Dateien:

Arten, Rechte und Bäume

unter Unix/Linux

Lars Oergel

24. Oktober 1999

Inhaltsverzeichnis

1 Inodes

Unix verwaltet Dateien mit Hilfe sog. *Inodes*. In diesen Inodes werden annähernd sämtliche Verwaltungsdaten einer Datei gespeichert. Dazu gehören

- die Dateigröße,
- der Dateityp,
- die Zugriffsberechtigungen,
- der Eigentümer,
- welcher Gruppe die Datei zugeordnet ist,
- das Gerät, auf dem die Datei liegt[1],
- die Inode–Nummer,
- die Anzahl der (Hard-)Links (d.h. der Namen),
- das Datum und die Zeit des letzten Zugriffs,
- das Datum und die Zeit der letzten Änderung des Datei–Inhalts,
- das Datum und die Zeit der letzten Änderung des Inodes an sich,
- und natürlich Informationen darüber, wo die Datei genau gespeichert ist.

Auch eine Datei, für die es mehrere, gleichberechtigte Namen gibt (Hard–Links), besitzt nur einen einzigen Inode, wovon man sich leicht mittels der *i*–Option des ls–Befehls überzeugen kann.

Die Inodes werden innerhalb eines Dateisystems (d.h. auf einer Partition) durchnumeriert, so daß durch das Tripel <Major–Nr.>, <Minor–Nr.> und <Inode-Nr.> jede Datei eindeutig identifiziert werden kann. Beim Anlegen des Filesystems auf einer Partition wird die maximale Anzahl der Inodes festgelegt. Daher kann ein Dateisystem nur eine bestimmte, maximale Anzahl von Dateien aufnehmen und ist auch dann schon „voll“, selbst, wenn eigentlich noch Plattenplatz verfügbar ist[2].

Der Linux–Befehl *stat* gibt übrigens alle Inode–Informationen über die Datei aus, mit deren Namen er als Parameter aufgerufen wurde.

Beispiel:

```
$ stat /
  File: "/"
  Size: 1024         Filetype: Directory
  Mode: (0755/drwxr-xr-x)         Uid: ( 0/ root)  Gid: ( 0/ root)
Device:  3,6   Inode: 2         Links: 16
Access: Sun Oct 24 00:17:23 1999(00000.00:36:19)
Modify: Sat Oct 23 20:07:07 1999(00000.04:46:35)
Change: Sat Oct 23 20:07:07 1999(00000.04:46:35)
```

[1]Das Gerät wird dabei über die Major- und Minor–Device–Number identifiziert – s. Kapitel 5.

[2]Daher sieht das Platten–Quotierungssystem auch eine maximale Anzahl von Inodes vor, die ein Benutzer belegen darf. Allerdings ist der Fall, daß wirklich alle Inodes „verbraucht“ sind, sehr selten.

2 Dateiarten

In Unix–Filesystemen gibt es sieben verschiedene Dateiarten:

1. Normale Dateien
2. Soft–Links
3. Verzeichnisse
4. Zeichenorientiert arbeitende Geräte (character devices)
5. Blockorientiert arbeitende Geräte (block devices)
6. FIFOs (= named Pipes)
7. Sockets

Wer hier die Hard–Links vermißt: Ein Hard–Link ist kein eigenständiger Dateityp, sondern beschreibt nur die Tatsache, daß eine Datei (d.h. ein Inode) mehrere Namen haben kann.

Wir wollen nun die einzelnen Dateiarten etwas genauer betrachten.

2.1 Normale Dateien

Normale Dateien sind, wie der Name schon sagt, „normale“ Datenspeicher, d.h. sie nehmen einfach einen unstrukturierten Datenstrom auf. Bei der Ausgabe des ls -l –Befehls sind sie in der ersten Spalte mit einem `-` gekennzeichnet.

Eine Besonderheit können sie allerdings doch haben: Sie können „löchrig“ sein, d.h. es muß sich bei den gespeicherten Daten nicht unbedingt um *einen* durchgehenden Datenstrom handeln. So ist es möglich, daß eine Datei zuerst 25 Nutzbytes enthält, dann ein 50 Bytes großes „Nichts“ folgt und danach weitere 25 Nutzbytes. Erreicht werden kann dieses, indem man bei programmgesteuerter Dateierzeugung mittels *seek*–Systemcall den Dateizeiger einfach entsprechende viele Bytes nach hinten setzt. Im Inode und beim ls–Befehl erscheint diese Datei dann mit 100 Bytes Länge, der du–Befehl ermittelt allerdings den echten Speicherbedarf.

