Einführung Unix/Linux

Kapitel: Dateisystem und Dateien

Jens Roesen <jens@roesen.org>

Würzburg, März 2010
Version: 0.1.5 – **sehr beta** –

Dieses Dokument wurde in vim (`http://www.vim.org`) bzw. TEXnicCenter (`http://www.texniccenter.org/`) geschrieben und mit LaTeX (`http://www.latex-project.org/`) formatiert und gesetzt.
Die jeweils aktuelle Version ist unter `http://www.roesen.org` erhältlich.

Inhaltsverzeichnis

Vorwort

Motivation

Im Rahmen interner Schulungsmassnahmen kam die Frage nach geeigneten Schukungsmaterialien bzw. einem Skript für Unix-Neulinge auf. Schulungsunterlagen und Skripte für Einsteiger, aber letztendlich hat mir bei den meisten entweder etwas gefehlt, oder es war für unseren Zweck viel zu viel irrelevanter Stoff. Da wir in der Hauptsache mit Solaris und Linux-Systemen arbeiten und dabei Themen wie X-Windows oder Druckerverwaltung komplett ausklammern können, aber auf Themen wie Netzwerke, Troubleshooting, Mailserver oder DNS-Server Wert legen, habe ich mich irgendwann hingesetzt und angefangen dieses Skript zu schreiben.

Es ist der Versuch Systemadministratoren mit Grundkenntnissen in Unix den Arbeitsalltag zu erleichtern und dabei zwei verschiedene Unix-Geschmacksrichtungen, nämlich Solaris[1] und Red Hat Enterprise Linux, gleichermassen zu betrachten.

Mir ist durchaus klar, dass nicht alles im Folgenden beschriebene „state of the art“ ist bzw. sein kann und sicher auch noch etliche Fehler übersehen wurden. Wer einen solchen findet, weiss wie man einige Aufgaben besser lösen kann oder bessere Beispiele kennt ist herzlich eingeladen mir eine Mail an <jens@roesen.org> zu schicken.

Zielgruppe

Dieses Kurzscript ist als Crashkurs zur Einführung in die Administration von Unix und Linux Systemen gedacht. Es wird dabei ausschliesslich auf der Konsole und ohne grafische Oberfläche gearbeitet. Die gezeigten Beispiele beziehen sich auf Systeme unter Sun Solaris und RedHat Enterprise Linux.

Ohne Vorkenntnisse und Erfahrung mit Unix und/oder Linux Systemen wird der angesprochene Stoff teils nur schwer zu verstehen sein. Als alleiniges Lehrskript für blutige Anfänger ist es daher nicht geeignet obwohl in einigen Kapiteln vereinzelt kurz auf Grundlagen eingegangen wird (z.B. Kapitel 2).

Aufbau des Skripts

Wirr. Durch und durch. Aber zu mehr ist momentan keine Zeit. Ich habe versucht die Kapitel und Themen in eine halbwegs sinnvolle Reihenfolge zu bringen. Im Lauf der Zeit wird da sicherlich noch einiges umgestellt werden.

[1] In der vorliegenden Version des Skripts nur bis Version 9.

Typographisches

Da sich alle Beispiele, Kommandos und Ausgaben auf der Konsole abspielen, werden diese Bereiche entsprechend formatiert um sich vom regulären Text abzusetzen. Nach dem Login als User root, mit dem wir in diesem Skript hauptsächlich arbeiten werden, landet man in einem Command Prompt der so aussehen koennte:

```
[root@server1 root]#
```

Da dieser Prompt ja nach name des Systems oder aktuellem Verzeichnis mal kürzer aber auch sehr viel länger sein kann, wird der root Prompt auf

```
#
```

verkuerzt. Bitte den Hash (`#`) hier **nicht** als Kommentarcharakter verstehen, der unter Linux/Unix z.B. in Shellskripten die folgende Zeile vor der Ausführung durch die Shell schützt. Der normale User-Prompt, falls er uns wirklich einmal begegnen sollte, wird analog dazu auf

```
$
```

zusammengestrichen.

Für Konsolenausgaben, Konfigurationsdateien oder Teile von Skripten wird eine nichtproportionale Schrift verwendet:

```
if [ -n "$_INIT_NET_IF" -a "$_INIT_NET_STRATEGY" = "dhcp" ]; then
        /sbin/dhcpagent -a
fi
```

Werden in einem Beispiel Konsoleneingaben vom Benutzer erwartet, wird die in einer nichtproportionale Schrift dargestellt, wobei die Benutzereingaben fett gedruckt sind:

```
# uname -a
SunOS zoidberg 5.9 Generic_118558-21 sun4u sparc SUNW,Sun-Blade-100
```

Kommandos, Dateinamen oder Benutzerkennungen im laufenden Text werden ebenfalls in einer nichtproportionalen Schrift dargestellt: „Mit dem Befehl `pwd` kann überprüft werden, in welchem Verzeichnis man sich gerade befindet.“.

Müssen in einem Beispiel noch Teile der erwarteten Benutzereingaben durch die richtigen Werte ersetzt werden, so wird dieser Teil in kursiver Nichproportionalschrift dargestellt: „Für jedes Interface welches beim boot konfiguriert werden soll muß eine Datei **/etc/hostname.*interface*** existieren.“.

Eigennamen, Personen oder Organisationen erscheinen manchmal (ich bin gerade zu faul alle Vorkommen entsprechend zu formatieren) in Kapitälchen: „Eine sehr große Rolle hierbei hat die University of California in Berkley (UCB) gespielt, an der Thompson im Winter 76/77 eine Vorlesung zum Thema Unix abhielt.“.

1 Dateisystem und Dateien

> You can use any editor you want, but remember that vi vi vi is the text editor of the beast!
>
> *(Richard Stallman, HOPE 2006)*

1.1 Dateien und Dateiattribute

Jede Datei in Unix verfügt über bestimmte Attribute, die festlegen wer alles mit der Datei arbeiten darf und welche Operationen erlaubt sind. Diese Attribute sind uns in Kapitel 2 schon ab und zu in den Ausgaben von `ls -l` begegnet. Sehen wir uns die Ausgabe einmal genauer an.

```
-rw-r--r-- 2 root root    0 2008-11-10 10:42 file1
```

Die ersten 10 Zeichen (`-rw-r--r--`) verraten uns um welchen Dateityp es sich handelt (mehr dazu in Kapitel 1.2) und welche Zugriffsrechte für die Benutzer(gruppen) gelten. Spalte drei und vier sagen uns welchem User und welcher Gruppe die Datei gehört (`root root`). Die übrigen Spalten geben Auskunft über den Link Count (`2`), Dateigröße (`0`), Zeitstempel des letzten Schreibzugriffs (`2008-11-10 10:42`) und natürlich den Dateinamen (`file1`).

