Unix/Linux - Kurzskript

Jens Roesen <drizzt@netzsheriff.de>

Würzburg, 17. Juli 2009
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.
Es ist irgendwann mal unter `http://www.netzsheriff.de` erhältlich.

Inhaltsverzeichnis

Vorwort

Motivation

Im Rahmen interner Schulungsmassnahmen kam irgendwann einmal die Frage nach einem Skript für Unix-Neulinge auf. Solche Skripte gibt es massenhaft, aber irgendwie hat mir bei den meisten entweder etwas gefehlt, oder es war für unseren Zweck zu viel. 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 Troubleshooting, Mailserver oder DNS-Server Wert legen, habe ich mich irgendwann eben hingesetzt und mehr oder weniger enthusiastisch und mit teils mehrmonatigen Pausen angefangen dieses Skript zu schreiben. Das geht jetzt schon Jahre so.

Es ist der Versuch Systemadministratoren mit Grundkenntnissen in Unix den Arbeitsalltag zu erleichtern und dabei zwei verschiedene Unix-Geschmacksrichtungen, nämlich Solaris 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 <drizzt@netzsheriff.de> 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.

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 Geschichtliches

"... the number of Unix installations has grown to 10, with more expected ... "

(Dennis Ritchie und Ken Thompson, Juni 1972)

1.1 Aller Anfang ist schwer...

Das mußten auch die Teilnehmer eines sehr ehrgeizigen Projektes erkennen, welches 1965 gemeinsam von den BELL-Laboratories, General Electric und dem MIT (Massachusetts Institute of Technology) angegangen wurde. Man wollte gemeinsam ein Betriebssystem entwickeln das mehreren Benutzern die gleichzeitige Arbeit am System ermöglichen sollte und auch auf zukünftigen, neueren Großrechnern eingesetzt werden konnte. Das Projekt wurde auf den Namen **Multics** (*Multiplexed Information and Computing Service*) getauft.

Leider blieben die erhofften Ergebnisse aus, und BELL zog sich 1969 auch wegen zusätzlicher finanzieller Engpässe aus dem Projekt zurück. Nur ein Gruppe unerschrockener — Ken Thompson, Dennis Ritchie, Doug McIlroy und J. F. Ossanna — arbeitete weiter, in der Hoffnung irgendwann doch noch die Früchte der jahrelangen Arbeit ernten zu können. Thompson entwickelte auf einer PDP-7 eine single-user Version von Multics. Brian Kernighan nannte diese Version spöttisch **Unics** (*Uniplexed Information and Computing Service*). Später wurde daraus dann **UNIX**[1] — so die Legende.

1.2 Eine neue Programmiersprache ward geboren

Die ersten Multics/Unics Versionen wurden noch in Assembler[2] geschrieben. Da dies jedoch nicht Thompsons Vorstellungen entsprach sah er sich nach anderen Möglichkeiten um. 1971 versuchte er sich an Fortran und versuchte damit sein mittlerweile stabil laufendes single-user Multics weiterzuentwickeln. Am Ende dieses ersten Entwicklungstages

[1] UNIX ist ein weltweit eingetragenes Wahrenzeichen von The Open Group (*http://www.opengroup.org/*), einem Konsortium mehrerer Firmen welche es sich zur Aufgabe gemacht haben 'Distributoren' und Kunden zusammenzubringen um so die weitere Entwicklung zu begünstigen.

[2] Bei Assembler handelt es sich um eine sehr hardwarenahe Programmiersprache, bei der jeder Befehl einem Maschinenbefehl entspricht.

unter Fortan Tages wickelte er jedoch alles schnell wieder ein, und begann damit sich seine eigene Programmiersprache zu schreiben. Versehentlich. Sie basierte auf **BCPL**[3], entstand eher zufällig beim Versuch BCPL auf die PDP-7 zu portieren und hieß schlicht und ergreifend **B**. Mit B war es zwar möglich Mutics/Unics zu verbessern, jedoch war auch B noch nicht die richtige Programmiersprache. B war langsam und für neuere Großrechner vom Typ PDP-11 nicht geeignet.

Brian Kernighan und Dennis Ritchie schrieben gemeinsam die heutzutage immer noch sehr weit verbreitete Programmiersprache **C** [4] in der heute immer noch alle vorhandenen Unix Versionen und Ableger geschrieben werden. In den Jahren von 1971 bis 1973 wurde das vorhandene System komplett in C umgeschrieben und auf eine PDP-11 portiert. Zu dieser Zeit wurde Unix nur von BELL im internen Gebrauch eingesetzt. Das sollte sich aber in den nächsten Jahren drastisch ändern. Unix Installationen Weltweit 1973: etwa 25.

1.3 "System V vs. BSD“

Im Jahr 1975 wurde damit begonnen, Unix gegen eine meist geringe Gebühr, die nur die Kopier- und Versandkosten abdeckte, auszuliefern. Hauptabnehmer waren Universitäten welche sich dem ebenfalls mit ausgelieferten Quellcodes von Unix annahmen um ihn weiterzuentwickeln. 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. Unix Installationen Weltweit 1977: etwa 500.

Das von Berkley erweiterte Unix wurde als **BSD**[5] bekannt. In Berkley wurde eine spezielle Gruppe ins Leben gerufen (Computer Systems Research Group — CSRG), die sich nur mit der Verbesserung der vorhandenen Unix Quellen beschäftigte. Große Teile der TCP/IP Implementierung stammen aus Berkley von denen einige direkt in speziell für die damalige DARPA[6] und somit den militärischen Bereich entworfene Versionen flossen.

Seit dieser Zeit unterscheidet man zwei Arten von Unix. Zum einen das aus den BELL-Laboratories (später AT&T) stammende **System V** und zum anderen das aus Berkely stammende **BSD**. Eine immer aktuelle grafische Aufschlüsselung der verschiedenen Entwicklungswege ist unter dem URL `http://www.levenez.com/unix/history.html` zu finden.

Seit 1983 vermarktet AT&T Unix als System V und weitere namhafte Firmen nahmen es unter anderem Namen in ihr Programm auf: bei Hewlett Packard sprach man von Unix als HP-UX, bei Siemens von SIMUX, DEC nannte es ULTRIX, SGI seines

[3] Basic Combined Programming Language wurde 1966 von Martin Richards entwickelt und 1967 am MIT erstmalig implementiert

[4] Wer C lernen möchte, dem sei das Buch "Programmieren in C“ — erschinen im Hanser Verlag (`http://www.hanser.de`), ISBN 3-446-15497-3 — von Brian Kernighan und Dennis Ritchie empfohlen.

[5] Berkley Software Distribution

[6] Defense Advanced Research Projects Agency

IRIX, Sun arbeitete mit SunOS und später mit Solaris und IBM entwickelte AIX. Unix Installationen Weltweit 1984: etwa 100.000.

1.4 Unix heute

Mittlerweile sind wir bei Unix System V Release 4[7] und BSD 4.4[8] angekommen. Zudem sind etliche nicht-kommerzielle Unix Ableger mit riesiger Fan-Gemeinde entstanden. Bei Linux handelt es sich dabei um den wohl bekanntesten Vertreter. Es werden weltweit etliche verschiedenen Linux Distributionen[9] angeboten, die man sich entweder kostenlos aus dem Internet holen kann oder gegen teils relativ geringe Beträge auf CD bestellen kann. Neben Linux, welches auf System V basiert, erfreuen sich noch die diversen BSD Ableger[10] großer Beliebtheit — besonders im Server Bereich.

[7] SVR4, veröffentlicht 1989, gilt heute noch als Unix-Standard

[8] veröffentlicht 1990

[9] Zu den hierzulande bekanntesten gehören SuSE Linux *(http://www.novell.com/de-de/linux/)*, Debian Linux *(http://www.debian.de)*, RedHat Linux *(http://www.redhat.de)* und Ubuntu *(http://www.ubuntu.com)*

[10] FreeBSD *(http://www.freebsd.org/)*, NetBSD *(http://www.netbsd.org)* und OpenBSD *(http://www.openbsd.org/)*

2 Unix Grundlagen

> Unix is user-friendly. It just isn't promiscuous about which users it's friendly with.
>
> *(Stephen King)*

2.1 Login am System

Da es sich bei Unix-Systemen um Multiuser-Systeme handelt, muß man sich in aller Regel zuerst am System anmelden bevor man mit ihm arbeiten kann. Dazu benötigt man z.B. einen Benutzernamen um sich gegenüber dem System zu authentifizieren und ein beispielsweise ein Passwort, ein Zertifikat oder einen Key für eine erfolgreiche Authorisierung.

Anmelden kann man sich entweder lokal am System oder, wie im Serverbetrieb eher üblich, „remote“ über eine Netzwerverbindung. Je nach Anmeldeart können verschiedene Authentifizierungsmethoden eingesetzt werden. In der Regel greift man über eine SSH Verbindung mit Passwort oder Public-Key Authentifizierung auf ein System zu. Eine Remoteanmeldung per Telnet sollte heutzutage in jedem Fall vermieden werden, da Telnet im Klartext übertragen wird und somit Passwörter etc. leicht mitgelesen werden können.

Nach dem Login wird die für den Benutzer vorgegebene Shell gestartet und man landet automatisch im eigenen Homedirectory, in dem sich einige für den Login relevante Daten befinden und ausserdem alle persönlichen Daten gespeichert werden. Welche Shell beim Login gestartet wird und welches das Homedir ist, wird in der Datei `/etc/passwd` festgelegt. Dazu in einem späteren Kapitel eventuell mehr :P.

Die Shell stellt die Schnittstelle zwischen dem Benutzer und dem Kernel, dem Betriebssystemkern, dar. Sie nimmt alle Befehle entgegen, leitet sie entsprechend weiter und gibt uns über verschidene „Kanäle“ Rückmeldungen. Auf all das werden wir im Kapitel „Arbeiten auf der Shell“ genauer eingehen.

2.2 Befehle und Kommandos

Befehle unter Unix bestehen oft nicht nur aus einem einzelnen Wort, dem Befehl oder Programmnamen an sich, sondern auch aus zusätzlichen Optionen und Argumenten. In Abbildung 2.1 auf Seite 5 habe ich einmal einen typischen Befehl inklusive Prompt in die einzelnen Teile zerlegt. `tar` ist das Programm welches gestartet wird, die Optionen

`-xzfv` sagen `tar` wie es mit dem Argument `irssi_0.8.5.tar.gz`, in diesem Fall einer Datei, verfahren soll.

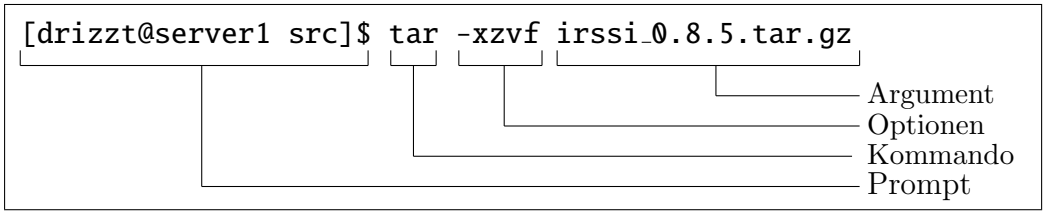

Abbildung 2.1: Unix Befehlssyntax

Viele der im täglichen Betrieb verwendeten Befehle sind sogenannte „shell builtins“. Diese Befehle sind in die Shell integriert und es werden keine externen Befehle, also irgendwo im Filesystem befindlichen ausführbare Dateien, gestartet. Shell builtins arbeiten schneller als externe Befehle, da sie direkt in die Shell integriert sind und nicht erst geladen werden müssen. Der Nachteil an builtins ist, das sie nicht einzeln modifiziert oder upgedatet werden können. Falls das nötig sein sollte, muss die komplette Shell ersetzt werden. Beispiele für builtins sind `cd`, `pwd`, `logout` oder `echo`. Hilfe zu den builtins bekommt man mittels `help <builtin>`:

```
# help pwd
pwd: pwd [-LP]
    Print the current working directory.  With the -P option, pwd prints
    the physical directory, without any symbolic links; the -L option
    makes pwd follow symbolic links.
```

Die meisten builtins stehen zusätzlich auch als externe Programme zur Verfügung (z.B. `/bin/pwd`). Eine grosse Ausnahme stellt hierbei der Befehl `cd` zum wechseln von Verzeichnissen dar, der nur als builtin existiert. Ob man es mit builtins oder externen Befehlen zu tun hat kann man mit `type` testen:

```
# type pwd
pwd is a shell builtin
# type tar
tar is /bin/tar
```

2.3 Erste Schritte im System

Da wir eingeloggt sind nun grob wissen wie Befehle aufgebaut sind können wir die ersten Schritte unternehmen. Dazu werde ich einige der grundlegenderen Befehle vorstellen mit denen man täglich arbeitet. Ich werden die Befehle nicht bis ins allerkleinste Detail beschreiben. Weitere Optionen und Hilfen erhält man bei fast allen Befehlen über *befehl* `--help` oder über die entsprechende Manpage die mit `man <befehl>` aufgerufen wird. Mit den Manpages beschäftigen wir uns in Abschnitt 2.4 noch genauer.

2.3.1 pwd - aktuelles Verzeichnis abfragen

`pwd` gibt den absoluten Pfad des aktuellen Arbeitsverzeichnisses an. Nicht mehr und nicht weniger:

```
# pwd
/usr/local/bin
```

2.3.2 ls - Verzeichnisinhalte anzeigen lassen

Um sich den Inhalt eines Verzeichnisses anzeigen zu lassen benutzt man das Kommando `ls`. Dadurch werden alle Datei und Unterverzeichnisnamen im aktuellen Arbeitsverzeichnis angezeigt, deren Namen nicht mit einem '`.`' beginnen.

```
# ls
ausfuehrbar      symlink  testdir          testfile
```

Um auch versteckte Dateien, solche mit einem `.` am Anfang vom Dateinamen, anzeigen zu lassen, wird die Option `-a` benutzt. Dabei steht '`.`' für das aktuelle und '`..`' für das übergeordnete Verzeichnis.

```
# ls -a
.                ..               .versteckt       ausfuehrbar      symlink
testdir          testfile
```

Um sich erweiterte Informationen (Dazu mehr in Kapitel 4) anzeigen zu lassen wird die Option `-l` verwendet. `-l` wird auch sehr gerne mit `-a` kombiniert.

```
# ls -la
total 3314
drwxr-xr-x   3 drizzt   drizzt       512 Sep  5 16:38 .
drwx------  44 drizzt   drizzt      4608 Sep  5 15:28 ..
-rw-r--r--   1 drizzt   drizzt    671744 Sep  5 16:38 .versteckt
-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
```

2.3.3 cd - Verzeichnisse wechseln

Um in ein anderes Verzeichnis zu wechseln verwendet man den Befehl `cd <zielverzeichnis>`. Wird `cd` ohne Argument verwendet, wechselt man automatisch in sein Homedir.

```
# pwd
/usr/local/bin
# cd
# pwd
/root
```

Gibt man als Argument den Namen des Verzeichnisses an, in da es zu wechseln gilt, muß zwischen *relativen* und *absoluten* Pfadnamen unterscheiden. Ein absoluter Pfad beginnt mit einem `/` — es wird also von der Wurzel, root, des Verzeichnis-/Dateibaums ausgegangen. Relative Pfade beginnen nicht mit einem `/`. In diesem Fall wird vom aktuellen Arbeitsverzeichnis ausgegangen.

```
# pwd
/
# cd /home/drizzt/
# pwd
/home/drizzt
# cd Unix/testdir/
# pwd
/home/drizzt/Unix/testdir
```

Wird als Shell die Bash verwendet kann man mit `cd -` in das Verzeichnis zurückwechseln, in dem man sich vor dem letzten Verzeichniswechsel befand. Welche Shell man gerade verwendet kann man durch den Befehl `echo $0` anzeigen lassen[1]

```
# echo $0
-bash
# pwd
/root
# cd /usr/local/bin
# pwd
/usr/local/bin
# cd -
/root
# pwd
/root
```

2.3.4 whoami und who - wer bin ich und wer ist sonst noch da

Falls man sich mal nicht so sicher sein sollte wer man selber ist (kommt nach stundenlangem Getippe durchaus vor) kann man sich vom System mit `whoami` (Linux) bzw. `who am i` (Linux und Solaris) helfen lassen.

```
$ whoami
drizzt
$ who am i
drizzt   pts/6        2008-11-11 15:18 (192.168.1.70)
```

Wer ausser einem noch auf dem System unterwegs ist, wird durch den Befehl `who` angezeigt.

```
$ who
vt100 tty1         2008-10-29 14:13
vt100 pts/2        2008-10-29 14:14 (:0:S.1)
vt100 pts/3        2008-10-29 14:14 (:0:S.2)
drizzt   pts/6        2008-11-11 15:18 (192.168.1.70)
```

[1] In der C-Shell funktioniert dies leider nicht interaktiv, wohl aber über ein Shellscript. Interaktiv wird die Fehlermeldung `No file for $0` ausgegeben.