2.2 Soft–Links

Soft–Links verweisen symbolisch auf andere Dateien. Dieser Verweis wird einfach in Form des (absoluten oder relativen) Dateinamens, auf den der Link verweisen soll, gespeichert. Das ls -l – Listing zeichnet Soft–Links mit einem `l` aus.

Die Verwendung von Soft–Links im Vergleich zu Hard–Links hat Vor- und Nachteile:

- Einerseits können Soft–Links auf Dateien auf anderen Filesystemen verweisen, bei Hard–Links geht das nicht. Ein Hard–Link verweist ja nicht auf einen Dateinamen, sondern direkt auf denselben Inode wie die referenzierte Datei (und Inode–Numern sind ja nur innerhalb eines Filesystems eindeutig).

- Außerdem sind Softlinks per se besser dokumentiert: Es ist leicht zu erkennen, daß ein Softlink vorliegt und auch die referenzierte Datei ist leicht auszumachen, bei Hard–Links ist das etwas schwieriger[3].
- Andererseits sind Soft–Links nicht mit der „echten“ Datei gleichzusetzen: Wird sie gelöscht, weist der Soft–Link ins Leere, einem Hard–Link kann das nicht passieren, da eine Datei solange gespeichert bleibt, bis sie ihren letzten Namen (d.h. Hard–Link) verliert.

2.3 Verzeichnisse

Verzeichnisse dürften ja bekannt sein. Sie werden von ls mit einem `d` gekennzeichnet und intern genauso wie eine normale Datei gespeichert (deshalb gibts auch eine Größenangabe beim ls–Befehl), der Inhalt wird eben nur anders interpretiert.

2.4 Character Devices

Character Devices stellen Namen, Zugriffsrechte und Zugriffsmöglichkeiten für zeichenorientiert arbeitende Gerätetreiber des Unix–Kernels zur Verfügung. Ihnen und den Block Devices ist das ganze Kapitel 5 gewidmet. Beim ls–Listing erscheinen sie mit einem `c` in der ersten Spalte.

2.5 Block Devices

Analog stellen Block Devices Namen, Zugriffsrechte und Zugriffsmöglichkeiten für blockorientiert arbeitende Gerätetreiber des Unix–Kernels zur Verfügung. Ihr Buchstabe ist das `b`.

2.6 FIFOs

FIFO ist eine Abkürzung und steht für „First in, first out“. FIFOs, die aus gutem Grund auch „benannte Pipes“ heißen, stellen eine Kommunikationsmöglichkeit in einer Richtung für „nicht–verwandte“ (bezüglich der Eltern–Kind–Beziehung der Prozeßhierarchie) zur Verfügung. Der ls–Befehl kennzeichnet sie mit einem `p`.

Die Vorgehensweise ist folgende: Wenn zwei Prozesse Daten in einer Richtung austauschen wollen (wie bei einer unbenannten Pipe einer Shell: ein Prozeß sendet, der andere empfängt), muß zunächst die benannte Pipe mit Hilfe des *mkfifo*–Befehls irgendwo im Dateisystem angelegt werden. Dann muß beiden Prozessen noch der Name der Pipe bekannt sein. Der sendende Prozeß öffnet dann einfach die Pipe zum Schreiben, der empfangende Prozeß öffnet sie zum Lesen, und schon kann der Datenaustausch stattfinden: Der Unix–Kernel übermittelt alle Daten, die der Schreiber in den FIFO hineinpackt in der Sendereihenfolge dem Leser. Natürlich muß man dabei noch auf die entsprechend gesetzten Zugriffsrechte der Pipe (und die Benutzer- bzw. Gruppenzugehörigkeit der beteiligten Prozesse) achten. Ein Beispiel für FIFOs findet man im /dev–Verzeichnis unter dem Namen *initctl*. Über diese Pipe reden einige wichtige Systemprozesse miteinander[4].

[3] Tip: `find -inum <inode-nr> ...`

[4] An dieser Kommunikation ist auch der wichtigste Prozeß im gesamten System, der Prozeß mit der Nummer 1 mit Namen init beteiligt.