Wenn eine neue Datei angelegt wird, gehört sie normalerweise dem User der sie angelegt hat bzw. dem der Prozess gehoert welcher die Datei angelegt hat sowie der Gruppe zu der dieser User gehört. Die Datei aus dem Beispiel oben gehoert also dem User `root` und der Gruppe `root`. Dateieigentümer und Gruppe können durch die Befehle `chown` und `chgrp` geändert werden.

```
# ls -l file1
-rw-r--r-- 1 root root 0 2008-11-10 10:42 file1
# chown drizzt file1
# ls -l file1
-rw-r--r-- 1 drizzt root 0 2008-11-10 10:42 file1
#
```

Auf den meisten Unix Systemen kann allerdings nur der Superuser `chown` ausführen, da es sonst möglich wäre Dateien anderen Usern „unterzuschieben“ um so Quotabeschränkungen zu umgehen.

```
$ chown root file1
chown: changing ownership of ‘file1’: Operation not permitted
```

Analog zur Syntax von `chown` wird auch `chgrp` benutzt. Will man sowohl Eigentümer als auch Gruppe ändern kann dies in einem Rutsch mit `chown <newuser>:<newgroup> <file>` gemacht werden.

In Kombination mit Eigentümer und Gruppe wird der Zugriff auf eine Datei zusätzlich über die Dateirechte definiert. Es können drei mögliche Berechtigungen vergeben werden: lesen/read, schreiben/write und ausführen/execute. Sie werden durch einzelne Zeichen repräsentiert: r für read, w für write und x für execute. Diese Berechtigungen haben bei Dateien und Verzeichnissen unterschiedliche Bedeutungen. Tabelle 1.1 auf Seite 2 vergleicht die verschiedenen Bedeutungen.

Unix Dateirechte		
Dateirecht	Bedeutung für eine Datei	Bedeutung für ein Verzeichnis
read	Inhalt anzeigen	Inhalt anzeigen (mit `ls`)
write	Editieren und speichern	Dateien erstellen, löschen, umbenennen usw.
execute	ausführen	In das Verzeichnis wechseln (`cd`)

Tabelle 1.1: Unix Dateirechte

Am Anfang dieses Abschnitts haben wir schon gelernt, dass die ersten 10 Zeichen der Ausgabe von `ls -l` Dateityp und Berechtigungen anzeigen. Das erste Zeichen, in unserem Fall ein „-“, verrät uns den Dateityp (Die verschiedenen Dateitypen werden im nächsten Abschnitt genauer erklaert.). Die folgenden neun Zeichen stellen die eigentlichen Dateiberechtigungen dar und muessen in drei Gruppen zu je drei Zeichen für die drei möglichen Berechtigungen gelesen werden. Die ersten drei Zeichen symbolisieren die Berechtigungen für den Eigentümer/user (`u`) der Datei, die zweite Zeichengruppe legt die Berechtigung für die Gruppenmitglieder/group (`g`) der Gruppe zu der die Datei gehoert fest und die letzten drei Zeichen gelten für alle anderen User/other (`o`). Ich versuche das mal in einem quick and dirty Diagramm aufzuzeigen bis ich ein hübscheres gemalt habe:

```
-rw-r--r--
|\_/\_/\_/
| |  |  |__ Berechtigungen fuer "other"
| |  |_____ Berechtigungen fuer "group"
| |________ Berechtigungen fuer "owner"
|__________ Dateityp
```

Für unsere Datei `file1` heißt das:

- `rw-` der Eigentümer darf die Datei lesen, schreiben aber nicht ausführen
- `r--` die Gruppenmitglieder dürfen die Datei lesen aber weder schreiben noch sie ausführen
- `r--` alle anderen dürfen die Datei lesen aber weder schreiben noch ausführen

Diese Berechtigungen können mittels `chmod` geändert werden. Welche Berechtigungen letztendlich gesetzt werden wird über eine Kombination aus Benutzergruppe, Operator und Dateirecht definiert. Als Benutzergruppen kommen `u`, `g`, `o` und `a` (all, umfasst alle drei Gruppen) in Frage. Gültige Operatoren sind `+` oder `-` zum hinzufuegen bzw. löschen eines Dateirechts und `=` zum setzen exakter Rechte. Mögliche Dateirechte sind natürlich `r`, `w` und `x`. Mit ein paar Beispielen wird das schnell klarer. Um allen (`a`) Usern Leserechte (`r`) für `file1` zu geben (`+`) nutzt man

```
# chmod a+r file1
```

Will man der Gruppe (`g`) und allen anderen Usern (`o`) die Leserechte für `file1` entziehen (`-`) startet man `chmod` wie folgt

```
# chmod go-r file1
```

Für Verzeichnisse ändert man die Dateiberechtigungen genau so, man muss nur die geänderte Bedeutung im Vergleich zu normalen Dateien beachten (vergleiche Tabelle 1.1).

Weil das so alles viel zu einfach wär, können all diese Dateirechte nicht nur „symbolisch“ sondern auch „numerisch“ (oktal) vergeben werden. Dies erleichtert das setzen von absoluten Berechtigungen über einen `chmod` Aufruf. Den einzelnen Berechtigungen r, w und x werden dazu die Werte 4, 2 und 1 zugeodnet und schlicht addiert. So erhält man eine Zahl `n`. Für jedes Tripel rwx gibt es dann nur noch eine Summe der zugehörigen Werte, einen Wert `n`. `chmod` wird dann als `chmod` ***`nnnn datei`*** aufgerufen (das erste `n` ist optional und setzt Sonderattribute). Ein `chmod 764 file1` setzte demnach volle Rechte für den Eigentümer (7 = 4 (read) + 2 (write) + 1 (execute)), für die Gruppe nur Lese- und Schreibrechte (6 = 4 (read) + 2 (write)). Alle anderen dürfen nur lesen (4 (read)). Die Notation `764` entspricht also `rwxrw-r--`, `531` entspricht `r-x-wx--x`, `740` entspricht `rwxr-----` usw.. Tabelle 1.2 fasst alle möglichen Werte für `n` zusammen.

`n`	Rechte	Zusammensetzung
`7`	read, write und execute	4 + 2 + 1
`6`	read und write	4 + 2
`5`	read und execute	4 + 1
`4`	read	4
`3`	write und execute	2 + 1
`2`	write	2
`1`	execute	1
`0`	nüscht	

Tabelle 1.2: Mögliche Werte für `n`.

