Linux Übung:

Grundlagen

Kursunterlagen

-Universitäten Göttingen und Freiburg-

u.a. Dirk von Suchodoletz
dirk@goe.net

überarbeitet von Antonia Blanke
toni@uni-math.gwdg.de

24. Februar 2004

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Zu diesen Unterlagen

Diese Unterlagen wurden aus Anlass einer Vorlesung und eines Fortbildungskurses zum Thema Linux und Systemadministration zusammengestellt. Viele Teile stammen aus unterschiedlichen Quellen im Netz und wurden teilweise direkt übernommen, da sie sehr gut hier hineinpassten. Das "Self-Linux"-Projekt[1] war eine gute Quelle, denn es bietet eine vernünftige Ergänzung. Leider habe ich von vielen Stellen die URL nicht wiedergefunden, so dass es noch keine vernünftige Quellensammlung gibt. Sollte es Probleme mit den Rechten übernommener Texte geben, bitte ich Sie, mich einfach zu benachrichtigen. Fast alle der Textausschnitte stammen aus GPL'ten Vorlagen. Die GPL (GNU Public License) möchte ich auch auf diese Unterlagen angewandt sehen...

Ich habe daher nichts dagegen, wenn diese Unterlagen ganz oder zum Teil für andere Projekte Verwendung finden. Ich möchte dann nur auf den Haftungsausschluss hinweisen und bitten, die Quellen zu überprüfen und gegebenenfalls zu kennzeichnen. Ich hoffe, dass diese Kursunterlagen den Einstieg in das Betriebssystem Linux erleichtern. Die Beispiele habe ich in den meisten Fällen selbst ausprobiert und Angaben aus Konfigurationsdateien stammen aus meiner eigenen Praxis. Trotzdem können Fehler enthalten sein, die ich dann zu entschuldigen bitte. Diese Unterlagen werden regelmäßig aktualisiert, trotzdem werden nicht immer alle Teile auf dem gleichen jüngsten Stand sein.

Die (Kurz-)beschreibung einiger Programme kommt durchaus mehrfach vor: Einmal im laufenden Text, wenn einzelne Bereiche eines Linux-Systems mehr oder weniger ausführlich behandelt werden und in der Liste wichtiger Kommandos am Ende dieses Skriptes. Das ist durchaus so beabsichtigt! Um aber zuviele Wiederholungen zu vermeiden, wurde versucht, jeweils verschiedene Beispiele zu wählen, wenn welche angegeben werden. Ebenso wiederholen sich zum Teil die Beschreibungen wichtiger Systemkomponenten: So spielt die Shell natürlich beim Erst-Login eine wesentliche Rolle und wird später nochmals ausführlich behandelt - trotzdem würde ich nicht auf die ersten einführenden Worte verzichten wollen :-))

[1] www.selflinux.org

1.2 Geschichte

Damit nicht schon die erste Stunde in Stress ausartet, hat es sich eingebürgert, eine kurze Geschichte oder ähnliches vorzuschalten: 1991 kauft sich der finnische Student Linus Torvalds einen 386er PC, um mit den Möglichkeiten des 80386 Prozessors[2] zu experimentieren. Zunächst entwickelte er Linux unter Minix, einem Lehrbetriebssystem von A. Tanenbaum (von diesem stammt der Ausspruch: "Linux is obsolete". Sonst ist er aber eher für ein ordentliches Netzwerkbuch bekannt). Er schrieb hierzu in der Newsgroup `comp.os.minix`: *"... As I mentioned a month ago, I'm working on a free version of a Minix-look-alike for AT-386 computers. It has finally reached the stage where it's even usable (though may not be, depending on what you want), and I am willing to put out the sources for wider distribution. It is just version 0.02... but I've successfully run bash, gcc, gnu-make, gnu-sed, compress, etc. under it."*

Aus den ersten Versuchen wird schnell ein einfacher, aber brauchbarer multitasking fähiger Betriebssystemkern. "Linux" ist genaugenommen ein Betriebssystemkern (engl. "kernel"). Der Name geht auf "Linus' Unix" zurück und zeigt damit an, dass Linux ein unix-artiger Kernel ist. 1992 stellt Linus Torvalds die Kernel-Version 0.12 über anonymes FTP ins Netz, womit eine breitere Zahl von Testern Zugriff erhält. Schnell wird die Zahl der per Mail kommunizierenden Tester und Anwender so gross, dass der nötige Austausch nicht mehr allein über diesen Weg zu bewältigen ist. Zur besseren Kommunikation wird die Usenet-Gruppe `alt.os.linux` eingerichtet. Newsgroups waren vor der Ausbreitung des World Wide Web das dominierende Diskussions- und Kommunikationsmedium. Die eingerichtete Newsgroup schuf das Forum für eine explosionsartige Weiterentwicklung des Systems im ganzen Internet. Linus Torvalds koordiniert fortan die Weiterentwicklung des Kernels.

Nachdem der C-Compiler "gcc" und weitere wichtige Entwicklungswerkzeuge unter Linux liefen, konnte Linux unter Unix weiterentwickelt werden. Der Betriebssystemkern wurde von Grund auf neu programmiert und inzwischen modularisiert.

1994 wird die Version 1.0 des Kernels herausgebracht. Nun ist dieser bereits netzwerkfähig und die Zahl seiner Anwender steigt auf 100.000. Ein weiterer wichtiger Schritt geschieht ebenfalls in diesem Jahr: Die Anpassung einer grafischen Benutzerschnittstelle auf Linux. Diese wird von einer weiteren Non-Profit-Gruppe, dem XFree86-Projekt, geleistet. Torvalds stellt nun den Quelltext des Linux-Kernels offiziell unter die GPL. Die Marke "Linux" ist inzwischen in vielen Ländern eingetragenes Warenzeichen von Linus Torvalds. Somit ist die freie Existenz und Weiterentwicklung von Linux gesichert. In der Zwischenzeit erfolgt die Portierung auf die Plattformen Digital (DEC) und Sun Sparc.

Zwei Jahre nach dem Release der Version 1.0 wird die Majornumber um eins erhöht und die Version 2.0 des Linux-Kernels freigegeben. Dieser ist nun in der Lage mehrere Prozessoren gleichzeitig anzusprechen. Mit diesem Schritt verläßt Linux

[2] Der 80386 ist der 32 bit Nachfolger des 16 bittigen 286ers. Er bringt wesentliche "Systemeigenschaften" mit, wie Speicherschutz gegen unberechtigte oder fehlerhafte Zugriffe im Multi-User-Betrieb, Systemschutz gegen Verändern von Programmen und Daten. Weiterhin kennt er einen Real(Address)Mode, in dem der Prozessor als schneller "32 bit 8086" arbeitet. Hinzu kommen Protected(Virtual Address) Mode mit Speicherverwaltung und 4-Ebenen Schutzmechanismus (Privilegienlevel der Programmausführung). Linux ist deshalb auch nicht ohne wesentliche Einschränkungen auf 286er CPUs lauffähig.

langsam den Experimentalstatus und wird zu einer ernstzunehmenden Betriebssystemalternative. Auf der PC-Plattform gehört es neben den verschiedenen BSDs zur Software mit der besten TCP/IP-Unterstützung. Erste Firmen beginnen ihre Produkte für Linux zu portieren, womit die Zahl der kommerziell verfügbaren Software-Pakete steigt.

Im Jahre 1998 startet ein wesentlicher Schritt zur Verbreitung von Linux für den Desktop: Das KDE-Projekt wird aus der Taufe gehoben. Ein Jahr später kommt mit GNOME ein weiteres ernstzunehmendes Desktop-Projekt hinzu. In diesem Jahr erscheint auch die Kernel-Version 2.2 mit verbessertem Support für symmetrisches Multiprozessing (SMP) und renoviertem Netzwerk-Code. Eine erste Soundunterstützung realisiert ein Jahr später das Open Sound System (OSS), außerdem wird Samba, zur Einbindung von Windows-Netzwerken, in einer neuen Version 2.0 veröffentlicht.

Das Jahr 2000 bringt Linux einen weiteren Schritt dem Desktop näher: XFree86 wird in der Version 4.0 veröffentlicht und KDE 2.0 erscheint. 2001 wird die derzeitig aktuelle Kernel-Linie 2.4.X eröffnet. Der Kernel kann nun bis zu 64 GByte RAM ansprechen und unterstützt 64 Bit-Dateisysteme. Ebenso sind USB-Unterstützung und Journaling Filesysteme realisiert; Samba erscheint in der Version 2.2. Die nächste Runde der Desktop-Entwicklung erfolgt 2002 mit der Verfügbarkeit von KDE 3.0 und GNOME 2.0. Die OpenSource-Projekte Mozilla und OpenOffice erscheinen in stabilen Releases.

Die Weiterentwicklung fand und findet unter Beteiligung von vielen interessierten Programmierern im Internet statt; Linus Torvalds, der inzwischen bei der Prozessorschmiede Transmeta arbeitet, ist und war nie der einzige Entwickler.

1.3 Linux/Unix

Im allgemeinen Sprachgebrauch wird nicht zwischen dem Betriebssystemkern, dem Betriebssytem oder den Linux-Distributionen unterschieden, was manchmal für Verwirrung sorgt ... Der Betriebssystemkern von Linux ist in der Programmiersprache C geschrieben und liegt im Quelltext vor. Ohne zusätzliche Programme (Software) ist ein Betriebssystemkern ziemlich nutzlos. Um die Fähigkeiten des Betriebssystemkerns komfortabel und sinnvoll nutzen zu können, benötigt man mindestens:

- eine Reihe von Programmen für systemnahe Aufgaben (Systemsoftware)
- Programme zur Erkennung und Behebung von Fehlern
- den Zugriffsschutz, d.h. die Abfrage des Geheimwortes (Passwort)
- eine Befehlszeile zum Start weiterer Programme
- ein System zum automatischen Start der oben genannten Programme

Diese absolut notwendigen Komponenten bezeichnet man landläufig als "Betriebssystem". Derzeit wird in den meisten Fällen die Software des GNU-Projektes[3] in Verbindung mit dem Linux-Kernel verwendet. Das GNU-Projekt entwickelt bereits

[3] http://www.gnu.org

seit 1984 freie Software. Heute sind beide Komponenten, der Linux-Kernel und die GNU-Betriebssoftware, kaum noch voneinander zu trennen. Beide Teile haben sich gegenseitig vorangebracht und gegenseitig befruchtet. Korrekterweise müsste man also eigentlich, wenn man das Betriebssystem nennt, von "GNU/Linux" sprechen, da hier immer der Kernel (Linux) und die OS-Softwaretools (GNU) gemeinsam gemeint sind. Umgangssprachlich auch dem Drang nach Abkürzung folgend vereint dem Begriff "Linux" die Kombination aus Kernel und Betriebssystem. Dies soll keinesfalls eine Herabsetzung des GNU-Projektes sein, welches die wesentlichen Grundlagen überhaupt erst geliefert hat. Eher trägt die Abkürzung der Tatsache Rechnung, dass diese Benennung heutzutage von den meisten Anwendern und Journalisten verstanden und benutzt wird.