2.7 Sockets

Ebenfalls zur Kommunikation zwischen nicht zwangsläufig verwandten Prozessen dienen die Sockets. Sie sind allerdings deutlich mächtiger als die named Pipes: Sie stellen universelle Kommunikationsendpunkte dar, über die (in einer anderen Betriebsart) sogar die weltweite TCP/IP–Kommunikation im Internet realisiert wird[5]. Sockets erscheinen mit einem `s` in der ls–Ausgabe.

Sockets werden nicht mit Hilfe eines speziellen Befehls vor Beginn der Kommunikation angelegt. Sie erscheinen vielmehr in dem Augenblick im Filesystem, in dem die beteiligten Prozesse durch das Aufrufen mehrerer System–Calls[6] zuerst die Sockets und dann die Kommunikation etablieren. Sockets werden u.a. innerhalb des X–Window–Systems verwendet und finden sich dann z.B. als `X0` im /tmp/.X11-unix–Verzeichnis. Aber auch im /dev–Verzeichnis werden wir mit `printer` und `log` fündig.

3 Zugriffsrechte

Jede Datei und jedes Verzeichnis in einem Unix–Dateisystem ist mit Zugriffsinformationen versehen, die kontrollieren, welche Benutzer bestimmte Aktionen durchführen dürfen.

Grundsätzlich werden dabei immer drei verschiedene Personen bzw. Personengruppen betrachtet: Der Eigentümer der Datei oder des Verzeichnisses (das ist der „Erschaffer“), eine bestimmte Benutzergruppe, der die Datei oder das Verzeichnis zugewiesen wurde[7] und „der Rest der Welt“. Jeder dieser drei Personen/-gruppen können nun bestimmte Rechte gewährt oder entzogen werden. Der ls -l – Befehl gibt die sog. `r`-, `w`- und `x`–Rechte für jede der drei Gruppen aus.

Beispiel:

```
$ ls -l /etc/inittab
-rw-r--r--   1 root     root         1756 Aug 27 05:37 /etc/inittab
```

Die erste Spalte kennzeichnet bekanntermaßen den Dateityp, die neun weiteren Spalten enthalten jeweils ein `r`, `w` oder `x` im Falle des gewährten Rechtes, ansonsten ein `-`, zunächst für den Eigentümer, dann die zugeordnete Gruppe (das ist das zweite „root“) und zuletzt den Rest.

Änderungen an den Zugriffsrechten können mit Hilfe des *chmod*–Befehls vorgenommen werden, s. dazu `man chmod`.

Bei der Betrachtung der einzelnen Rechte ist nun allerdings zu unterscheiden, ob es sich um eine Datei oder ein Verzeichnis handelt. Betrachten wir zunächst die Dateien.

[5] Informationen über momentan im System aktive Sockets kann man daher auch konsequenterweise mit dem netstat -a – Befehl erhalten.

[6] s. `man socket` und Verweise

[7] Wenn der „Erschaffer“ nur Mitglied einer einzigen Benutzergruppe ist, ist das automatisch diejenige, ansonsten kann sich der Benutzer aussuchen, welcher Benutzergruppe eine Datei oder ein Verzeichnis zugewiesen sein soll.

3.1 Dateien

Bei den Dateien bedeuten die `r`-, `w`- und `x`–Rechte im einzelnen:

1. Das *R*ead – Recht

   erlaubt es der entsprechenden Person oder Benutzergruppe, die Datei zum Lesen zu öffnen.

2. Das *W*rite – Recht

   erlaubt es der entsprechenden Person oder Benutzergruppe, die Datei zum Schreiben zu öffnen. Allerdings kann mit diesem Recht **nicht** festgelegt werden, ob die Datei gelöscht werden darf, dafür ist ausschließlich das Schreib–Recht im entsprechenden Verzeichnis nötig!!

3. Das e*X*ecute – Recht

   erlaubt es der entsprechenden Person oder Benutzergruppe, das in dieser Datei möglicherweise enthaltene Programm (oder zu interpretierende Skript) auszuführen.