Selbst damit haben wir das Ende der Fahnenstange noch nicht erreicht. Es gibt noch etliche Sonderattribute wie setuid, setgid, sticky bit und file locking, aber die werden erst in einer späteren Version des Skripts besprochen. „Mut zur Lücke!“[1]

[1] Klaus Koch, Deutschlehrer und Weinkenner, während einer Klassenarbeit 1992

1.2 Dateien und Dateitypen

Unter Unix kann man fast immer von „eveything is a file“ ausgehen. Das bedeutet, das nicht nur „normale Daten“ wie Texte oder Bilder als Datei, File, behandelt werden, sondern auch Verzeichnisse, Laufwerke, Drucker oder Mittel zur Kommunikation zwischen Prozessen (Interprocess, IPC). Um welche Art File es sich handelt erkennt man anhand des ersten Zeichens der Ausgabe von `ls -l` bzw. kann man es mit dem Befehl `file` abfragen.

```
# ls -l
total 4
-rw-r--r-- 2 root root    0 2008-11-10 10:42 file1
drwxr-xr-x 2 root root 4096 2008-11-10 10:43 subdir1
lrwxrwxrwx 1 root root    5 2008-11-10 10:42 symlink1 -> file1
# file symlink1
symlink1: symbolic link to ‘file1’
```

Tabelle 1.3 listet einige der möglichen Dateitypen und ihre Darstellung im `ls` auf. Die wichtigsten Typen werden im Folgenden kurz besprochen.

Jede Datei hat einen Namen und einen Inode in dem das System Informationen über die Datei speichert. Diese Inodes sind durchnumeriert und werden beim Anlegen des Dateisystems erzeugt. Innerhalb eines Inodes wird auf die Datenblöcke auf dem Laufwerk in denen der eigentliche Dateiinhalt geschreichert ist verwiesen. Ein Dateisystem kann nur so viele Dateien haben, wie es Inodes gibt. Dies muss z.B. bei Proxyservern mit vielen kleinen Dateien bedacht werden, da sonst der Plattenspeicher beispielsweise nur zu 20% belegt ist, aber aufgrund der aufgebrauchten Inodes keine weiteren Dateien angelegt werden und der Dienst seine Arbeit einstellt.

Unix Dateitypen		
Dateityp	**`ls -l`**	**Erzeugt mit**
normale Datei	`-`	verschiedene Möglichkeiten (`touch, vi,` Kompiler usw.)
Verzeichnis	`d`	`mkdir`
symbolischer Link	`l`	`ln -s`
Gerätedatei	`b` bzw. `c`	`mknod, dfsadm,`, automatisch
Sockets	`s`	kein Befehl
FIFO	`p`	`mkfifo`

Tabelle 1.3: Unix Dateitypen

1.2.1 normale Dateien

Bei den normalen Dateien handelt es sich um den gängigsten Dateityp. Mit ihm werden wir es meistens zu tun haben. Dabei kann es sich um ein Textfile, ein Bild, ein Programm, eine Konfigurationsdatei für einen Dienst, ein Logfile oder oder oder handeln.

1.2.2 Verzeichnisse

Während eine normale Datei alle möglichen Daten beinhalten kann, findet man in einem Verzeichnis (hier bitte vom Verzeichnis als Datei im Sinn von „everything is a file“ denken) nur einen Datentyp. Ein Verzeichnis ist letztendlich eine kleine Liste der Dateien mit zugehörigen Inode-Nummern, die in ihm/unter ihm zu finden sind. In einem Verzeichnis sind also direkt keine Dateien gespeichert.

1.2.3 Links

Links begegnen uns meisetns in Form von symbolischen Links. Ein symbolischer Link ist ein Verweis auf eine andere Datei. Dieser Verweis kann als absoluter oder relativer Pfad angegeben werden. Da sie mit Pfaden arbeiten, funktionieren symbolische Links auch über Partitions/Laufwerksgrenzen hinweg. Sie teilen sich ihren Inode nicht mit der Datei auf die sie verweisen. Bei Hardlinks sieht das anders aus. Jede Datei verfügt ueber mindestens einen Hardlink, der Verknüpfung zwischen Dateiname und Inode. Jeder zusätzliche mit `ln` angelegte Hardlink teilt sich den Inode mit der Quelldatei. Daher sind Hardlinks auch auf Partitionsgrenzen beschränkt.

Wenn wir `ls` zusaetzlich noch mit dem Schalter `-i` aufrufen, werden auch die Inodenummern der Dateien ausgegeben.

```
# ls -li
total 0
49479 -rw-r--r-- 2 drizzt drizzt 2616 2008-11-10 10:42 file1
49482 -rw-r--r-- 1 drizzt drizzt   24 2008-11-10 11:59 file2
49479 -rw-r--r-- 2 drizzt drizzt 2616 2008-11-10 10:42 hardlink1_file1
49480 lrwxrwxrwx 1 drizzt drizzt    5 2008-11-10 10:42 symlink1 -> file1
```

Hier kann man sehen, das sich **file1** und `hardlink1_file1` den selben Inode mit der Nummer 49479 teilen, während der symbolische Link `symlink1` einen eigenen Inode Besitzt und nur auf `file1` verweist. Daher wirken sich Änderungen z.B. an den Dateirechten einer Datei auch 1:1 auf alle Hardlinks dieser Datei aus. Symbolische Links sind davon nicht betroffen. Wir sehen auch, dass der Link-Zähler bei **file1** und `hardlink1_file1` um 1 hochgezählt wurde und die Dateigrösse natürlich identisch ist, während die Dateigroesse des symbolischen Links der Anzahl der Zeichen des Verweises ist.

1.2.4 Gerätedateien

Gerätedateien stellen Schnittstellen zur Hardware dar. Alle auf diese Gerätedateien angewandten Operationen werden and das entsprechende Gerät weitergeleitet. So kann man z.B. die Ausgabe eines `cat`-Aufrufs direkt an die Gerätedatei eines Druckers umleiten und so die Ausgabe drucken. Inodes von Gerätedateien verweisen nicht auf Datenblöcke, sondern enthalten Informationen die auf das Device verweisen. Da wo man im `ls` bei anderen Dateitypen die Dateigrösse angezeigt bekommt, befinden sich bei Garätedateien zwei Zahlen, die major und minor device number.

```
# ls -l hda*
total 0
brw-rw---- 1 root disk 80, 0 2008-10-24 08:20 hda
brw-rw---- 1 root disk 80, 1 2008-10-24 08:21 hda1
brw-rw---- 1 root disk 80, 2 2008-10-24 08:20 hda2
brw-rw---- 1 root disk 80, 3 2008-10-24 08:21 hda3
brw-rw---- 1 root disk 80, 4 2008-10-24 08:20 hda4
brw-rw---- 1 root disk 80, 5 2008-10-24 08:21 hda5
brw-rw---- 1 root disk 80, 6 2008-10-24 08:21 hda6
```

Dieses Beispiel zeigt uns die Garätedateien der ersten Festplatte in System und ihrer Partitionen. Die erste Zahl, `80`, ist die major device number und legt den Gerätetyp und zu verwendende Treiber fest. Die Zweite Zahl, die minor device number, ist die Gerätenummer und läßt so eine Unterscheidung mehrerer verwandter oder ähnlicher Geräte zu.

1.2.5 FIFOs und Sockets

FIFOs heissen eigentlich named pipes und stellen eine Erweiterung des Pipelining Konzepts („unnamed/anonymous pipe“) dar, welches wir in Kapitel 3 kennengelernt haben. Sie dienen der Interprozesskommunikation (IPC) und können bei Bedarf auch per Hand angelegt werden.