2.3.5 date - Datum und Uhrzeit

Mit dem Befehl `date` kann man sich die aktuelle Uhrzeit und das Datum anzeigen lassen.

```
# date
Thu Sep  5 10:41:43 MEST 2004
```

Die Ausgabe der Datums und der Uhrzeit kann auf vielerlei Arten formatiert werden[2]

```
# date '+%d.%m.%y %H:%M:%S'
05.09.04 10:43:29
```

Der User `root` kann mittels `date` die Systemzeit manuell setzen. Eine genaue Uhrzeit ist kein Luxus sondern später beim Troubleshooting ein sehr wichtiges Hilfsmittel. Dabei müssen auch eventuell unterschiedliche Zeitzonen[3] berücksichtigt werden.

2.3.6 cat - Dateiinhalte anzeigen lassen

Der Befehl `cat` liest eine Datei ein und gibt deren Inhalt auf dem Monitor wieder. Der Schalter `-n` gibt zusätzlich noch die Zeilennummern aus, während `-b` die Ausgabe der Zeilennummern bei Leerzeilen unterdrückt:

```
# cat testfile
Zeile 1

Zeile 3
# cat -n testfile
     1  Zeile 1
     2
     3  Zeile 3
# cat -b testfile
     1  Zeile 1

     2  Zeile 3
```

2.3.7 less & more - Dateiinhalte seitenweise anzeigen lassen

Wenn man sich lange Textdateien mittels `cat` anzeigen läßt rauscht die Ausgabe förmlich an einem vorbei. Man kann in diesem Fall die Datei mittels `more`[4] (`more <dateiname>`) oder `less`[5] (`less <dateiname>`) seitenweise ausgeben lassen.

Es gibt gibt verschiedene Möglichkeiten innerhalb von mit `more` und `less` angezeigten Dateien zu navigieren oder zu suchen. In Tabelle 2.1 sind die wichtigsten gegenübergestellt. Grade bei `less` gibt es oft mehrere Möglichkeiten ein und dasselbe auszuführen. In diesen Fällen hab ich mich auf die gängigsten Methoden beschränkt.

Wie man sieht ist `less` um einiges mächtiger als `more` und versteht oft auch Kommandos die wir beim Arbeiten mit dem Editor `vi` noch kennenlernen werden.

[2] Die Formatierungsmöglichkeiten sind in der Manpage von `date` genauer beschrieben.
[3] Siehe auch `http://de.wikipedia.org/wiki/Zeitzone` oder `http://www.zeitzonen.de/`
[4] `more` ist normalerweise auf jedem Unix System verfügbar.
[5] `less` gehört leider immer noch nicht bei jedem System zum Standard-Befehlsrepertiore

Befehle innerhalb von `more` und `less`		
Auswirkung	`more`	`less`
Eine Bildschirmseite runter	`SPACE z`	`SPACE f`
Eine Bildschirmseite rauf	`b`	`b`
Eine Zeile runter	`RETURN`	`RETURN e`
Eine Zeile rauf		`y k`
Eine halbe Seite weiter		`d`
Eine halbe Seite zurück		`u`
Hilfeseite aufrufen	`h`	`h H`
Abwärts nach `pattern` suchen	`/pattern`	`/pattern`
Aufwärts nach `pattern` suchen		`?pattern`
Zur nächsten Fundstelle von `pattern` weiter unten im Text springen	`n`	`n`
Zur nächsten Fundstelle von `pattern` weiter oben im Text springen		`N`
Suchmarkierungen löschen		`ESC-u`
An den Anfang springen		`g`
Ans Ende springen		`G`
Zur Zeile n springen	`:n`	`:n pn`
Datei `file` anzeigen		`e: file`
Datei in `$EDITOR` bearbeiten	`v`	`v`
Beenden	`q`	`q :q Q :Q ZZ`

Tabelle 2.1: Befehle innerhalb von `more` und `less`

2.3.8 `passwd` - eigenes Passwort wechseln

Passwörter sollten regelmäßig gewechselt werden und möglichst sicher sein. Unter Unix wird das eigene Passwort mit dem Befehl `passwd` geändert. Wenn man als normaler, unprivilegierter User `passwd` aufruft muss man zunächst sein aktuelles und danach zwei mal das neue Passwort eingeben. Als Superuser root entfällt der erste Schritt und mann kann auch mit `passwd <username>` die Passwörter anderer User ändern.

Bitte bei der Auswahl eines Passworts wirklich darauf achten, dass es sicher ist. Sicher bedeutet mindestens acht Zeichen lang, Gross- und Kleinbuchstaben sowie Ziffern und Sonderzeichen sollten enthalten sein. Am einfachsten nimmt man sich einen Satz und Kombiniert die Anfangsbuchstaben der einzelnen Worte zu einem Passwort. Aus „Ein sicheres Passwort besteht aus mindestens 8 Zeichen!“ wird so das Passwort „EsPbam8Z!“.

2.3.9 `logout` - Ausloggen

Aus einem System ausloggen kann man sich mit dem Befehl `logout` oder der Tastenkombination `CTRL-D` (`^D`). Genauer gesagt beendet man damit die aktuelle Shell. Wenn

man also aus der `sh` heraus erst noch die `bash` gestartet hat muss man sich zwei mal ausloggen um das System letztendlich zu verlassen.

2.4 Manpages, die schnell verfügbare Hilfe

Normalerweise sind auf jedem Unix System auch die Manpages installiert. Dabei handelt es sich um eine Sammlung von etlichen Dokumenten mit oft sehr genauer und detailierter Dokumentation zu Befehlen oder Systemdateien. Die Manpages sind in momentan 12 Bereiche unterteilt.

Zum anzeigen dieser „online Dokumentation“ benutzt man den Befehl `man`. Leider empfinden viele User das Lesen der Manpages als nervig, langwierig oder gar zu kompliziert. Man findet jedoch kurzfristig — und oft auch langfristig — keine besseren Anleitungen.

Es existiert zu nahezu jedem Unix Kommando bzw. Systembestandteil eine Manpage. Man ruft sie auf, indem man einfach `man` gefolgt von dem Kommandonamen aufruft, zu dem man Hilfe braucht. Abbildung 2.2 auf Seite 12 zeit die erste Bildschirmseite von `man man`.

Die Navigation innerhalb der Manpage ist der Navigation innerhalb von `more` und `less` sehr ähnlich, da die Manpage in `more` bzw. `less` gepiped wird. Durch ein „`-`“ als Option wird `man` angewiesen sein Ausgabe in `cat` zu pipen und die Manpage rauscht in einem Schwung durch. Mit `h` kann man sich in `man` eine Hilfeseite mit Befehlen anzeigen lassen auf der man auch prompt mit `@(#)more.help 1.5` bzw. `SUMMARY OF LESS COMMANDS` begrüßt wird.

Mit `e` oder `RETURN` scrollt man eine Zeile weiter runter, mit `y` eine Zeile weiter rauf. Ganze Seiten runter geht es mit `f` oder `SPACE` und wieder rauf geht's mit `b`. Nach `pattern` sucht man mit `/pattern` wobei mit `n` zum nächsten Fundort von `pattern` weiter runter gesprungen wird und mit `N` zum nächsten Fundort weiter oben in der Manpage.

Wenn das System keine Manpage mit dem angegebenen Befehl findet, kann man innerhalb der Manpages nach Begriffen (`keyword`) suchen. Dazu muessen die Manpages vom Superuser indiziert werden. Das geschieht mittels `catman` bzw. `mandb` oder `makewhatis`. Anschliessend kann man sich mit `whatis` eine einzeilige Zusammenfassung zum Suchbegriff ausgeben lassen.

```
# whatis hostname
hostname        hostname (1)    - set or print name of current host system
```

Eine erweiterte Suche bietet `apropos`. Damit werden die Manpages mit zugehöriger Sektion ausgegeben, bei denen `keyword` entweder im `NAME` oder im `DESCRIPTION` Bereich der Manpage vorkommt.

```
# apropos hostname
check-hostname  check-hostname (1m) - check if sendmail can determine the system's
                                      fully-qualified host name
ethers          ethers (4)      - Ethernet address to hostname database or domain
gethostname     gethostname (3c)    - get or set name of current host
```

```
gethostname     gethostname (3xnet) - get name of current host
hostname        hostname (1)    - set or print name of current host system
```

```
MAN(1)                          Manual pager utils                          MAN(1)

NAME
       man - an interface to the on-line reference manuals

SYNOPSIS
       man  [-c|-w|-tZ]  [-H[browser]]  [-T[device]] [-X[dpi]] [-adhu7V] [-i|-I]
       [-m system[,...]] [-L locale] [-p string] [-C file] [-M path] [-P pager]
       [-r prompt]  [-S list] [-e extension] [--warnings [warnings]] [[section]
       page ...] ...
       man  -l [-7] [-tZ] [-H[browser]] [-T[device]] [-X[dpi]] [-p string]
       [-P pager] [-r prompt] [--warnings[warnings]] file ...
       man -k [apropos options] regexp ...
       man -f [whatis options] page ...

DESCRIPTION
       man  is the systemâs manual pager. Each page argument given to man is
       normally the name of a program, utility or function.  The manual page
       associated  with each of these arguments is then found and displayed.
       A section, if provided, will direct man to look only in that  section
       of  the manual.  The default action is to search in all of the avail
       able sections, following a pre-defined order and  to  show  only  the
       first page found, even if page exists in several sections.

       The  table  below shows the section numbers of the manual followed by
       the types of pages they contain.

       1   Executable programs or shell commands
       2   System calls (functions provided by the kernel)
       3   Library calls (functions within program libraries)
       4   Special files (usually found in /dev)
       5   File formats and conventions eg /etc/passwd
       6   Games
       7   Miscellaneous (including macro  packages  and  convenâ[m
           tions), e.g. man(7), groff(7)
       8   System administration commands (usually only for root)
       9   Kernel routines [Non standard]

       A manual page consists of several parts.

       They  may  be  labelled  NAME, SYNOPSIS, DESCRIPTION, OPTIONS, FILES,
       SEE ALSO, BUGS, and AUTHOR.
 [...]
```

Abbildung 2.2: Die man-Manpage

3 Arbeiten auf der Shell

> We have persistent objects,
> they're called files.
>
> *(Ken Thompson)*

3.1 Die Shell als Schnittstelle zum Kernel

Wie schon in Kapitel 2 erwähnt wurde, umgibt die Shell den Kernel und stellt somit die Schnittstelle zwischen User und Betriebssystem dar. Einfach gesprochen ist die Shell ein simples Programm welches die Befehle des Users entgegennimmt und für den Kernel übersetzt. Dabei bietet sie dem Benutzer mittels diverser grundlegender Programmierwerkzeuge (Schleifen, Variablen, bedingte Verzweigungen usw.) sogar die Möglichkeit selber kleinere „Programme“, sogenannte Shellscripte, zu schreiben. Dazu jedoch später mehr. Welche Shell nach einem erfolgreichen Login gestartet wird, wird für jeden User in der Datei `/etc/passwd` definiert. Sollte einem diese Shell nicht gefallen, kann man entweder den Systemadministrator bitten eine andere Loginshell einzutragen oder selber nach dem Login die gewünschte Shell selber starten.

3.2 Die verschiedenen Shells

3.2.1 Bourne-Shell - Das Urgestein

Die Bourne-Shell (`/bin/sh`) ist die älteste der heute noch gebräuchlichen Shells. Sie wurde von Steven Bourne bei AT&T entwickelt und erstmals mit Unix Version 7 in 1979 ausgeliefert. Für den User `root` ist sie die Standardloginshell (`/bin/sh` bzw. `/sbin/sh`). Normale User verwenden sie jedoch selten, da die Bourne-Shell im Vergleich zu den anderen Shells relativ unkomfortabel ist. Aufgrund ihrer geringen Systembelastung und großen Verfügbarkeit ist sie für die Entwicklung von Shellscripten immer noch erste Wahl. Der Defaultprompt für normale User ist das Dollarzeichen (`$`).

3.2.2 Bash

Die wohl bekannteste und am weiten verbreitetste Shell ist die BASH — Bourne Again Shell (`/bin/bash`). Sie ist im Rahmen des GNU Projekts[1] ursprünglich von Brian Fox

[1] GNU ist ein rekursives Akronym für „GNU's not UNIX“. Das GNU Projekt hat es sich seit 1984 zur Aufgabe gemacht ein freies Unix-artiges Betriebssystem zu entwickeln. Urheber dieses Projekts war Richard Stallman von der Free Software Foundation (FSF). `http://www.gnu.org`

entwickelt und am 10. Januar 1988 offiziell vorgestellt worden. Die BASH übernahm ihrerseits die besten Features der bisher bekannten Shells und fügte ihr noch einige weitere hinzu. Dabei wurde viel Wert auf eine möglichst gute Konfigurierbarkeit der Arbeitsumgebung gelegt. Bereits vorhandenen Bourne-Shell Shell-Scripte können natürlich ebenfalls weiter verwendet werden. Sie gehört allerdings immer noch nicht zur Standardinstallation eines jeden Unix.

3.2.3 C-Shell - für den C-Programmierer unter uns

Billy Joy entwickelte in Berkley die in ihrer Syntax der Programmiersprache C sehr ähnliche C-Shell (`/bin/csh`). Sie bietet gegenüber der Bourne-Shell mehr Komfort durch z.B. eine command-line history, job control und der Möglichkeit Aliase zu vergeben. Der normale Prompt ist der Hostname gefolgt von einem Prozentzeichen (`hostname%`). Mittlerweile wird sie meiner Erfahrung nach jedoch eher selten eingesetzt — jedenfalls kenne ich keinen überzeugten C-Shell User.

3.2.4 Korn-Shell

Die Korn-Shell (`/bin/ksh`) haben wir Herrn David Korn von den AT&T Bell Laboratories zu verdanken. In ihr wurden etliche Neuerungen der C-Shell übernommen und um Features wie command-line editing erweitert. Dabei ist sie weiterhin voll kompatibel zur Bourne-Shell, so daß bereits vorhandene Shell-Scripte auch weiterhin verwendet werden können. Die Korn-Shell ist auf vielen Systemen noch die Standardshell für normale User.

3.3 Metazeichen

Beim Arbeiten mit der Shell muss man die Bedeutung diverser Sonderzeichen, der Metazeichen, kennen. Metazeichen stehen bei der Verarbeitung der Eingaben nicht für sich selber sondern werden von der Shell entsprechend ersetzt.

Das bekannteste Metazeichen ist wohl das Sternchen „`*`“, oder auch Asterisk genannt. Das `*` wird von der Shell zu jedem erdenklichen und passenden Zeicher ersetzt. Das schliesst auch „kein Zeichen“ ein. Besonders oft wird das bei der Dateinamenerweiterung, dem sogenannten „Globbing“ benutzt. Das „`?`“ hingegen wird nur zu einem einzigen Zeichen erweitert.

