Original und Link unterschieden (geht ja eben auch gar nicht) so daß eine Datei mit nur einem Namen hier eine 1 anzeigt. Die zweite Datei des obigen Beispiels hat also 5 Namen, das könnte bedeuten, daß die Datei erstellt wurde und anschließend vier Hardlinks auf sie erstellt wurden. Insgesamt existieren also 5 Namenseinträge für diese Datei.

Da die Hardlinks sozusagen innerhalb der Mechanismen eines Dateisystems arbeiten indem sie eigentlich nur Verweise auf schon bestehende Inodes sind, arbeiten diese Links nur innerhalb der Grenzen eines Dateisystems also einer Partition. Es ist also nicht möglich, Hardlinks auf einer zweiten Partition zu erstellen, die auf eine Datei zeigen, die auf der ersten Partition liegt!

Eine weitere Einschränkung von Hardlinks ist, daß es nicht möglich ist, Hardlinks auf Verzeichnisse zu legen. Es existiert zwar die Option -d (oder -F oder --directory) des Befehls ln, die die Fähigkeit erlauben soll, auf allen Linux-Dateisystemen ist dieses Feature aber verboten.

Symbolische Links

Die genannten Einschränkungen der Hardlinks (keine Links auf Verzeichnisse/ keine Links über die Dateisystemgrenze hinaus) können mit sogenannten *symbolischen Links* umgangen werden. Symbolische Links arbeiten nicht auf der Dateisystemebene, sondern sind einfach Dateien, die nichts anderes enthalten, als den Pfad zu der Datei (oder dem Verzeichnis), auf die sie zeigen. Damit sie als Links zu erkennen sind, haben sie einen eigenen Dateityp (`l`), der es dem Betriebssystem klar macht, daß es sich hier um einen Link und nicht um eine reguläre Datei handelt.

Das `ls -l` Kommando zeigt einen symbolischen Link also als solchen an:

```
-rw-r--r--   1 root     root       276295 Apr 21 19:46 Datei1
lrwxrwxrwx   1 root     root            6 Apr 21 19:46 Datei2 -> Datei1
```

Sowohl an der Angabe des Dateityps (`l`, als auch am Dateinamen, dem ein Pfeilsymbol und das Ziel des Links folgt, ist ersichtlich, daß es sich hier um einen symbolischen Link handelt. Genauso ist ersichtlich, worauf der Link zeigt. Beim Hardlink konnten wir Original und Link nicht unterscheiden, beim symbolischen Link sind sie eindeutig unterscheidbar.

Weil symbolische Links nicht auf der Ebene der Dateisysteme selbst arbeiten reagieren sie aber auch in anderen Beziehungen völlig anders, als die Hardlinks:

- Wenn die Datei (oder das Verzeichnis), auf die ein symbolischer Link zeigt nicht mehr existiert (z.B. gelöscht wurde), dann zeigt der symbolische Link "ins Leere", der Link existiert zwar weiter, er funktioniert aber nicht mehr.
- Wird ein symbolischer Link mit einer relativen Pfadangabe erstellt und anschließend in ein anderes Verzeichnis kopiert, dann wird er womöglich nicht mehr weiter funktionieren, weil vom neuen Verzeichnis aus dieser Pfad nicht existiert.

Andererseits sind die symbolischen Links nicht eingeschränkt, was die Grenzen eines Dateisystems angeht oder die Verwendung für Verzeichnisse.

Das Programm ln

Um Links anzulegen, existiert das Programm ln. Die Aufrufform ist einfach,

```
ln [-s] Datei [Link]
```

Wird das Programm `ln` ohne den Parameter `-s` oder `--symbolic` aufgerufen, so wird ein Hardlink erstellt, mit einem dieser beiden Optionsschaltern wird ein symbolischer Link erstellt.

Wenn ein Hardlink erstellt wird, so muß die Datei, auf die der Link verweist existieren, wenn ein symbolischer Link erstellt wird, so gilt das nicht.

Wird beim Aufruf von ln der Linkname weggelassen, so wird ein Link mit gleichem Namen wie die Datei im aktuellen Verzeichnis erzeugt. Das setzt aber natürlich voraus, daß die Datei nicht im aktuellen Verzeichnis liegt.

Werden mehr als zwei Dateinamen angegeben (Datei und Link), so muß der letzte Parameter ein Verzeichnisname sein. In diesem Verzeichnis werden dann Links auf all die Dateien angelegt, die vor diesem letzten Parameter angegeben wurden.

Normalerweise überschreibt das Programm ln keine Dateien, wenn ein Link angelegt werden soll, dessen Namen schon existiert. ln bietet aber eine Fülle von Optionen, die diese Eigenschaft ändert, inclusive der Frage der automatischen Umbenennung von überschriebenen Dateien.

Identifikation von Hardlinks

Wie oben schon erwähnt, gibt es keine Möglichkeit, zwischen Hardlink und Original zu unterscheiden, weil es ja kein Original gibt, sondern nur mehrere Namen, die alle auf die gleiche Inode verweisen. Wenn wir nun herausfinden wollen, welche Dateinamen alle die selbe Inode verwenden, gibt es da durchaus eine Möglichkeit:

Zunächst einmal müssen wir herausbekommen, welche Inode von unserer Datei überhaupt belegt wird. Das `ls`-Kommando bietet uns mit der Option `-i` die Möglichkeit, das zu erfahren. Ein `ls -i` gibt uns zu den Dateinamen die verwendeten Inode-Nummern mit aus.

Dann können wir mit dem Programm `find` nach allen Dateien suchen, die diese Inode-Nummer benutzen. Allerdings sollten wir das ausschließlich innerhalb der Partition tun, auf der die Datei liegt, die wir untersuchen. Denn zufälligerweise kann ja eine ganz andere Datei auf einem ganz anderen Dateisystem (also auf einer anderen Partition) die selbe Inode-Nummer benutzen.