Sockets (auch Unix Domain Socket oder IPC Socket) kommen ebenfalls bei IPC zum tragen. Sie arbeiten, grob gesagt, ähnlich wie Netzwerkverbindungen, Network Sockets, aber die Daten verlassen den Host dabei nie.

1.3 Verzeichnisstruktur

Um einen Überblick über die vielen tausend Dateien eines Unix Betriebssystems zu behalten, benötigt man ein Grundgerüst an dem sich die Entwickler orientieren und die Verzeichnisstruktur anpassen können. Bei vielen Betriebssystemen der Unix Familie ähnelt sich diese Struktur sehr stark , da sie sich alle am File System Hierarchy Standard[2] (FHS) orientieren.

Logisch betrachtet liegen alle Verzeichnisse unterhalb des sogenannten Wurzelverzeichnisses `root` (\). All diese Unterverzeichnisse können physikalisch auf ein und derselben Partition liegen, oder ab über mehrere Partitionen, Festplatten oder gar über das Netzwerk verteilt sein.

Tabelle 1.4 auf Seite 7 fasst die wichtigsten Verzeichnisse kurz zusammen. Für eine genauere Betrachtung, muss man sich mit der Dokumentation des betreffenden Systems auseinandersetzen.

[2] http://www.pathname.com/fhs/

Verzeichnisstruktur unterhalb von \	
Verzeichnis	**Inhalt**
bin	Wichtige (System)programme die feru den Bootprozess und im Singleusermode verfügbar sein müssen. Unter Solaris ist /bin ein symbolischer Link auf /usr/bin
boot	unter Linux liegen hier die zum Booten benötigten Daten wie Konfigurationen für den Bootloader und der Kernel
dev	Hier liegen alle Gerätedateien. Die Verteilung auf die Unterverzeichnisse von /dev unterscheidet sich zwischen den Verschiedenen Systemen.
etc	Systemspezifische Konfigurationsdateien.
home	Homeverzeichnisse der einzelnen User.
mnt	Temporäre Mountpoints
opt	Optionale Softwarepakete
proc	Pseudofilesystem mit System- und Prozessinformationen
root	Heimatverzeichnis des Superusers root
sbin	Software zur Systemadministration und beim Booten benötigte Programme.
tmp	Platz für temporäre Dateien. Unter Solaris ist /tmp eine Ramdisk und hier gespeicherte Dateien überleben einen Reboot nicht.
usr	Die meisten systemunkritischen Anwendungen und Bibliotheken sowie die man-Pages liegen hier. Nachträgliche Softwareinstallationen landen oft in `/usr/local`. `usr` wird heute oft als Akronym fuer „UNIX System Resources“ gesehen und sollte menn möglicht read-only gemountet sein.
var	Hier liegen variable, sich häufig ändernde Dateien wie Logfiles, Mailqueues, Proxy Caches usw..

Tabelle 1.4: Unix Verzeichnisstruktur

1.4 Arbeiten mit Dateien

Nun werden wir uns einige Befehle ansehen, mit denen wir im normalen Administrationsalltag regelmässig Dateien modifizieren, bearbeiten oder sonstwie verwursten.

1.4.1 `mv` - Dateien verschieben und umbenennen

Mit `mv` können Daten verschoben und umbenannt werden. Dabei muß ebenso wie bei `rm` darauf geachtet werden, daß man nicht aus versehen wichtige Daten überschreibt. `mv` warnt den Benutzer nicht bevor Daten überschrieben werden! Um eine Warnung zu erzwingen muß die Option `-i` benutzt werden.

Um Daten zu verschieben ruft man `mv` auf und übergibt ihm zuerst den oder die Namen der Daten die verschoben werden sollen gefolgt vom Ziel.

```
# ls -l
total 7142
-rwxr--r--   1 drizzt   drizzt    999424 Sep  5 16:38 ausfuehrbar
-rw-r--r--   1 drizzt   drizzt   2637824 Sep  5 17:13 move_me.txt
lrwxrwxrwx   1 drizzt   drizzt        12 Sep  2 13:04 symlink -> /home/drizzt
drwxr-xr-x   3 drizzt   drizzt       512 Sep  5 16:35 testdir
-rw-r--r--   1 drizzt   drizzt        17 Aug 24 22:40 testfile
# mv move_me.txt testdir/

# ls -l testdir/
total 5170
-rw-r--r--   1 drizzt   drizzt   2637824 Sep  5 17:13 move_me.txt
drwxr-xr-x   3 drizzt   drizzt       512 Aug 25 08:29 testdir2
```

Es können auch mehrere Dateien verschoben werden.

```
# mv move_me.txt move_me_2.txt move_me_3.txt testdir/
# ls -l testdir/
total 9074
-rw-r--r--   1 drizzt   drizzt   2637824 Sep  5 17:13 move_me.txt
-rw-r--r--   1 drizzt   drizzt    991232 Sep  5 17:14 move_me_2.txt
-rw-r--r--   1 drizzt   drizzt    991232 Sep  5 17:14 move_me_3.txt
drwxr-xr-x   3 drizzt   drizzt       512 Aug 25 08:29 testdir2
```

Zum umbenennen von Dateien und Verzeichnissen gibt man einfach den alten und den gewünschten neuen Namen an.

```
# ls -l
total 1990
-rwxr--r--   1 drizzt   drizzt    999424 Sep  5 16:38 ausfuehrbar
-rw-r--r--   1 drizzt   drizzt      8192 Sep  5 17:22 rename
lrwxrwxrwx   1 drizzt   drizzt        12 Sep  2 13:04 symlink -> /home/drizzt
drwxr-xr-x   3 drizzt   drizzt       512 Sep  5 17:22 testdir
-rw-r--r--   1 drizzt   drizzt        17 Aug 24 22:40 testfile

# mv rename newname
# ls -l
total 1990
-rwxr--r--   1 drizzt   drizzt    999424 Sep  5 16:38 ausfuehrbar
-rw-r--r--   1 drizzt   drizzt      8192 Sep  5 17:22 newname
lrwxrwxrwx   1 drizzt   drizzt        12 Sep  2 13:04 symlink -> /home/drizzt
drwxr-xr-x   3 drizzt   drizzt       512 Sep  5 17:22 testdir
-rw-r--r--   1 drizzt   drizzt        17 Aug 24 22:40 testfile

# mv newname testdir/newername
# ls -l testdir/
total 18
-rw-r--r--   1 drizzt   drizzt      8192 Sep  5 17:22 newername
drwxr-xr-x   3 drizzt   drizzt       512 Aug 25 08:29 testdir2
```

1.4.2 cp - Dateien kopieren

Man kann seine Daten natürlich nicht nur verschieben, sondern auch kopieren. Dazu dient der Befehl `cp`. Die Syntax ist dabei der von `mv` sehr ähnlich.