In Tabelle 3.4 auf Seite 22 werden einige der wichtigsten Metazeichen zusammengefasst und erklärt.

3.4 Ein- und Ausgabeumlenkung in der Shell

Auf der Konsole ist es möglich die Ausgaben eines Programms bzw. die Eingaben die es erwartet zu beeinflussen. Dazu dient das Ein-/Ausgabekonzept von Unix, welches drei vordefinierte Datenkanäle besitzt: `stdin(0)`, `stdout(1)` und `stderr(2)`. Wird nichts

anderes angegeben erfolgen die Ausgaben standardmässig auf dem Bildschirm (standard out/output - `stdout`) und die benötigten Eingaben werden von der Tastatur gelesen (standard in/input - `stdin`). So gesehen verhalten sich viele Programme wie ein Filter: sie lesen Daten aus einer Eingabedatei ein, bearbeiten sie und schreiben sie in eine Ausgabedatei.

`ls` gibt den Verzeichnisinhalt normalerweise direkt auf standard out aus. Die Shell bietet uns jedoch die Möglichkeit die Ausgabe z.B. in eine Datei oder auf einen Drucker umzuleiten. Dies erreichen wir durch das '>'. Wenn ein > und ein Dateiname einem Befehl angehangen wird, werden alle Ausgaben statt auf standard out `stdout(1)` in diese Datei geschrieben.

```
# ls -l verzeichnisinhalt.txt
verzeichnisinhalt.txt: No such file or directory
# ls > verzeichnisinhalt.txt
# ls verzeichnisinhalt.txt
-rw-r--r--  1 root  root  3598 Dec 11 10:42 verzeichnisinhalt.txt
#
```

Die Datei `verzeichnisinhalt.txt` hat noch nicht existiert. Durch das Umlenken der Ausgabe von `ls` wurde sie angelegt und mit den Ausgaben von **ls -l** gefüllt.
Existiert die Datei noch nicht wird sie angelegt. Existiert die angegebene Datei bereits wird sie überschrieben!

```
# echo "Eine Zeile Text" > zeilentext.txt
# cat zeilentext.txt
Eine Zeile Text
# echo "Eine neue Zeile Text ueberschreibt die alte" > zeilentext.txt
# cat zeilentext.txt
Eine neu Zeile Text ueberschreibt die alte
#
```

Der Befehl `echo` gibt einfach den Text aus den man ihm als Argument übergibt. Im ersten Schritt wurde durch die Umlenkung der Ausgabe *Eine Zeile Text* die Datei `zeilentext.txt` erzeugt. Im zweiten Schritt wurde durch das erneute Umlenken nach `zeilentext.txt` der Inhalt überschrieben. Will man das Überschreiben verhindern, muss man mit '>>' umlenken. Dann werden die umgelenkten Ausgaben ans Ende der Datei angehangen.

Ebensoleicht kann man auch standard in `stdin(0)` umleiten bzw. sich die erwarteten Eingaben die sonst über die Tastatur erfolgen würden aus einer Datei holen:

```
# mail drizzt@netzsheriff.de < zeilentext.txt
```

So maile ich mir den Inhalt der Datei `zeilentext.txt` aus dem Beispiel von oben zu.

All diese Umlenkungen lassen sich auch kombinieren. Besonders häufig benutzt man dies z.B. bei Cronjobs:

```
1 2 * * * [ -x /usr/sbin/rtc ] && /usr/sbin/rtc -c > /dev/null 2>&1
```

Cronjobs sehen wir uns später noch genauer an. An der Stelle reicht es erstmal wenn ich frech behaupte durch diesen Eintrag würde täglich um 02:01 Uhr geschaut ob die Datei `/usr/sbin/rtc` ausführbar ist. Falls dies zutrifft wird sie mit der Option `-c` gestartet. Uns interessiert dann nur noch der Ausdruck `> /dev/null 2>&1`. Was `> /dev/null` bewirkt sollte klar sein: alle möglichen Ausgaben von `rtc` werden in das Nulldevice `/dev/null` umgeleitet. Einfach gesagt: auf den Müll damit. Mit `2>&1` zusammen wird es eine sogenannte gekoppelte Umleitung. Ein `2>` mit eintsprechender Zieldatei dahinter würde standard error `stderr(2)` umleiten:

```
# ls verzeichnisinhalt.txt
verzeichnisinhalt.txt: No such file or directory
# ls verzeichnisinhalt.txt 2> ls-error.txt
# cat ls-error.txt
verzeichnisinhalt.txt: No such file or directory
#
```

Mit `2>&1` leite ich `stderr` genauso um wie `stdout` zuvor: in unserem Cronjob-Beispiel von oben also nach `/dev/null`.

Getrennte Umlenkungen sind ebenso möglich:

```
# cat zeilentext.txt nichtvorhanden.txt
Eine neu Zeile Text ueberschreibt die alte Zeile
cat: cannot open nichtvorhanden.txt
# cat zeilentext.txt nichtvorhanden.txt > stdout.txt 2> stderr.txt
# cat stdout.txt
Eine neu Zeile Text ueberschreibt die alte
# cat stderr.txt
cat: cannot open nichtvorhanden.txt
#
```

Tabelle 3.1 auf Seite 17 fasst die Möglichkeiten noch einmal zusammen.

3.5 Pipes

Das Konzept der Pipelines unter Unix ermöglicht es, die Ausgaben (`stdout`) eines Prozesses direkt als erwartete Eingabe (`stdin`) eines anderen Prozesses zu nutzen. Dabei wird der vertikale Strich „`|`“, auch „Pipe Zeichen“ oder schlicht „Pipe“ genannt, als Signal für das Erstellen einer Pipe zwischen zwei Prozessen genutzt. Das einfachste Beispiel und der wohl am häufigsten auftretende Fall ist, die Ausgabe eines Prozesses an `less` zu übergeben:

```
# ps aux | less
```

Damit wird die unter Umständen mehrere Bildschirmseiten lange Prozessliste an `less` übergeben und so seitenweise dargestellt.

Ein anderes Beispiel aus dem Admin-Alltag:

```
# mailq | grep MAILER-DAEMON | awk '{print $1}' | xargs postsuper -d
```

Ein-/Ausgabeumleitung in der Shell	
Umlenken von...	**Bourne/Korn/Bourne-Again-Shell (Bash)**
`stdin`	`< file`
`stdout`	`> file`
`stderr`	`2> file`
`stdout` und `stderr` in selbe Datei	`> file 2>&1` (Reihenfolge beachten)
getrennte Umlenkung von `stdout` und `stderr` in verschiedene Dateien	`> file 2> file2`
Ausgabe an bestehende Datei anhängen	`>> file`

Tabelle 3.1: Ein-/Ausgabe Umlenkungsmöglichkeiten

Hier wird die aktuelle Mailqueue eines Mailservers mit `mailq` abgefragt. In der `mailq` Ausgabe wird mit `grep` nach allen Zeilen mit `MAILER-DAEMON` gesucht und diese werden an `awk` übergeben. `awk` gibt schliesslich nur das erste Feld dieser Zeilen, die Postfix Queue-ID der Mails, aus und diese werden mittels `xargs` an `postsuper -d` übergeben und so aus der Queue gelöscht.

3.6 Verknüpfen von Kommandos

In der Shell können mehrere Kommandos miteinander verknüpft werden um diese z.B. stur hintereinander ablaufen zu lassen oder aber auch die Verarbeitung der einzelnen Kommandos an Bedingungen knüpfen. Der einfachste Weg zwei Kommandos miteinander zu verknüpfen geht über ein einfaches `;`.

```
# echo "echo 1"; echo "gefolgt von echo 2"
echo 1
gefolgt von echo 2
#
```

Hier werden die beiden `echo`s einfach hintereinander weg ausgeführt, egal was kommt bzw. egal ob das erste `echo` erfolgreich oder mit einem Fehler beendet wurde.

In manchen Fällen ist es aber sinnvoll das nächste Kommando nur dann auszuführen, falls das vorhergehende erfolgreich (bzw. eben nicht erfolgreich) war. Ob ein Kommando erfolgreich war bzw. ohne Fehler abgeschlossen wurde wird in einer Variablen[2], dem

[2]Mit Variablen beschäftigen wir uns später - so dieses Script je fertiggestellt wird - noch genauer. Fuer den Moment reicht es aus sich eine Variable als eine Art Speicher für Informationen vorzustellen und zu wissen das Variablen mit einem `$` beginnen. In Variablen sind u.a. wichtige Werte für die Shell wie der Path/Pfad (`$PATH`) oder das aktuelle Verzeichnis (`$PWD`) gespeichert. Man kann den Wert/Inhalt einer Variablen mit `echo $<`*`variablenname`*`>` abfragen.

sogenannten Rückgabewert `$?`, gespeichert. Wenn `$?` den Wert 0 hat wurde das letzte Kommando ohne Fehler ausgefuehrt. Ist der Wert von `$?` ungleich 0 trat ein Fehler auf.

Im folgenden Beispiel war das `echo` erfolgreich und das `cat` nicht:

```
# echo Test
Test
# echo $?
0
# cat nichtexistentedatei.txt
cat: cannot open nichtexistentedatei.txt
# echo $?
2
#
```

Um zu erreichen das ein Kommando nur dann ausgeführt wird falls das vorausgehende erfolgreich war, verknüpfe ich beide mittels `&&`. Soll das zweite Kommando nur ausgeführt werden falls das erste fehlschlägt verknüpft man beide mittels `||`. Hier ein paar Beispiele:

```
# echo Test && echo Test bestanden
Test
Test bestanden
#

# cat nichtexistentedatei.txt || echo ERROR
cat: cannot open nichtexistentedatei.txt
ERROR
e nich#

# cat nichtexistentedatei.txt && echo ERROR
cat: cannot open nichtexistentedatei.txt
#
```

So einfach ist das ;) In Tabelle 3.2 auf Seite 19 findet sich nochmal eine Übersicht zum Thema.

3.7 Quoting und Escapen

Im Abschnitt 3.3 haben wir schon einige Sonder- oder Metazeichen kennengelernt. Die Shell ersetzt diese Zeichen automatisch. Nun kommen aber im täglichen Betrieb durchaus Situationen vor, in denen wir nicht wollen das diese Zeichen ersetzt werden. In einem solchen Fall müssen wir diese Sonderzeichen vor der Shell schützen indem wir sie quoten oder escapen.

Quoten kommt vom englischen to quote, etwas/jemanden Zitieren bzw. etwas in Anführungszeichen setzen, und genau das wird auch gemacht. Man kann mit doppelten (`"..."`) oder einfachen Anführungszeichen (`'...'`) sowie mit dem Backslash (\) quoten. Je nach Methode ändert sich auch das Ergebnis.

Der Backslash (\) schützt nur das unmittelbar folgende Zeichen. Abgesehn vom Zeilenumbruch newline werden alle Zeichen geschützt.

Kommandoverknüpfungen in der Shell	
Verknüpfung mit...	**Auswirkung**
`;`	– das `;` bewirkt, daß die einzelnen dadurch getrennten Kommandos nacheinander ausgeführt werden – Der Rückgabewert (`$?`) des gesamten Kommandos entspricht dem Rückgabewert des zuletzt ausgeführten
`&`	– durch das Anhängen eines `&` an ein Kommando wird bewirkt, daß dieses Kommando im Hintergrund ausgeführt wird – die Shell meldet sich nach Absetzen des Kommandos sofort wieder mit einem Prompt, bereit weitere Kommandos entgegenzunehmen und auszuführen, während das mit `&` gestartete Kommando noch läuft – `stdout` und `stderr` eines mit `&` gestarteten Kommandos werden – falls nicht umgeleitet – ganz normal ausgegeben
`k1 || k2`	– `k2` wird nur dann ausgeführt wenn der Rückgabewert `$?` von `k1` ungleich `0` ist – `k1` also nicht fehlerfrei abgelaufen ist – Der Rückgabewert (`$?`) des gesamten Kommandos entspricht dem Rückgabewert des zuletzt ausgeführten
`k1 && k2`	– `k2` wird nur dann ausgeführt wenn der Rückgabewert `$?` von `k1` gleich `0` ist – `k1` also fehlerfrei abgelaufen ist – Der Rückgabewert (`$?`) des gesamten Kommandos entspricht dem Rückgabewert des zuletzt ausgeführten

Tabelle 3.2: Kommandoverknüpfungen

Mit den doppelten Anführungszeichen (`"..."`) (auch „weak quotes“ genannt) schützt man bis zum nächsten doppelten Anfürungszeichen alle Sonderzeichen abgesehen von doppelten Anführungszeichen, dem Backslash, dem Dollarzeichen und den „backticks“ (`‘...‘`).

Die einfachen Anführungszeichen (`'...'`) (auch „strong quotes“) schützen bis zum nächsten einfachen Anführungszeichen alles ausser einfachen Anführungszeichen.

Will man auf die filename generation, das Ersetzen von `*` zu Dateinamen, ganz verzichten, kann man es mit `set -f` abschalten und bei Bedarf mit `set +f` wieder aktivieren.

3.8 Umgebungsvariablen

Wie bei allen anderen Betriebssystemen werden auch unter Unix viele Eigenschaften der Arbeitsumgebung durch Variablen, den Umgebungsvariablen, festgelegt. Sie legen

Quoting in der Shell	
\	– das direkt rechts vom Backslash stehende Zeichen verliert seine besondere Bedeutung – Bsp.: `echo \*` oder `echo \$LOGNAME`
`'...'`	– Text innerhalb dieser einfachen Anführungszeichen wird von der Shell nicht verändert, **alle** Sonder-/Metazeichen verlieren ihre besondere Bedeutung – Variablen werden **nicht** durch ihren Wert ersetzt
`"..."`	– alle Sonderzeichen **außer** `$` verlieren ihre besondere Bedeutung – Variablen innerhalb von `"..."` werden durch ihren Wert ersetzt

Tabelle 3.3: Quoting in der Shell

fest in welchen Verzeichnissen nach ausführbaren Dateien gesucht werden soll, welche Terminalvariante zum Einsatz kommt oder welcher Texteditor genutzt wird.

Welche Umgebungsvariablen aktuell gesetzt sind, kann man sich mit `env` oder `set`anzeigen lassen bzw. sie gezielt mit `echo $<variablenname>` abfragen.

```
# env
TERM=vt100
SHELL=/bin/bash
USER=root
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
PWD=/root
HOME=/root
LOGNAME=drizzt
_=/usr/bin/env

# echo $SHELL
/bin/bash
```

Umgebungsvariablen werden bei Start einer Shell automatisch durch das einlesen und ausführen einiger Scripte gesetzt. Da wir hauptsächlich mit der Bash arbeiten beschränke ich mich auf die Dateien die für die Bash relevant sind. Bei anderen Shells läuft das alles jedoch in einem ähnlichen Schema ab.

In jedem Homedir sollte es eine Datei `.bash_profile` geben („sollte“, nicht „muss“). Diese wird bei jedem Start der Bash eingelesen und die darin enthaltenen Anweisungen werden ausgeführt. Hier einmal die `.bash_profile` von einem RHEL3:

```
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi
```

```
# User specific environment and startup programs

PATH=$PATH:$HOME/bin
BASH_ENV=$HOME/.bashrc
USERNAME="root"

export USERNAME BASH_ENV PATH
```

In diesem Beispiel werden hauptsächlich zwei Funktionen ausgeführt. Im oberen Teil wird mit einer einfachen `if`-Abfrage geprüft ob eine Datei `.bashrc` im Homedir existiert. Sollte dies der Fall sein, so wird diese Datei „gesourced“. Sie wird eingelesen und ausgeführt, genauso wie die `.bash_profile` grade eingelesen wird. Im unteren Teil werden dann einige Umgebungsvariablen angepasst und anschliessend durch `export` gesetzt.