3.2 Verzeichnisse

Bei den Verzeichnissen ist die „Rechtslage" nicht ganz so offensichtlich wie bei Dateien, so bedeuten die `r`-, `w`- und `x`–Rechte im einzelnen:

1. Das *R*ead – Recht

   erlaubt es der entsprechenden Person oder Benutzergruppe, das Verzeichnis zu lesen. Es darf also mit `ls <Verzeichnisname>` aufgelistet werden[8]. Allerdings kann mit diesem Recht **nicht** festgelegt werden, ob das Verzeichnis auch mit dem cd–Befehl zum aktuellen Verzeichnis gemacht werden darf, dafür ist das „Ausführ"–Recht des entsprechenden Verzeichnisses nötig!!

2. Das *W*rite – Recht

   erlaubt es der entsprechenden Person oder Benutzergruppe, im Verzeichnis zu schreiben. Es dürfen also neue Dateien angelegt und alte Dateien umbenannt oder gelöscht werden!! Um diese Aktionen ausführen zu können, sind **keinerlei** Rechte an den beteiligten Dateien oder die Eigentümerschaft nötig!!

3. Das e*X*ecute – Recht

   erlaubt es der entsprechenden Person oder Benutzergruppe, dieses Verzeichnis mit dem cd–Befehl zum aktuellen Verzeichnis zu machen. Allerdings kann mit diesem Recht **nicht** festgelegt werden, ob der Verzeichnisinhalt mit ls angezeigt werden kann, dafür ist das Lese–Recht des entsprechenden Verzeichnisses nötig!! Bei Kenntnis von Datei- oder Verzeichnisnamen, die sich innerhalb dieses Verzeichnisses befinden, ist allerdings das Arbeiten (sofern die erforderlichen Rechte gewährt sind) mit den Dateien oder Verzeichnissen uneingeschränkt möglich!!

[8] „Spaß" bzw. Sinn macht das ohne X–Recht allerdings nicht: Die Ausgabe des ls–Befehls ist mit Fehlermeldungen durchsetzt, da der *stat*–Systemcall, der Informationen über die Dateien bzw. Inodes liefert, nur für Dateien ausgeführt werden darf, die in einem Verzeichnis liegen, für das der Aufrufer das X–Recht hat. Daher kann auch auf keine Datei in einem solchen Verzeichnis lesend oder schreibend zugegriffen werden, unabhängig von den Dateirechten!

3.3 Spezielles

Zusätzlich zu den genannten Lese-, Schreib- und Ausführrechten gibt es noch einige Spezialbits, die nun besprochen werden sollen.

3.3.1 Set–UID–Bit

Das sog. „set user ID"–Bit ist nur bei ausführbaren Dateien sinnvoll: Ist eine Datei mit diesem Bit versehen, so läuft sie unabhängig davon, wer sie aufruft, mit den Berechtigungen des Eigentümers der Datei. Dazu wird die sog. *effective UID* für die Laufzeit des Programmes auf die UID des Eigentümers der Datei gesetzt. Die *real UID* verbleibt dabei unberührt. Normalerweise sind effective UID und real UID identisch, nur während der Laufzeit von set–UID–Programmen unterscheiden sie sich[9]. Zur Prüfung der Zugriffsberechtigungen auf Dateien oder Verzeichnisse wird immer die effective UID benutzt.

Dieses Programm darf dann also z.B. auf Dateien oder Verzeichnisse zugreifen, auf die zwar der Eigentümer der Datei, nicht aber der Aufrufer Zugriffsrechte hat. Dieser Mechanismus ist bei vielen Unix–Systembefehlen anzutreffen, so z.B. beim passwd–Befehl, unter dessen Kontrolle ja jeder Systembenutzer die ansonsten nur für root beschreibbare /etc/passwd–Datei modifizieren kann.

Allerdings birgt die Anwendung des suid–Bits auch Sicherheitsgefahren, da ein „schlecht" programmiertes Programm es einem geschickten Angreifer evtl. erlaubt, Dinge unter der entsprechenden Benutzerkennung zu tun, an die der Programmierer nicht gedacht hatte. Aus diesem Grunde ist auch speziell unter Linux der suid–Mechanismus für Shell- und andere -Skripte (d.h. für alle zu interpretierenden Dateien) im Kernel deaktiviert, d.h. er funktioniert unter Linux nur bei echten Binaries!

Das gesetzte suid–Bit wird unter Unix traditionellerweise „über" dem X–Recht des Eigentümers angezeigt: Sind X–Recht und suid–Bit einer Datei gesetzt, steht an der dritten Stelle der Berechtigungen ein kleines `s`. Ist nur das suid–Bit gesetzt, das X–Recht jedoch nicht, erscheint ein großes `S`. Will man beim chmod–Befehl das suid–Bit mittels Oktalzahlnotation setzen, so ist die Bitmaske `4000` zu verwenden.