Die Installation des Linux-Betriebssystems ist weitgehend automatisiert und menügeführt. "Linux-Distributionen" sind inzwischen einfach zu installierende Sammlungen von Programmen "für Linux"; sie enthalten außerdem das Betriebssystem in der aktuellen Fassung. Linux-Distributionen werden von verschiedenen Firmen oder Interessengruppen in leicht unterschiedlicher und wechselnder Zusammensetzung, bzw. Qualität angeboten.

Es gibt keine "offizielle" Linux-Distribution und somit läßt sich von Linux nicht sprechen wie von MacOS, DOS, OS/2 oder den verschiedenen Windows-Versionen. Viele der Systemprogramme, die mit heutigen Linux-Distributionen ausgeliefert werden, gab es bereits vor Linux. Sie wurden meistens von Systemadministratoren und -entwicklern geschrieben, um bestimmte Aufgaben zu erleichtern oder die mitgelieferten Systemprogramme der Unix-Hersteller zu verbessern und in ihrem Funktionsumfang zu erweitern. Diese Programme wurden zumeist im Quelltext (Sourcecode) weitergegeben und meist unter die GNU Public License gestellt.

Das Internet schaffte die Infrastruktur zur sekundenschnellen Kommunikation der Software-Entwickler; erst dadurch wurden so große Projekte wie Linux überhaupt möglich.

1.4 Linux im Vergleich zu Windows/DOS

Linux ist wie andere Unixe auch ein Multi-User-Betriebssystem. Dieses mag für den Einsatz am heimischen Rechner, wo man/frau sowieso die einzige Person am Einschalter ist, erstmal etwas übertrieben sein, aber wenn man sich das Sicherheitskonzept ansieht, wird man sehen, dass die Unterteilung in mehrere Benutzer durchaus Sinn macht. Inzwischen entwickelt sich auch das Microsoft-Universum immer stärker in Richtung Serverbetriebssystem, welches mit vielen Einschränkungen, die im Folgenden genannt werden, aufräumt. Trotzdem bleiben eine Reihe von Wünschen offen und sei es nur, für ein bestimmtes Problem einen Blick in den Sourcecode werfen zu können. Mit der Einführung der Serverfähigkeit geht im Gegenzug jedoch ein Teil des "Komforts" verloren, der jedem Benutzer erlaubte, wirklich alles (inklusive des Total-GAUs) mit seinem Windows anzustellen. Nun bekommt man dafür schonmal schnell ein Problem, ein bekanntes Brennprogramm unter Windows XP für Normalbenutzer bereitzustellen.

Mit Windows in irgendeiner Form hat wahrscheinlich jede(r) schon einmal gearbeitet. Natürlich ist Linux ein ganz anderes System. Hier sollen einmal die wichtigsten

Unterschiede und Besonderheiten erläutert werden.
"Echtes" Multitasking: Linux ist ein Mehrbenutzer System, das heißt es können sich mehrere Benutzer einen Computer teilen und das sogar gleichzeitig. Man kann nämlich an einen Computer viele Terminals anschließen oder auch über ein Netzwerk mit einer Terminalemulation vom PC aus auf diesen Computer zugreifen. Jeder Benutzer teilt sich CPU (Rechenleistung) und Arbeitsspeicher mit den anderen Benutzern und deren Programmen.

Das Risiko, jeden Benutzer jede Systemdatei ändern zu lassen, kann man sich an der seit etlichen Jahren ungelösten Virenfrage für DOS/Windowssysteme ansehen. Deshalb gibt es mindestens einen priviligierten Benutzer "root" auf einem Linuxsystem und weitere unpriviligierte Benutzer für normale User- bzw. Systemprozesse. So wird sichergestellt, dass nicht unbedachterweise Konfigurationen oder Systemdateien verändert, manipuliert oder gelöscht werden.

Dateisysteme und Speichermedien: Festplatten, ihre Partitionen und eine Vielzahl von Wechselmedien existieren natürlich auch unter Linux. Es gibt nur keine Laufwerksbuchstaben, sondern einen Verzeichnisbaum. Dieser Verzeichnisbaum fängt mit /, daher auch die Bezeichnung "root" (Wurzel). Eventuelle "Laufwerke" sind an den Übergängen zu Unterverzeichnissen zu finden (beispielsweise /cdrom/ oder /floppy/, bei SuSe-Distribution im Unterverzeichnis /media/). Unter Unix könnten zum Beispiel */etc* und */usr* auf der ersten Festplatte sein und */home* könnte sich auf einer weiteren befinden. So lassen sich Probleme "überlaufender" Partitionen meistens schnell beheben, indem in stark belegte Bereiche einfach eine weitere Partition eingelinkt wird. Sollte generell die Festplatte zu klein werden, ist eine einfache Kopie des Dateisystems kein Problen: Einzig der Bootsektor muss anschliessend neu geschrieben werden.

Im eben Gezeigten wird auch schon deutlich, dass der DOS-gewohnte \ ("Backslash") hier durch einen / ("Slash") ersetzt wurde, da der Backslash in der Unix-Shell eine ganz andere Bedeutung besitzt.

Dateisysteme gibt es viele. Linux kann auf DOS oder Windows-(VFAT-)formatierte Festplatten ebenso zugreifen wie auf Atari-, CD-ROM-, Apple- oder Windows-NT-Dateisysteme. Bei Linuxinstallationen wird jedoch hauptsächlich folgende Dateisystem verwendet: EXT3 (EXT2 mit Journaling-Erweiterung), ReiserFS, JFS oder XFS, welches lange Dateinamen und Zuordnung von Benutzern/Besitzern und Dateien ermöglicht.

Achtung Falle (oder Selbstverständlichkeit?!): Linux unterscheidet in Groß- und Kleinschreibung! */home/ich/Hallo* und */home/ich/hallo* sind zwei verschiedene Dateien! Probleme können auftreten, wenn unter Linux ein Dateisystem ohne diese Unterscheidung verfügbar gemacht wird, z.B. ein VFAT gemountet oder CIFS über das Netz eingebunden wird. Hier findet dann einstellbar eine generelle Umstellung auf Groß- oder Kleinschreibung statt.

Wichtiger Hinweis: Bevor der Computer ausgeschaltet wird, muss man ihm zuerst "gute Nacht" (oder weniger romantisch "reicht jetzt") sagen, d.h. das Sitzungsende signalisieren, damit Linux eventuelle im Arbeitsspeicher gepufferte Dateien auf die Festplatten schreibt und sich korrekt beendet[4]. Zum Teil werden dabei die ATX-Features der Maschine ausgenutzt, jedoch sind nicht immer alle Möglichkeiten unter-

[4] Dieses ist evtl. bereits von WindowsNT/2000/XP bekannt

stützt.

Geräte und Treiber: Unter Linux findet sich für jedes Gerät im Verzeichnis */dev* eine Datei. */dev/ttyS0* ist zum Beispiel die erste serielle Schnittstelle. */dev/hda* bezeichnet die Masterfestplatte am ersten IDE-Controller, */dev/hda1* verweist auf die erste Partition dieser Platte. Dies scheint auf den ersten Blick völliger Blödsinn zu sein, erleichtert aber die Programmierung wesentlich. Dem Programmierer kann es egal sein, ob die Ausgabe seines Programms auf den Bildschirm, in eine Datei oder auf ein Drucker ausgegeben wird. Auch Treiber gibt es unter Linux. Sie werden entweder fest in den Systemkern (Kernel) eincompiliert oder als Zusatzmodule geladen[5].

1.5 Anmerkungen

Alle Kommandos werden üblicherweise ohne Nachfrage ausgeführt, also sollte gewisse Vorsicht geboten oder Toleranz gegenüber verschwundenen Daten angebracht sein. Liegen wichtige Daten auf dem System, so sollten regelmässige Backups erfolgen (das gilt für alle Betriebssysteme). Viele Befehle zur Systemadministration sind dem Superuser ("root") vorbehalten.

Zur Erhöhung der Lesbarkeit werden alle ausführbaren Dateien oder Systembefehle durch **Fettdruck**, z.B. **ls** oder **dhcpd** hervorgehoben. Alle Konfigurationsdateien oder Verzeichnisse werden *italic* gesetzt, z.B. */home/name/*. Beispiele für Kommandoeingaben, wie `last, sort, less` werden in Courier gesetzt.

Um den Lesefluss nicht stark zu stören, werden viele Erläuterungen als Fussnoten angefügt. Auch alle Links zu den entsprechenden Webseiten sind hier zu finden.

1.6 GNU und Free Software Foundation

GNU-Software unterliegt sämtlich dem GNU-Copyright! Dieses GNU-Copyright ist auch als COPYLEFT bekannt, da der Inhalt dieses Titels besagt, dass diese Software frei kopierbar ist, nicht verkauft werden darf, und dass entsprechender Quellcode veröffentlicht werden muss. Ferner ist alles, was mit GNU-Produkten erstellt wird (GNU-Teile enthält, z.B. Libraries nach einem Compilevorgang), selbst wieder ein GNU-Produkt. Ein Beispiel:

Meier schreibt ein nettes Programm. Er kompiliert es mit dem GNU-Compiler, den er kostenlos bekommen hat und der GNU-Libraries in das Compilat einbindet. Nun ist sein Programm (das Binary) ebenfalls GNU. Er darf es für sich behalten oder samt Quellcode weitergeben, ganz egal, er darf es nur nicht verkaufen und auch niemand sonst darf es. Bei einer (organisierten) Verteilung darf aber eine Gebühr für das Bereitstellen des Datenträgers erhoben werden. Das sind dann die Kosten einer Linux-Distribution.

Natürlich ist auf GNU/Linux-Systemen nicht nur Software lauffähig und einsetzbar, die unter der GPL steht; es gibt viele andere freie Lizenzen, unter denen Software stehen kann.

---

[5] Dieses wird - wie auch vieles andere - in einem weiteren Abschnitt ausführlicher behandelt

1.7 Begriffserklärungen

Im folgenden werden einige Begriffe und Abkürzungen erläutert, die im Text häufig verwendet werden. Kurze Erläuterungen zu wichtigen Shell-Kommandos, Systembefehlen, Diensten und grafischen Programmen sind am Ende dieses Textes zu finden.

Bibliothek Eine Software-Bibliothek benennt eine Sammlung von wiederkehrenden Funktionen. Da viele Programme gleiche oder ähnliche Funktionen benötigen, wäre es sehr ineffektiv gleiche Teile immer wieder neu zu programmieren. Daher werden diese Teile in eine externe Datei ausgelagert. Da mehrere Programme darauf zugreifen, kann die Grösse des einzelnen Programms auf der Festplatte verkleinert werden. Fehlt die Bibliothek, funktioniert jedoch das gesamte Programm nicht mehr.