Oft gibt es Dateien wie `/etc/bashrc` oder `/etc/profile` über die systemweite Umgebungsvariablen gesetzt werden. Im Beispiel oben wurde getestet ob die Datei `/.bashrc` existiert. In dieser wiederum gab es eine Anweisung, über die eine eventuell vorhandene `/etc/bashrc` eingelesen wird.

Alle Umgebungsvariablen lassen sich natürlich auch per Hand setzen. In der sh setzt man sie mit *<variablenname>=<variablenwert>* und macht sie danach mit `export` *<variablenname>* bekannt bzw. kombiniert man das in der bash gleich zu `export` *<variablenname>=<variablenwert>*. Unter csh oder tcsh reicht zum setzen und bekanntmachen `setenv` *<variablenname> <variableninhalt>*.

Metazeichen der Shell	
*	– der Asterisk – wird zu **keinem bzw. einem oder mehreren** beliebigen Zeichen für Dateinamen, ausgenommen Dateien mit einem „.“ am Beginn – versteckte Dateien werden also von diesem Metazeichen nicht erfaßt
?	– wird zu **einem** beliebigen Zeichen eines Dateinamens expandiert
[s]	– wird mit **einem** Zeichen aus der Zeichenkette `s` entsprechend den verfügbaren Dateinamen ersetzt Bsp.: `t[mu]p` würde auf die Dateien `tmp` und `tup` zutreffen – sofern sie existieren
[!s]	– wird durch ein Zeichen ersetzt, welches nicht in `s` enthalten ist
[c1-cn]	– wird durch genau ein Zeichen aus den Zeichen `c1-cn` ersetzt – Bsp.: `[a-z]` wäre ein Kleinbuchstabe von a bis z
{s1,s2,...}	– das Ergebnis besteht aus Zeichenketten in denen die in den geschweiften Klammern und durch Komma getrennten Zeichenketten enthalten sind – Bsp.: bei `ls h{an,un}d` würde der Ausdruck zu `ls hand hund` ersetzt werden sofern diese Dateien vorhanden sind – *nicht in der Bourne- und Korn-Shell verfügbar*
$	– das Zeichen beschreibt den Beginn einer Variablen deren Wert an dieser Stelle eingesetzt werden soll – Bsp.: `echo $HOME`
‘kommando‘	– man nennt sie Backticks – der Befehl innerhalb der Backticks wird ausgeführt und der Ausdruck ‘`kommando`‘ wird durch die Ausgabe von `kommando` ersetzt – Bsp.: `echo ‘pwd‘`
SPACE, TAB	– `SPACE` (Leertaste) und `TAB` (Tabulatortaste) werden in der Shell als Trennzeichen (Whitespaces) benutzt – mehrere Trennzeichen hintereinander werden von der Shell zu einem verkürzt
$(kommando)	– `kommando` wird in einer Sub-Shell ausgeführt – die neu gestartete Sub-Shell wird direkt nach Beendigung von `kommando` wieder geschlossen – da `kommando` in einer extra Shell gestartet wird, wirken sich mögliche Änderungen an Shell- und Umgebungsvariablen durch `kommando` nur auf diese neu gestartete Shell aus

Tabelle 3.4: Metazeichen in der Shell

4 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)*

4.1 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 4.2) 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 4.1 auf Seite 24 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 4.1: 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 4.1).

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 4.2 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 4.2: 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

4.2 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 4.3 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 4.3: Unix Dateitypen

4.2.1 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.

4.2.2 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.

4.2.3 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.

4.2.4 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.

4.2.5 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.

4.3 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 4.4 auf Seite 29 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 4.4: Unix Verzeichnisstruktur

4.4 Unix Dateisysteme

4.5 Arbeiten mit Dateien

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

4.5.1 `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
```

4.5.2 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
```

4.5.3 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]

4.5.4 `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
```

4.5.5 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.

4.5.6 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.

4.5.7 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.

4.5.8 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.

5 Systemadministration

> LINUX: You will spend countless hours figuring out how to do the simplest things. What could be more fun?
>
> *(dumbentia.com)*

In diesem Kapitel will ich einige wenige wichtige Themen zur Systemadministration ansprechen. Das sind bei weitem nicht alle relevanten Gebiete der Systemadministration. Themen wie Kernel kompilieren oder Benutzerverwaltung fallen vorerst unter den Tisch.

5.1 Boot Prozess und init-System

Bootstrapping, von „To pull oneself up by one's bootstraps.“, bezeichnet den Vorgang den wir allgemein als Booten bezeichnen. Mittels eines einfachen Systems, wie dem BIOS oder später dem Boot-Loader, wird ein kompliziertes System gestartet. Der Ausdruck bootstrapping geht zurueck auf Baron Münchhausen, der sich selber samt Gaul an einen Schopf aus dem Morast zog, und die Science-Fiction Kurzgeschichte „By His Bootraps“ von Robert A. Heinlein.

Der Boot Prozess der meisten Unix Systeme läuft relativ ähnlich ab. Sie verwenden alle SysVinit, das init-System von Unix System V. Nur Ubuntu, Fedora seit Version 9 und der Experimental Branch von Debian nutzen statt SysVinit mittlerweile das für Ubuntu entwickelte upstart[1].

Nachdem bei Intel/x86 Systemen der POST[2] vom BIOS[3] abgeschlossen wurde, werden die angeschlossenen Devices nach Bootsektoren (z.B. dem MBR[4] durchsucht. Der im Bootsektor liegende Boot-Loader wird gestartet und dieser lädt das Kernel-Image. Der Kernel erkennt die Hardware, bindet die Laufwerke ein und startet schliesslich den init-Prozess.

Bei Sun Sparc Systemen kontrolliert der Boot-PROM[5] und der auf ihm gespeicherte Monitor namens OpenBoot das System bis der Solaris Kernel geladen werden kann. Der

[1] `http://upstart.ubuntu.com/`
[2] Power On Self-Test
[3] Basic Input Output System
[4] Master Boot Record, die ersten 512 Byte einer Festplatte mit Partitionstabelle und Boot-Loader.
[5] Programmable Read-Only Memory

Boot-PROM lädt den `bootblk`, dieser startet das Programm ufsboot welches schliesslich den Kernel einliest. Sobald der Kernel mit Hilfe von ufsboot genug Module geladen hat um das Root-FS zu mounten, startet den init-Prozess.Der init-Prozess, PID 1, sucht in der Datei /etc/inittab nach einem Eintrag für den default Runlevel.

```
id:3:initdefault:
```

Ein Runlevel in einem SysV-Unix-System beschreibt eine Art Systemzustand. Ein System kann so in einen definierten Zustand gebracht werden um z.B. spezielle Wartungsarbeiten im Single-User Mode durchzuführen. Es gibt normalerweise sieben Runlevel, 0 bis 6, von denen nur die Runlevel 0 (System-Halt), S (Single-User Mode, meist ein Synonym für Runlevel 1) und 6 (Reboot) auf allen SysV-Systemen identisch sind. Alle übrigen Runlevel werden je nach Distribution und Hersteller anders benutzt.

Einem Runlevel sind Dienste zugeordnet welche beim Booten nach und nach gestartet werden. Dabei werden die einzelnen Runlevel gegebenenfalls nacheinander komplett durchlaufen, bis das System seinen default Runlevel erreicht hat. Unter Solaris ist Runlevel 3 default[6]. Unter Linux ist es mit installierter graphischer Benutzeroberfläche Runlevel 5, ohne GUI ebenfalls Runlevel 3. Tabelle 5.1 stellt die Runlevel von Red Hat und Solaris gegenüber.

Red Hat und Solaris Runlevel		
Runlevel	Red Hat	Solaris
S	Single-User Mode, wie Runlevel 1	Single-User Mode, wie Runlevel 1
0	Halt/Shutdown	Shutdown, OpenBoot PROM Modus (`ok`)
1	Single-User Mode	Single-User Mode
2	Multiuser, einige Netzwerkservices werden gestartet	Multiuser ohne NFS Server
3	Multiuser mit Netzwerk	Multiuser mit NFS
4	nicht definiert, frei verfügbar	nicht definiert, frei verfügbar
5	Multiuser mit X11 und Netzwerk	Poweroff, ähnlich Runlevel 0 aber es wird noch sofern möglich das Netzteil abgeschaltet
6	Reboot	Reboot

Tabelle 5.1: Unix Runlevel

Die Funktion spezieller Runlevel wie Q, a, b, c oder U kann der Manpage zu `init` entnommen werden. In welchem Runlevel sich das System befinder kann man mit `runlevel` (Linux) oder `who -r` (Linux & Solaris) abfragen.

Wenn das System in den Runlevel 3 bootet, werden nacheinander die darunterliegenden Runlevel durchlaufen und die entsprechenden Startscripte/init-Scripte, all die mit

[6]Dies hat sich mit Solaris 10 geändert. Dort wurden „milestones“ eingeführt, über die Services gruppiert werden. „If services are files, then milestones are directories.“

einem S gefolgt von einer Zahl am Anfang des Dateinamens, der Reihe nach gestartet. Die init-scripte liegen unter Solaris alle in `/etc` bzw. `/etc/rc.*/` und `/etc/init.d/`[7]. Die Scripte aus den einzelnen Verzeichnissen (z.B. `/etc/rc3.d`) sind Hardlinks auf die Scripte in `/etc/init.d/` wie man am identischen Inode erkennt:

```
# ls -li /etc/rc3.d/S15nfs.server /etc/init.d/nfs.server
     788 -rwxr--r--   6 root     sys       2769 Apr  7  2002 /etc/init.d/nfs.server
     788 -rwxr--r--   6 root     sys       2769 Apr  7  2002 /etc/rc3.d/S15nfs.server
```

Unter Linux liegen die init-scripte unter `/etc/rc.d/rc.*/` bzw. `/etc/rc.d/init.d/`. Es gibt jedoch auch Hardlinks (RHEL 3) bzw. Symlinks (RHEL 5) von `/etc/rc.*` nach `/etc/rc.d/rc.*`, was die Umstellung von Solaris auf Linux erleichtert.

```
[RHEL3]# ls -ldi /etc/rc3.d/ /etc/rc.d/rc3.d/
5390340 drwxr-xr-x    2 root     root         4096 Jan 24  2007 /etc/rc3.d/
5390340 drwxr-xr-x    2 root     root         4096 Jan 24  2007 /etc/rc.d/rc3.d/

[RHEL5]# ls -ldi /etc/rc3.d/ /etc/rc.d/rc3.d/
655428 lrwxrwxrwx    1 root     root           10 Nov 3 11:38 /etc/rc3.d -> rc.d/rc3.d
655422 drwxr-xr-x    2 root     root         4096 Nov 3 12:03 /etc/rc.d/rc3.d
```

Alle diese init-Scripte können auch zum starten, stoppen oder reinitialisieren von Services genutzt werden. Dazu werden sie, je nach Bedarf mit einer der Optionen `start`, `stop`, `reload`, `status` oder `restart` aufgerufen. Welche Optionen unterstützt werden erfährt man in der Regel durch das Starten eines init-Scripts ohne Optionen.

```
# /etc/rc3.d/S80sendmail
Usage: /etc/rc3.d/S80sendmail start|stop|restart|condrestart|status
```

Nun bei Bedarf die passende Option anhängen und ausführen.

```
# /etc/rc3.d/S80sendmail restart
Shutting down sm-client:                                   [  OK  ]
Shutting down sendmail:                                    [  OK  ]
Starting sendmail:                                         [  OK  ]
Starting sm-client:                                        [  OK  ]
```

Über die `/etc/inittab` werden, besonders unter Linux, außer den Runleveln noch andere Einstellung vorgenommen. Die Virtuellen Consolen werden definiert, Tastenkombinationen wie CTRL-ALT-DEL werden abgefangen und Maßnahmen bei UPS powerfail bzw. powerrestore vorgegeben.

Zusätzlich zu den runlevelspezifischen Scripten in den `rc.*` Verzeichnissen gibt es unter Linux noch das Script `/etc/rc.local`. Dieses Script wird nach alles anderen init-Scripts aber noch vor dem Login-Prompt ausgeführt. Kommandos oder Dienste die in jedem Runlevel benötigt werden können auch in `rc.local` geschrieben werden.

[7] Seit Solaris 10 werden viele Services nicht mehr über die init-Scripte sondern über das Solaris Service Management Facility (`smf(5)`) gestartet.

5.2 Logging und Logfiles

Die verschiedenen Programme und Dienste eines Unix-Systems sind teilweise durchaus geschwätzig. Teilweise wollen sie einem nur mitteilen, dass sie genau das was wir von ihnen erwarten gemacht haben. Eine Mail zugestellt, eine Proxyanfrage beantwortet usw.. Viel wichtiger sind Meldungen und Warnungen über Probleme, fehlgeschlagene Aktionen oder gar kritische Fehler wie eine Kernel panic. All diese Meldungen werden in verschiedenen Logfiles protokolliert. Verantwortlich dafür ist der Syslog-Daemon `syslogd`.

Welche Meldungen `syslogd` protokolliert und wohin diese gespeichert werden, wird über die Datei `/etc/syslog.conf` konfiguriert. Die Meldungen werden dabei nach zwei Gesichtspunkten betrachtet: welcher Dienst hat die Meldung geschickt und wie wichtig ist sie. Man spricht von Facility und Priotity (auch level oder severity). Bei der Kombination aus `facility.priority` spricht man vom Selector. Das Ziel der Meldungen wird auch als action bezeichnet. Tabelle 5.2 listet alle vorhandenen Facilities auf und Tabelle 5.3 auf Seite 45 stellt die möglichen Priorities je nach Relevanz in absteigender Reihenfolge zusammen. Einträge in der `syslog.conf` haben folgendes Format:

```
facility.priority          destination
```

syslog Facilities	
facility	Bedeutung
`auth` bzw. `authpriv`	Security und Authorisationsmeldungen. Unter Linux wird `authpriv` statt `auth` verwendet
`cron`	Meldungen von cron und at (Siehe Kapitel 5.3
`daemon`	Meldungen von Daemons ohne eigene Kategorie
`ftp`	Meldungen des ftp-Daemons (nur Linux)
`kern`	Kernel-Meldungen
`lpr`	Medungen des Druckersystems
`mail`	Meldungen des Mailsystems
`news`	Meldungen Usenet
`syslog`	Interne Meldungen vom `syslogd`
`user`	Medlungen von User Prozessen, default facility
`uupc`	Unix to Unix Copy System
`local0-7`	frei verwendbar
*	alle facilities

Tabelle 5.2: syslog facilities

Pro Zeile kann man mehrere „Selectoren“ verwenden wenn diese durch ein „`;`“ getrennt werden[8]. Zusätzlich kann das Verhalten vom `syslogd` noch über das Asterisk (`*`), das Ausrufezeichen (`!`)und das Gleichheitszeichen (`=`) beeinflußt werden.

Ein einfaches Beispiel für einen `syslog.conf` Eintrag ist folgender:

[8]Hier das Semikolon nicht mit dem Komma verwechseln. Ein `kern.err,mail.warn` wird alle Kernel und Mail Meldungen ab dem Level `warn` loggen.

syslog Priorities	
priority	Bedeutung
`emerg`	Worst Case. System Panic. All hands abandon ship!
`alert`	Alarm der sofortiges Eingreifen erfordert
`crit`	kritischer Zustand eines Dienstes oder Systembestandteils
`err`	allgemeiner Fehler
`warning` bzw. `warn`	Warnungen, nicht gravierend aber auch nicht normal
`notice`	besondere Situation aber kein Fehler. Einträge mit priority `notice` müssen in einer extra Zeile erfolgen
`info`	normaler Betrieb
`debug`	Debug Meldungen
`none`	keinerlei Meldung von entsprechender facility protokollieren

Tabelle 5.3: syslog priorities

```
mail.info          /var/log/mail
```

Damit werden alle Meldung des Mailsystems mit der Priorität info und höher in die Datei `/var/log/mail` geschrieben. Sollen nur die Meldung von genau einer Priorität erfasst werden, wird die mit dem = angezeigt: `mail.=info`. Folgender Eintrag würde jede `crit` Meldungen aller Facilities nach `/var/log/critical` schreiben, ausgenommen alle Kernel-Meldungen:

```
*.=crit;kern.none          /var/log/critical
```

Um Meldung nur bis zu einer bestimmten Priorität zu erfassen, gibt nach die ersten nicht mehr zu loggende Priorität mit vorangestelltem `!` an. Im folgenden Beispiel werden alle Kernel-Meldungen bis zur Priorität `warning` nach `/var/adm/kernel-info` geschrieben, alle Meldungen ab `err` werden nicht mitgeschrieben.