Um eine Datei zu kopieren übergibt man `cp` einfach den Namen der Originaldatei und den der Kopie.

```
# ls -l
total 1974
-rwxr--r--   1 drizzt   drizzt    999424 Sep  5 16:38 ausfuehrbar
lrwxrwxrwx   1 drizzt   drizzt        12 Sep  2 13:04 symlink -> /home/drizzt
drwxr-xr-x   3 drizzt   drizzt       512 Sep  5 17:23 testdir
-rw-r--r--   1 drizzt   drizzt        17 Aug 24 22:40 testfile
# cp testfile testfile_2
# ls -l
total 1976
-rwxr--r--   1 drizzt   drizzt    999424 Sep  5 16:38 ausfuehrbar
lrwxrwxrwx   1 drizzt   drizzt        12 Sep  2 13:04 symlink -> /home/drizzt
drwxr-xr-x   3 drizzt   drizzt       512 Sep  5 17:23 testdir
-rw-r--r--   1 drizzt   drizzt        17 Aug 24 22:40 testfile
-rw-r--r--   1 drizzt   drizzt        17 Sep  6 10:20 testfile_2
```

Ebenso wie bei `mv` kann man durch die Option `-i` ein ungewolltes Überschreiben von bereits vorhandenen Daten verhindern. Um Verzeichnisbäume zu kopieren muß die Option `-r` benutzt werden.

Es können natürlich auch mehrere Dateien auf einmal kopiert werden.

```
# cp testfile testfile_2 ausfuehrbar testdir/
# ls -l testdir/
total 1990
-rwxr--r--   1 drizzt   drizzt    999424 Sep  6 10:26 ausfuehrbar
-rw-r--r--   1 drizzt   drizzt      8192 Sep  5 17:22 newername
drwxr-xr-x   3 drizzt   drizzt       512 Aug 25 08:29 testdir2
-rw-r--r--   1 drizzt   drizzt        17 Sep  6 10:26 testfile
-rw-r--r--   1 drizzt   drizzt        17 Sep  6 10:26 testfile_2
```

1.4.3 rm & rmdir - Dateien und Verzeichnisse löschen

Den Befehl `rm` sollte man immer mit Bedacht einsetzen! **Gelöschte Daten sind gelöscht und bleiben gelöscht!** Es gibt unter Unix kein 'undelete' und keinen 'Papierkorb' wie es einige von anderen Betriebssystemen gewohnt sind.

Wird `rm` ohne weitere Optionen benutzt werden die angegebenen Dateien ohne Rückfrage gelöscht, so es die gesetzten Permissions erlauben.

```
# ls -l
total 9846
-rwxr--r--   1 drizzt   drizzt    999424 Sep  5 16:38 ausfuehrbar
-rw-r--r--   1 drizzt   drizzt   1351680 Sep  5 16:42 loeschen_1.txt
-rw-r--r--   1 drizzt   drizzt   1196032 Sep  5 16:42 loeschen_2.txt
-rw-r--r--   1 drizzt   drizzt    450560 Sep  5 16:42 loeschen_3.txt
```

```
-rw-r--r--   1 drizzt   drizzt    999424 Sep  5 16:42 loeschen_4.txt
lrwxrwxrwx   1 drizzt   drizzt        12 Sep  2 13:04 symlink -> /home/drizzt
drwxr-xr-x   3 drizzt   drizzt       512 Sep  5 16:35 testdir
-rw-r--r--   1 drizzt   drizzt        17 Aug 24 22:40 testfile
# rm loeschen_*
# ls -l
total 1974
-rwxr--r--   1 drizzt   drizzt    999424 Sep  5 16:38 ausfuehrbar
lrwxrwxrwx   1 drizzt   drizzt        12 Sep  2 13:04 symlink -> /home/drizzt
drwxr-xr-x   3 drizzt   drizzt       512 Sep  5 16:35 testdir
-rw-r--r--   1 drizzt   drizzt        17 Aug 24 22:40 testfile
```

Möchte man, daß sich das System den Löschvorgang der betreffenden Datei nocheinmal explizit vom Benutzer bestätigen läßt, muß man mit der Option `-i` arbeiten.

```
# ls -l
total 9846
-rwxr--r--   1 drizzt   drizzt    999424 Sep  5 16:38 ausfuehrbar
-rw-r--r--   1 drizzt   drizzt   1351680 Sep  5 16:45 loeschen_1.txt
-rw-r--r--   1 drizzt   drizzt   1196032 Sep  5 16:45 loeschen_2.txt
-rw-r--r--   1 drizzt   drizzt    450560 Sep  5 16:45 loeschen_3.txt
-rw-r--r--   1 drizzt   drizzt    999424 Sep  5 16:45 loeschen_4.txt
lrwxrwxrwx   1 drizzt   drizzt        12 Sep  2 13:04 symlink -> /home/drizzt
drwxr-xr-x   3 drizzt   drizzt       512 Sep  5 16:35 testdir
-rw-r--r--   1 drizzt   drizzt        17 Aug 24 22:40 testfile

# rm -i loeschen_*
rm: remove loeschen_1.txt (yes/no)? y
rm: remove loeschen_2.txt (yes/no)? y
rm: remove loeschen_3.txt (yes/no)? y
rm: remove loeschen_4.txt (yes/no)? n

# ls -l
total 3942
-rwxr--r--   1 drizzt   drizzt    999424 Sep  5 16:38 ausfuehrbar
-rw-r--r--   1 drizzt   drizzt    999424 Sep  5 16:45 loeschen_4.txt
lrwxrwxrwx   1 drizzt   drizzt        12 Sep  2 13:04 symlink -> /home/drizzt
drwxr-xr-x   3 drizzt   drizzt       512 Sep  5 16:35 testdir
-rw-r--r--   1 drizzt   drizzt        17 Aug 24 22:40 testfile
```

Um schreibgeschützte Dateien zu löschen benutzt man entweder `rm` ohne weitere Optionen und bestätigt dann den Löschvorgang für jeder schreibgeschützte Datei oder man verwendet die Option `-f` (force).

```
# ls -l
total 3942
-rwxr--r--   1 drizzt   drizzt    999424 Sep  5 16:38 ausfuehrbar
-r--r--r--   1 drizzt   drizzt    999424 Sep  5 16:45 schreibgeschuetzt.txt
lrwxrwxrwx   1 drizzt   drizzt        12 Sep  2 13:04 symlink -> /home/drizzt
drwxr-xr-x   3 drizzt   drizzt       512 Sep  5 16:35 testdir
-rw-r--r--   1 drizzt   drizzt        17 Aug 24 22:40 testfile
# rm schreibgeschuetzt.txt
```

```
rm: schreibgeschuetzt.txt: override protection 444 (yes/no)? n

# rm -f schreibgeschuetzt.txt
# ls -l
total 1974
-rwxr--r--   1 drizzt   drizzt    999424 Sep  5 16:38 ausfuehrbar
lrwxrwxrwx   1 drizzt   drizzt        12 Sep  2 13:04 symlink -> /home/drizzt
drwxr-xr-x   3 drizzt   drizzt       512 Sep  5 16:35 testdir
-rw-r--r--   1 drizzt   drizzt        17 Aug 24 22:40 testfile
```