Spielen wir es einmal durch: Wir befinden uns im Verzeichnis `/usr/local/data` und das `ls -l` Kommando zeigt uns eine Datei mit Namen `BEISPIEL.DAT` folgendermaßen an:

```
-rw-r--r--   5 root     root         1895 Apr  8 15:02 BEISPIEL.DAT
```

Aus der Angabe direkt nach dem Zugriffsmodus entnehmen wir, daß es insgesamt fünf Dateinamen gibt, die auf ein und dieselbe Inode verweisen. Also neben dieser Datei noch vier weitere. Zunächst müssen wir wissen, welche Inode diese Datei überhaupt benutzt. Dazu geben wir den Befehl

```
ls -i BEISPIEL.DAT
```

ein und bekommen die folgende Ausgabe:

```
92550 BEISPIEL.DAT
```

Die benutzte Inode ist also die Inode Nummer 92550. Jetzt müssen wir herausbekommen, auf welcher Partition wir uns eigentlich befinden und wo sie im Verzeichnisbaum eingehängt ist. Wir geben den Befehl `df` ohne weitere Parameter ein. Die Ausgabe lautet:

```
/dev/hda2              2071328   1047620    918484  53% /
/dev/hda5              3099108   1737192   1204484  59% /usr
/dev/hda6              2071296    767708   1198364  39% /opt
/dev/hda7              2071296    215212   1750860  11% /home
```

Nachdem wir uns im Verzeichnis `/usr/local/data` befinden, können wir also aus dieser Ausgabe schließen, daß wir uns auf der Partition `/dev/hda5` befinden und daß diese Partition ins Verzeichnis `/usr` gemountet ist. Jetzt rufen wir den `find`-Befehl auf und weisen ihn an, ab dem Verzeichnis `/usr` alle Dateien zu suchen, die die Inode 92550 benutzen. Zusätzlich weisen wir ihn noch darauf hin, daß er nur dieses eine Dateisystem durchsuchen soll. Dazu kennt find die Option `-xdev`. Wir geben folgenden Befehl ein:

```
find /usr -xdev -inum 92550 -print
```

Der Befehl `find` sucht ab dem Verzeichnis `/usr` aber nur innerhalb der Partition (`-xdev`) alle Dateien, die die Inode 92550 (`-inum 92550`) besitzen. Die gefundenen Dateien werden ausgegeben (`-print`). Die Anweisung `-print` hätten wir unter Linux auch weglassen dürfen, es ist hier die voreingestellte Aktion. Das Ergebnis sieht dann etwa wie folgt aus:

```
/usr/lib/Beispiel/Datei.txt
/usr/local/data/BEISPIEL.DAT
/usr/local/lib/Beispiel/Noch_eine
/usr/openwin/Hier_auch
/usr/src/abc
```

Hätten wir dem `find`-Befehl statt der Aktion `-print` die Aktion `-ls` mitgegeben, die alle gefundenen Dateien im selben Format wie `ls -dils` ausgibt, dann hätten wir die folgende Ausgabe bekommen:

```
 92550    5 -rw-r--r--   1 root     root         1895 Apr  8 15:02 /usr/lib/Beispiel/
Datei.txt
 92550    5 -rw-r--r--   1 root     root         1895 Apr  8 15:02 /usr/local/data/
BEISPIEL.DAT
 92550    5 -rw-r--r--   1 root     root         1895 Apr  8 15:02 /usr/local/lib/
Beispiel/Noch_eine
 92550    5 -rw-r--r--   1 root     root         1895 Apr  8 15:02 /usr/openwin/
Hier_auch
 92550    5 -rw-r--r--   1 root     root         1895 Apr  8 15:02 /usr/src/abc
```

Eine weitere Möglichkeit der Identifikation von Hardlinks wäre es, ein `ls -ilR` Kommando (langes Listing mit Inode-Nummern, rekursiv) auszuführen und dessen Ausgabe an `grep` weiterzuleiten. Grep würde dann nach der entsprechenden Inode-Nummer am Zeilenanfang suchen. Also für unser obiges Beispiel etwas in der Art:

```
ls -ilR /usr | grep "^ *92550"
```

Das hat allerdings den Nachteil, daß wir nicht festlegen können, daß nur die Dateien innerhalb einer Partition gesucht werden. Zum anderen ist diese Art der Suche wesentlich langsamer als die mit dem `find`-Befehl.

Kopieren symbolischer Links

Wenn ein symbolischer Link mit dem Befehl `cp` kopiert wird, ohne daß dazu bestimmte Optionen gegeben werden, so wird nicht etwa der Link kopiert, sondern die Datei, auf die der Link zeigt. Das Ergebnis (die Zieldatei) ist jetzt also kein Link, sondern eine reguläre Datei. Man spricht in diesem Zusammenhang von der Dereferenzierung eines Links.

Dereferenzierung bedeutet, daß der Link zurückverfolgt wird, und durch das ausgetaucht wird, auf das er zeigt.

Um das zu vermeiden, muß dem `cp`-Befehl eine spezielle Option mitgegeben werden, die diese Dereferenzierung unterbindet. Wenn wir also einen symbolischen Link als solchen kopieren wollen, das heißt wenn das Ziel der Kopieraktion wiederum ein Link sein soll, so müssen wir dem Kopierbefehl die Option `-d` oder `--no-dereference` mitgeben.

Aber bitte Vorsicht walten lassen! Wenn ein symbolischer Link als Link kopiert wird, dann wird der Link genauso kopiert, wie er war. Falls er keine absolute, sondern eine relative Pfadangabe zu dem Objekt beinhaltet, auf das er verweist, so wird die Kopie genau die Selbe Pfadangabe haben. Es ist nicht gewährleistet, daß diese Angabe von der neuen Lokalität aus immer noch stimmt.