```
kern.!err          /var/adm/kernel-info
```

Dabei darf man „`!`“ und „`=`“ auch zu einem „alles, ausser genau dieser Priorität“ kombinieren: `mail.!=info`.

Als Ziel der `syslog`-Meldungen können neben normalen Dateien auch named Pipes/FIFOs (gekennzeichnet durch ein vorangestelltes „`|`“), andere Hosts (gekennzeichnet durch ein „`@`“ vor dem Hostnamen), einzelne bzw. mehrere durch „,“ getrennte User oder auch alle eingeloggten User („`*`“) angegeben werden.

Die meisten Logfiles liegen unter `/var/log` (Linux, Solaris) bzw. `/var/adm` (Solaris).

5.3 Prozessmanagement

Die Überwachung von Prozessen und, zu einem gewissen Teil, auch die Beeinflussung der Prozessprioritäten gehören zum täglichen Handwerk eines Admins. Als Prozess verstehen wir im wesentlichen ein laufendes Programm ohne Betrachtung um was es sich dabei genau handelt.

5.3.1 Prozessattribute und Eigenschaften

Jeder Prozess verfügt über diverse Attribute, über die das Betriebssystem die diversen Prozesse verwaltet. Diese Attribute sind auch für uns als Admins sehr wichtig, um das Zusammenspiel der Prozesse untereinander zu verstehen.

Die Prozess ID (PID) eines jeden Prozesses ist einzigartig auf dem jeweiligen System. Es sollten auf einem System nie zwei unterschiedliche Prozesse mit derselben PID existieren. Da jeder Prozess durch einen anderen Prozess gestartet wurde, ist für jeden Child-Prozess auch die PID seines Parent-Process bekannt, die PPID. Wie Dateien gehoeren Prozesse auch immer einem User und einer Gruppe. Nur der Eigentümer eines Prozesses und der Superuser root darf die Prozesseigenschaften ändern oder ihn beenden. Die „Besitzverhältnisse“ werden in der Regel vom Parent an seine Child-Prozesse vererbt. All diese und noch weitere Attribute wie die Priotität mit der der Prozess Rechnzeit von der Prozessverwaltung, dem Scheduler, zugewiesen bekommt, in welcher Umgebung er läuft oder welchen Status er momentan innehat ist im Pseudifilesystem unter `/proc`[9] abgelegt. Für jeden laufenden Prozess gibt es unter `/proc` ein Verzeichnis `PID`.

```
# ls /proc/2194/
cmdline  cpu  cwd  environ  exe  fd  maps  mem  mounts  root  stat  statm  status
```

Normalerweise muß man aber selber in den `/proc/$PID` Verzeichnissen nichts ändern.

5.3.2 Prozesslisten abfragen

Alle Prozessinformationen können durch diverse Tools abgefragt werden. Das bekannteste davon ist `ps`, process status. Wird ps ohne Optionen aufgerufen zeigt es lediglich die Prozesse an, die demselben User gehören und auf demselben Terminal (`TTY`) wie `ps` gestartet wurden.

```
# ps
   PID TTY      TIME CMD
 27986 pts/2    0:00 bash
 28126 pts/2    0:00 ps
```

Bisschen mager. Die gebräuchlichsten Aufrufe sind unter Solaris `ps -afe` (full listing aller laufenden Prozesse, siehe Abbbildung 5.1 auf Seite 47) und unter Linux `ps aux` (alle laufenden Prozesse im user-oriented format, siehe Abbbildung 5.2 auf Seite 48). Es gibt etliche Möglichkeiten `ps` zu beeinflussen. Am besten man sieht sich die Manpage an und experimentiert etwas herum.

```
# ps -afe
     UID   PID  PPID  C    STIME TTY    TIME CMD
    root     0     0  0   Mar 04 ?      0:03 sched
    root     1     0  0   Mar 04 ?      0:21 /etc/init -
    root     2     0  0   Mar 04 ?      0:03 pageout
    root     3     0  0   Mar 04 ?     1532:16 fsflush
    root   338     1  0   Mar 04 ?      0:00 /usr/lib/saf/sac -t 300
    root   221     1  0   Mar 04 ?     14:43 /usr/sbin/nscd
 postfix  3964   454  0   Sep 30 ?      1:20 qmgr -l -t fifo -u
  nobody 14815   164  0   Mar 10 ?      0:01 fs
    root 23178   164  0   Feb 24 ?      0:00 in.tnamed
    root 29010   292  0 09:42:49 ?      0:01 /usr/sbin/sshd2
    root 14812   164  0   Mar 10 ?      0:01 rpc.metamedd
  drizzt 24714 24705  0   Mar 26 pts/6 63:43 /opt/Irssi/bin/irssi
    root 14817   164  0   Mar 10 ?      0:02 rpc.ttdbserverd
  [...]
```

Abbildung 5.1: Gekürzte Ausgabe von `ps -afe` unter Solaris

`ps` wird häufig in Kombination mit `grep` verwendet um nach Prozessen zu suchen. Wenn man eine Übersicht über die laufenden Prozesse haben will bietet sich das Tool `top` an. `top` ist nicht automatisch auf jeder Solaris Installation verfügbar. Seit Solaris 8 gibt es jedoch `prstat`[10] welches ebenfalls seinen Zweck erfüllt.

`top` zeigt nicht nur eine automatisch aktualisierte Prozessliste an, sondern auch Informationen über die CPU Auslastung, Prozessprioritäten, Speicherverbrauch, IOwait, die Uptime und Load des Systems usw.. Schickt man `top` ein `?` gelangt man zu einer kleinen Hilfeseite. Man kann direkt aus `top` heraus viele der alltäglichen Administrationsaufgaben laufende Prozesse betreffend erledigen. Es lassen ich Prozesse killen (`k`) oder neu priorisieren (`r`). Ebenso läßt sich die Darstellung anpassen. Sortierung nach Speicherverbrauch (`M`) oder Ausblenden bestimmer Informationen sind möglich.

5.3.3 Vorder- und Hintergrundprozesse

Laufende (interaktive) Prozesse können vom Anwender beliebig zwischen Vorder- und Hintergrund verschoben werden. Lang laufende `grep`-Suchen können so im Hintergrund weiterlaufen während man im Vordergrund weiter arbeitet. Man kann einen Prozess entweder direkt so starten, dass er direkt im Hintergrund läuft oder den laufenden Prozess in den Hintergrund verschieben.

Um einen Prozess direkt im Hintergrund zu starten hängt man einfach ein „Ampersand“ `&` an dem Befehl an.

```
# grep -i kernel /var/adm/messages &
```

[9] Manpage zu `proc` in der Sektion 5 bei Linux und Sektion 4 bei Solaris.

[10] Topping top in Solaris 8 with prstat, `http://developers.sun.com/solaris/articles/prstat.html`

```
# ps aux
USER     PID %CPU %MEM    VSZ   RSS TTY   STAT START   TIME COMMAND
root       1  0.0  0.1   2064   404 ?     Ss   08:12   0:00 init [5]
root       2  0.0  0.0      0     0 ?     S<   08:12   0:00 [migration/0]
root       4  0.0  0.0      0     0 ?     S<   08:12   0:00 [watchdog/0]
root       5  0.0  0.0      0     0 ?     S<   08:12   0:00 [events/0]
root      82  0.0  0.0      0     0 ?     S<   08:12   0:00 [cqueue/0]
root      85  0.0  0.0      0     0 ?     S<   08:12   0:00 [khubd]
root     379  0.0  0.0      0     0 ?     S<   08:13   0:00 [kauditd]
root     413  0.0  0.1   2992   284 ?     S<s  08:13   0:00 /sbin/udevd -d
root     690  0.0  0.0      0     0 ?     S<   08:13   0:00 [kgameportd]
root    4185  0.0  0.2  12048   484 ?     S<sl 08:13   0:00 /sbin/audispd
root    4201  0.0  0.1  10236   276 ?     Ss   08:13   0:00 /usr/sbin/restorecond
root    4212  0.0  0.2   1720   492 ?     Ss   08:13   0:00 syslogd -m 0
root    4215  0.0  0.1   1672   292 ?     Ss   08:13   0:00 klogd -x
root    4241  0.0  0.1   2156   380 ?     Ss   08:13   0:00 mcstransd
rpc     4254  0.0  0.1   1804   420 ?     Ss   08:13   0:00 portmap
root    4289  0.0  0.2   1920   620 ?     Ss   08:13   0:00 rpc.statd
root    4325  0.0  0.1   5432   316 ?     Ss   08:13   0:00 rpc.idmapd
dbus    4344  0.0  0.4  13120   916 ?     Ssl  08:13   0:00 dbus-daemon --system
root    4456  0.0  0.4  10832  1052 ?     Ssl  08:13   0:00 automount
root    4475  0.0  0.1   1672   368 ?     Ss   08:13   0:00 /usr/sbin/acpid
root    4506  0.0  0.3   6988   804 ?     Ss   08:13   0:00 /usr/sbin/sshd
[...]
```

Abbildung 5.2: Gekürzte Ausgabe von `ps aux` unter Linux

```
[1] 23016
```

Laufende Prozesse verschiebt man mit `CRTL-Z (^Z)` in den Hintergrund. Diese werden damit allerdings auch gestoppt und können mit `bg` wieder getsartet werden.

```
# some-process
^Z
[1]+  Stopped                 some-process
# bg
[1]+ some-process &
```

Welche Prozesse derzeit im Hintergrund laufen und welche Job-Nummer ihnen zugewiesen wurde kann man sich ueber das Kommando `jobs` anzeigen lassen.

```
# jobs
[1]-  Exit 1                  grep -i kernel /var/adm/messages
[2]+  Stopped                 vi hostname.qfe5
```

Die Ausgaben von `jobs` unterscheiden sich dabei je nach Shell etwas. Mit `fg %<`*job nummer*`>` kann ich mir einen Prozess wieder in den Vordergrund holen.

```
top - 11:23:27 up 22 days,  4:02,  2 users,  load average: 0.00, 0.00, 0.00
Tasks: 100 total,   1 running,  99 sleeping,   0 stopped,   0 zombie
Cpu0  :  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu1  :  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu2  :  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu3  :  0.3%us,  0.0%sy,  0.0%ni, 99.3%id,  0.3%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   4025648k total,   657796k used,  3367852k free,   219540k buffers
Swap:  4883752k total,        0k used,  4883752k free,   248144k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
16724 drizzt    20   0  2308 1116  860 R    0  0.0   0:00.10 top ´
    1 root      20   0  2712 1736  612 S    0  0.0   0:01.98 init
    2 root      15  -5     0    0    0 S    0  0.0   0:00.00 kthreadd
    3 root      RT  -5     0    0    0 S    0  0.0   0:00.01 migration/0
    4 root      15  -5     0    0    0 S    0  0.0   0:00.09 ksoftirqd/0
    5 root      RT  -5     0    0    0 S    0  0.0   0:00.00 watchdog/0
    6 root      RT  -5     0    0    0 S    0  0.0   0:00.01 migration/1
    7 root      15  -5     0    0    0 S    0  0.0   0:00.04 ksoftirqd/1
    8 root      RT  -5     0    0    0 S    0  0.0   0:00.00 watchdog/1
    9 root      RT  -5     0    0    0 S    0  0.0   0:00.01 migration/2
   10 root      15  -5     0    0    0 S    0  0.0   0:00.10 ksoftirqd/2
   11 root      RT  -5     0    0    0 S    0  0.0   0:00.00 watchdog/2
   12 root      RT  -5     0    0    0 S    0  0.0   0:00.01 migration/3
   13 root      15  -5     0    0    0 S    0  0.0   0:00.14 ksoftirqd/3
```

Abbildung 5.3: Gekürzte Ausgabe von `top` auf einer Mehrprozessormaschine

5.3.4 Prozesse beenden

Sollte einmal nötig einen einen Prozess außerplanmäßig beenden zu müssen kommen die Befehle `kill`, `killall`[11] und `pkill` ins Spiel. Diese drei Tools erlauben es dem Admin Signale[12] an Prozesse zu schicken und sie so zu beeinflussen. Dabei muss `kill` mit einer PID aufgerufen werden während sich `killall` und `pkill` auch mit Namen von Prozessen zufriedengeben. Ein einfaches `kill <PID>` sendet ein SIGTERM (termination signal) an den Prozess. Sollte das nicht ausreichen um den Prozess zu beenden, kann man mit `kill -9 <PID>` ein SIGKILL (kill signal) senden nachdem der Prozess beendet sein sollte. Mit einem SIGHUP Signal (hangup signal, `kill -1 <PID>`) kann man Daemons (z.B. `inetd` dazu veranlassen ihre Konfiguration neu einzulesen. `killall` und `pkill` arbeiten ähnlich, werden aber statt mit der PID mit dem Namen des zu killenden Prozesses aufgerufen. Wer gerne Fussnoten überliest weil da eh nur Mist drinsteht sei nochmal drauf hingewiesen, dass ein `killall` unter Solaris alle laufenden Prozelle killt. Ohne wenn und aber.

[11] **ACHTUNG!** `killall` unter Solaris stoppt alle aktiven Prozesse.

[12] Eine Liste der möglichen Signale findet man unter Solaris mit `man -s 3head signal` und unter Linux mit `man 7 signal`.

5.3.5 Prozesse priorisieren

Eher selten muss man die Priorität mit der ein Prozess Rechenzeit zugewiesen bekommt per Hand anpassen. Normalerweise weist der Scheduler den Prozessen abhängig von ihrer Priorität im round robin Verfahren Rechenzeit zu. Die Priorität wird durch eine ganze Zahl symbolisiert und ist umso höher je kleiner die Zahl ist. Pro Prozess gibt es zwei Prioritätswerte: die nice number stellt die erwünschte Priorität dar und die wirkliche aktuelle Priorität mit der der Prozess behandelt wird. Die nice number, `NI` in der Ausgabe von `ps` oder `top`, kann vom Prozesseigentümer und vom `root` angepasst werden. Die aktuelle Priorität wird im `ps` oder `top` unter `PRI` aufgeführt.

Die angestrebte nice number kann entweder beim Starten des Prozesses mit vorangestelltem `nice` direkt angeben oder nachträglich mit `renice` erbitten.

5.3.6 Zeitgesteuerte Prozesse mit `cron` und `at`

Unter Unix gibt es zwei Daemons, die einem Sysadmin erlauben den Start eines Prozesses an bestimmte Zeitvorganen zu knüpfen. Um bestimmte Prozesse in einer festen Zeitreihe ablaufen zu lassen vertraut man diese dem `cron`-System an. Soll ein Prozess nur einmalig zu einer vorgegebenen Zeit laufen bedient man sich des `at-Daemons.`

Das `cron` System

Viele Vorgänge und Aufgaben, wie wöchentliche Backups oder Statistiken über den Mailverkehr, müssen regelmäßig wiederholt werden. Alle diese Aufgaben kann man mit sogenannten Cronjobs zu vorgegebenen Zeiten automatsich starten. Dazu startet das System beim booten den Dienst `cron`. Wer letztendlich `cron` benutzen darf wird über die Dateien `/etc/cron.allow` und `/etc/cron.deny` geregelt. Unter Solaris liegen diese beiden Dateien in `/etc/cron.d/`. Existiert die `cron.allow` muss ein normaler User in ihr gelistet sein, um `cron` nutzen zu dürfen. Existiert sie nicht, darf er nicht in der `cron.deny` gelistet weis wenn er `cron` benutzen will. Existieren weder `cron.allow` noch `cron.deny` darf nur der Superuser `root` Cronjobs erstellen.

Cronjobs werden in `crontab`-Dateien gespeichert. Diese liegen unter `/var/spool/cron` (Linux) bzw. `/var/spool/cron/crontabs/` (Solaris). Man kann sich seine aktuelle `crontab`-Datei mit `crontab -l` anzeigen lassen oder sie mit `crontab -e` im $EDITOR bearbeiten. `crontab`-Dateien werden von `cron` zeilenweise gelesen. Jede Zeile stellt eine von `cron` durchzuführende Aufgabe dar und ist folgendermaßen aufgebaut:

```
minute      stunde      tag-im-monat      monat      wochentag      kommando
```

Die ersten fünf Felder definieren den Zeitpunkt an dem `kommando` ausgeführt werden soll. In Tabelle 5.4 ist aufgeführt welche Werte in die einzelnen Felder eingesetzt werden können.

Statt absoluter Werte können auch durch Komma getrennte Listen (`0,10,20,30`), Bereiche (`8-18`) und Schrittweiten mittels `/n` angegeben werden. Sehen wir uns ein paar Beispiele an.

crontab Zeitsteuerung		
Feld	Bedeutung	mögliche Werte
minuten	Minuten der Stunde	`0-59` und *
stunde	Stunde vom Tag	`0-23` (`0`=Mitternacht) und *
tag-im-monat	Monatstag numerisch	`1-31` und *
monat	Monat numerisch	`1-12` und *
wochentag	Wochentag numerisch	`0-6` (`0`=Sonntag) und *

Tabelle 5.4: crontab Zeitsteuerung

```
15     20     *     *     6    echo "‘Samstag, 20:15 Uhr."’
```

Dieser Job würde jeden Samstag (Eintrag `6` als Wochentag) umd 20:15 Uhr gestartet.