Desktop Der X-Server selbst bringt nur die Fähigkeit mit, Grafikausgaben (auch netzwerktransparent) zu realisieren. Es sind jedoch zusätzliche Programme notwendig, um den Arbeitskomfort zu realisieren. Dazu dient ein Desktop wie KDE oder GNOME. Dieser ermöglicht das Arbeiten, wie man es von Apple-OS oder Windows her kennt. Man verfügt über eine Arbeitsfläche mit Fenstern und kann Programme über das “Anklicken” von Icons starten.

FTP Das File Transfer Protocal ist eines der ältesten Möglichkeiten, mittels TCP/IP Dateien zwischen Rechnern zu kopieren. Es verwendet das verbindungsorientierte TCP und verwendet Port 23. Ein Nachteil von FTP ist die unverschlüsselte Übertragung sowohl der Dateien, als auch des Passworts.

LAN Local Area Network. Meint Netzwerke einer geringen bis mittleren Ausbreitung, die sich üblicherweise der Ethernet-, ATM-, TokenRing- oder FDDI-Technologie bedienen.

Linux bezeichnet eigentlich nur den Kernel. Ein Kernel ist ein Stück Software, das die Kommunikation zwischen den einzelnen Hardwarekomponenten und den Anwenderprogrammen implementiert. Das mag trivial klingen, ist aber eine sehr komplexe Aufgabe. Jedes OS (Operating System) hat einen Kernel, nur werden die wenigsten nach dessen Namen benannt.

NFS Network File System. NFS ist ein UDP-basiertes Protokoll, das Dateisysteme über ein TCP/IP-Netzwerk zur Verfügung stellen kann.

Perl Perl ist eine freie interpretierte Skriptsprache, die sich im Bereich der Stringverarbeitung durchgesetzt hat. Sie steht unter allen gängigen Unix-Architekturen, aber auch unter Mac-OS und Windows zur Verfügung. Diese Programmiersprache kann durch Module erweitert werden. Inzwischen stehen Module für fast jeden Anwendungsfall[6] in weltweit zugänglichen Archiven[7] zur Verfügung.

Server Der Server ist in erster Linie ein Diensteanbieter im klassichen TCP/IP-Client-Server-Modell, d.h. er stellt, meistens zentral, bestimmte Funktionalitäten, wie Mail-, File- und Webdienste oder Applikationen zur Verfügung. Benutzer können sich an einem Server anmelden, werden aber nur in den seltensten Fällen physisch vor dem Gerät sitzen.

Shell (engl. für Muschel) Nach dem Einloggen befindet man sich in einer Shell. Dies ist ein Programm, das zwischen dem Benutzer und dem System arbeitet. Von dieser Aufgabe, dem Benutzer eine abgeschlossene Arbeitsumgebung zur Verfügung zu stellen, stammt der Name. Innerhalb der Shell hat man die Möglichkeit, Befehle und Programme aufzurufen. Zudem verfügt jede Shell über eine Programmiersprache, so dass Skripte zur Arbeitserleichterung geschrieben werden können.

SSH Secure Shell zur Verbindung zu einem anderen Rechner. Diese ist dem Telnet auf jeden Fall vorzuziehen, da sie verschlüsselt erfolgt. Das Programm auf der Serverseite heisst üblicherweise **sshd**, die Clientapplikation **ssh**.

Telnet Eines der ersten Protokolle der TCP/IP-Suite, um sich an entfernten Rechnern anmelden zu können. Telnet verwendet als Transportprotokoll TCP und arbeitet auf Port 21. Der Daemon, d.h. der Hintergrundprozess, der den Telnet-Dienst auf einem Rechner anbietet, heisst üblicherweise **(in.)telnetd** und wird meistens über den Internet-Super-Daemon **(x)inetd** gestartet. Die Clientapplikation heisst einfach **telnet**.

X-Server realisiert die Schnittstellen für das Graphical User Interface (GUI). Die grafische Oberfläche (unter Unix X-Server, X Window System oder X11 genannt) ist nicht Teil des Betriebssystems, sondern ein eigenständiges Programm.

XDMCP Das X display message control protocol steuert die Grafikschnittstelle auf Unix-Systemen. Diese Schnittstelle ist netzwerktransparent. Dabei erfolgt die Ausgabe der Grafikoberfläche des Servers lokal auf der Maschine. Die Benutzereingaben durch Tastatur und Maus werden über XDMCP an den Server weitergereicht.

[6]z.B. die Umsetzung von Netzwerkprotokollen oder die Schnittstellen zu bestimmten Anwendungen

[7]Das CPAN

Kapitel 2

Erste Schritte

2.1 Die erste Sitzung

Sitzung kommt von Sitzen. Jetzt kommt schon die erste Entscheidung: Entweder man setzt sich an die "Konsole", also direkt an den Bildschirm des Linux Computers (vor die reale Hardware). Oder man geht "per **telnet**" oder besser (weil verschlüsselt) "per **ssh**", also von irgendeinem entfernten PC aus über Netzwerk an die Sache heran. Dann sitzt man jedoch nicht mehr direkt an der Hardware, was aber kein Problem darstellt, solange man nicht an bestimmte Komponenten, wie z.B. das CD-Rom-Laufwerk heran möchte. Sollte kein Netz zur Verfügung stehen, erübrigt sich diese Entscheidung. An dieser Stelle sollte darauf hingewiesen werden, dass sich eine Netzwerk-Sitzung immer über das sogenannte Loopback-Netzwerk-Interface realisieren lässt. Denn das Loopback-Interface ist auf jedem Linux-Rechner unabhängig von einer realen Internet- oder LAN-Anbindung eingerichet. Das Einloggen geschieht dann mittels `ssh localhost` oder `ssh 127.0.0.1`. In diesem Beispiel sieht man bereits ein wichtiges Charakteristikum einer Linux-Maschine. Selbst wenn sie nicht an ein Netzwerk angeschlossen ist, steht das Netzwerkprotokoll TCP/IP zur Verfügung. Es bildet für viele Dienste überhaupt die Grundlage ihrer Funktion. Weiterhin kann ein Rechner fast immer auf zwei Wegen angesprochen werden: Die direkte Form geschieht über die IP-Adresse, im Beispiel über die *127.0.0.1*. Meistens kann der Rechner auch über seinen Namen adressiert werden, dieses setzt jedoch die Möglichkeit der Namensauflösung (DNS), d.h. der Zuordnung einer IP-Adresse zu einem Namen, voraus. Der Rechnername für die Maschine kann beliebig festgelegt werden. Er kann aber unter Umständen nur auf der Maschine selbst bekannt sein. Zusätzlich heisst jede Maschine selbst auch noch "localhost". Localhost entspricht fast immer der IP-Nummer *127.0.0.1*.

2.1.1 Die Konsole

An dieser sitzt man/frau üblicherweise, wenn der Rechner als Workstation eingerichtet ist. Einige Server hingegen werden vielleicht nur über das Netzwerk erreichbar sein, weil sie meistens an gesonderten Orten stehen. Hierin liegt ein Vorteil von Linux, denn es macht fast keinen Unterschied, ob man direkt am Rechner sitzt oder sich über das Netzwerk einloggt. So könnte der Login-Prompt an einer Linuxmaschine aussehen:

```
            WELCOME TO

                  --      --  ---  --  --   --  --   --     --
                 |  |    |  ||    |  ||  | |  ||  |_|  |   |  |
                 |  |    |  ||       ||  | |  |       /    |__|
                 |  |___ |  ||  |    ||  |_|  | /  _        __
                 |______||__||__| |__|  _____/ |__| |__|   |__|

            hermes.test.local                              console:tty1

hermes login:
```

Der Text oberhalb der Login-Aufforderung wird in der Datei */etc/issue* eingetragen. Mittels spezieller Control-Sequenzen kann der Name des Rechners und der Konsole automatisch ermittelt und übernommen werden. Das Login auf einem anderen Rechner über ein Netzwerk sollte immer mit **ssh** (Secure Shell) erfolgen, damit die eingegebenen Passwörter nicht abgehört werden können. Der Befehl **telnet** sollte nur in äussersten Notfällen zur Anwendung kommen[1]. Ein Login über das Netzwerk stellt man in der nachstehenden Weise her:

```
dirk@linux:~/lak> ssh -l dsuchod login.gwdg.de
dsuchod@login.gwdg.de's password:
```

Soll auch für das Netzwerk-Login ein kurzer Text oder Infoscreen erscheinen, muss hierfür die Konfigurationsdatei des SSH-Servers geeignet angepasst werden. Meistens stehen über SSH- oder Telnetverbindungen nicht alle Control-Sequenzen der Shell zur Verfügung. Hierzu jedoch später mehr.

2.2 Wichtige Tastenkombinationen

Wenn man direkt vor der Maschine sitzt, stehen einige spezielle Tastenkombinationen zur Verfügung:

[Alt]-[F1] bis [Alt]-[F6] Umschalten der virtuellen Text-Konsolen. Man hat mehrere virtuelle Bildschirme (Konsolen) und kann sich mehrfach am System anmelden und parallel arbeiten. Die Zahl der Textkonsolen wird in der */etc/inittab* definiert.

[Alt]-[F7] Grafische Oberfläche (meistens die nächste Konsole nach der "letzten" Textkonsole. Der Standard sieht fast immer so aus. Jedoch kann jede Maschine individuell über die */etc/inittab* konfiguriert werden.

[Alt]-[F10] Hier landen üblicherweise die aktuellen Syslogmeldungen, wie sie auch in die Systemlog-Datei */var/log/messages* geschrieben werden. Konfiguriert wird dies über die Datei */etc/syslog.conf*.

[Alt]-[b] je ein Wort rückwärts ("backward") mit dem Cursor bewegen.

[1] leider wird auch bei neuesten Windowssystemen keine SecureShell automatisch installiert

[Alt]-[f] je ein Wort vorwärts ("forward") mit dem Cursor bewegen.

[Strg]-[Alt]-[F1] Umschalten von X11 aus auf die erste virtuelle Konsole.

[Strg]-[Alt]-[BACKSPACE] Beenden der grafischen Oberfläche ohne den Umweg des Sessionmanagers. Diese Tastenkombination spricht direkt den X-Server (das Programm zur Steuerung des grafischen Displays) an und kann in dessen Konfigurationsdatei:
/etc/X11/XF86Config ausgeschaltet werden.

[Strg]-[Alt]-[Entf] Reboot des Rechners (steht nicht unter der grafischen Oberfläche zur Verfügung). Diese Funktionsweise wird in der */etc/inittab* ein- oder ausgeschaltet.

[Strg]-[k] löscht bis zum Ende der Zeile.

[Strg]-[l] löscht den Bildschirm. Dieses kann gleichfalls durch die Eingabe des Kommandos **clear** erreicht werden.

[Cursor]-Tasten Editieren in der Kommandozeile sowie Wiederholen von Befehlen (History-Funktion der meisten Unix-Kommandointerpreter mit Pfeil-hoch).