Beispiel:

```
$ ls -l /usr/bin/passwd
-r-s--x--x  1 root   root     10704 Apr 14  1999 /usr/bin/passwd
```

3.3.2 Set–GID–Bit

Das nächste Spezialbit ist sowohl bei ausführbaren Dateien als auch bei Verzeichnissen sinnvoll: Das sog. „set group ID"–Bit bewirkt bei Dateien analog dem suid–Bit, daß das Programm mit einer anderen effektiven GID läuft (nämlich von der Gruppe, der die Datei zugeordnet ist), auch hier bleibt wieder die reale GID unverändert.

Wenn es für ein Verzeichnis gesetzt ist, bewirkt es, daß alle Dateien und Verzeichnisse, die in diesem Verzeichnis (von berechtigten Benutzern) angelegt werden, default–mäßig derselben Gruppe wie das Verzeichnis selbst zugeordnet sind. Ist ein Benutzer Mitglied mehrerer Gruppen, so kann er ja normalerweise mit dem newgrp–Befehl einstellen, mit welcher Gruppenzugehörigkeit die vom ihm angelegten Dateien und

[9] s. dazu auch `man id`

Verzeichnisse default–mäßig versehen werden und die Zuordnung außerdem später mit dem chgrp–Befehl verändern.

Das sgid–Bit kann bei Verzeichnissen also eingesetzt werden, um dafür zu sorgen, daß Dateien in Verzeichnissen, in denen explizit eine ganze Gruppe Schreibrechte haben soll, auch stets mit der entsprechenden Gruppenzugehörigkeit angelegt werden, so daß kein Gruppenmitglied sich darum kümmern muß, ob es nun gerade die „richtige“[10] Gruppe als aktive Gruppe (mit dem newgrp–Befehl) eingestellt hatte. Dieses Bit wirkt sozusagen rekursiv: Auch, wenn man Unterverzeichnisse in dem entsprechenden Verzeichnis anlegt, wird die Gruppenzugehörigkeit entsprechend gesetzt und damit dann auch wieder in diesem Unterverzeichnis usw.

Das gesetzte sgid–Bit wird unter Unix traditionellerweise „über“ dem X–Recht der Gruppe angezeigt: Sind X–Recht und sgid–Bit gesetzt, steht an der sechsten Stelle der Berechtigungen ein kleines `s`, ist nur das das sgid–Bit gesetzt, das X–Recht jedoch nicht, erscheint ein großes `S`. Will man beim chmod–Befehl das sgid–Bit mittels Oktalzahlnotation setzen, so ist die Bitmaske `2000` zu verwenden.

Beispiel:

```
$ ls -l /usr/bin/man
-rwxr-sr-x  1 root    man     32320 Apr 10  1999 /usr/bin/man
```

3.3.3 Sticky–Bit

Nun zum letzten Spezialbit, das heute[11] nur noch bei Verzeichnissen Sinn macht: Das sog. „Sticky“–Bit sorgt dafür, daß auch in Verzeichnissen, die welt–beschreibbar sind, nicht jeder alle Dateien löschen kann, sondern immer nur der Eigentümer seine eigenen Dateien. Dieses Bit ist äußerst sinnvoll bei /tmp- und /var/mail–Verzeichnissen, die zwar aus naheliegenden Gründen welt–beschreibbar sind, in denen aber jeder nur seine eigenen Dateien löschen können soll.

Das gesetzte Sticky–Bit wird unter Unix traditionellerweise „über“ dem X–Recht der „Others“ (d.h. dem Rest der Welt) angezeigt: Sind X–Recht und Sticky–Bit eines Verzeichnisses gesetzt, steht an der neunten Stelle der Berechtigungen ein kleines `t`, ist nur das Sticky–Bit gesetzt, das X–Recht jedoch nicht, erscheint ein großes `T`. Will man beim chmod–Befehl das Sticky–Bit mittels Oktalzahlnotation setzen, so ist die Bitmaske `1000` zu verwenden.