```
0,10,20,30,40,50  8-18  *  *  1-5  echo "‘Alle 10 Minuten, Wochentags, 8 bis 18 Uhr."’
```

Die Liste `0,10,20,30,40,50` für „alle vollen 10 Minuten“ könnte man auch als `0-59/10` bzw. `*/10` schreiben.

```
0  0  1,15  *  *  echo "‘Jeden 1. und 15. eines jeden Monats genau um Mitternacht."’
```

Bereiche und Listen lassen sich durch Komma getrennt ebenfalls kombinieren: `0,4,8-16,20` oder `8-12,14-18`.

Bei den auzuführenden Kommandos müssen Prozentzeichen „`%`“ falls vorhanden mit einem Backslash escaped werden, da sie sonst als Zeilentrenner interpretiert werden. Leerzeilen oder Zeilen beginnend mit „#“ (Kommentare) werden nicht ausgewertet.

Für Cronjobs werden automatisch einige Umgebungsvariablen gesetzt. Benutzername (`LOGNAME`) und Homedir (`HOME`) werden aus der `/etc/passwd` übernommen. Als default Shell wird unter Solaris die Bourne Shell verwendet (`SHELL=/bin/sh`), Linux übernimmt (je nach Distribution) auch diesen Wert aus der `passwd`. Der Pfad (`PATH`) ist fuer normaler User `/usr/bin`, für `root` hingegen `/usr/sbin:/usr/bin`. Cronjob `PATH` oder der der Shell unter scheiden sich also. Sollen für die durch `cron` ausgeführten Kommandos besondere Umgebungsvariablen gelten, so müssen diesen am Anfang der `crontab`-Datei bzw. im Kommandofeld durch `VARIABLE=WERT` gesetzt werden. Systemweit können diese Einstellungen unter Solaris über die Datei `/etc/default/cron` erfolgen.

Normalerweise werden alls durch `cron` gestarteten Job über `syslog` Protokolliert und die Ausgaben oder Fehlermeldungen an den Eigentümer des Jobs (`LOGNAME`) gemailt. Soll die Protokollierung über `syslog` ausbleiben, seltzt man ein „`-`“ vor das Kommando. Will man auf die gemailte Ausgabe verzichten, kann man `stdout` und `stderr` wie in Kapitel 3.4 angesprochen umleiten. Hier nocheinmal das Beispiel aus Kapitel 3.4.

```
1 2 * * * [ -x /usr/sbin/rtc ] && /usr/sbin/rtc -c > /dev/null 2>&1
```

Mittlerweile sollte jeder diesen Eintrag, abgesehen vielleicht vom `test`-Ausdruck `[ -x /usr/sbin/rtc ]`[13], komplett verstehen und beschreiben können.

[13]Mit `test` können Konditionen überprüft und zur Bedingung gemacht werden. Die `[...]` umschließen die zu testende Kondition. In diesem Fall mird mit `-x` geprüft, ob die Datei `/usr/bin/rtc` existiert und ausführbar ist.

Das at-System

Soll ein Job nur einmalig zu einer bestimmten Zeit laufen bietet sich als Scheduler der Dienst `at` an. Er wird ebenso wie `cron` beim booten gestartet. Analog zu `cron` wird über die Dateien `/etc/at.allow` und `/etc/at.deny` (Linux) bzw. `/etc/cron.d/at.allow` und `/etc/cron.d/at.deny`[14] festgelegt wer mit `at` arbeiten darf.

Als Argument verarbeitet `at` Zeitangaben, die durchaus komplex sein können. Tabelle 5.5 führt einige Beispiele auf.

Beispiele für `at` Zeitangaben	
Beispiel	Bedeutung
at now + 20 minutes	Job startet in 20 Minuten ab Eingabe.
at teatime Dec 24	Job startet am 24. Dezember um 16 Uhr
at 15:45 20.03.09	Job startet am 20. März 2009 um 15:45 Uhr
at 2 am ZULU + 3 days	Job startet um 2 AM nach ZULU (GMT) Zeit plus drei Tage.

Tabelle 5.5: crontab Zeitsteuerung

Wenn die Zeit in der Form „`HH:MM`“ angegeben wird, startet `at` den Job zum nächstmoglichen Zeitpunkt der auf `HH:MM` zutrifft. Wenn man um 16:10 Uhr einen Job für 16:00 Uhr einstellt wird dieser am nächsten Tag gestartet.

```
# date
Sun Nov 12 16:10:30 CET 2008
# at 16:00
at> echo foobar
at> <EOT>
job 4 at 2008-11-12 16:00
```

Ausser Zeitangaben in der Form `HH:MM` sind `now`, `teatime`, `noon` und `midnight` zulässig. Zusätzlich kann der Starttermin durch „`+ anzahl zeiteinheit`“ verschoben werden, wobei als Zeiteinheiten `minute(s)`, `hour(s)`, `day(s)` oder `week(s)` akzeptiert werden.

Nachdem man `at <zeit>` gestaretet hat, gelang man zu einem extra Prompt(„`at>` “) in dem na die auszuführenden Kommandos eingibt. Um diesen Eingabemodus zu verlassen und den `at`-Job zu speichern drückt man `CTRL-D` (`^D`). Alternativ können durch den Schalter `-f` auch Textdateien mit den auszuführenden Kommandos eingelesen werden. Danach gibt `at` die ID des Jobs sowie den Ausführungszeitpunkt aus. Den Stand bzw. die Anzahl der Jobs in der `at`-Queue zeigt der Befehl `atq` an. Mittels `atrm <job-id>` kann man einen Job löschen.

`at` speichert auch die wichtigsten Umgebungsvariablen, die zum Zeitpunkt der Joberstellung gültig waren[15].

Wie auch bei `cron` werden `stdout` und `stderr` der Kommandos aus dem `at`-Job per Mail dem Job-Eigentümer zugeschickt.

[14] Laut Manpage sollen die Dateien in `/usr/lib/cron/` liegen, allerdings ist `/usr/lib/cron/` ein Symlink nach `/etc/cron.d/`.

[15] Ausgenommen `TERM`, `DISPLAY`, _, `EUID` und `UID`

5.4 Softwarepakete

5.4.1 Pakete installieren

5.4.2 Pakete Patchen

5.4.3 Pakete updaten/upgraden

Falls eine Software veraltet ist und gegen eine neue mit zusätzlichen Features ausgetauscht werden soll, müssen die entsprechenden Pakete geupdatet werden. Unter Solaris werden neue Features in Software häufig über die (Recommended) Patches hinzugefügt[16]. Red Hat patcht, soweit ich weiss, bei Bedarf die Software des entsprechenden Releases fügt aber keine neuen Features hinzu oder ermöglicht gar einen Versionswechsel. In solchen Fällen muss man sich die Software entweder selber kompilieren, was momentan den Umfang des Skripts sprengen würde, oder man nimmt fertige Pakete von „Drittanbietern“.

Im Folgenden will ich so ein Update eines Pakets mal am Beispiel vom Postfix unter Red Hat Enterprise Linux 3 zeigen. Standardmaessig ist der Postfix bei RHEL3 in der Version 2.0.16 dabei:

```
# rpm -q postfix
postfix-2.0.16-14.RHEL3
```

Fuer Red Hat gibt es halboffizielle Postfix RPMs die von Simon J[17]. Mudd auf seiner Postfix Seite unter `http://postfix.wl0.org/` bereitgestellt werden. Ich hab mich für die Version 2.2.4 entschieden und das RPM auf den Server heruntergeladen. Vor dem Upgrade wird der laufende Postfix gestoppt.

```
# postfix stop
postfix/postfix-script: stopping the Postfix mail system
```

Das neue RPM kann entweder mit `rpm -Uvh` oder `rpm -Fvh` eingespielt werden. Der Switch `-U` macht ein Upgrade oder installiert das Paket falls voher noch keine andere Version installiert war. Der Switch `-F`, freshen, macht nur ein Upgrade bei schon installierten Paketen. Zusätzlich rufen wir `rpm` noch mit dem Switch `--repackage` auf. Dadurch wird eine Sicherungskopie der installierten Pakets[18] nach `/var/spool/repackage`[19].

```
# rpm -Uvh --repackage postfix-2.2.4-2.rhel3.i386.rpm
Preparing...                ########################################### [100%]
Repackaging...
   1:postfix                ########################################### [100%]
Upgrading...
```

[16] „A patch can also provide a new feature or an enhancement to a particular software release.“, siehe `http://www.sun.com/bigadmin/sundocs/articles/patch-types.jsp`

[17] Das Zwischeninitial ist der Doktortitel des kleinen Mannes

[18] Die angepassten Konfigurationsdateien werden nicht mit gesichert!

[19] Im Normalfall landet es dort. Wird über das Macro `%_repackage_dir` vorgegeben

```
   1:postfix               ########################################### [100%]
warning: /etc/postfix/main.cf created as /etc/postfix/main.cf.rpmnew
[...]
Editing /etc/postfix/master.cf, adding missing entry for tlsmgr service

    Note: the following files or directories still exist but are no
    longer part of Postfix:

     /etc/postfix/pcre_table /etc/postfix/regexp_table
     /usr/share/man/man8/nqmgr.8.gz
     /usr/share/doc/postfix-2.0.16/samples/sample-aliases.cf
     /usr/share/doc/postfix-2.0.16/samples/sample-auth.cf
     [...]
```

Nachdem RPM fehlerfrei durchgelaufen ist können wir überprüfen welches Paket installiert ist und den Service wieder starten.

```
# rpm -q postfix
postfix-2.2.4-2.rhel3
# postfix start
postfix/postfix-script: starting the Postfix mail system
```

Das Backuppaket wurde auch erstellt, so dass ein roll-back möglich wär.