[Tab] Die Tabulatortaste dient der Kommando- bzw. Dateinamen-Ergänzung (Typecompletion, spezielle Funktion des Kommandointerpreters **bash**). Wird der Anfang eines Befehls oder Dateinamens gegeben, so kann dieser damit automatisch vervollständigt werden.

[Strg]-[A] oder [Pos1] springt an den Anfang einer Kommandozeile. Dies ist nützlich, wenn man feststellt, dass man sich am Anfang vertippt hat und nicht den weiten Weg mit dem Cursor zurücklegen möchte.

[Strg]-[C] Bricht die meisten Kommandos in ihrer Ausführung ab, so dass der zuvor eingegebene Befehl nicht weiter ausgeführt wird. Ausnahmen sind Dateibetrachter, z.B. das Kommando **less** in bestimmten Situationen oder der Editor **vi** und natürlich Programme mit grafischen Oberflächen, außer man ist in der aufrufenden Shell.

[Strg]-[D] beendet die Bash. Ist diese die Haupt-Shell (Shell, die vom Login-Prozess gestartet wurde - siehe die Ausgabe von `echo $SHLVL` und $SHELL), so loggt man sich aus oder schliesst im grafischen Modus das Terminalfenster. Diese Tastenkombination entspricht den Befehlen **logout** oder **exit**.

[Strg]-[E] oder [Ende] springt analog zu [Strg]-[A] an das Ende einer Kommandozeile.

[Strg]-[R] Rückwärts suchen in der Liste der bis dahin eingegebenen Befehle. Hierbei erfolgt mit jedem danach eingegebenen Zeichen die Suche nach dem nächsten passenden Eintrag in der Liste der Befehle.

[Strg]-[Z] Stoppt laufende Prozesse, die aus der Shell aufgerufen wurden und gibt den Prompt an den Benutzer zurück.

[Shift]-[PgUP] Erlaubt das "Hochschieben" des Terminalbildschirms, um bereits nach oben herausgeschobene Bildschirmbereiche wieder sichtbar zu machen.

[Shift]-[PgDOWN] Gegenstück zum eben Genannten: Erlaubt das Blättern nach unten.

2.2.1 Login

Jede(r) BenutzerIn meldet sich mit einer Kombination aus Benutzername und Passwort an der Maschine an und bekommt erst dann eine Umgebung (Shell, s. Kap. 2.2.2, S.12) zur Kommunikation mit dem System zur Verfügung gestellt. Wie auch für Datei- und Verzeichnisnamen, so gilt auch hier, dass nach Gross- und Kleinschreibung bei Benutzername und Passwort unterschieden wird. Die Kommandozeile wird aus historischen Gründen als "shell" ("Muschel") bezeichnet; diese Bezeichnung wird im Folgenden verwendet. Nach dem Login wird üblicherweise die Datei */etc/motd* (Message of the Day) angezeigt.

2.2.2 Shell / Kommandointerpreter

Ist man auf der Kommandozeile angelangt, so findet man ein weisses bzw. schwarzes Kästchen oder einen nervös blinkenden Unterstrich vor; beide werden als "Cursor" bezeichnet und markieren die Stelle, an der die eingetippten Zeichen erscheinen.

Die Shell oder auch der Kommandointerpreter ist ein Prozess der nach dem Anmelden (Login) an einer Unix-Maschine gestartet wird und die Interaktion des Benutzers mit dem Betriebssystem erlaubt. Diese Shell bringt bereits etliche Funktionalitäten mit. Sie interpretiert die Kommandoaufrufe seitens der Benutzer und erlaubt einfache Skript- (bzW. Batch-)programmierung zur Erleichterung wiederkehrender Tätigkeiten. Sie stellt weiterhin über Umgebungsvariablen Programmen eine ganze Reihe von Systeminformationen, wie Benutzername, Home-Verzeichnis und Rechnername zur Verfügung.

Die Kommandozeile beginnt meistens etwa so:

```
meier@hermes:~/kursunterlagen >
```

Zu Anfang steht der Username, mit dem man/frau am System angemeldet ist. Dann folgt der Rechnername und anschliessend das Verzeichnis, in dem man sich befindet. All das, zusammen mit dem ">" bezeichnet man als Prompt. Das Aussehen des Prompts kann verändert werden, wenn die steuernde Umgebungsvariable "PS1" entsprechend angepasst wird. Dieses geschieht entweder systemweit in der Datei */etc/profile* bzw. in einer Datei, die aus dieser heraus aufgerufenen wird oder durch einen Eintrag in der *.profile* des Benutzers. Dieses Verfahren läßt sich bei vielen Unix-Programmen wiederfinden: Es gibt systemweite Konfigurationsdateien und die Möglichkeit für den Benutzer eigene Einstellungen vorzunehmen, die komplett unabhängig von anderen Benutzern oder von den Systemeinstellungen sind.

Die Shell dient zum Aufruf der Programme, mit denen man eigentlich arbeiten will. Durch Ausgabe der Eingabeaufforderung, des sogenannten Prompts zeigt die Shell an, dass sie bereit ist, Kommandos entgegenzunehmen. Ein typischer Programmaufruf sieht etwa so aus:

```
meier@hermes:~/kursunterlagen > less kurs01.txt
```

Zuerst steht das Kommando, dann folgt nach einem Leerzeichen (white space) der Dateiname. Es ist dabei immer auf die Leerzeichen zu achten, da sonst die Shell die Kommandos nicht von ihren Optionen oder nachfolgenden Dateinamen unterscheiden kann. Beispielsweise hätte die Eingabe von **lesskurs01.txt** die Fehlermeldung "lesskurs01.txt: Command not found" verursacht, da die Shell nach einem (nicht vorhandenen) Befehl aus den Zeichen "lesskurs01.txt" am Stück sucht. Ausnahmen sind spezielle Zeichen, die von der Shell anders interpretiert werden. Dazu zählt zum Beispiel das Semikolon ";" für das Trennen von Kommandos in einer einzigen Zeile. Weitere Ausführungen hierzu finden sich in Kap. 3, S. 29.

Befehle bestehen aus Kommandonamen, Optionen und Argumenten, die an den Befehl angehängt werden. Beispiel: `ls -alh /home/test`. Hier ist der Befehl ls, die Option ist alh und das Argument ist die Datei /home/test. Optionen sind Anweisungen an den Befehl, seine Arbeitsweise gegenüber der Voreinstellung zu ändern. Optionen werden meist mit "-" eingeleitet. Für bestimmte Kommandos und Situationen findet man auch "- -". Klassisches Beispiel ist der Aufruf der Kurz-Hilfe-Option, die mit "-h" oder auch "- -help" erreichbar ist. Eine Option besteht gewöhnlich aus einem Buchstaben. Mehrere Optionen können aneinandergereiht werden, wie im obigen Beispiel die drei Optionen "a", "l" und "h". Jeder Befehl hat seine eigenen Optionen, die man in eigenen Hilfeseiten (mit "man befehlsname") nachschlagen kann. Argumente sind Zeichenketten, die vom Befehl interpretiert werden. Meist geben sie Dateien an, mit denen etwas gemacht werden soll.

Nach dem Drücken der "Enter"-Taste[2] wird der Befehl ausgeführt. Fehler in der Eingabezeile können mit Hilfe der Taste "Backspace"[3] oder "Delete" korrigiert werden. Die meisten Shells erlauben die Benutzung der Cursor-Tasten (Pfeil-Tasten) zur Korrektur der Eingabezeile.

2.2.3 Ändern des Passworts

Normalerweise wird das Passwort bei der Einrichtung eines Accounts - des Benutzerkontos auf einem Rechner - vom Administrator oder automatisch vergeben. Dieses fällt entweder sehr kryptisch aus oder ist zu einfach. Deshalb sollte dieses beim Erst-Login neu gesetzt werden. Das Passwort kann mit dem Befehl **passwd** geändert

[2] oder auch "Return"; Bezeichnung deshalb, weil der Prompt nach Ausführung des Kommandos wieder erscheint und eine neue Eingabe möglich wird

[3] Löschen eines Zeichens rückwärts

werden. Dies gilt jedoch meist nur auf Maschinen, die nicht an eine zentrale Benutzerverwaltung angeschlossen sind. Das Kommando **passwd** wirkt auf die Datei */etc/shadow* in der auf einer Linuxmaschine die Passwörter aller Benutzer verschlüsselt gespeichert sind. Dabei wird man noch einmal nach dem alten Passwort gefragt, es sein denn man ist der Systemadministrator. Damit wird verhindert, dass nicht in einem unbeobachteten Moment ein Witzbold schnell das Passwort ändern kann. Danach muss man zweimal das neue Passwort eingeben - als Schutz vor Tippfehlern, da das Passwort wie beim Einloggen "blind" eingegeben wird.

Das erste Passwort, das bei der Einrichtung eines Accounts zugeteilt wird, sollte aus Sicherheitsgründen schnellstmöglich geändert werden. Aber auch später ist ein regelmäßiges Ändern des Passworts ratsam, um einen Missbrauch des eigenen Accounts zu erschweren. Man sollte bei der Auswahl eines Passworts eine Reihe von Regeln beachten:

- Eine ziemlich unclevere Idee ist die Benutzung von Namen, Geburtstage, Firmennamen, Telefon- und Autonummern. Sie sind leicht zu raten und daher ungeeignet. Gleiches gilt für Abwandlungen der User-ID.

- Die einzelnen Zeichen des Passworts sollten auf der Tastatur nicht direkt nebeneinanderliegen, damit das Passwort beim Eingeben nicht einfach mitgelesen werden kann.

- Es sollte ein Passwort gewählt werden, das in keinem Wörterbuch steht. Es sollte daher mindestens ein nicht-alphanumerisches Zeichen, d.h. ein Sonderzeichen, nicht einen Buchstaben oder eine Zahl, enthalten und sowohl aus Gross- als auch Kleinbuchstaben bestehen.

- Ein Paßwort sollte aus mindestens acht Zeichen bestehen. Dieses ist die Maximalzahl von Zeichen eines klassischen Unixsystems. Viele Systeme erlauben durch Konfiguration meistens keine Passwörter, die kürzer sind als sechs Zeichen und melden bereits zu einfache Kombinationen.

- Beliebte Methoden, um auf ein einfach zu merkendes aber schwer zu ratendes Passwort zu kommen, sind das Ersetzen von Zeichen (etwa 's' durch '$', 'i' durch '!' etc.) oder Paßwörter aus Anfangsbuchstaben von Sätzen[4] zu bilden.

- Man sollte sein Passwort niemals aufschreiben oder weitersagen und vor dem Verlassen des Computers immer darauf achten, dass man ausgeloggt ist.

2.2.4 Umsehen am Kommando-Prompt

Für die ersten Schritte im System sollte man vielleicht einfach mal nacheinander folgende Kommandos ausprobieren, um sich mit der Funktionsweise von Kommandos, Shell und dem Linux-System vertraut zu machen:

[4] z.B. "Ich stehe fast immer um acht Uhr auf" zu "!$fiu8Ua" abwandeln. Dieses Satz trifft zwar nicht auf den Autor dieses Textes zu, bietet aber ein besseres Beispiel als zweistellige Uhrzeiten :-))

- **ps** - Prozess-Status: Zeigt die gerade unter der eigenen User-ID laufenden Prozesse an. Mit `kill ProzessID` kann man einzelne Prozesse beenden. Eine ausführliche Beschreibung der Linux-Prozessverwaltung ist in Kap. 5 auf S. 69 zu finden.