Beispiel:

```
$ ls -ld /tmp
drwxrwxrwt  8 root   root      1024 Oct 24 04:12 /tmp
```

[10] in dem Sinne, daß auch alle Dateien in diesem durch eine Gruppe beschreibbaren Verzeichnis der Gruppe zugeordnet sind, so daß alle Gruppenmitglieder nicht nur in dieses Verzeichnis schreiben dürfen, sondern daß auch alle Gruppenmitglieder auf alle Dateien und Unterverzeichnisse in diesem Verzeichnis zugreifen dürfen

[11] Früher bewirkte das Sticky–Bit bei ausführbaren Dateien, daß der Programmcode nach Beendigung des Programmes im Speicher verblieb.

4 Reise durchs Dateisystem

Wir wollen uns nun die typische[12] Struktur eines Linux–Dateisystems anschauen.

Zunächst betrachten wir nur Unterverzeichnisse im Root–Verzeichnis, anschließend werden wir uns das i.d.R. umfangreichste Unterverzeichnis, nämlich das /usr–Verzeichnis anschauen.

4.1 Das Root – Verzeichnis

Im Root–Verzeichnis selber liegen normalerweise höchstens zwei bis drei echte Dateien, nämlich ein oder zwei Linux–Kernels (z.B. /vmlinux oder /zImage) und vielleicht noch eine sog. System–Map[13]. Neuerdings werden aber auch diese Dateien in Unterverzeichnissen (in diesem Falle /boot) abgelegt, so daß sich im Root-Verzeichnis nur noch Unterverzeichnisse befinden. Diese Unterverzeichnisse heißen im einzelnen:

- /bin

  Hier liegen „wichtige“[14] Binaries für normale Benutzer und den Systemverwalter.

- /boot

  beinhaltet Dateien, die unmittelbar für den Systemstart bzw. dessen Konfiguration benötigt werden.

- /dev

  beinhaltet alle Gerätedateien[15].

- /etc

  „Wichtige“ Systemkonfigurationsdateien werden hier abgelegt.

- /home

  Homeverzeichnisse normaler Benutzer. In größeren, vernetzten Umgebungen ist dieses meist nur ein sog. Mountpoint, an dem eine weitere Partition oder (Netz-)Festplatte mit den eigentlichen Benutzerdaten gemountet wird[16].

- /lib

  Hier liegen die dynamischen Lader, „wichtige“ Bibliotheken und alle Kernelmodule (d.h. modularisierte Gerätetreiber).

[12] Da Linux ja ein „chaotisch“ entstandenes System ist, und sich im Bereich des Filesystem–Aufbaus doch eine gewisse Standardisierung lohnt, wurde der ***Filesystem Hierarchy Standard*** [1] ins Leben gerufen, der Empfehlungen für den Aufbau eines Filesystems gibt, an dem sich mittlerweile auch fast alle Linux–Distributionen „orientieren“.

[13] Das ist eine Datei, die eine Liste beinhaltet, die die Namen der Kernelroutinen den entsprechenden Adressen im Kernel zuordnet. Mit diesen Informationen können bei sehr systeminternen Verwaltungsarbeiten anstatt nichtssagender Hex–Adressen die Namen von Kernel–Routinen ausgegeben werden — eine solche System–Map ist aber zum reinen Betrieb eines Linux–Systems nicht nötig.

[14] „Wichtig“ heißt im gesamten Dateisystem–Kapitel, daß es sich um Dateien handelt, die entweder zum Starten des Systems oder zu Wartungs- und Reparaturarbeiten nötig sind. „Unwichtig“ sind in diesem Sinne reine Anwendungen, Dokumentation, graphische Oberflächen, Daten normaler Benutzer, usw.

[15] s. dazu auch Kapitel 5

[16] s. /etc/fstab, df–Befehl, mount–Befehl bzw. autofs

- /mnt

  Mountpoint zum vorübergehenden Mounten anderer Filesysteme, z.B. CD-Roms oder (Netz-)Festplatten.

- /proc

  Virtuelles Filesystem, das eine Schnittstelle zum Linux–Kernel darstellt.

- /root

  Hier ruht das Homeverzeichnis des Systemverwalters.

- /sbin

  „Wichtige" Binaries für den Systemverwalter (und interessierte Benutzer – die haben das allerdings i.d.R. nicht im Pfad, müssen diese Befehle also mit absoluter Pfadangabe aufrufen).