```
# ls -l /var/spool/repackage/
total 2164
-rw-r--r--   1 root  root  2211513 Oct 29 10:22 postfix-2.0.16-14.RHEL3.i386.rpm
```

5.4.4 Pakete deinstallieren

6 Netzwerke

> A name indicates what we seek.
> An address indicates where it is.
> A route indicates how we get there.
>
> *(Jon Postel, RFC 791)*

In diesem Kapitel werden wir kurz das OSI Modell[1], das Adress Resolution Protocol[2] und die IP Adressierung wiederholen, bevor wir uns der Konfiguration von Netzwerkinterfaces widmen. Dabei betrachten wir nur IPv4[3] und lassen IPv6[4] außen vor.

6.1 Das Open Systems Interconnect Modell

Bereits 1979 begann die Entwicklung eines mehrschichtigen Referenzmodells mit dessen Hilfe Kommunikationsprotokolle beschrieben und entwickelt werden können. Dieses Modell regelt welche Anforderungen die im einzelnen beteiligten Protokolle erfüllen müssen um eine reibungslose Kommunikation zu gewährleisten. Die Standardisierung des Referenzmodells erfolgte 1983.

Das OSI Modell besteht aus sieben Schichten von Diensten die eine Grundlage für Netzwerkommunikation und Kommunikationsprotokolle bilden. Jede Schicht (layer) muss bei der Kommunikation die ihr zugedachten Aufgaben erfüllen und reicht danach die bearbeiteten Daten an die nächst niedrigere (beim senden) bzw. nächst höhere (beim empfangen) Schicht weiter. Jede Schickt verpackt die Daten zusätzlich indem sie einen Header und einen Footer hinzufügt. Dieser Vorgang wird data encapsulation genannt und ermöglichet die Kommunikation der einzelnen Layer untereinander.

Abbildung 6.1 auf Seite 56 zeigt den Aufbau des Modell mit einigen bekannten Protokoll- und Servicebeispielen.

6.1.1 Application Layer / Anwendungsschicht

Der Application layer stellt als oberste Schicht des OSI Modells den darüberliegenden Anwendung eine Schnitstelle zum Netzwerk dar. Dies können Browser, Mail Clients und

[1] Durch die Internationale Organisation für Normung (ISO) im Jahrt 1983 standardisiert.
[2] RFC 826 http://tools.ietf.org/html/rfc826
[3] RFC 791 http://tools.ietf.org/html/rfc791
[4] Unter anderem RFC 2460 http://tools.ietf.org/html/rfc2460

Open Systems Interconnect Modell				
OSI Schicht		Aufgabe	Protokoll	Einheit
7	Application	Anwendungsorientiert	HTTP, FTP, SMTP	Daten
6	Presentation			
5	Session			
4	Transport	Transportorientiert	TCP UDP	Segmente
3	Network		IP ICMP IGMP	Pakete
2	Data Link		Ethernet, Token Ring	Frames
1	Physical			Bits

Abbildung 6.1: OSI Schichtenmodell

ähnliches sein.

6.1.2 Presentation Layer / Darstellungsschicht

Diese Schicht setzt die verschiedenen Datenformate beteiligter Systeme in eine einheitliche Form um und stellt sicher, dass die Daten von der Anwendungsschicht des Kommunikationspartners auch gelesen werden können.

6.1.3 Session Layer / Sitzungsschicht

Diese schickt kümmert sich um die Verbindung, session, und die Prozesskommunikation beteiligter Systeme.

6.1.4 Transport Layer / Transportschicht

Die Transportschicht stellt eine Ende-zu-Ende Kommunikation für Sender und Empfänger zur Verfügung und sorgt dafür, dass die Daten auch beim Empfänger ankommen.

6.1.5 Network Layer / Vermittlungsschicht

Die Vermittlungsschicht kümmert sich um die Weiterleitung der Datenpakete, stellt die Adressierung (z.B. IP) sicher und verwaltet die Routingtabellen.

6.1.6 Data Link Layer / Sicherungsschicht

Die Sicherungsschicht soll eine zuverlässige und fehlerfreie Kommunikation durch die unter ihr liegenden Netzwerkschichten gewährleisten. Die übetragenden Frames enthalten eine Prüfsumme und können so bei Bedarf neu angefordert werden. Die Sicherungsschicht ist in zwei Unterschichten geteilt: die obere namens Logical Link Control (LLC) und darunter die Media Access Control (MAC).

6.1.7 Physical Layer / Bitübertragungsschicht

Die unterste Schickt des OSI Modells definiert die zugrundeliegende Hardware und die Art der Signale welche für die Datenübertragung genutzt werden.

6.2 TCP/IP Referenzmodell

Das TCP/IP Modell ist wie das OSI Referenzmodell in Schichten aufgebaut. Es baut auf dem vom US Verteidigungsministerium 1970 entwickelten DoD-Schichtenmodell auf.

Im Gegensatz zum OSI Modell besteht das TCP/IP Modell aus nur vier Schichten, die sich aber auf im OSI Modell wiederfinden. Abbildung 6.2 beschreibt die vier Schichten und nennt die vergleichbaren OSI Schichten.

TCP/IP Schichtenmodell		
TCP/IP-Schicht	≡ OSI-Schicht	Protokoll Beispiel
Application layer	5-7	HTTP, FTP, SMTP
Transport Layer	4	TCP, UDP, SPX
Internet Layer	3	IPv4, IPv6
Link Layer	1-2	Ethernet, FDDI, Token Ring

Abbildung 6.2: TCP/IP Schichtenmodell

6.2.1 Application layer / Anwendungsschicht

Die Anwendungsschicht beinhaltet Protokolle, die entsprechenden Anwendungen Dienst zur Verfügung stellen.

6.2.2 Transport Layer / Transportschicht

Das wichtigste Protokoll der Transportschicht, das Transmission Control Protocol (TCP), stellt eine End-zu-End-Verbindung her. Es arbeitet verbindungsorientiert und soll so eine zuverlässige Datenübertragung ohne Paketverluste garantieren. Verbindungen werden über einen three-way-handshake (`SYN`, `SYN,ACK`, `ACK,DATA`) hergestellt. Zu dieser Schicht gehoren aber auch „unzuverlässige“ Protokolle wie das verbindungslose User datagram Protocol (UDP).

6.2.3 Internet Layer / Internetschicht

Die Internetschicht kümmert sich um das Routing und die Vermittlung der Segmente und Pakete höherliegender Schichten als Datagramm zum nächsten Wegpunkt/Hop. Sie ist außerdem für die Adressierung der beteiligten Systeme sowie für Fragmentierung und Defragmentierung der Datagramme verantwortlich.

6.2.4 Link layer / Netzzugangsschicht

Die Netzzugangsschicht des TCP/IP-Models beinhaltet keine eigenen Protokolle, sondern verlässt sich auf die unteren beiden Schichten des OSI-Modells.

6.3 IP Adressierung und Subnetting

Im Grunde setze ich voraus, daß jeder der dieses Skript liest wenigstens in Ansätzen die IP-Adressierung verstanden hat. Wer noch völlig unbefleckt auf diesem Gebiet ist soll sich entweder im Internet schlau machen oder das Buch „TCP/IP Netzwerkadministration“ aus dem O'Reilly Verlag (ISBN 3897211793) besorgen. Ich werden das Thema, genauer gesagt die IPv4 Adressierung, daher nur kurz zur Wiederholung anschneiden.

6.3.1 IP-Adressen und Subnetze

IP-Adressen sind 32 Bit lang und werden in der Regel als „dotted decimal“ notiert: vier ganze Dezimalzahlen zwischen 0 und 255 und je durch einen Punkt getrennt. Jede Zahl stellt ein Oktett bzw. Byte dar. Hier im Vergleich die IP-Adresse 192.168.2.28 einmal binär und dezimal:

$$\underbrace{11000000}_{192.}.\underbrace{10101000}_{168.}.\underbrace{00000010}_{2.}.\underbrace{00011100}_{28}$$

Das erste Oktett aus den 8 Bits 11000000 entspricht dezimal also dem Wert 192. Wieso? Jedes der 8 Bits entspricht einem Wert. Von links nach rechts[5]: 128 64 32 16 8 4 2 1. Sind nun die ersten beiden Bits je 1 und alle folgenden Bits 0 muß man für den Dezimalwert die Werte der ersten beiden Bits addieren. 128+64=192. Genauso für das zweite Oktett 10101000: 128+32+8=168. Und so weiter. Eigentlich ganz einfach.

Zu jeder IP-Adresse gehoert auch eine Subnetzmaske welche die Größe des Netzwers in dem sich unsere IP befindet bestimmt. Diese Subnetzmaske teilt die IP in einen unveränderlichen Netzwerk- und einen veränderlichen Hostteil. Subnetzmasken können entweder ebenfalls dottet decimal oder als CIDR-Suffix gaschrieben werden. CIDR[6], Classless Inter-Domain Routing, wurde eingeführt um die IP-Adressräume effizienter zu nutzen und Routingtabellen verkleinern zu können. Vor CIDR gab es drei Subnetzgrößen. Class-A Netz (Subnetzmaske 255.0.0.0) mit 16777214 Hosts pro Netz, Class-B (255.255.0.0) mit 65534 Hosts pro Netz und schliesslich Class-C (255.255.255.0) mit 254 Hosts pro Netz. Wenn Example Corp. nun also ein offizielles Netz mit 600 Hosts gebrauchte hätte, wäre ein Class-C Netz zu klein und ein Class-B Netz eine gigantische Verschwendung von Adressen gewesen. Natürlich hätte man Example Corp. entsprechend viele Class-C Netze zuweisen können, dadurch wären aber die Routingtabellen weiter angewachsen.

[5] Ja, es wird binär von rechts nach links, von klein nach groß, gelesen, aber das finde ich persönlich beim Thema IP und Subnetze nur verwirrender.

[6] RFC 1518 (http://tools.ietf.org/html/rfc1518) und RFC 1519 (http://tools.ietf.org/html/rfc1519) von 1993

Ein Subnetzmaske bei der man früher von einem Class-C Netz gesprochen hat lässt sich also dotted decimal als 255.255.255.0 oder nach CIDR als Suffix /24 an die IP-Adresse anfügen: 192.168.2.28/24. Warum die 24? Weil bei einer Netzmaske von 255.255.255.0 die ersten 3 Oktette oder eben die ersten 24 Bits auf 1 stehen.

$$\underbrace{11111111}_{255.}.\underbrace{11111111}_{255.}.\underbrace{11111111}_{255.}.\underbrace{00000000}_{0}$$

Der Netzanteil der IP-Adresse ist, wie vorhin schon erwähnt, unveränderlich. Das heisst die IP-Adresse alle Hosts aus diesem Netz (192.168.2.0/24) beginnen mit 192.168.2. und unterscheiden sich erst in der letzten Zahl der IP, dem Hostteil.

$$\underbrace{192\ .\ 168\ .\ 2\ .}_{Netzwerkteil}\ \underbrace{28}_{Hostteil}$$

In jedem Netz mit n IP-Adressen koennen maximal n-2 Hosts existieren, denn die erste IP des Netzes ist fuer die Netzwerdadresse und die letzte IP fuer die Broadcastadresse reserviert[7]. In unserem Beispiel sind von den 256 möglichen IPs (192.168.2.0 bis 192.168.2.255) 254 für Hosts nutzbar.

6.3.2 Subnetting

Da Subnetting vielen angehenden (und teils auch gestandenen Admins) ein Dorn im Auge ist werde ich das Thema auch nicht herumkommen. Solltest du Subnetting mit links und verbundenen Augen beherrschen überspring dieses Teil, denn ich verwende letztendlich eine etwas andere Art der Berechnung als die „offiziell“ propagierte und will mögliche Verwirrung vermeiden.

In den meisten Lehrbüchern wird auf die Formel $2^n - 2$ bzw. 2^n verwiesen. Wenn man ein Subnetz mit Suffix /27 hat sind 27 Bits für den Netteil und die übrigen 5 Bits $(32-27 = 5)$ für den Hostteil vorgesehen. Pro Subnet können $2^5-2 = 30$ Hosts existieren. Mit Betrachtung auf ein Class-C Netz mit Suffix /24 verwendet unser Subnetz mit Suffix /27 zusätzliche 3 Bits für die Netzmaske. Es kann also $2^3-2 = 6$ bzw., da wir anständige Hardware besitzen die `ip subnet-zero`[8] verstehen, $2^3 = 8$ Subnetze. Auf die Weise, mit Bits zählen und Potenzen bilden, kommt man prima durch jedes Subnetting. Ich persönlich hab das zwar verstanden, fand es aber nicht sonderlich anschaulich und hab in all den Jahren so manchen Auszubildenden und CCNA-Anwärter daran verzweifeln sehen. Also: $2^n - 2$ und 2^n my ass!

Bleiben wir bei dem Beispiel von oben: ein Netz mit eine Netzmaske /27. Die 3 zusätzlichen Bytes im Vergleich zu /24 bewirken eine in dotted decimal geschriebene Netzmaske von 255.255.255.224. Von den 256 IPs (0 bis 255) eines /24 Netzes ziehe ich die 224 der /27 Netzmaske ab und ich komme auf meine 32 IPs pro Subnetz. Abzüglich

[7] Theoretisch kann die Netzadresse auch frei benutzt werden, solang sich keine Windows 9x Hosts im Netz befinden. Aber laut sagen darf man das eigentlich nicht. Jehova!

[8] Früher sollten das erste (all-zeros) und letzte (all-ones) Subnet nicht benutzt werden. Heute ist das allerdings kein Problem mehr. Siehe auch `http://www.cisco.com/application/pdf/paws/13711/40.pdf`

Netzwerkadresse und Broadcast bleiben 30 IPs fuer Hosts. Ich kann 256/32 = 8 /27 Subnetze bilden. Wer ungern teilt kann sich auch die Reihe „128 64 32 16 8 4 2“ vorstellen und dann einfach „an der 16 spiegeln“: bei 32 IPs gibt es 8 Netze, bei 64 Netzen nur 4 IPs pro Netz usw.. Klingt komisch, funktioniert aber auch. Supernetting ist so auch möglich.

Letztendlich gibt es verschiedene Möglichkeiten Subnetting im Kopf zu betreiben. Wichtig ist nur, das man es verstanden hat und seine Auswirkungen versteht. Zur Not gibt es ja auch noch `ipcalc`[9].

```
# ipcalc -n 192.168.2.1/27
Address:   192.168.2.1          11000000.10101000.00000010.000 00001
Netmask:   255.255.255.224 = 27 11111111.11111111.11111111.111 00000
Wildcard:  0.0.0.31             00000000.00000000.00000000.000 11111
=>
Network:   192.168.2.0/27       11000000.10101000.00000010.000 00000
HostMin:   192.168.2.1          11000000.10101000.00000010.000 00001
HostMax:   192.168.2.30         11000000.10101000.00000010.000 11110
Broadcast: 192.168.2.31         11000000.10101000.00000010.000 11111
Hosts/Net: 30                    Class C, Private Internet
```

6.4 ARP - Address Resolution Protocol

Das Address Resolution Protocol ist ein Protokoll der Netzzugangsschicht des TCP/IP Modells und ermittelt die zu einer IP-Adresse zugehörige 48Bit lange Hardwareadresse, auch MAC-Adresse, eines Netzwerkgeräts im lokalen Netz. Falls die MAC-Adresse eines Zielrechners noch nicht in der ARP-Tabelle ist, schickt der Host einen ARP-Request als Paket mit der Destination-IP des gesuchten Rechners per ARP-Broadcast (`ff-ff-ff-ff-ff-ff`) an alle Rechner im Lan. Findet ein Rechner seine eigene IP als Empfänger in dem ARP-Request antwortet er dem Absender mit seiner MAC-Adresse.

```
14:34:22.611372 arp who-has server-b.example.com tell server-a.example.com
14:34:22.611641 arp reply server-b.example.com is-at 00:03:ba:09:bf:51
```

Ist der Empfänger eines IP-Pakets im lokalen Netz, werden alle Pakete für diesen Empfänger direkt an seine MAC-Adresse gesendet. Liegt der Empfänger ausserhalb des LAN werden die Pakete an die MAC-Adresse des entsprechenden Gateways gesendet, die Destination-IP wird dabei nicht geändert.

Die ARP-Tabelle kann mit arp (Linux) bzw. arp -a (Solaris, unter Linux Ausgabe im BSD-Stil) abgefragt und bei Bedarf manipuliert werden. Der Switch -n unterdrückt die Namensauflösung.

```
# arp -a
Net to Media Table: IPv4
Device   IP Address               Mask      Flags   Phys Addr
------ -------------------- --------------- ----- ---------------
```

[9] Online Tool und Download unter `http://jodies.de/ipcalc`

```
eri0   torwaechter          255.255.255.255       00:14:4f:1f:b1:eb
eri0   subint               255.255.255.255 SP    00:03:ba:24:92:a9
eri0   oeko-pix             255.255.255.255       00:0b:46:22:25:91
eri0   gandhi               255.255.255.255       00:03:ba:6c:e4:0d
eri0   pix                  255.255.255.255       00:03:6b:f6:70:e6
eri0   zoidberg             255.255.255.255 SP    00:03:ba:25:6c:a9
eri0   BASE-ADDRESS.MCAST.NET 240.0.0.0       SM    01:00:5e:00:00:00
```

Wie lange Einträge in der ARP-Tabelle bestehen bleiben ist unterschiedlich. Unter Solaris liegt das `arp_cleanup_interval` von `/dev/arp` normalerweise bei 300000 Millisekunden (5 Minuten). Unter Linux liegt der Defaultwert für die `gc_stale_time`[10] bei 60 Sekunden. Notfalls muß ein Eintrag mit `arp -d <`*ipadresse*`>` manuell aus der Tabelle gelöscht werden.

6.5 Routing

Wenn man einen Host erreichen will, der nicht im eigenen (Sub)netz liegt, müssen die Pakete an jemanden geschickt werden, der entweder den Weg zum Ziel kennt oder zumindest eine Ahnung hat in welche Richtung es geht. Das ist der Router. Während Hubs nur auf Layer 1 und Switche nur auf Layer 2 des OSI-Modells arbeiten und daher Pakete nur im lokalen Netz verteilen, arbeitet ein Router auf Layer 3 und ist in mehreren Netzen beheimatet zwischen denen er die Pakete weiterleiten kann.

Um also Daten in entferne Netze zu schicken, muss ein Host zumindest einen Router kennen, an den er diese Pakete schicken kann. Das ist der Defaultrouter. Der Host adressiert die Daten an die IP-Adresse des Zielhosts, schickt sie aber, wie im letzten Abschnitt schon erwähnt, an die MAC-Adresse vom Router. Dieser sucht in seiner Routing Tabelle nach dem besten Weg zum Ziel, im Zweifel ist das der Defaultrouter des Routers, und leitet sie entsprechend weiter. Das geht solange bis das Paket das Ziel erreicht hat oder aus irgendeinem Grund nicht zugestellt werden kann[11]. Die Ziel IP wird auf all diesen Wegen nicht geändert[12]. Dieser Vorgang des Entscheiden über den besten Weg und das anschließende Weiterreichen der Pakete von einer Zwischenstation, einem Hop, zum nächsten wird „Forwarding“ genannt. „Routing“ bezeichnet ansich den gesamten Weg durch das Netz bis zum Ziel.

Das Defaultgateway wird unter Solaris über die Datei `/etc/defaultrouter` vorgegeben. Sie entält pro Zeile die IP oder den Hostnamen[13] eines oder mehrerer Defaultgateways. Unter Linux wird der Defaultrouter in der Datei `/etc/sysconfig/network`[14] über den Parameter `GATEWAY=<`*gateway-IP*`>` vorgegeben.

Die Routing Tabelle eines Hosts kann man sich mit dem Befehl `netstat` und den Optionen `-rvn` anzeigen lassen. Dabei bedeutet das `-rvn` das die Ausgabe der Routing-

[10] Dieser Parameter entspricht nicht dem absoluten ARP-Timeout.
[11] In dem Fall wird der letzte beteiligte Router wahrscheinlich eine Internet Control Message Protocol (ICMP) Nachricht wie „Destination Unreachable“ an den Absender schicken
[12] Solange keine Network Address Translation (NAT) beteiligt ist.
[13] Dieser muss dann über die `/etc/hosts` auflösbar sein.
[14] Siehe auch Kapitel 6.6.2

tabelle ausführlich, verbose, und ohne Namensauflösung erfolgen soll. Folgendes Beispiel zeigt die Routingtabelle eines Solaris Hosts mit einem Interface `eri0` mit der IP 192.168.2.200/27. Das Defaultgateway ist die 192.168.2.193. Als Gateway für das Subnetz 192.168.2.192/27 in dem sich der Host befindet ist die eigene IP angegeben, da dieses Netz direkt erreicht werden kann. Directly connected, wie der Cisco sagt.