- **w** - Eine Art "Who": Liefert eine Liste der gerade am System angemeldeter Benutzer mit zusätzlichen Informationen, welche Prozesse diese gerade am Laufen haben. Darüberhinaus erzählt einem das Kommando die Auslastung der Maschine, wie lange sie schon läuft und von woher die Benutzer angemeldet sind.

- **uptime** - Laufzeit der Maschine: Ausschliesslich Informationen zur aktuellen Last auf der Maschine und die vergangene Zeit nach dem letzten Neustart.

- **uname** - Name des Betriebssystems: Ausführlichere Informationen erhält man durch `uname -a`.

- **pwd** - Pfadangabe: Gibt an, wo man sich gerade aktuell im Dateisystembaum befindet. Dies kann nützlich sein, wenn diese Information nicht durch den Prompt bereitgestellt wird.

- **whoami** - Eigene User-ID: Vergessliche Benutzer können mit diesem Kommando ihren Account-Namen anzeigen lassen.

- **last** - Wer war als letztes da: Meldet alle zuletzt angemeldeten Benutzer. Diese Liste kann verdammt lang sein und sollte durch `last | less` besser lesbar gemacht werden.

- **chsh** - Shell ändern: Es sind meistens mehr als eine Shell installiert und jede(r) hat vielleicht andere Vorlieben, was die Features von Shells anbetrifft. Das Ändern der Default-Shell funktioniert so ähnlich wie beim **passwd**, nur wenn die Maschine nicht an zentralen Verwaltungs- und Authentifizierungsstrukturen hängt.

- **chfn** - Name ändern: Hiermit kann man Daten zu seinem Account ändern, jedoch nicht seinen Accoutnamen selbst! Es gilt das zu **passwd** und **chsh** Gesagte.

- **ls** - Inhalt des Verzeichnisses: Listet alle Dateinamen im aktuellen Verzeichnis auf.

- **finger** - "Who" netzwerktransparent: Einfach aufgerufen liefert das Kommando ähnliche Informationen wie **w** oder **who**. Ein Aufruf von `finger @rechnername` liefert, wenn es nicht abgeschaltet ist[5], die auf einem bestimmten Rechner angemeldeten Benutzer.

- **date** - Datum und Uhrzeit: Liefert die aktuelle Systemzeit mit Stunden, Minuten, Sekunden und das aktuelle Datum inklusive Information zur Zeitzone.

[5] wegen häufigen Missbrauchs eher noch ein Relikt aus Zeiten, als die Welt der Internets noch in Ordnung war :-)

- **export** - Variablenliste: Zeigt den Inhalt aller Variablen, die an Programme oder weitere Shells weitergereicht werden können. Hier findet man z.B. die Variablen "USER" für den eigenen Accountnamen, "UID" für die eigene User-ID, die zum Accountnamen passt, "PATH" für den Suchpfad nach ausführbaren Programmen und etliche weitere.

- **free** - Freier Speicher: Zeigt die Belegung des Arbeitsspeichers an. Die Menge des freien Speichers ist meistens recht gering, da Dateien für schnelleren Zugriff zwischengespeichert werden.

- **df** - Disk Free: Zeigt für die einzelnen Bereiche des Dateisystems die Belegung der Festplatte(n) an.

- **id** - Gibt Auskunft über die Benutzer-ID in der gerade laufenden Shell:
  `uid=500(dirk) gid=100(users) groups=100(users),`
  `14(uucp),16(dialout),17(audio),33(video)`.

2.2.5 Ordnung in Verzeichnissen

Die meisten Aufgaben unter UNIX und Linux beinhalten das Anlegen, Löschen, kopieren und Umbenennen von Verzeichnissen und Dateien. An dieser Stelle werden die wichtigsten Befehle hierfür nur kurz genannt; eine ausführliche Beschreibung ist in Kap. 4.8 auf S. 55 zu finden.
Die wichtigsten Befehle sind:
mkdir "make directory", **rmdir** "remove directory", **cp** "copy", **mv** "move" und **rm** "remove". Im aktuellen Verzeichnis erzeugt man ein neues Unterverzeichnis mittels:
`mkdir name_des_neuen_ver- zeichnisses`.
Dieses kann analog mit:
`rmdir name_des_neuen_verzeichnisses` wieder entsorgt werden.

Eine Datei zu kopieren geschieht mittels **cp** in den folgenden Ausprägungen:
`cp existierende_ datei neuer_dateiname`
`cp existierende_datei existierendes_verzeichnis`
`cp existierende_datei existierendes_verzeichnis/neuer_dateiname`.
In vielen Aspekten analog arbeitet das Kommando **mv**, wobei die Quelldatei anschliessend nicht mehr unter ihrem ursprünglichen Namen existiert. Bei beiden Programmen wird eine eventuell existierende Zieldatei ohne Warnung überschrieben. Gelöscht werden kann eine Datei durch `rm dateiname`.

Eine einfache Aufstellung des Verzeichnisinhalts liefert **ls** mit vielen Zusatzinformationen `ls -alh`, wobei zusätzlich die Dateigröße nicht einfach in Byte sondern "(h)uman readable" in Kilo-, Mega- oder GigaByte angegeben wird. Die aktuelle Belegung eines Verzeichnisses liefert **du** "disk usage", wobei Unterverzeichnisse nochmal separat angezeigt werden.

2.2.6 Finden von Dateien

Um eine Datei zu suchen und eventuell zu finden, verwendet man das Kommando **find**. Abstrakt startet man die Suche so:

`find "Startverzeichnis" -name Dateiname`.
Man sollte das Startverzeichnis geeignet wählen, wenn man ungefähr weiss, wo die Datei grob liegen könnte. Sonst beginnt man am besten mit /. Das ist nicht immer ratsam, da es sehr lange dauern kann und man Fehlermeldungen für jedes Verzeichnis erhält, auf das man keinen Lesezugriff hat.

Es muss nicht unbedingt der Dateiname bekannt sein, man kann auch nach völlig anderen Kriterien einen Dateibaum durchforsten:
`find -type d -exec chmod a+x \ ;`

Das Beispiel sucht nach allen Dateien vom Typ "Verzeichnis" und führt auf diesen eine Operation aus. Die Operation wird nach der Find-Option "-exec" angegeben (in diesem Beispiel werden die Zugriffsrechte aller Verzeichnisse gesetzt). Damit **find** weiss, wo die Operation endet, wird diese mit einem Semikolon beendet. Damit jedoch die Shell dieses nicht interpretiert (als Trennzeichen von Kommandos) muss es entsprechend "escaped" werden, was mit dem Backslash ("\") geschieht.

Zum Finden von Dateien, Verzeichnissen, Programmen, etc. ist auch der Befehl **locate** sehr hilfreich. Dieser Behehl sucht den Datei- oder Verzeichnisnamen in einer Datenbank ; ob diese existiert und wie oft sie aktualisiert wird, hängt vom Systemadministrator ab. Beispiel: `locate XFree86`. Auf SuSE-Linux-Systemen findet man noch das kleine Skript **rpmlocate**, welches nicht ein **updatedb** benötigt, da es direkt auf die RPM-Datenbank zugreift.

2.2.7 Abmelden durch Ausloggen

Damit nicht nach dem Beenden einer Arbeitssitzung irgendjemand Blödsinn mit dem eigenen Account anstellen kann - was einem selbst in Rechnung gestellt werden könnte - sollte man sich unbedingt abmelden. Dies geschieht in Abhängigkeit von der Konfiguration und der verwendeten Shell auf verschiedene Weise: Mit [Strg]-[D] lassen sich viele Shells schliessen. Handelt es sich dabei um die automatisch beim Login gestartete Shell, so loggt man sich damit automatisch aus. Ähnliche Auswirkungen haben auch die Kommandos **logout** oder **exit**. Unter der grafischen Benutzeroberfläche genügt das einfache Schliessen der Shell nicht, sondern man verwendet entweder den geeigneten Menüpunkt des Windowmanagers oder beendet die X-Session mit [Strg]-[Alt]-[Backspace]. Man sollte auch immer daran denken, dass man an mehreren Konsolen gleichzeitig angemeldet sein kann und diese dann ebenfalls schliessen sollte.

2.2.8 Alternative zur Kommandozeile

Für Anfänger ist der **mc** ("midnight commander", stark angelehnt an den Norton Commander, der aus der DOS-Welt noch bekannt sein könnte) eine große Hilfe; er ersetzt die Kenntnis vieler Linux-Kommandos. Der **mc** läuft als eigenständiger Prozess in der Shell aus der er aufgerufen wurde (der Befehl hierfür ist einfach **mc**) Im Textmodus des **mc** findet man am unteren Bildschirmrand die Befehle mit jeweils einer Zahl davor. Drückt man die Funktionstaste mit der entsprechenden Zahl (z.B. F4 zum Bearbeiten), so wird der entsprechende Befehl ausgeführt. Die restliche Bedienung ist recht schnell verstanden: Mit den Pfeiltasten navigiert man durch die

Verzeichnislisten und durch die Menüs, mit "Enter" öffnet man Dateien bzw. aktiviert einen Menüpunkt. Die Tabulatortaste wechselt zwischen den beiden Fenstern.

Unter der grafischen Oberfläche "GNOME" wird derzeit der **gmc**, die grafische Ausgabe des **mc**, als ein weiterer Filemanager neben dem **nautilus** eingesetzt. Die Bedienung dieses Programms ist erfreulich intuitiv, jedoch lassen sich hiermit schwer Automatisierungen von wiederkehrenden Abläufen vornehmen.

2.3 Editoren und Dateibetrachter

2.3.1 Dateibetrachter

Die Programme **more**, **less** und **cat** dienen zum seitenweisen Anzeigen von Texten, sie sind kleiner und schneller als normale Editoren. Mit der Umgebungsvariable *PAGER*[6] bestimmt man, welches dieser Programme zum Anzeigen von Text benutzt werden soll. Die Variable kann durch `export PAGER=”less -MMI”` umdefiniert werden.

cat ist der denkbar einfachste Dateibetrachter überhaupt und eignet sich eigentlich nur zum Ansehen sehr kurzer Dateien, die über wenige Zeilen nicht hinausgehen. Da es im Gegensatz zu den im folgenden vorgestellten Betrachtern nicht interaktiv bedient wird, kann **cat** gut in Shellskripten verwendet werden.

more ist ein sehr einfacher Pager und wird in seiner Funktionalität von **less** bei weitem übertroffen. Weiterblättern kann man mit der Leertaste (seitenweise) oder der Return-Taste (zeilenweise), zurückblättern geht mit der Taste "b" (für back). Am Ende der Datei kehrt man bei **more** automatisch zum Prompt zurück.

less Nur **less** lässt sich interaktiv mit den Cursortasten bedienen. Es wird durch die Taste "q" beendet. Gesucht werden kann in **less** wie beim **vi** mittels "/Suchwort" vorwärts und mittels "?Suchwort" rückwärts.

head und tail Das Kommando **head** zeigt die ersten Zeilen und **tail** die letzten Zeilen einer Datei an.