- /tmp

  Wie der Name schon sagt: Sowohl Benutzer als auch das System legen hier vorübergehend anfallende Daten ab (und löschen sie hoffentlich auch wieder).

- /usr

  Hier liegen weniger wichtige Anwendungen und deren Konfigurations- und Datendateien.

- /var

  Ablage für variable Daten, also z.B. Caches jeglicher Natur, Mails, Druckjobs, Lockdateien, Protokolldaten usw.

Sinn und Zweck dieser Einteilung ist nicht nur, eine einheitliche Norm zu schaffen, nach der alle Linux–Distributionen aufgebaut sein sollen (erleichtert ungemein das distributionsübergreifende Suchen nach einer Datei). Die Daten sollen außerdem gleichzeitig so im Filesystem verteilt sein, daß man die /home-, /usr- und /var–Verzeichnisse nicht zum Booten, Warten und Reparieren von Systemen braucht.

Das hat mehrere Vorteile: Die eigentliche Root–Partition kann relativ klein sein, wodurch die Wahrscheinlichkeit eines Plattenfehlers in ihrem Bereich sinkt (und solange die Root–Partition noch funktioniert, kann man wenigstens das System noch starten und Reparaturarbeiten beginnen), zusätzlich können zumindest die /home- und die /usr–Unterverzeichnisse und auch Teile der /var–Struktur auf zentralen Netzwerkservern liegen und von vielen Clients gleichzeitig benutzt werden.

4.2 Das /usr – Verzeichnis

Nun schauen wir uns die wichtigsten Unterverzeichnisse im /usr–Verzeichnis an:

- /usr/X11R6

  Hier liegen annähernd sämtliche Bestandteile des graphischen X–Window–Systems.

- /usr/bin

  dient weniger wichtigen Anwendungen für normale Benutzer und den Systemverwalter als Aufenthaltsort.

- /usr/doc

  Eigentlich ein sehr wichtiger Bestandteil eines jeden Linux–Systems: Unmengen an Dokumentation, ja ganze Bücher warten in diesem Verzeichnis darauf, gelesen zu werden!

- /usr/etc

  Hier könnten Konfigurationsdateien der weniger wichtigen Programme liegen, aber leider wird dieses Verzeichnis nur allzu selten und benutzt und stattdessen sämtlicher Konfigurationskrempel im /etc–Verzeichnis abgeladen.

- /usr/include

  Hier liegen jede Menge Headerfiles für alle möglichen Bibliotheken.

- /usr/info

  Hier wird Dokumentation im sog. TeXinfo–Format abgelegt[17].

- /usr/lib

  ist für weniger wichtige Bibliotheken und sonstige Datendateien (z.B. Konsol–Fonts und -Keymaps) die richtige Adresse.

- /usr/local

  Nochmal eine ganze Struktur mit bin, doc, etc, include, info, lib, man, sbin und src für eigene Installationen außerhalb der jeweiligen Linux–Distribution (ist pflegeleichter bei Distributionsneuinstallation oder -update).

- /usr/man

  Ablage der Man–Pages. Die Man–Pages untergliedern sich in neun Kapitel: User Commands, System Calls, Library Functions, Devices, File Formats, Games, Miscellaneous, System Administration, Internal Kernel Routines.

- /usr/sbin

  für weniger wichtige Binaries für den Systemverwalter (und interessierte Benutzer).

- /usr/share

  Hier liegen architekturunabhängige Daten wie z.B. Icons oder auch Fonts.

- /usr/src

  Sourcecodes, z.B. /usr/src/linux: Linux–Kernel.

[17] Das ist ein Dokumentationsformat, das es erlaubt, aus derselben Quelldatei sowohl online abrufbare Info–Seiten zu generieren, als auch ein hübsch auszudruckendes Büchlein zu machen.

5 Gerätedateien

Nun wollen wir diese mysteriösen Gerätedateien etwas genauer unter die Lupe nehmen.

5.1 Character Devices

Character Device–Files (oder auch „unbuffered special files") stehen für Geräte, deren Treiber zeichenorientiert, d.h. ungepuffert arbeiten. Das sind Geräte wie z.B. Terminals (TTYs), Mäuse, Modems und Drucker.

Sie werden entweder bei der Systeminstallation oder nachträglich vom Systemverwalter mit Hilfe des Befehls

```
mknod <device-filename> c <major-device-no.> <minor-device-no.>
```

angelegt. Zur Bedeutung der sog. Haupt- und Untergerätenummern kommen wir noch.