Am Rande bemerkt: Der `cp`-Befehl ist selbst auch in der Lage, Links zu erzeugen statt Dateien zu kopieren. Mit der Option `-l` erzeugt er Hardlinks statt Kopien und mit `-s` symbolische Links.

Verwenden von Links zur Systemadministration

In der Systemverwaltung werden Links auf vielfältige Weise eingesetzt. Dabei kommen sowohl Hard- als auch symbolische Links zur Anwendung. Ein paar Beispiele:

Vermeidung versehentlichen Löschens mit Hardlinks
: Wichtige Systemdateien können an andere Orte gelinkt werden, wo sie sozusagen eine Versicherung gegen versehentliches Löschen bieten. Nehmen wir an, Sie erstellen ein Verzeichnis `/etc2` und verlinken alle wichtigen Dateien in `/etc` mit Hardlinks in dieses Verzeichnis. Sollte jetzt eine Datei in `/etc` gelöscht werden, so steht sie uns immer noch in der aktuellsten Version in `/etc2` zur Verfügung.

Zentrale Verwaltung wichtiger Startdateien in den Userverzeichnissen
: Jeder User hat in seinem Home-Verzeichnis verschiedene Startdateien oder andere Konfigurationsdateien. Soll sichergestellt werden, daß alle User immer die gleichen Einstellungen besitzen, so könnten all diese Dateien Hardlinks auf einen Prototyp sein. Wenn der Systemverwalter jetzt für alle User eine Veränderung vornehmen will, so muß er nur eine dieser Dateien (etwa den Prototyp) verändern und alle Dateien der User sind mitverändert.

Symbolische Links auf Verzeichnisse
: Wenn bestimmte Teile des Verzeichnisbaums ReadOnly gemountet sein sollen, aber andererseits

Unterverzeichnisse dieser Teile beschreibbar sein müssen, so können diese Unterverzeichnisse symbolische Links auf andere Unterverzeichnisse sein, die nicht auf ReadOnly-gemounteten Partitionen liegen. Das `/usr` Verzeichnis wird z.B. gerne ReadOnly gemountet, enthält aber Verzeichnisse wie `tmp` oder `spool`, die beschreibbar sein müssen. Diese Verzeichnisse sind oft symbolische Links auf entsprechende Verzeichnisse unter `/var`. Dort sind diese Verzeichnisse tatsächlich beschreibbar.

1.104.8 - Auffinden von Systemdateien und Platzieren von Dateien an den korrekten Ort

Beschreibung: Prüfungskandidaten sollten ausreichend vertraut mit dem *Filesystem Hierarchy Standard*, einschließlich typischer Speicherort von Dateien und der Einteilung der Verzeichnisse, sein. Dieses Lernziel beinhaltet das Auffinden von Dateien und Kommandos auf einem Linux-System.

Die wichtigsten Dateien, Bezeichnungen und Anwendungen:

- **find**
- **locate**
- **slocate**
- **updatedb**
- **whereis**
- **which**
- `/etc/updatedb.conf`

Die Standard-Hierarchie der Unix-Dateisysteme war und ist schon immer einer der großen Vorteile gegenüber anderen Systemen gewesen. Hat man einmal ein Unix-System egal welcher Art kennengelernt, so findet man sich nahezu auf jedem anderen Unix-System mühelos zurecht. Linux bietet einen solchen Standard genauso an und es gibt weltweite Bestrebungen, die Positionen der Dateien und Verzeichnisse innerhalb des Dateisystems, sowie deren Namen festzulegen. Trotzdem finden sich immer Unterschiede zwischen verschiedenen Unixen und selbst zwischen verschiedenen Linux-Distributionen. Um mit diesen Unterschieden umgehen zu können finden sich verschiedene Werkzeuge, die hier näher besprochen werden sollen.

Die Standard-Hierarchie des Dateisystems

Um zu gewährleisten, daß die verschiedenen Linux-Distributionen eine einheitliche Dateisystem-Hierarchie benutzen, gibt es den Versuch, einen Standard für diese Hierarchie zu erarbeiten, den *Filesystem Hierarchy Standard*, der unter www.pathname.com/fhs/ genau nachzulesen ist.

Die wesentlichen Aufgaben dieses Standards sind die Festlegungen, welche Dateien in welche Verzeichnisse gehören und zwar dergestalt, daß es möglich ist, bestimmte Verzeichnisse mit statischen Daten als ReadOnly-Dateisystem einzuhängen, um eine größtmögliche Stabilität des Systems zu gewährleisten.

Im folgenden sollen die wesentlichen Verzeichnisse kurz beschrieben werden, um ihre Aufgaben bzw. die Philosophie, die hinter ihnen steckt zu klären:

Das Wurzeldateisystem /

Dieses Dateisystem enthält die Wurzel des gesammten Systems. In einem professionellen System wird versucht, dieses Dateisystem möglichst klein zu halten. Es soll die Verzeichnisse enthalten, die notwendig sind, um ein System auch nach einem Ausfall anderer Partitionen noch hochfahren und reparieren zu können.

Weil dieses Wurzeldateisystem selbst sehr viele systemspezifische Dateien enthält, ist es nicht geeignet, über ein Netz von mehreren Rechnern verwendet zu werden.

Viele der Unterverzeichnisse dieses Dateisystems sind nur Mountpoints, um andere Partitionen dort einzuhängen. Einige Verzeichnisse sollten aber unter keinen Umständen auf anderen Plattenpartitionen liegen, da sie sonst nicht zur Verfügung stehen, wenn während des Bootvorgangs bisher nur das Wurzelverzeichnis gemountet ist. Ein simples Beispiel ist das Programm *mount* selbst. Dieses Programm muß natürlich irgendwo auf der Partition liegen, die das Wurzelverzeichnis enthält, sonst steht es in dem Moment nicht zur Verfügung, wo andere Platten eingehängt werden sollen!