2.3.2 Texteditoren

Texteditoren spielen unter Linux eine grosse Rolle, da sie häufig zum Anlegen und Ändern von Dateien benötigt werden. Fast alle Konfigurationsdateien verschiedener Programme und Services sind Textdateien, welche auf die konkreten Bedürfnisse der User oder des Systems angepasst werden müssen oder können.

[6] s. Kap. 3.4, S. 38

2.3.3 Joe

Der zu Beginn sicherlich am einfachsten zu bedienende Texteditor ist **joe**, der in seiner Kommandosyntax wordstar-kompatibel ist. Er eignet sich insbesondere für User, die sich partout nicht den **vi** aneignen wollen, aber dennoch eine Möglichkeit suchen, von der Konsole aus einfache Bearbeitungen an Textdateien vorzunehmen. Mit **vi** gemeinsam hat **joe** zum einen den Vorteil, auch dann zu laufen, wenn sonst nichts mehr geht, zum anderen die genügsamen Ansprüche an die Hardware. Im Gegensatz zum **vi**, dessen Bedienungskonzept manchem Neuling als recht kryptisch erscheint, ist **joe** aber etwas konventioneller.

Der Editor wird einfach von der Konsole aus gestartet mit `joe <Dateiname>` Dann erscheint der Dateiinhalt im Editierbereich. Zum Eingeben einfach die Tastatur verwenden, der Cursor kann mit den Cursortasten bewegt werden. **joe** legt von jeder geänderten Datei eine Sicherungskopie an, die aus dem Dateinamen mit angehängtem (Tilde) besteht, wenn man es ihm nicht in der Konfigurationsdatei */etc/joerc* explizit verbietet.

Tasten	**Funktion**	**Tasten**	**Funktion**
Strg-kh	Hilfefenster	Strg-kf	Suchen
Strg-kr	Datei einlesen	Strg-l	Weitersuchen
Strg-c	Beenden	Strg-kx	Speichern/Beenden
Strg-kd	Speichern unter	Strg-kb	Markierung Anfang
Strg-kc	Kopieren	Strg-kk	Markierung Ende
Strg-km	Verschieben	Strg-ky	Block löschen
Strg-ku	Dateianfang	Strg-kv	Dateiende
Strg-y	Zeile löschen		

Tabelle 2.1: Wichtige Tastenkürzel für **joe**

2.3.4 Vi

vi oder in der grafischen Version **gvim** ist für den Anfänger nicht so leicht zu bedienen, eignet sich aber besonders für die Arbeit über recht langsame Netzwerkverbindungen. Bis es soweit ist, liegt aber ein steiniger Weg vor einem. Vieles, was man bisher von Editoren wusste, gilt beim **vi** nicht mehr. Alles wird mit Hilfe der Tastatur gemacht, die Maus verliert im **vi** ihre Bedeutung als zentrales Hilfsmittel. Diesem Effekt ist es zu verdanken, dass der vi immer zur Verfügung steht. Wenn nichts mehr geht, geht immer noch ein vi. Daher ist er Teil der meisten Rettungssysteme. Da man in einem Notfall keine Zeit hat, sich in den vi einzuarbeiten, sollte man dies schon vorher einmal tun.

Damit der Umstieg zum vi nicht zu schmerzhaft ist, gibt es eine Vielzahl von vi-Clonen. Der oben genannte **gvim** ist eine angenehme Mischung aus der Stärke des vi mit der Einfachheit von **gedit**. **gvim** kann entweder wie vi über die Tastatur bedient werden oder auch mit Hilfe der Maus.

Hier erfolgt nur eine kurze Einführung[7] Aus dem Eingabemodus gelangt man

[7] für eine ausführliche Anleitung siehe: http://www.selflinux.org/selflinux-devel/html/vim.html

mit [ESC] in den Kommandomodus. Die wohl wichtigste Tastenkombination ist die zum Verlassen des Editors :-)) (ohne die vielleicht versehentlich geöffnete Datei zu schreiben): Nacheinander sind [ESC],[:],[q],([!]) zu drücken.

Tasten	**Funktion**	**Tasten**	**Funktion**
x	Zeichen löschen	/Muster	Suche vorwärts
dd	Zeile löschen	?Muster	Suche rückwärts
:w	Datei sichern		
:q	»vi« beenden		
i	Eingabe (insert)		
r	Eingabe (overwrite)		

Tabelle 2.2: Wichtige Tastenkürzel des **vi**

2.3.5 Emacs

Emacs ist wohl einer der flexibelsten Editoren unter Linux/Unix. Es gibt umfangreiche Unterstützungen für die verschiedensten Programmiersprachen und Formatierungen. Der Editor emacs verfügt über eine mächtige Lisp-Unterstützung für weitestgehende Konfiguration. Funktionen lassen sich entweder durch Benutzung der Maus oder durch Tastenkombinationen aktivieren. In einer Emacs-Sitzung kann man viele Dinge parallel erledigen: Dateien eines Verzeichnisses umbenennen, Dateien bearbeiten, die eigentliche Aufgabe eines Editors, den Compiler für eine Programmiersprache starten ... Jede dieser Aufgaben erledigt Emacs in einem eigenen (Daten-)Puffer. Ein spezieller Puffer ist "scratch", der für Notizen gedacht ist, die nicht gespeichert werden sollen. Ruft man den Emacs ohne Dateinamen auf, wird per default der Puffer Scratch gezeigt.

Tasten	**Funktion**	**Tasten**	**Funktion**
Strg-g	Vorgang abbrechen	Strg-x Strg-s	Buffer speichern
Strg-s	Wort suchen	Strg-x Strg-c	Beenden/Verlassen
Strg-_	Undo	Strg-x Strg-f	Datei laden
Strg-w	Text ausschneiden	Strg-x Strg-i	Datei einfügen
Strg-SPC	Marke setzen	Strg-x 1	Fenster schliessen
Strg-y	Text einfügen	Strg-h m	Bearbeitungsmodus
F10	Menü (versionsabhängig)		anzeigen

Tabelle 2.3: Wichtige Tastenkürzel für **emacs**

2.4 Suchen und Finden von Hilfe

Selbst erfahrene UNIX-/Linux-Anwender kennen nur einen Teil der Befehle mitsamt ihren Optionen. Daher gibt es auf jedem UNIX-System mehrere Möglichkeiten, Informationen über Kommandos abzurufen. Die klassischen Befehle hierfür sind **man**,

apropos und **whatis**. Die meisten Programme bringen selbst eine eingebaute Mini-Hilfe mit, welche mit "-h" oder "--help" zu aktivieren ist: `ls --help` gibt zum Beispiel einen Kurzüberblick zu den Fähigkeiten des List-Kommandos.
Manualpages:

- enthalten die abstrakte Syntax eines Kommandos oder Anwendungsprogrammes.
- enthalten eine Liste aller Optionen und deren Wirkung.
- enthalten Querverweise zu anderen Befehlen/Manual Pages.
- enthalten manchmal Beispiele.
- sind nicht zum Lernen, sondern eher zum Nachschlagen geeignet.
- werden aufgerufen durch `man Kommandoname`
- werden seitenweise mithilfe eines Pagers (**less**, **more**) angezeigt

Das folgende Beispiel zeigt eine typische Man-Page für das Kommando **cp**. Defaultmäßig wird die Man-Page in Englisch angezeigt, ist die entsprechende Umgebungsvariable "LANG" entsprechend gesetzt, kann auch jede andere installierte Sprache angezeigt werden, wie hier im Beispiel Deutsch - **man cp** liefert:

```
CP(1)              Dienstprogramme für Benutzer              CP(1)



BEZEICHNUNG
       cp - (copy) kopiert eine oder mehrere Dateien

SYNTAX
       cp [Optionen] Quelle Ziel

       cp [Optionen] Quelle ... Verzeichnis

OPTIONEN
       -a     (archiv) das gleiche wie -dpR

       -b     (backup) sichert Dateien im Zielverzeichnis vor dem
              Überschreiben

       -d     (no-dereference) kopiert die Links  und  nicht  die
              Dateien, auf die der Link zeigt

       -f     (force)    Dateien    im    Zielverzeichnis    werden
              überschrieben
[... eine Menge weiterer Optionen ...]
```

```
       -V {numbered, existing, simple}
              (version-control)  erhält  auch  frühere  Versionen
              einer Datei, indem  jeweils  neue  Backups  erzeugt
              werden.  Die  Art  der Backups kann auch durch die
              Umgebungsvariable VERSION_CONTROL bestimmt  werden.
              Die  Option  -V überschattet VERSION_CONTROL.  Wenn
              weder  die  Option  noch  die  Environmentvariable
              gesetzt  sind, wird existing benutzt. Gültige Werte
              für VERSION_CONTROL sind:

              numbered
                     Backups werden nummeriert

              existing
                     Backups werden nur  für  Dateien  nummeriert
                     die bereits nummerierte Backups haben.

              simple es werden immer einfache Backups gemacht

SIEHE AUCH
       das LunetIX Linuxhandbuch

AUTOR
       Torbjorn Granlund, David MacKenzie und Jim Meyering

LunetIX Linuxhandbuch      1. Juli 1993                 CP(1)
```

Das Kommando **apropos** liefert die Antwort auf die Frage “Was findet man alles zum Stichwort xyz?”. Es gibt eine Liste der zu diesem Stichwort verfügbaren Manual Pages aus. Aufgerufen wird es durch **apropos Stichwort**. Das Kommando **whatis** liefert eine einzeilige Kurzinformation, wenn der Name des Kommandos bekannt ist:

```
dirk@test:~/kurs> whatis cp
cp (1)               - copy files and directories
```

Ein weiteres kurzes Kommando, `which Befehlname`, liefert den vollständigen Pfad zu einem ausführbaren Programm, wenn es sich im Suchpfad befindet. Der Suchpfad wird durch die Shell-Umgebungsvariable “PATH” bestimmt.

2.5 Remote Login, Execution

2.5.1 Verbindung mit anderen Rechnern

Mit dem Kommando **telnet** oder **rsh** (für remote Shell) kann man sich auf einem anderen Rechner einloggen und den eigenen Rechner als “Terminal” benutzen, wie schon in einem frühen Abschnitt beschrieben wurde. Das Einloggen mit **telnet** hat den gravierenden Nachteil, dass die Kommunikation zwischen den Rechnern und damit auch die Authentifizierung mit Login-ID und Passwort unverschlüsselt über das Netzwerk erfolgt und daher leicht mitgelesen werden kann. Daher sollte man die Secure Shell (**ssh**) einsetzen, die eine verschlüsselte Verbindung mittels eines sogenannten Public Key-Kryptoverfahrens ermöglicht.