Verzeichnisse können entweder mit `rm` oder mit `rmdir` gelöscht werden. Um ein Verzeichnis mit `rmdir` löschen zu können, muss das betreffende Verzeichnis leer sein.

```
# rmdir removedir/
rmdir: directory "removedir/": Directory not empty
# ls -l removedir/
total 1600
-rw-r--r--   1 drizzt   drizzt    811008 Sep  5 16:52 nicht_leer
```

Man kann jetzt entweder alle Dateien und Unterverzeichnisse einzeln löschen oder das Verzeichnis mittels `rm` und den Optionen `-f` (force) und `-r` (recursive) auf einen Rutsch löschen.

```
# rm -rf removedir/
# cd removedir
bash: cd: removedir: No such file or directory
```

Vor dem Bestätigen des Befehls aber immer noch mal gründlich nachdenken damit keine wichtigen Daten verloren gehen und kontrollieren, daß man auch im richtigen Verzeichnis ist! Das gilt besonders für so nette Aktionen wie `rm -rf ..`. Dadurch würde `rm` in das Übergeordnete Verzeichnis und von da aus rekursiv alles löschen. Als User `root` z.B. im Verzeichnis `/bin` ausgeführt würde das fatal enden.

Ich habe Leute erlebt, die `rm -rf` vorschnell bestätigt haben und sich anschließend über etliche Gigabyte an neuem freien Festplattenplatz freuen konnten.[3]

1.4.4 `mkdir` - Verzeichnisse erstellen

Mit `mkdir` werden neue Verzeichnisse erstellt:

```
# cd testdir
bash: cd: testdir: No such file or directory

# mkdir testdir
# cd testdir
```

[3] Die betreffende Person glaubte mittels `rm -rf /mnt` die gemountete Windows NT Partition unmounten zu können. Daß bei ihm die Verwunderung über das bootunwillige NT und bei uns das Gelächter groß war kann sich jetzt wohl jeder denken :o)

Durch den Schalter `-p` werden auch eventuell nicht vorhandene übergeordnete Verzeichnisse angelegt:

```
# cd testdir/testdir2/testdir3
bash: cd: testdir/testdir2/testdir3: No such file or directory
# cd testdir/testdir2
bash: cd: testdir/testdir2: No such file or directory

# mkdir -p testdir/testdir2/testdir3
# cd testdir/testdir2/testdir3
# pwd
/export/home/drizzt/Unix/testdir/testdir2/testdir3
```

1.4.5 touch - Dateien „anfassen“

Der Befehl `touch` wird immer mit einer bzw. mehreren Dateinamen als Argument aufgerufen. Existiert keine Datei mit dem angegebenen Namen, wird automatisch eine leere Datei angelegt:

```
# ls -l
total 0
# touch testfile
# ls -l
total 0
-rw-r--r--   1 drizzt   drizzt         0 Aug 24 21:15 testfile
```

Das Anlegen der Datei kann mit dem Schalter `-c` verhindert werden:

```
# ls -l
total 0
-rw-r--r--   1 drizzt   drizzt         0 Aug 24 21:15 testfile
# touch -c testfile2
# ls -l
total 0
-rw-r--r--   1 drizzt   drizzt         0 Aug 24 21:15 testfile
```

Existiert die angegebene Datei bereits, kann man mittels `touch` die Zeitstempel des letzten Zugriffs und der letzten Bearbeitung aktualisieren bzw. gezielt neu setzen:

```
# ls -lc
total 0
-rw-r--r--   1 drizzt   drizzt         0 Aug 24 21:15 testfile
# date
Sat Aug 24 22:06:03 MEST 2002
# touch testfile
# ls -lc
total 0
-rw-r--r--   1 drizzt   drizzt         0 Aug 24 22:05 testfile
```

Durch den Schalter `-a` läßt sich gezielt der Zeitstempel des letzten Zugriffs und mittels `-m` der letzten Änderung setzen. Will man statt dem Aktuellen Datum und der aktuellen Zeit eigene Daten verwenden, kann man sie in Verbindung mit dem Schalter `-t [[CC]YY]MMDDhhmm.ss` setzen.

1.4.6 find - Dateien suchen

`find` gehört mit zu den mächtigsten[4][5] Befehlen unter Unix. Es ist auf jeden Fall ratsam sich die man-Page zu `find` gründlich durchzulesen.

Die `find` Syntax unterscheidet sich etwas von den Befehlen, die wir bisher angesprochen haben.

`find` *verzeichnis expression*

verzeichnis gibt an wo die Suche gestartet werden soll. `find` durchsucht dabei auch alle Unterverzeichnisse von *verzeichnis*.

Eine einfache Suche nach einer Datei mit dem Namen `testfile` beginnend in meinem Homedir — welches in diesem Beispiel auch das aktuelle Arbeitsverzeichnis ist — würde folgendermaßen aussehen:

```
# find . -name testfile
./Unix/testfile
```

Der '`.`' steht für das aktuelle Verzeichnis und bei `-name testfile` handelt es sich um die *expression*. `find` versteht die normalen Metazeichen der Shell, allerdings müssen sie in Verwendung mit `find` durch Quoting vor der Shell geschützt werden. Bei einem `find . -name test*` in unserem Beispielverzeichnis würde der Befehl zu `find . -name testdir/ testfile testfile_2` expandiert werden und wir eine entsprechende Fehlermeldung bekommen. Das `test*` muß also gequotet werden:

```
# find . -name test*
find: bad option testfile
find: path-list predicate-list
# find . -name 'test*'
./testfile
./testdir
./testdir/testdir2
./testdir/testdir2/testdir3
./testfile_2
#
```

Wenn man `find` noch ein `-ls` übergibt werden nicht nur die Namen der gefundenen Dateien ausgegeben, sondern auch andere Attribute — ähnlich einem `ls -l`.

```
# find . -name testfile -ls
435368    1 -rw-r--r--   1 drizzt   drizzt         17 Aug 24 22:40 ./Unix/testfile
```

Dabei handelt es sich um die *Inode Nummer*, die *Gößе in Kilobyte*, die *Dateirechte*, die *Anzahl der Hard Links*, die *Namen von Dateiinhaber und Gruppe*, die *Größе in Byte* und den *Zeitstempel der letzten Modifikation.*

[4] Dieses „mächtig“ widme ich Herrn Jürgen Küffner. Hallo Jürgen, das hier ist für Dich.

[5] Diese Fussnote habe ich im Jahr 2002 geschrieben, und heute, 2008, hab ich nicht die blasseste Ahnung warum...