In der folgenden Darstellung wird immer auch beschrieben, ob ein Verzeichnis geeignet ist, auf eine andere Partition ausgelagert zu werden, oder ob es zwingend auf der Wurzelpartition sein muß.

Ein weiterer Gesichtspunkt ist die Frage, ob ein Verzeichnis statische oder dynamische Dateien enthält. Wenn in einem Verzeichnis ausschließlich statische Dateien liegen - also Dateien, die im laufenden Betrieb nicht verändert werden müssen - so bietet es sich an, dieses Verzeichnis ReadOnly zu mounten. Das Wurzeldateisystem enthält zwingend sowohl statische, als auch dynamische Dateien, muß also ReadWrite gemountet sein.

/bin - Grundlegende Programmdateien

Das Verzeichnis /bin enthält grundlegende Programmdateien, die während des Bootvorgangs und zur Reparatur des Systems zwingend notwendig sind. Es muß auf der Wurzelpartition positioniert sein.

/boot - Statische Dateien für den Boot-Loader

Das Verzeichnis /boot enthält die statischen Dateien, die für den Bootloader notwendig sind, um das System überhaupt zu booten. Es ist durchaus möglich, dieses Verzeichnis auf eine eigene kleine Partition zu positionieren. Bei Platten mit mehr als 1024 Zylindern und einem älteren Bootloader ist es sogar notwendig, dieses Verzeichnis auf eine Partition unterhalb des 1024ten Zylinders zu legen. Es macht in der Regel keinen Sinn, diese Partition übers Netz zu teilen.

/dev - Gerätedateien

Dieses Verzeichnis enthält die Gerätedateien, die notwendig sind, um auf jede Art von Hardware zugreifen zu können. Es muß zwingend auf der Wurzelpartition angelegt sein, sonst sind Hardwarezugriffe während des Bootvorgangs unmöglich, was ihn sofort zum Abbruch führen würde. Die hier liegenden Dateien benötigen keinen physikalischen Speicherplatz auf der Partition, wohl aber Inodes.

/etc - Rechnerspezifische Konfigurationsdateien

Auch dieses Verzeichnis enthält Informationen, die notwendigerweise beim Booten gebraucht werden, also muß es auf der Wurzelpartition liegen. Hier finden sich alle wichtigen Konfigurationsdateien des Systems, unter anderem eben auch die Datei `fstab`, die beschreibt, welche Partitionen wohin gemountet werden müssen.

/home - Die Heimatverzeichnisse der User

Hier liegen die Heimatverzeichnisse der einzelnen User. Dieses Verzeichnis ist geradezu dafür vorgesehen, auf einer eigenen Partition zu liegen, damit die User nicht durch eine volle Platte das System zum Absturz bringen können. Alles was hier liegt ist dynamisch, hier werden die Daten der User abgelegt. Sehr sinnvoll ist es auch, dieses Verzeichnis über das Netz mit mehreren Rechnern zu teilen, so daß die verschiedenen User auf jedem Rechner ihre Daten vorfinden.

/lib - Grundlegende Libraries und Kernelmodule

Dieses Verzeichnis muß beim Systemstart zur Verfügung stehen, darf also auf keinen Fall auf einer anderen Partition als der Wurzelpartition liegen. Es enthält sowohl wichtige *shared libraries*, also Programmbibliotheken, ohne die die wichtigsten Programme des Systems nicht arbeiten können, als auch die Kernelmodule, mit anderen Worten die Gerätetreiber. Alles Informationen, die beim Booten zur Verfügung stehen müssen.

/mnt - Ein leerer Mountpoint

Das ist einfach nur ein leeres Verzeichnis, um temporär andere Dateisysteme dorthinein zu mounten. Die meisten modernen Linux-Distributionen enthalten neben diesem Verzeichnis noch mindestens `/cdrom` und `/floppy`, die auch nur leere Verzeichnisse sind, gedacht um eine CDROM bzw. Diskette dort hinein zu mounten.

/opt - Platz für große zusätzliche Programmpakete

Was früher in `/usr` abgelegt wurde, wird heute standardmäßig in dieses Verzeichnis abgelegt: große zusätzliche Programmpakete wie z.B. Netscape, StarOffice, KDE, Postgres usw. Nichts was zum Booten notwendig wäre, also ist dieses Dateisystem bestens geeignet, um auf einer eigenen (großen) Partition Platz zu finden.

Dieses Verzeichnis enthält zumeist statische Daten, kann also im normalen Betriebsablauf ReadOnly gemountet werden. Es enthält keine rechnerspezifischen Daten, ist also auch geeignet, um gemeinsam im Netz genutzt zu werden.

proc - Prozessinformationen
In diesem Verzeichnis finden sich Dateien, die eigentlich keine sind. Es handelt sich hier um Schnittstellen zum Kernel, die es ermöglichen, bestimmte Informationen vom Kernel zu erhalten und bestimmte Einstellungen im laufenden Betrieb des Kernels zu setzen.

Außerdem besitzt hier jeder laufende Prozess ein Unterverzeichnis, das den Namen der PID trägt und prozess-spezifische Informationen bereitstellt.

/root - Heimat des Systemverwalters
Dieses Verzeichnis ist das Home-Verzeichnis des Systemverwalters. Weil der Systemverwalter im Notfall, etwa bei einer Reparatur des Systems im Single User Mode auf seine Dateien zugreifen können muß, ist dieses Verzeichnis eben nicht unter `/home` sondern auf der Wurzelpartition direkt abgelegt. Es eignet sich also nicht dazu, auf eine andere Partition ausgelagert zu werden.