2.5.2 Filetransfer per Secure Copy

Kopieren mit **ssh**; zum Beispiel:
`scp user@dozent:~/lak.pdf kurs/` kopiert die Datei lak.pdf aus dem Homeverzeichnis von user auf dem Rechner dozent in das Verzeichnis kurs auf dem lokalen Rechner.
Umgekehrt kopiert:
`scp -r test user@rechner01:/tmp` das Verzeichnis test rekursiv (d.h. mit allen unterverzeichnissen und Dateien) vom lokalen Rechner in das Verzeichnis /tmp auf Rechner01.

2.6 Aufgaben

2.6.1 Dokumentation/Hilfe

1. Man nenne das Kommando (und den Aufruf am Beispiel der Shell “bash”) zum Anzeigen von Manpages!

2. Wo finden sich unter einer Suse-Linux-Installation üblicherweise weitere Hilfetexte, Howto’s und Unterlagen?

3. Welche Möglichkeiten stehen alle zur Verfügung, um sich Hilfen, Informationen und Hinweise zu Programmen und Softwarepaketen anzeigen zu lassen?

2.6.2 Zentrale Kommandos

1. Man ermittele seine eigene UserID und GruppenID (numerisch und als Accountname) und die des Systemadministrators!

2. Wie lautet das Kommando zum Wechseln der Verzeichnisse? Wie kommt man in die oberste Verzeichnisebene?

3. Mit welchem Kommando legt man ein Verzeichnis an und wie lösche ich ein nichtleeres Verzeichnis komplett?

4. Lesen Sie Ihre aktuelle DISPLAY-Variable aus und geben Sie diese anschliessend an!

5. **who**: Wer ist da? Welche Befehle zeigen ähnliche Informationen an? Wie bekommt man heraus, wer sich in letzter Zeit an der Maschine eingeloggt hat? Wie kann man dafür sorgen, dass sich diese Liste bequem durchblättern läßt?

6. Wie kann ich meinen "Full Name" (Room Number, Work Phone, Home Phone) ändern? Wo wird festgelegt, ob normale User diese Felder wirklich ändern können? Weshalb klappt dieses in großen Umgebungen mit einer zentralen Passwortverwaltung meistens nicht mehr?

7. Welche(s) Kommando(s) liefern mir Datum und Systemzeit? Wie kann man diese (neu) setzen, welche Voraussetzungen sind dafür notwendig?

8. Wofür dienen die Kommandos **find** und **locate**? Worin bestehen die Unterschiede zwischen ihnen?

9. Mit welchem Kommando erhalte ich die Ausgabe: `6:39pm an 2 Tage 5:28, 10 Benutzer, Durchschnittslast: 0,01, 0,04, 0,23`?

10. Welche Kommandos (Mehrzahl!) zeigen mir an, wer gerade mit mir an einer Maschine eingeloggt ist?

2.6.3 Die Shell

1. Was sind Shell Prompt und Terminal-Fenster?

2. Wie setzt man eine Umgebungsvariable (und sorgt dafür, dass sie auch in abgeleiteten Shells vorhanden ist)? Wie zeigt man die Liste aller in der aktuellen Shell definierten Variablen an?

3. Wie "schneidet" man aus der */etc/passwd* die Zeile mit der eigenen UserID heraus?

4. Woher bezieht die Shell die Informationen zu Username, numerischer UserID und Home-Verzeichnis? Welcher Prozess ist bei der Anmeldung involviert?

5. Wie findet man heraus, in welchem Pfad man sich befindet? In welcher Shellvariablen kann diese Information angezeigt werden (damit man immer weiss, in welchem Verzeichnis man gerade ist)?

6. Welche Möglichkeiten der Abmeldung von einer Shell oder allgemeiner von einer Konsole stehen zur Verfügung?

7. Welche Shellvariablen sind nur lesbar und warum kann man sie nicht einfach überschreiben, z.B. mit `export VARIABLE=”neuer Wert”`?

8. Welche verschiedenen Möglichkeiten stehen zur Verfügung, um an (vor einiger Zeit) eingegebene Befehle und Befehlsfolgen wieder heranzukommen?

9. Wo werden Shellvariablen definiert? Wie kann ein Benutzer seine eigenen Definitionen speichern, damit sie bei jedem Login oder Shellaufruf automatisch aktiviert werden?

10. Wann sind ausführbare Programme, wie `ls -la /etc` einfach ausführbar? Wann muss der absolute Pfad, wie z.B. `/opt/mozilla/mozilla` eingegeben werden, um ein Programm zu starten?

11. Verbinde die untenstehenden Blöcke geeignet! Ordne die Umgebungsvariablen ihrem Sinn zu! (2 Punkte)

PATH	Numerische ID des gerade angemeldeten Benutzers
HOME	String / Benutzername des gerade angemeldeten Benutzers
DISPLAY	Einstellung für den Default-Editor
UID	Pfadvariable, gibt an, wo nach ausführbaren Programmen gesucht wird
USER	Homeverzeichnis des gerade angemeldeten Benutzers
EDITOR	Gerade ausgeführte Shell mit absolutem Pfad
SHELL	Variable zur Weiterleitung grafischer Ausgaben über das Netz

12. Was verbindet das Kommando **which** mit der Shellvariablen $PATH?

13. Was verbindet das Kommando **cd** (ohne weitere Optionen und Argumente) mit der Shellvariablen $HOME?

2.6.4 Kommandos auf Dateien

1. Wie bekommt man heraus, von welchem Typ eine Datei ist? Spielen dabei Dateiendungen eine Rolle?

2. Man nenne die Befehle:
   a) Zum Kopieren
   b) Zum Umbenennen/Verschieben
   c) Zum Löschen von Dateien!
   d) Zum Löschen eines leeren Verzeichnisses!
   e) Was gibt man ein, wenn man sich die Anleitung zum Befehl **ls** ansehen will?
   f) Wo gibt es darüberhinaus Informationen wenn der Versuch aus e) fehlschlug?

3. Wie erzeuge ich in meinem Homeverzeichnis ein Verzeichnis *uebung* mit einem Unterverzeichnis *nummer01*, welches wiederum ein Unterverzeichnis *aufgabe01* enthält? Wie lösche ich diese Verzeichnisse wieder (alle auf einmal direkt auf oberster Ebene meines Homeverzeichnisses)?

2.6.5 Wichtige Tastaturkürzel

1. Mit welcher Magic-Sysrequest-Taste kann ich den Abschuss (kill) aller Prozesse bis auf den Init-Prozess erreichen?? Wo wird es gesteuert, ob es funktioniert oder nicht? Wo findet man die Informationen hierzu?

2. Wie wechselt man von der grafischen Konsole in den Textmodus (und umgekehrt)?

3. Wie beende ich X direkt per Tastatur? Welche Probleme handele ich mir damit evtl. ein?

4. In der Shell (Textmodus oder “xterm”): Wie kann man “hochblättern” um bereits aus dem Terminal “verschwundene” Zeilen wieder sichtbar zu machen?

5. Wie lässt sich während einer grafischen Sitzung die Bildschirmauflösung unter X verändern?

6. Wie lässt sich in der Shell ein Bereits vor kurzem ausgeführtes Kommando erneut aufrufen (History-Funktion)?

7. Man verbinde die untenstehenden Blöcke geeignet: Links steht eine Tastenkombination und rechts wofür sie gut sein könnte!

Cursor nach oben	Grafische Oberfläche ”hart” beenden
CRTL-E	Gerade in der Shell laufendes Programm abbrechen
CTRL-D	Auf die erste Textkonsole wechseln
CTRL-C	An den Anfang der Kommandozeile springen
Shift-Bild nach oben	Den Inhalt der Pfadvariablen wiederherstellen
CTRL-ALT-F1	Shell beenden / Ausloggen
CTRL-ALT-BACKSPACE	An das Ende der Kommandozeile springen
CRTL-A	In der Command-History nach oben blättern
Pos1	Nach oben ”verschwundenen” Bildinhalt wieder sichtbar machen

2.6.6 Editoren

1. Wie heisst Ihr/Dein Lieblings-Texteditor (verordnete Standardeditor)? Wie kann ich in diesem:
   a) Ihn verlassen ohne zu speichern?
   b) Verlassen und dabei speichern?
   c) Einen Block kopieren,
   d) verschieben,
   e) löschen?

2. Editieren Sie eine Datei mit dem Texteditor Ihrer Wahl: Nennen Sie sie *uebung1.txt* und schreiben Sie in diese Datei: Ihren Usernamen (Zeile 1), Vollständigen Namen (Zeile 2)!

3. Tauschen Sie in einem Übungstext (z.B. diesem Skript) die Umlaute gegen ihre Umschrift (ae, ue ...) aus!

4. Hängen Sie an die Übungsdatei den veränderten Übungstext (in Teilen) an!

Kapitel 3

Shell und Shellprogrammierung

3.1 Einleitung

Sehr viele administrative Aufgaben laufen unter Unix/Linux durch Skripte. Diese Skripte sind üblicherweise Batches [1] von Programmabfolgen und Kontrollstrukturen, die von der Shell interpretiert werden. Möchte man überprüfen, welche Rechner gerade in einem bestimmten Netzwerkbereich erreichbar sind, kann das z.B. durch folgendes Skript geschehen:

```
declare -i i ; i=0
while [ $i -lt 255 ] ;
  do ping -c1 -w1 172.16.20.$i 1>/dev/null \
        && echo 172.16.20.$i is alive ; i=$i+1
done
```

Die Init-Skripte zum Starten und Stoppen von Diensten sind ebenfalls Batches von Kommandos, die im Shell-Kontext ausgeführt werden; hier exemplarisch ein Ausschnitt aus der */etc/init.d/nfsserver*:

```
[...]
case "$1" in
    start)
      PARAMS=3
      test "$USE_KERNEL_NFSD_NUMBER" -gt 0 && PARAMS="$USE_KERNEL_NFSD_\
NUMBER"

      echo -n "Starting kernel based NFS server"
      /usr/sbin/exportfs -r
      rc_status
      /usr/sbin/rpc.nfsd $PARAMS
      rc_status
      startproc /usr/sbin/rpc.mountd
      rc_status -v
      ;;
```

[1] engl. Stapel, ursprünglich von der sequenziellen Programmabarbeitung mit Lochkarten

```
    stop)
      echo -n "Shutting down kernel based NFS server"
      /usr/sbin/exportfs -au
      killproc -n -KILL nfsd
      rc_status
      killproc    -TERM /usr/sbin/rpc.mountd
      rc_status -v
      ;;
[... some more lines ...]
    *)
        echo "Usage: $0 {start|stop|status|try-restart|restart|\
 force-reload|reload}"
        exit 1
        ;;
esac
rc_exit
```

3.2 Einige Bash-Grundlagen

3.2.1 Die Standardshell

An dieser Stelle soll nur kurz auf die Linux-Standard-Shell, die Bourne Again Shell **bash**, eingegangen werden. Weitere Informationen findet man in der Man-Page (**man bash**). Die **bash** verfügt, neben der obligatorischen Manpage, auch über eine eingebaute Hilfe Funktion. Mit dem Kommando **help** wird eine Übersicht der **bash** eigenen Kommandos angezeigt. Shell-Skripte sind das A und O der UNIX-/Linux-Systemadministration. Fast alle Administrationsaufgaben werden automatisch von Skripten abgearbeitet. Für Administratoren ist es daher unerlässlich, wenigstens Grundzüge der Shell-Programmierung zu beherrschen. Shell-Skripte besitzen eine erweiterte Syntax: Es stehen mathematische Operationen und Vergleichsoperationen wie in anderen Programmiersprachen zur Verfügung. Shells kennen verschiedene Kontrollstrukturen.