Zusätzlich zum Datei-/Verzeichnisnamen kann man die Suche auch nach weiteren Kriterien einschränken. Unter anderem ist es möglich `find` nur nach bestimmten Dateitypen suchen zu lassen. Dazu wird die Expression `-type n` verwendet, wobei *n* u.a. die Werte `f` für plain File, `d` für Verzeichnisse und `l` für symbolische Links zugewiesen werden. Um z.B. vom aktuellen Verzeichnis ausgehend nach allen normalen Dateien zu suchen nimmt man `find . -type f`

```
# ls -l
total 1976
-rwxr--r--   1 drizzt   drizzt    999424 Sep  5 16:38 ausfuehrbar
lrwxrwxrwx   1 drizzt   drizzt        12 Sep  2 13:04 symlink -> /home/drizzt
drwxr-xr-x   3 drizzt   drizzt       512 Sep  6 10:28 testdir
-rw-r--r--   1 drizzt   drizzt        17 Aug 24 22:40 testfile
-rw-r--r--   1 drizzt   drizzt        17 Sep  6 10:20 testfile_2
# find . -type f
./testfile
./testdir/testdir2/testdir3/blafasel
./testdir/newername
./.versteckt
./ausfuehrbar
./testfile_2
#
```

Da `find` normalerweise alle Unterverzeichnisse durchsucht werden hier in diesem Beispiel auch entsprechende Dateien in `./testdir` bzw. `./testdir/testdir2/testdir3/` gefunden.

Die Suche nach Dateien/Verzeichnissen mit bestimmten Rechten ist mittels der Expression `-perm [-]mode` möglich. Das - ist dabei optional und besagt, daß die betreffenden Dateien mindestens die in `mode` angegebenen Rechte besitzen müssen. Andernfalls wird nach Dateien/Verzeichnissen mit genau den in `mode` angegebenen Rechten gesucht. Um beispielsweise alle Dateien zu finden, die mindestens `rwx` für den Eigentümer und `r-x` für die Gruppe haben wählt man den Befehl `find . -perm -750`

```
# find . -perm -750
.
./testdir
./testdir/testdir2
./testdir/testdir2/testdir3
./symlink
# ls -l
total 1976
-rwxr--r--   1 drizzt   drizzt    999424 Sep  5 16:38 ausfuehrbar
lrwxrwxrwx   1 drizzt   drizzt        12 Sep  2 13:04 symlink -> /home/drizzt
drwxr-xr-x   3 drizzt   drizzt       512 Sep  6 10:28 testdir
-rw-r--r--   1 drizzt   drizzt        17 Aug 24 22:40 testfile
-rw-r--r--   1 drizzt   drizzt        17 Sep  6 10:20 testfile_2
#
```

Natürlich können die diversen Expressions auch kombiniert werden. Im folgenden Beispiel wird nach regulären Dateien (`-type f`) die der Gruppe `drizzt` (`-group drizzt`)

gehören und mindestens die Rechte `r-xr--r--` (`-perm -544`) haben gesucht. Über die gefundenen Dateien hätten wir gerne mehr Informationen als normal (`-ls`):

```
# find . -type f -group drizzt -perm -544 -ls
435395  984 -rwxr--r--   1 drizzt   drizzt     999424 Sep  5 16:38 ./ausfuehrbar
#
```

Außerdem ist es möglich `find` zu sagen, was es mit allen gefundenen Dateien machen soll. Dazu wird folgende Syntax verwendet:

```
find verzeichnis expression -exec kommando {} \;
```

bzw.

```
find verzeichnis expression -ok kommando {} \;
```

Dabei steht {} als Platzhalter für den Pfad-/Dateinamen und wird entsprechend ersetzt. Das Ende von `kommando` muß durch `\;` markiert werden. Der Unterschied zwischen `-exec` und `-ok` besteht darin, daß bei `-exec` der Befehl `kommando` sofort ausgeführt wird und bei `-ok` für jede gefundene Datei die Ausführung von `kommando` erst vom Benutzer durch ein `y` bestätigt werden muß. Beispiel zur Expression `-exec`:

```
# ls -l testdir/
total 2
drwxr-xr-x   3 drizzt   drizzt       512 Aug 25 08:29 testdir2
# ls -l
total 1976
-rwxr--r--   1 drizzt   drizzt    999424 Sep  5 16:38 ausfuehrbar
lrwxrwxrwx   1 drizzt   drizzt        12 Sep  2 13:04 symlink -> /home/drizzt
drwxr-xr-x   3 drizzt   drizzt       512 Dec 15 16:38 testdir
-rw-r--r--   1 drizzt   drizzt        17 Aug 24 22:40 testfile
-rw-r--r--   1 drizzt   drizzt        17 Sep  6 10:20 testfile_2
# find . -type f -perm -700 -exec cp {} testdir/ \;
# ls -l testdir/
total 1970
-rwxr--r--   1 drizzt   drizzt    999424 Dec 15 16:40 ausfuehrbar
drwxr-xr-x   3 drizzt   drizzt       512 Aug 25 08:29 testdir2
#
```

Der Befehl `find . -type f -perm -700 -exec cp {} testdir/ \;` sucht nach allen normalen Dateien bei denen mindestens die Rechte für den Besitzer bei `rwx` liegen und kopiert diese dann und das Unterverzeichnis `testdir/`. Im aktuellen Verzeichnis trifft das nur auf die Datei `ausfuehrbar` zu, die auch entsprechend nach `testdir/` umkopiert wird.

Nun lassen wir `find` im Verzeichnis `testdir/` mit den gleichen Kriterien nach Dateien suchen (hier wird nur unsere `ausfuehren`-Kopie aus dem vorherigen Beispiel gefunden) und danach interaktiv löschen.

```
# find testdir/ -type f -perm -700 -ok rm {} \;
< rm ... testdir/ausfuehrbar >?   y
# ls -l testdir/
total 2
drwxr-xr-x   3 drizzt   drizzt       512 Aug 25 08:29 testdir2
#
```

Wenn man im `-exec` Teil die gefundenen Dateien mit einer älteren Version von `grep` (siehe nächster Abschnitt) durchsuchen lassen will, sollte man zusätzlich zu {} als Platzhalter fuer den Dateinamen der gefundenen Datei noch `/dev/null` dazusetzen:

`find . -exec grep` *pattern* {} `/dev/null \;`

Das hat einen ganz einfachen Grund: wenn `grep` nur eine Datei zum durchsuchen übergeben wird, gibt er bei einem Match den Dateinamen nicht zusaetzlich aus. Warum auch, er wurde nur in einer Datei gesucht und der dumme Benutzer an der Tastatur wird sich ja wohl merken können wo er eben gesucht hat. Wenn `find` nun aber mehrere passende Dateien findet und `grep` in mehr als einer Datein fündig wird, bekommen wir letztendlich nur eine Liste der Zeilen mit einem Match, wissen aber nicht zu welcher Datei diese gehören. Durch das zusätzliche durchsuchen von `/dev/null` erzwingen wir quasi die Angabe des Dateinamens in dem das Suchpattern gefunden wurde. Bei neueren Versionen von `grep` kann man die Ausgabe des Dateinamens mit der Option `-o` erzwingen und sich den `/dev/null` Teil sparen.