Bei älteren Unix-Systemen hatte der Systemverwalter kein Home-Verzeichnis, sondern die Wurzel des Systems (root) diente ihm als solches. Das führte aber dazu, daß auf der Wurzel dann einige Dateien lagen, die den Überblick über die Konsistenz des Systems erschwerten. Aus diesem Grund wurde dieses Verzeichnis eingeführt.

/sbin - Grundlegende Systemprogramme
Hier liegen wichtige Systemprogramme, die im Gegensatz zu den Programmen in `/bin` nicht für alle User, sondern nur für den Systemverwalter nötig sind. Alles, was hier liegt ist für den Bootvorgang nötig, darf also nicht auf einer anderen Partition als der Wurzelpartition liegen.

/tmp - Temporärer Speicherplatz
Dieses Verzeichnis ist ein Platz, in dem alle User Temporärdateien anlegen können. Es sollte das Sticky-Bit gesetzt haben, damit boshafte User nicht die Dateien anderer User löschen können. Es ist kein Problem, dieses Verzeichnis auf eine andere Partition zu legen, es enthält nichts, was für den Bootvorgang nötig wäre. Das Verzeichnis ist logischerweise nur für dynamische Dateien benötigt, darf also nicht ReadOnly gemountet werden.

/usr - Die zweite Dateihierarchie
Das /usr-Verzeichnis ist die zweite Hauptsektion des Unix-Dateisystems. Es enthält übers Netz teilbare, statische Daten. Es ist übliche Praxis, dieses Verzeichnis auf eine eigene Partition zu legen, die im Netz geteilt verwendet wird und jeweils ReadOnly gemountet ist. Große Programmpakete sollten - mit Ausnahme des X11-Systems - nicht hier, sondern unter `/opt` abgelegt werden. Der genaue Inhalt dieses Dateisystems wird im nächsten Abschnitt dargestellt.

/var - variable Daten
Frühere Unixe haben bestimmte variable Daten, wie etwa ein Temporärverzeichnis, die Systemlogbücher, Spoolverzeichnisse, usw. im `/usr` Verzeichnis abgelegt. Da dieses Verzeichnis heute unbedingt ReadOnly mountbar sein muß, wurde es nötig, einen Platz für variable (dynamische) Daten zu schaffen. Diese Aufgabe übernimmt heute das Verzeichnis /var.

Damit auch ältere Programme, die noch immer ins `/usr`-Verzeichnis schreiben wollen, nicht abstürzen, existieren heute zumeist die folgenden symbolischen Links im `/usr`-Verzeichnis, die ins `/var`-Verzeichnis verweisen:

- `/usr/spool -> /var/spool`
- `/usr/tmp -> /var/tmp`

Es ist durchaus möglich, dieses Verzeichnis auf eine eigene Partition zu legen und auch der zumindest teilweisen gemeinsamen Nutzung im Netz steht nichts im Weg. Allerdings enthält das `/var`-Verzeichnis einige Rechnerspezifische Daten, die im Netz nicht umbedingt geteilt werden sollten (etwa `/var/lock` - die Lockfiles oder `/var/run` - die ProzessIDs verschiedener Programme).

Das /usr Dateisystem

Die zweite Ebene der Dateisystemhierarchie liegt im `/usr` Verzeichnis. Hier finden sich sehr ähnliche

Verzeichnisse, wie auf der Wurzel des Systems. Allerdings sind alle Daten, die hier liegen statisch, müssen also im normalen Systembetrieb nicht verändert werden. Im einzelnen sind hier folgende Unterverzeichnisse wichtig:

X11R6 - Das X Window System, Version 11, Release 6
Hier liegen verschiedene Verzeichnisse, die wiederum die Binärdateien (bin), die Libraries (lib), Include-Dateien (include) und vieles mehr beinhalten, die alle zum X11-System gehören. Das ist das einzige große Programmpaket, das immer noch unter /usr liegt und nicht unter /opt.

bin - Die meisten User Kommandos
Hier liegen all die Userkommandos, die nicht unbedingt während des Systemstarts oder einer Notfallreparatur benötigt werden.

games - Spiele
Hier finden sich Spiele und andere zum Teil erzieherische Software.

include - Die Include-Dateien des C-Compilers
Hier finden sich die ganzen Include-Dateien, die sowohl für den C-, als auch für den C++ Compiler notwendig sind. Das Verzeichnis enthält auch viele Unterverzeichnisse, die wiederum Includedateien enthalten.

lib - Bibliotheken für Programme und Pakete
Dieses Verzeichnis enthält Objektdateien, Programmbibliotheken und interne Binärdateien, die nicht direkt von Usern aufgerufen werden sollen. Anwendungen können unter `/usr/lib` ein eigenes Verzeichnis haben, um ihre Daten dort abzulegen.

local - Die dritte, lokale Hierarchie
Dieses Verzeichnis sollte nach der Hauptinstallation leer sein. Hier ist Platz für eine weitere Dateisystemhierarchie, mit dem gleichen Inhalt wie `/usr` selbst. Nur sollten hier die lokalen Besonderheiten abgelegt sein, statt der systemspezifischen Standards.

sbin - Nicht lebensnotwendige Systemprogramme
Hier finden sich die Systemprogramme, die nicht für alle User, sondern nur für den Systemverwalter gedacht sind und die nicht während des Boot- oder Reparaturvorgangs benötigt werden.

share - Geteilte Daten
Hier liegen architekturunabhängige Daten, wie etwa Handbuchseiten, Dokumentationen, Wörterbücher, Sound usw.

src - Der Quellcode für Programme
Aller Quellcode für Systemprogramme, inclusive der des Systems selbst ist hier zu finden. Um hier Code zu übersetzen muß natürlich Schreibmöglichkeit existieren. Aus diesem Grund wird dieses Verzeichnis entweder als eigene Partition realisiert, die ReadWrite gemountet wird, oder der Systemverwalter remountet das /usr Verzeichnis ReadWrite, um Programme zu übersetzen.