5.2 Block Devices

Block Device–Files (oder auch „buffered special files") stehen für Geräte, deren Treiber blockorientiert, d.h. gepuffert[18] arbeiten. Dazu gehören Geräte wie Disketten- und CD–Rom–Laufwerke, Festplatten oder auch Streamer.

Auch sie werden entweder bei der Systeminstallation oder nachträglich vom Systemverwalter mit Hilfe des Befehls

```
mknod <device-filename> b <major-device-no.> <minor-device-no.>
```

angelegt.

5.3 Gemeinsamkeiten

Gerätedateien sind keine echten Dateien, außer dem Inode belegen sie keinen Plattenplatz[19].

Ein Zeichen- oder Blockgerätetreiber wird im Kernel über die Haupt- und Untergerätenummer identifiziert[20]. Da ein Treiber ja möglicherweise mehrere Geräte verwaltet, z.B. mehrere Festplatten(-partitionen), mehrere serielle Schnittstellen oder mehrere (Pseudo–)TTYs, gibt es zusätzlich die Untergerätenummer. Mit ihrer Hilfe kann ein Treiber die von ihm verwalteten Geräte unterscheiden. So kann jedes einzelne Gerät über das Tripel <block/character>, <major-device-no.> und <minor-device-no.> identifiziert werden[21].

[18] Es wird also nicht Byte für Byte vom/zum Gerät übertragen, sondern immer eine gewisse Menge, z.B. 512 Bytes oder 1 KB auf einmal.
[19] Was man wiederum mit `stat` leicht überprüfen kann.
[20] s. `cat /proc/devices`
[21] Es kann also jede Hauptgerätenr. zweimal geben: als Block Device und als Character Device.

Die Gerätedateien erfüllen nun viererlei Aufgaben:

1. Zum einen kann man mit ihnen den unhandlichen Tripeln zur Geräteidentifizierung, z.B. <char>, 3, 0 leichter zu merkende Namen zuordnen, z.B. /dev/hda (HardDisk A – die erste Festplatte).
2. Zweitens kann man durch die (Pseudo-)Dateien die übliche Rechtevergabe des Unix–Dateisystems verwenden, um festlegen zu können, welcher Benutzer (direkten) Zugriff auf bestimmte Geräte hat.
3. Drittens dienen die Dateien auch zur Kommunikation mit den Geräten: Der Linux–Kernel fängt jeden lesenden oder schreibenden Zugriff auf die Gerätedateien ab und leitet den Datenstrom zum angesprochenen Gerät um. Durch dieses Verhalten ist der Zugriff auf Geräte insbesondere für Programmierer relativ einfach zu realisieren.
4. Und viertens bieten viele Gerätetreiber noch weitere spezielle Steuerungskommandos über den sog. „ioctl"–Systemcall an, mit dem geräte- und treiber–spezifische Befehle erteilt werden können[22]. Manche Gerätetreiber bieten gar nur diese Möglichkeit zur Kommunikation mit ihrem Gerät.

Hier noch einige Beispiele für Gerätedateien:

Char/ Block	**Haupt-ger.nr.**	**Unter-ger.nr.**	**/dev**	**Bedeutung**
c	4	64	ttyS0	1. Serielle Schnittstelle, oft Maus
c	4	65	ttyS1	2. Serielle Schnittstelle, oft Modem
c	10	1	psaux	PS/2–Mausanschluß
c	6	0	lp0	1. Parallele Schnittstelle, i.d.R. Drucker
b	3	0	hda	1. (E)IDE–Festplatte
b	3	64	hdb	2. (E)IDE–Festplatte/ATAPI–CD–Rom
b	8	0	sda	1. SCSI–Festplatte
b	8	16	sdb	2. SCSI–Festplatte
b	11	0	scd0	1. SCSI–CD–Rom

Literatur

[1] Filesystem Hierarchy Standard Group. *Filesystem Hierarchy Standard*, 1997. http://www.pathname.com/fhs.

[2] Kai Petzke. *Unix für jedermann – Einführung in Linux*. Bernd–Michael Paschke Verlag, 1995. Online–Version unter http://www.bmp.de/ISBN/3-929711-07-9.

[22] z.B. das Zurückspulen eines Streamerbandes