```
# netstat -rvn

IRE Table: IPv4
  Destination        Mask           Gateway     Device Mxfrg  Rtt  Ref Flg  Out  In/Fwd
--------------- --------------- --------------- ------ ----- ----- --- --- ----- ------
192.168.2.192   255.255.255.224 192.168.2.200   eri0    1500*    0   1 U    22632      0
224.0.0.0       240.0.0.0       192.168.2.200   eri0    1500*    0   1 U        0      0
default         0.0.0.0         192.168.2.193           1500*    0   1 UG  224238      0
127.0.0.1       255.255.255.255 127.0.0.1       lo0     8232*    0   2 UH      38      0
```

Unter Linux tut es auch schon ein einfaches `route` zum Abfragen der Tabelle.´

```
# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.3.80    *               255.255.255.240 U     0      0        0 eth0
169.254.0.0     *               255.255.0.0     U     0      0        0 eth0
default         install-pc      0.0.0.0         UG    0      0        0 eth0
```

Die Flags (Flg) haben folgende Bedeutung: `U` - Route ist up, `H` - Hostroute und `G` - Route zeigt auf ein Gateway.

Das Solaris `route` bietet mit dem „sub-command“ (betrachtes wir es als Option oder Switch) `get` einen praktischen Weg, um auf Solaris Hosts die über mehrere Netzinterfaces verfügen oder gar selber routen[15] die Route zu einem Ziel zu erfragen.

```
# route get www.example.com
   route to: www.example.com
destination: default
       mask: default
    gateway: 192.168.2.193
  interface: eri0
      flags: <UP,GATEWAY,DONE,STATIC>
 recvpipe  sendpipe  ssthresh    rtt,ms rttvar,ms  hopcount      mtu     expire
       0         0         0         0         0         0      1500          0
```

6.6 Interfacekonfiguration

Viele der grundsätzlichen Einstellungen eines Interfaces werden sowohl unter Solaris als auch unter Linux mit ifconfig vorgenommen bzw. abgefragt. Mit ifconfig -a werden alle

[15] Erkennt das init-Script `/etc/rc2.d/S69inet` beim starten mehr als zwei Netzwerkinterfaces wird IP-Forwadring automatisch aktiviert. Bei Bedarf kann es mit `/usr/sbin/ndd -set /dev/ip ip_forwarding 0` wieder deaktiviert werden.

konfigurierten Interfaces abgefragt und die momentan aktiven Einstellungen ausgegeben. Einzelne Interfaces lassen sich gezielt mit `infconfig` *`interface`* abfragen. Dabei ähneln sich die Ausgaben von Solaris

```
# ifconfig hme0
hme0: flags=1000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4> mtu 1500 index 2
        inet 192.168.200.154 netmask fffffffc broadcast 192.168.200.155
        ether 8:0:20:a2:8d:d6
```

und Linux

```
# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:0A:E4:7E:4B:8A
          inet addr:192.168.178.178  Bcast:192.168.178.183  Mask:255.255.255.248
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:120870162 errors:0 dropped:0 overruns:0 frame:0
          TX packets:114609847 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:3045188142 (2904.1 Mb)  TX bytes:2423390401 (2311.1 Mb)
          Base address:0x2400 Memory:dd220000-dd240000
```

Diese Einstellungen können mit **ifconfig** auch geändert werden. Ein **ifconfig hme0 10.1.2.108 mask 255.255.255.128** wuerde z.B. dem Interface **hme0** die IP 10.1.2.108 in einem 25-Bit Subnetz zuweisen. Diese Änderung würde einen reboot der Maschine allerdings nicht überstehen. Solaris und die diversen Linux Distributionen gehen da verschiedene Wege.

6.6.1 Interfacekonfiguration unter Solaris

Um ein Interface bereits beim booten richtig ins das System einzubinden müssen unter Solaris einige Dateien konfiguriert werden. Für jedes Interface welches beim boot konfiguriert werden soll muß eine Datei `/etc/hostname.`*`interface`* existieren. In dieser Datei existiert nur ein Eintrag: entweder der Hostname (wird in die IP aufgelöst) oder die IP-Adresse des Interfaces. Gegebenenfalls muss in der Datei `/etc/netmasks` noch die passende Netzmaske angegeben werden, damit Netzmaske ind Broadcast für das Interface richtig und nicht class-full gesetzt werden.

Angenommen das Interface `eri0` soll die IP 192.168.2.12 bekommen und das Netz hat eine 27-Bit Maske. In die Datei `/etc/hostname.eri0` enhält folgenden Eintrag:

```
192.168.2.12
```

bzw.

```
eri0
```

falls der Hostname `eri0` z.B. über einen Eintrag

```
192.168.2.12  eri0
```

in der `/etc/hosts` aufgelöst werden kann. Zusätzlich muss die Netzmaske des Netzes noch in der `/etc/netmasks` angegeben werden:

```
192.168.2.0  255.255.255.224
```

Damit wird das Interface nach dem booten richtig initialisiert. Duplex-Verfahren und Übertragungsgeschwindigkeit werden mittels Autonegotiation ausgehandelt. Da das in der Praxis aber oft nicht so klappt wie gedacht und letztendlich das eine Interface auf 100 full und das andere auf 100 halb oder noch grausameren Einstellungen endet, ist es sicherer das Interface fest einzustellen. Dazu müssen mit `ndd` die Treibereinstellungen des Interfaces geändert werden. Ein kleines Initskript welches beim booten ausgeführt wird kann dies erledigen:

```
#!/bin/sh
ndd -set /dev/ce instance 0
ndd -set /dev/ce adv_1000fdx_cap 0
ndd -set /dev/ce adv_1000hdx_cap 0
ndd -set /dev/ce adv_100fdx_cap 1
ndd -set /dev/ce adv_100hdx_cap 0
ndd -set /dev/ce adv_10fdx_cap 0
ndd -set /dev/ce adv_10hdx_cap 0
ndd -set /dev/ce adv_autoneg_cap 0
```

Der erste Befehl stetzt die Instanz fuer `/dev/ce` auf `0`, also alle nachfolgenden `ndd -set` Anweisungen wirken sich auf das Interface `ce0` aus. Der Vollständigkeit halber sei erwähnt, dass für den Fall das alle Instanzen eines Interfaces gleich eingestellt werden sollen, die entsprechenden Anweisungen auch in die `/etc/system` bzw. in die *`interface`*`.cfg` unter `/plattform/$(uname -m)/kernel/drv` eingetragen werden können. Der Weg über ein Initskript ist allerdings der gebräuchlichste und sollte daher auch gewählt werden. Mit `ndd -get` können die aktuellen Einstellungen ausgelesen werden. Einfacher geht das allerdings mit dem Skript `nicstatus.sh`[16]:

```
# nicstatus.sh

      Link:  Auto-Neg:   Status:   Speed:    Mode:  Ethernet Address:
---------------------------------------------------------------------
      eri0         ON        UP    100MB      FDX    0:3:ba:24:6c:a9
```

Die Defaultroute wird über den entsprechenden Eintrag in `/etc/defaultrouter` konfiguriert.

6.6.2 Interfacekonfiguration unter Linux

Unter RHEL spielt sich die Netzwerkkonfiguration in /etc/sysconfig und dessen Unterverzeichnissen ab. Die Grundlage der Netwerkkonfiguration bildet die Datei `/etc/sysconfig/network`. Sie enthält maximal sechs Angaben, in den meisten Fällen aber mindestens diese drei:

[16] http://www.sun.com/bigadmin/scripts/submittedScripts/nicstatus.txt

```
NETWORKING=yes
HOSTNAME=server1.example.com
GATEWAY=192.168.2.1
```

`HOSTNAME` kann auch über DHCP vergeben werden. Ausserdem können noch `NETWORKING_IPV6`, `NISDOMAIN` und `GATEWAYDEV` (Interface über das `GATEWAY` erreicht werden kann) definiert werden falls nötig.

Über die Skripte in `/etc/sysconfig/network-scripts` werden alle nötigen Netzwerinformationen bezogen und gesteuert. Für uns sind an dieser Stelle in erster Linie die `ifcfg-`*`interface`* Skripte interessant.

Um beispielsweise dem Interface `eth0` die IP 192.168.2.13 im Netz 192.168.2.0/27 zu vergeben würde folgende `ifcfg.eth0` ausreichen:

```
DEVICE=eth0
BOOTPROTO=static
IPADDR=192.168.2.12
NETMASK=255.255.255.224
ONBOOT=yes
TYPE=Ethernet
```

Zusätzlich können noch weitere Optionen wie die Broadcast-Adresse, DNS-Server oder Interfacebonding übergeben werden. Interessant ist im Vergleich zur Netzwerkkonfiguration unter Solaris noch die Option `ETHTOOL_OPTS`. Unter Linux werden Duplex-Modus und Übertragungsgeschwindigkeit mit `ethtool` bzw. `mii-tool` gesetzt und abgefragt:

```
# ethtool eth0
Settings for eth0:
        Supported ports: [ MII ]
        Supported link modes:   10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
                                1000baseT/Half 1000baseT/Full
        Supports auto-negotiation: Yes
        Advertised link modes:  10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
                                1000baseT/Half 1000baseT/Full
        Advertised auto-negotiation: Yes
        Speed: 100Mb/s
        Duplex: Full
        Port: Twisted Pair
        PHYAD: 1
        Transceiver: internal
        Auto-negotiation: on
        Supports Wake-on: g
        Wake-on: d
        Current message level: 0x000000ff (255)
        Link detected: yes

# mii-tool
eth0: negotiated 100baseTx-FD, link ok
eth1: no link
```

Der Befehl um das Interface `eth0` mit `ethtool` fest auf 100 Mbit full duplex einzustellen lautet `ethtool -s eth0 speed 100 duplex full autoneg off`. Mit `mii-tool` erreicht man dasselbe durch ein `mii-tool -F 100baseTx-FD`.

Um diese Änderung bootfest zu machen könnte man wie unter Solaris die entsprechenden Befehle in ein Initscript schreiben. Einfacher geht es aber über die `ifcfg.`*`interface`* und die Option `ETHTOO_OPTS`:

```
DEVICE=eth0
BOOTPROTO=static
IPADDR=192.168.2.12
NETMASK=255.255.255.224
ONBOOT=yes
TYPE=Ethernet
ETHTOOL_OPTS="autoneg off speed 100 duplex full"
```

6.7 Statische Routen

Oft ist notwendig einzelne Hosts bzw. Netze unabhängig vom Defaultgateway zu routen. Es gibt verschiedene Möglichkeiten um statische Routen einzupflegen, die auch nach einem Reboot der Maschine bestehen bleiben. Letztendlich nutzen wir unter Solaris und Linux den gleichen Ansatz über ein Init-Script.

6.7.1 statische Routen unter Solaris

Der einfachste Weg ohne einen zusätzlichen Routingprozess starten zu müssen führt über ein Init-Script. Man legt dazu z.B. unter `/etc/rc2.d` eine für den Superuser `root` ausführbare Datei `S77routen` an. In diese schreibt man die `route` Befehle die man letztendlich auch auf der Shell verwenden würde.

```
# cat S77routen
#!/sbin/sh
#
# static routes
#
route add net 172.16.0.0 -netmask 255.255.0.0 192.168.230.2 1
route add net 172.19.0.0 -netmask 255.255.0.0 192.168.230.2 1
route add net 192.168.3.0  -netmask 255.255.255.0 192.168.100.51 1
route add net 192.168.222.0 -netmask 255.255.255.0 192.168.220.1 1
route add net 10.17.0.0 -netmask 255.255.0.0 192.168.100.254 1
```

6.7.2 statische Routen unter Linux

Statische Routen in Linux[17] können pro Interface gesetzt werden. Für jedes Interface über das statische Routen gesetzt werden, wird unter `/etc/sysconfig/network-scripts`

[17] Bitte dran denken, dass wir hier von Red Hat Enterprise Linux reden. Bei anderen Distributionen können durchaus andere Vorgehensweisen nötig sein.

eine Datei `route-<interface>` angelegt. Dort werden über die Einträge `GATEWAYn`, `NETMASKn` und `ADDRESSn` die statischen Roten gesetzt, wobei `n` eine Zahl ist und dazu dient die Einträge zu verknüpfen. Sehen wir uns das Ganze mal im Beispiel an. So sieht die aktuelle Routingtabelle aus:

```
# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.2.28    *               255.255.255.240 U     0      0        0 eth0
169.254.0.0     *               255.255.0.0     U     0      0        0 eth0
default         gateway         0.0.0.0         UG    0      0        0 eth0
```

Jetzt füllen wir die Datei `/etc/sysconfig/network-scripts/route-eth0` mit folgendem Inhalt:

```
GATEWAY0=192.168.2.42
NETMASK0=255.255.255.255
ADDRESS0=172.16.10.8

GATEWAY1=192.168.2.23
NETMASK1=255.255.255.255
ADDRESS1=10.10.10.40
```

Nach einem Reboot bzw. Restart des Netzwerks mittels `service network restart` sieht die Routingtabelle wie folgt aus:

```
# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
172.16.10.8     192.168.2.42    255.255.255.255 UGH   0      0        0 eth0
10.10.10.40     192.168.2.23    255.255.255.255 UGH   0      0        0 eth0
192.168.2.28    *               255.255.255.240 U     0      0        0 eth0
169.254.0.0     *               255.255.0.0     U     0      0        0 eth0
default         gateway         0.0.0.0         UG    0      0        0 eth0
```

Die beiden neuen Hostrouten sind jetzt aktiv und bootfest.

Stattdessen kann man natürlich auch denselben Weg wie ich ihn oben für Solaris beschrieben habe gehen, und unter `/etc/rc3.d` ein entsprechendes Init-Script anlegen[18].

```
#!/bin/sh
#
# static routes
#
route add -net 172.16.0.0 netmask 255.255.0.0 gw 192.168.230.2 metric 1
route add -net 172.19.0.0 netmask 255.255.0.0 gw 192.168.230.2 metric 1
route add -host 192.168.3.15 gw 192.168.100.51 metric 1
```

Alternativ können die Routen auch in die `/etc/rc.local` gesetzt werden.

[18] Oder, wenn man genau ist, ein Script in `/etc/init.d/` anlegen und in `/etc/rc3.d` einen Symlink darauf setzen.

7 Postfix

I have been a happy man ever since January 1, 1990, when I no longer had an email address. I'd used email since about 1975, and it seems to me that 15 years of email is plenty for one lifetime.

(Donald E. Knuth vs. email)

8 Domain Name Service (DNS)

9 Squid Proxy

I just had to take the hypertext idea and connect it to the TCP and DNS ideas and – ta-da! – the World Wide Web.

(Sir Tim Berners-Lee)

Abbildungsverzeichnis

Tabellenverzeichnis