Das /var Dateisystem

Vereinfacht gesagt gehört alles, was nicht statische Daten ist aber eigentlich unter `/usr` zu finden wäre, heute in dieses Verzeichnis. Insbesondere ist hier zu finden:

lib
Variable Statusinformationen der Programmbibliotheken

lock
Lockfiles, die anzeigen, daß ein bestimmtes Gerät gerade in Benutzung ist.

log
Die Systemlogbücher und Unterverzeichnisse für die Logdateien einzelner Anwendungen (z.B. Webserver, Mailsystem)

run

Prozessinformationen über gerade laufende Prozesse. Viele Daemon-Prozesse schreiben hier eine Datei hinein, die nur die ProzessID des laufenden Daemons enthält.

spool
Das Spoolverzeichnis für alle verschiedenen Dienste, die Warteschlangen unterstützen. Dazu zählen z.B. Druckerdienste, Faxdienste, Maildienste und Programme wie at oder cron.

tmp
Das Temporärverzeichnis, das eigentlich in `/usr` liegt, dort aber nur ein symbolischer Link auf dieses Verzeichnis ist.

Auffinden von Dateien und Programmen

Die oben beschriebene Struktur des Linux-Dateisystems macht es zwar leichter, einzelne Dateien und Programme innerhalb des Systems zu finden, aber wir brauchen natürlich noch Mechanismen und Programme, die uns das Auffinden einzelner Dateien ermöglichen.

Bei dieser Frage müssen wir zwischen Programmen und Dateien unterscheiden. Ein Programm ist eine ausführbare Datei, im folgenden wird der Begriff aber etwas genauer definiert. Linux enthält, wie oben gesehen, einige Verzeichnisse, die für die Aufnahme von Programmen (Binärdateien und Scripts) vorgesehen sind. Zumeist enden diese Verzeichnisse mit den Buchstaben `bin`. Damit einzelne Programme von überall her aufrufbar sind, ohne jeweils ihren ganzen Pfad eingeben zu müssen, existiert ein sogenannter Programmsuchpfad. Das ist eine Liste all der Verzeichnisse, die von der Shell durchsucht werden, wenn ein Programmaufruf eingegeben wurde. Diese Liste ist in einer Umgebungsvariable namens `PATH` gespeichert. Ein Normaluser hat hier meist andere Einstellungen als der Systemverwalter, der in seinem Pfad noch die verschiedenen `sbin`-Einträge hat.

Um herauszufinden, welche Verzeichnisse im Suchpfad stehen reicht der Aufruf von

```
echo $PATH
```

Eine typische Ausgabe für den Systemverwalter wäre z.B

```
/sbin:/usr/sbin:/usr/local/sbin:/root/bin:/usr/local/bin:/usr/bin:/usr/X11R6/bin:
/bin:/usr/games/bin:/usr/games:/opt/gnome/bin:/opt/kde/bin:.
```

Ein Normaluser würde aber eher etwas in der Art vorfinden:

```
/usr/local/bin:/usr/bin:/usr/X11R6/bin:/bin:/usr/games/bin:/usr/games:
/opt/gnome/bin:/opt/kde/bin:.
```

Wenn wir jetzt Programme suchen, so existieren Tools, die nur diesen Suchpfad durchforsten und uns sagen, welche Programme von wo aus aufgerufen werden. Suchen wir hingegen Dateien, so sind wir gezwungen, den gesammten Bestand an Verzeichnissen zu durchsuchen, was natürlich erheblich mehr Arbeit ist und länger dauert.

Für die Programme ist noch anzumerken, daß die Shell, wenn sie einen Programmaufruf ausführt, den Suchpfad von vorne nach hinten (von links nach rechts) durchsucht und das erste gefundene Programm mit dem passenden Namen ausführt. Das kann zu Konflikten führen, wenn wir z.B. zwei Programme gleichen Namens auf dem System haben, das eine in /bin und das andere in /usr/bin. Nachdem /usr/bin im Suchpfad vor /bin steht wird immer das Programm in /usr/bin aufgerufen.

Programme finden mit which, whereis und type

Wenn wir Programme suchen, so stehen uns dafür drei Tools zur Verfügung: which, whereis und type. Alle durchsuchen jeweils nur den Suchpfad, finden also nur die Programme, die die Shell auch finden würde.

Das Programm **which** wird zusammen mit einem oder mehreren Programmnamen aufgerufen und gibt uns dann den vollen Pfad des gefundenen Programms zurück. which arbeitet exakt wie die Shell, durchsucht eben den Suchpfad (`PATH`) und gibt das erste Kommando mit passendem Namen samt Pfad zurück. Der Aufruf von `which mount` hätte also die folgende Ausgabe gebracht:

```
/bin/mount
```

Das Programm **type** ist eigentlich gar kein Programm sondern eine interne Funktion der Shell (bash). Es zeigt etwas mehr Information als which, weil auch die hash-Mechanismen der Shell berücksichtigt werden und angezeigt wird, ob das gewünschte Kommando eine Shellfunktion, ein Alias, ein interner Shell-Befehl oder ein Programm ist.

Auf vielen Systemen ist tatsächlich etwa das Programm which nur ein Alias auf `type -p`.

Die Ausgabe von `type mount` würde folgendermaßen aussehen:

```
mount is /bin/mount
```

Das Programm whereis ist schließlich neben dem Auffinden von Programmdateien auch in der Lage, die zugehörigen Handbuchseiten und den Quellcode (falls installiert) von Programmen zu lokalisieren.