Es gibt noch viele andere praktische Expressions für `find`. Beispielsweise `-inum` zur Suche nach bestimmten Inodes, `-mount` zur Beschränkung der suche auf Filesysteme/Partitionen oder `-mtime` um nach zuletzt geänderten Dateien zu suchen. Näheres zu den etlichen Möglichkeiten findet sich in der entsprechenden Manpage.

1.4.7 grep - Dateiinhalte durchsuchen

Ein weiteres, besonders in Verbindung mit regulären Ausdrücken (regular Expressions)[6], sehr mächtiges Werkzeug ist `grep` bzw. das erweiterte `egrep`. `grep` durchsucht eine/mehrere Datei(en) bzw. alles was ihm vorgeworfen wird (pipelining) nach einem Pattern. Gross- und Kleinschreibung werden dabei beachtet!

```
# grep root /etc/passwd
root:x:0:0:root:/root:/bin/bash
```

Wenn mehrere Dateien zum durchsuchen angegeben werden, nennt `grep` zu jedem Match auf das Pattern noch die Datei in der es fündig wurde. Für einzelne Dateien kann dies je nach `grep` Version auch durch die Option `-o` erzwungen werden.

[6] regExp sind ein grosses aber auch interessantes Thema. Interessenten sei das Eulenbuch „Reguläre Ausdrücke“ von Jeffrey Friedl empohlen. Erschienen bei O'Reilly, ISBN 3897217201. Einen kleinen Überblick bietet auch der entsprechende Wikipedia Artikel `http://de.wikipedia.org/wiki/Regexp`

```
# grep root /etc/passwd /etc/hosts /etc/shadow
/etc/passwd:root:x:0:1:Super-User:/root:/sbin/sh
/etc/shadow:root:c3581516868fb3b71746931cac66390e:12592::::::
```

Das waren nur sehr einfache Beispiele. Oft benötigte Optionen bzw. Switche sind `-i` zum ignorieren von Gross- und Kleinschreibung, `-v` zum suchen nach allem was NICHT dem angegebenen Pattern entspricht oder `-c` zum Zählen der gefundenen Vorkommen. Da nicht alle Optionen zwingend bei jeder `grep` bzw. `egrep` Version verhanden sind lohnt sich wie immer ein Blick in die Manpage. Gerade im Hinblick auf regExp, das `grep` weniger Metazeichen unterstützt als `egrep` und diese Teils auch anders angewandt werden müssen.

1.4.8 du und df - Plattenplatz kontrollieren

Die Befehle du (disk usage) und df (disk free) kommen spätestens dann zum Einsatz, wenn eine Partition im System voll ist. df gibt den noch freien Plattenplatz in Blöcken aus. Linux und Soaris unterscheiden sich in der Darstellung.
Linux:

```
# df
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/i2o/hda1          6538528    713116   5493272  12% /
varrun                 2012824        80   2012744   1% /var/run
varlock                2012824         0   2012824   0% /var/lock
udev                   2012824        48   2012776   1% /dev
devshm                 2012824         0   2012824   0% /dev/shm
/dev/i2o/hda3          4806936   3373116   1189632  74% /home
/dev/i2o/hda5          9614116    535216   8590528   6% /usr
/dev/i2o/hda6          9582284   3363904   5731616  37% /var
```

Solaris

```
# df
/                  (/dev/dsk/c0t0d0s0 ):  277654 blocks   114589 files
/usr               (/dev/dsk/c0t0d0s6 ): 2750422 blocks   407325 files
/proc              (/proc             ):       0 blocks     3833 files
/etc/mnttab        (mnttab            ):       0 blocks        0 files
/dev/fd            (fd                ):       0 blocks        0 files
/var               (/dev/dsk/c0t0d0s1 ): 2188572 blocks   463144 files
/var/run           (swap              ): 2119888 blocks    22812 files
/tmp               (swap              ): 2119888 blocks    22812 files
/export            (/dev/dsk/c0t0d0s7 ):15492714 blocks  1065076 files
/opt               (/dev/dsk/c0t0d0s5 ): 2892910 blocks   488557 files
```

Oft ist es sinnvoller sich die Belegung mittels Switch -h als „human readable form“ anzeigen zu lassen. Da nicht jede df Version unter Solaris diesen Switch kennt, sollte man sich auch -k für die Angabe in Kilobytes merken.

```
# df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/i2o/hda1         6.3G  697M  5.3G  12% /
varrun                2.0G   80K  2.0G   1% /var/run
varlock               2.0G     0  2.0G   0% /var/lock
udev                  2.0G   48K  2.0G   1% /dev
devshm                2.0G     0  2.0G   0% /dev/shm
/dev/i2o/hda3         4.6G  3.3G  1.2G  74% /home
/dev/i2o/hda5         9.2G  523M  8.2G   6% /usr
/dev/i2o/hda6         9.2G  3.3G  5.5G  37% /var
```

bzw.

```
# df -k
Filesystem            kbytes    used   avail capacity  Mounted on
/dev/dsk/c0t0d0s0     240399  101572  114788    47%    /
/dev/dsk/c0t0d0s6    4030518 2655307 1334906    67%    /usr
/proc                      0       0       0     0%    /proc
mnttab                     0       0       0     0%    /etc/mnttab
fd                         0       0       0     0%    /dev/fd
/dev/dsk/c0t0d0s1    1984230  889944 1034760    47%    /var
swap                 1060104     160 1059944     1%    /var/run
swap                 1060008      64 1059944     1%    /tmp
/dev/dsk/c0t0d0s7    9915436 2169084 7647198    23%    /export
/dev/dsk/c0t0d0s5    1984230  537775 1386929    28%    /opt
```

Nachdem man so z.B. die vollen Partitionen identifiziert hat, kann man mit `du` nachsehen wo der meiste Platz verbraucht wird. Auch bei `du` kann man die Ausgabe human readable bzw. in Kilobytes gestalten. Um beispielsweise die Gesamtgrösse der einzelnen Dateien bzw. Verzeichnisse unter `/var` (ohne jedes Unterverzeichnis extra aufzulisten) in Erfahrung zu bringen summieren wir die ermittelten Grössen mit der Option `-s` auf.

```
# du -sh /var/*
4.0M    /var/backups
675M    /var/cache
144M    /var/lib
4.0K    /var/local
0       /var/lock
85M     /var/log
16K     /var/lost+found
36K     /var/mail
4.0K    /var/opt
80K     /var/run
352K    /var/spool
4.0K    /var/tmp
2.2G    /var/www
```

Auf die Weise kann man sich recht einfach bis zum Übeltäter durchhangeln.