Der find-Befehl

Um jetzt nicht nur nach Programmen im Suchpfad zu suchen, sondern nach allen möglichen Dateien im System und um auch alle denkbaren Suchkriterien formulieren zu können, gibt es das Programm find.

find bietet aber neben der Fähigkeit Dateien zu suchen auch noch an, beliebige Aktionen mit den gefundenen Dateien auszuführen. Aus diesem Grund ist dieses Programm ein unentbehrliches Hilfsmittel für den Systemvewalter.

Die genauen Parameter, Tests und Aktionen von **find** sind auf der Handbuchseite aufgelistet, hier soll noch einmal in Beispielen erklärt werden, wie find aufgerufen wird und wie man damit vernünftig arbeiten kann.

Die einfache Aufrufform von **find** ist

find ***Startverzeichnis Test Aktion***

Dabei sucht **find** alle Verzeichnisse ab *Startverzeichnis* nach den Dateien ab, für die die Kriterien *Test* zutreffen. Für jede gefundene Datei wird dann die *Aktion* durchgeführt. Wenn die Aktion weggelassen wird, so wird standardmäßig die Aktion **-print** ausgeführt, d.h. der Dateiname der gefundenen Datei wird zusammen mit seinem Suchpfad (relativ zum angegebenenen Startverzeichnis) ausgegeben.

Um Dateien anhand ihres Namens zu finden, wird der Test **-name *Muster*** benutzt. Wenn der Name nicht vollständig bekannt ist, oder nach verschiedenen Dateien mit bestimmten Namen gesucht werden soll, dann kann das Suchmuster auch die Jokerzeichen verwenden, die auch die Shell benutzt (*,?,[...]). Aber Vorsicht: Das Namensmuster muß dann in Anführungszeichen gesetzt werden, weil sonst die Shell die Jokerzeichen interpretieren und durch evt. gefundene Dateien im aktuellen Verzeichnis ersetzen würde.

Um also z.B. alle Dateien des gesamten Systems zu finden, deren Namen mit einem großen oder kleinen M beginnt schreiben wir:

```
find / -name "[mM]*"
```

Der Slash steht für das Startverzeichnis, hier also das Wurzelverzeichnis. **-name** leitet den Namenstest ein und **"[mM]*"** ist das Suchmuster. Die Aktion haben wir hier einfach weggelassen, es wird also die **-print** Aktion durchgeführt. Das Ergebnis sieht dann etwa so aus:

```
/usr/X11R6/bin/macptopbm
/usr/X11R6/bin/mgrtopbm
/usr/X11R6/bin/mtvtoppm
/usr/X11R6/bin/makedepend
/usr/X11R6/bin/makeg
/usr/X11R6/bin/mergelib
```

```
/usr/X11R6/bin/mkdirhier
/usr/X11R6/bin/mkfontdir
/usr/X11R6/bin/mksusewmrc
/usr/X11R6/bin/manix
/usr/X11R6/bin/mirrormagic
/usr/X11R6/bin/mogrify
/usr/X11R6/bin/montage
/usr/X11R6/bin/mtv
/usr/X11R6/bin/mtvp
/usr/X11R6/bin/mpeg_play
/usr/X11R6/include/GL/MesaDrawingArea.h
/usr/X11R6/include/GL/MesaDrawingAreaP.h
/usr/X11R6/include/GL/MesaMDrawingArea.h
/usr/X11R6/include/GL/MesaMDrawingAreaP.h
/usr/X11R6/include/GL/MesaWorkstation.h
/usr/X11R6/include/GL/MesaWorkstationP.h
/usr/X11R6/include/X11/Xaw3d/MenuButtoP.h
...
```

Wenn mit Tests gearbeitet wird, die nummerische Parameter haben, dann gibt es eine Besonderheit. Wird die Zahl einfach, ohne Vorzeichen angegeben, dann heißt das, daß genau diese Zahl gemeint ist. Hat die Zahl als Vorzeichen ein Pluszeichen (+), dann heißt das, daß die Zahl oder eine größere Zahl gemeint ist, ein Minuszeichen bedeutet entsprechend die Zahl oder eine kleinere.

Der Test **-size** sucht Dateien nach ihrer Größe. Schreiben wir also

```
find . -size 12k
```

so sucht **find** nach allen Dateien im aktuellen Verzeichnis (.), die genau 12 Kilobyte groß sind. Wenn wir alle Dateien suchen, die höchstens 12 Kilobyte groß sind, so schreiben wir

```
find . -size -12k
```

und Dateien, die mindestens 12 Kilobyte groß sind finden wir mit

```
find . -size +12k
```

Wenn wir mehrere Tests angeben, so werden die Dateien gesucht, auf die alle Tests zutreffen, nicht die, die einen von beiden Tests bestehen. Der Befehl

```
find . -size +12k -name "M*"
```

sucht also nach allen Dateien im aktuellen Verzeichnis, die mindestens 12 Kilobyte groß sind **UND** deren Namen mit einem M beginnt.

Richtig interessant wird **find** erst mit der Anwendung von Aktionen. Die wichtigste und sicherlich meistgebrauchte Aktion ist **-exec**. Mit dieser Aktion lassen sich beliebige Unix-Kommandos ausführen, die mit den gefundenen Dateien angewendet werden. Dabei gibt es zwei wichtige Punkte zu beachten.

1. Die Befehlszeile der **-exec** Aktion muß mit einem **\;** abgeschlossen werden, damit eindeutig klar wird, was ist Befehlszeile und was weitere Aktionen. Der Strichpunkt muß mit einem Backslash versehen sein, damit ihn die Shell nicht interpretiert. Außerdem muß er durch ein Leerzeichen von der Befehlszeile getrennt sein.
2. Der Namen der gefundenen Datei wird mit einem **{}** dargestellt. Trifft **find** auf eben diese zwei geschweiften Klammern, so ersetzt es diese durch den gefundenen Dateinamen.

Um also etwa alle Dateien des ganzen /usr Verzeichnisses, die größer als 12 Kilobyte sind und deren Namen mit M beginnt in das Verzeichnis /tmp zu kopieren schreiben wir:

```
find /usr -size +12k -name "M*" -exec cp {} /tmp \;
```

Oder, um ein etwas praktischeres Beispiel zu nennen, wenn der Systemverwalter alle Dateien löschen will, die Usern gehören, die es nicht mehr gibt, dann kann er schreiben:

```
find / -nouser -exec rm {} \;
```

Das ist zweifellos ein gewisses Risiko, es könnten ja auch wichtige Dateien dabei sein, die durch einen Zufall einer UserID zugewiesen wurden, die nicht bekannt ist. Statt **-exec** können wir in so einem Fall auch **-ok** benutzen. Es macht exakt genau das Gleiche wie **-exec**, fragt aber bei jeder Ausführung vorher nach, ob es die Befehlszeile wirklich ausführen soll.

Die zentrale Dateidatenbank

Linux (und andere Unixe) verwalten eine zentrale Datenbank, die alle Dateien, die auf dem System gespeichert sind als Namenseintrag mit komplettem Pfad speichern und die so auch das Suchen nach Dateien ermöglicht. Damit diese Datenbank auf einem möglichst aktuellen Stand gehalten wird, muß z.B. täglich ein Programm laufen, das die gesammte Festplatte durchscannt und die gefundenen Dateinamen in dieser Datenbank abspeichert.

Dabei kommen unterschiedliche Techiken zum Zuge, es kann sein, daß die Datenbank nur in einer einzigen Datei liegt (z.B. `/var/lib/locatedb`) oder daß sie verteilt auf verschiedenen Dateisystemen vorliegt. Unter Linux ist bei den meisten Distributionen der Standort `/var/lib/locatedb` voreingestellt.

Das Programm, das diese Datenbank anlegt und täglich auffrischen sollte heißt **updatedb** und wird gewöhnlich via cron einmal täglich aufgerufen. Es erstellt eine Datenbank indem es alle angeschlossenen Laufwerke durchscannt und jede einzelne Datei samt Pfad speichert. Da das eine sehr aufwendige Aktion ist, wird dieser Programmaufruf in der Regel mitten in der Nacht ausgeführt und es existieren Möglichkeiten, bestimmte Verzeichnisse oder auch bestimmte Dateisysteme (wie etwa Netzlaufwerke) von der Suche auszunehmen.

In älteren Versionen von **updatedb** wurde diese Ausnahmeregelung in der Datei `/etc/updatedb.conf` vorgenommen. Hier wurden einfach Umgebungsvariablen definiert, die verschiedene Pfade ausschlossen, eine typische solche Datei könnte wie folgt aussehen:

```
# File: /etc/updatedb.conf
# This file sets environment variables which are used by updatedb

# filesystems which are pruned from updatedb database
PRUNEFS="NFS nfs afs proc smbfs autofs auto iso9660 ncpfs coda"
export PRUNEFS

# paths which are pruned from updatedb database
PRUNEPATHS="/tmp /usr/tmp /var/tmp /afs /amd /alex /var/spool"
export PRUNEPATHS

# netpaths which are added
NETPATHS=""
export NETPATHS
```

Die erste Anweisung (`PRUNEFS="NFS nfs afs proc smbfs ..."`) definiert Dateisystemtypen, die nicht durchsucht werden sollen. Die zweite (`PRUNEPATHS="/tmp /usr/tmp /var/tmp ..."`) definiert Pfade, die nicht durchsucht werden sollen.

Neuere Versionen von **updatedb** arbeiten mit Kommandozeilenoptionen, die den selben Effekt bieten.

Wozu diese Datenbank? Ein schnelles Auffinden von Dateien anhand von Namensmustern ist mit `find` nicht so einfach möglich. Denn `find` durchsucht ja tatsächlich die Festplatten nach den Dateien und das dauert. Da aber das Suchen nach Namensmustern sicherlich die häufigste Form ist, existiert das Programm **locate**. Dieses Programm durchsucht nicht die Festplatten, sondern eben die Datenbank, die durch **updatedb** erstellt wurde. Das geht enorm schnell und effektiv, hat jedoch den Nachteil, daß Dateien, die seit dem letzten Aufruf von **updatedb** erstellt wurden natürlich nicht gefunden werden können.

locate erlaubt die Verwendung von Suchmustern wie die Shell sie anbietet, also *, ?, [...]. Allerdings wird tatsächlich der ganze Dateinamenstring sammt Pfad durchsucht, ein Suchmuster von `foo*bar` würde also auch eine Datei `/usr/foo/bla/bar` finden.

Neben **locate** existiert alternativ auch das Programm **slocate**, das einen sichereren Zugriff gewährleistet. **slocate** erstellt die zentrale Datenbank und speichert aber zusätzlich auch die Eigentümerschaft und Zugriffsrechte der Dateien ab. So kann ein Normaluser nur die Dateien mit **slocate** finden, auf die er auch Zugriff hätte.

Insgesammt kann man zusammenfassen, daß locate und slocate eine schnelle und für normale Systemdateien durchaus zufriedenstellende Möglichkeit darstellt, Dateien anhand ihres Namens oder eines Suchmusters im Dateisystem zu finden. Wenn es aber darum geht, Dateien zu finden, die womöglich nach dem letzten Aufruf von updatedb erstellt wurden oder die Suchkriterien nicht den Namen der Datei benutzen, dann muß find benutzt werden.