Eine der nützlichsten Funktionen der **bash** ist die Möglichkeit, Programm- und Dateinamen zu vervollständigen. Dabei kann dieses zu jeder Zeit am Shell-Prompt geschehen: Man tippe ein paar Zeichen eines Befehls und betätige anschliessend die Tabulator-Taste. Wenn die Eingabe bis zu der Stelle, an der der Tabulator ausgelöst wurde, eindeutig war, es also kein weiteres ausführbares Programm gibt, das mit diesen Buchstaben beginnt, wird die Eingabe automatisch vervollständigt. Sollte es zwei oder mehr Mglichkeiten geben, so wird man mittels Signalton verständigt. Ein nochmaliges Drücken der besagten Taste zeigt nun alle erhältlichen Alternativen an. Man probiere es einfach an einem Beispiel aus: Nehmen wir an, man möchte sich die Datei */var/log/messages* ansehen. Dazu benutzt man z.B. das Programm **less**. Man tippe also einfach mal "le" und die Tabulator-Taste zweimal. Nun werden einige Programme angezeigt, welche mit der besagten Buchstabenkombination beginnen.

Nun tippe man so lange weitere Zeichen ein, bis die Eingabe eindeutig ist, in diesem Beispiel sollte ein zusätzliches "s" dem geforderten Zweck genügen. Jetzt tippe man nochmals die TAB-Taste, und der Befehl wird ergänzt. Was bei Befehlen vielleicht noch etwas banal aussieht, wird bei langen Dateinamen mit kryptischen Zeichen sehr schnell zu einer großen Erleichterung. Neben dieser Funktionalität beherrscht die Bash weitere Komplettierungen.

Wozu die **bash** den begonnenen Namen zu vervollständigen sucht, hängt von der getätigten Eingabe ab. Wenn die Eingabe mit einem $ beginnt, so versucht die Bash, einen Variablennamen daraus zu machen. Beginnt die Eingabe hingegen mit einer Tilde (~) , so versucht sie einen Benutzernamen zu bilden. Beginnt sie mit @, versucht sie die Eingabe zu einem Hostnamen zu vervollständigen. Wenn keine dieser Bedingungen zutrifft, sucht die bash nach einem Alias- oder Funktionsnamen. Selbstverständlich muss der Name, zu dem die **bash** vervollständigt - sei es nun eine Variable, ein Benutzername, ein Hostname, ein Alias, eine Funktion oder ein Pfad - auch wirklich existieren. Wenn alle Versuche, eine passende Vervollständigung zu erreichen, fehlschlagen, ertönt ein kurzer Signalton.

3.2.2 Aufbau der Kommandozeile

Der Bash-Prompt, vielfach auch als Eingabeaufforderung bezeichnet, kann zwar beliebig angepasst werden, doch auf den meisten Systemen ist es für normale Benutzer (also alle außer dem Systemadministrator) standardmäßig eine Zeichenfolge nach dem Schema:

```
benutzer@computer:verzeichnis>
```

Wenn man sich auf dem Computer im eigenen Heimatverzeichnis befindet, welches durch die Tilde (~) repräsentiert wird, lautet der Bash-Prompt entsprechend:

```
dirk@dozent:~>
```

Der Benutzername ist "dirk" (ein ganz normaler Linux-Benutzer ohne Administratorrechte) und der Rechnername lautet "dozent". Dahinter blinkt der Cursor, an dem die Eingabe erfolgen kann. Nachdem ein Kommando ausgeführt wurde, gelangt man wieder zum Prompt. Es gibt sehr viele Kommandos, wie man sich überzeugen kann, wenn man einfach zweimal auf die Tabulatortaste tippt. Einige sind in die Bash direkt eingebaut, z.B. das Kommando **echo** zur Ausgabe von Zeichenketten. Andere sind klassische ausführbare Programme oder Skripte, wie **mount** zum Einbinden von Dateisysteme, oder **updatedb** zum Erzeugen der Locate-Datenbank. Auch Programme für die grafische Benutzerschnittstelle, wie **openoffice** können von der Bash aus gestartet werden, wenn die Bash in einer Terminal-Emulation unter X11 läuft.

3.2.3 Die Kommando-Geschichte

Fast jede Linux-Shell verfügt über eine Liste der zuletzt abgesetzten Kommandos, eine sogenannte History. Selbst wenn man nur gelegentlich die Kommandozeile verwendet, erweist sich die History als ein ausgesprochen nützlicher Helfer. Das gilt umso mehr, wenn man ausgiebigen Gebrauch von der Shell macht. Die Möglichkeiten zur Nutzung der History entsprechen der Benutzung eines effizienten Editors und werden in ihrem vollen Umfang nur von den Wenigsten benutzt. Sie gehen weit über die Möglichkeiten beispielsweise von “doskey” hinaus, welches vielleicht noch aus DOS-Zeiten bekannt sein könnte.

Die zuletzt eingetippten Befehle können mit den Cursor-Tasten (Pfeil hoch und runter) ausgewählt und editiert werden. Eine Rückwärtssuche in allen gespeicherten Kommandoaufrufen kann mittels [STRG]-R erfolgen. Nach dem Abmelden von einer Shell wird die “Geschichte”, die History, in der Datei *.bash_history* im Home-Verzeichnis des jeweiligen Benutzers abgelegt. Möchte man nicht, dass andere die Command History nachvollziehen können, hilft ein Link dieser Datei nach */dev/null*. Für eine Sitzung kann man die History durch die Eingabe von:
`export HISTFILE=/dev/null` ebenfalls abschalten. Mit `!!` wird der letzte Befehl ausgeführt, mit !-2 wird der vorletzte mit !-3 der vorvorletzte usw. ausgeführt. Mit “!string” wird der letzte Befehl, der mit “string” begonnen hat, ausgeführt. Der eingebaute Bash-Befehl **history** zeigt eine Liste der zuletzt abgesetzten Kommandos an.

3.2.4 Ein- und Ausgabe

Programme verhalten sich meist so, dass sie bestimmte Daten aufnehmen, diese Daten auf irgendeine Weise verwenden, um schließlich wieder Daten auszugeben. Dieses Schema verdeutlicht sich noch bei interaktiven Programmen, die immer wieder Informationen vom Benutzer annehmen und ihm andere Informationen zurückliefern. Eine Shell ist ein typisches interaktives Programm. Zu diesem Zweck muß sie über einen Eingabekanal verfügen, über den sie Information aufnehmen kann. Dieser Eingabekanal existiert tatsächlich und erhält unter Linux die Bezeichnung Standardeingabe.

Womit aber ist die Standardeingabe verbunden? Üblicherweise wird es die Tastatur sein: Die Shell nimmt Zeichen für Zeichen von der Tastatur entgegen und gibt diese Zeichen auch sofort auf dem Bildschirm aus. Die Ausgabeseite auf dem Bildschirm kann von der Standardeingabe erledigt werden. Die Shell verfügt also über einen weiteren Kanal, der folgerichtig als Standardausgabe bezeichnet wird. Die Standardausgabe der Shell ist üblicherweise mit einer Konsole[2] oder einem grafischen Pseudo-Terminal, wie **xterm** oder **konsole** verbunden, so dass eingetippte Zeichen sichtbar werden.

Darüberhinaus existiert ein dritter Kanal, der eine besondere Aufgabe zu erfüllen hat, der sogenannte Standardfehlerkanal. Wie der Name schon sagt, dient der Kanal zur Ausgabe von Fehlermeldungen, wenn der Programmlauf aus irgendeinem Grund nicht ordnungsgemäss fortgesetzt werden konnte. Üblicherweise ist Standardfehler ebenfalls mit dem Bildschirm verbunden und schreibt daher seine Meldungen zwischen die gewöhnliche Ausgabe. Es macht jedoch Sinn, Stardardausgabe und Standardfehler voneinander zu trennen, um die Möglichkeit zu haben, gewöhnliche Ausgaben und Fehlerausgaben getrennt zu verarbeiten. Beispielsweise könnte man die Fehlerausgabe in eine Datei umlenken, um sie später zu analysieren, während die gewöhnliche Ausgabe weiterhin über den Bildschirm läuft.

An dieser Stelle gilt es, etwas Wichtiges zu verstehen: Standardeingabe, Standardausgabe und Standardfehler sind lediglich Kanäle, die mit irgendeiner Quelle

[2] Monitorausgabe im Textmodus

und irgendeinem Ziel verbunden sein können. Standardeingabe ist nicht gleich Tastatur. Und Standardausgabe ist nicht gleich Monitor. Es gibt viele andere Quellen und Ziele, mit denen diese Kanäle verbunden werden können, wie beispielsweise Dateien oder andere Programme. Bei einer Shell macht es jedoch Sinn, Tastatur und Monitor als Eingabe und Ausgabe zu verwenden, daher ist dies die Voreinstellung.

Die drei Standardkanäle werden von Linux wie Dateien behandelt. Für geöffnete Dateien verwaltet das System eine Liste von Dateideskriptoren, die mit fortlaufenden ganzen Zahlen bezeichnet werden. Die Zahlen von 0 bis 2 sind für die drei Standardkanäle vorbelegt:

Kanal	Bezeichnung	Nummer
Standardeingabe	stdin	0
Standardausgabe	stdout	1
Standardfehlerkanal	stderr	2

Tabelle 3.1: Standardkanäle der Shell

Bei der Umlenkung dieser Kanäle werden diese Bezeichnungen benötigt! Bereits sehr früh wurde das Prinzip der "Pipe" von den Mainframe-Betriebssystemen abgeschaut. Röhre ist hierfür keine schlechte Übersetzung. Man kann in diese etwas "hineinleiten" und am anderen Ende kommt etwas wieder heraus. Wie bereits beschrieben, gibt es sehr viele kleine, spezialisierte Programme unter Unix/Linux, die mit speziellen Parametern aufgerufen werden können. Daher entstand die Idee eine geeignete Schnittstelle zwischen diesen zu schaffen, um eine geeignete Hintereinanderschaltung dieser Tools zu erreichen.

Sinnvoll ist daher eine Schnittstelle zwischen diesen Programmen, welche Daten austauscht oder die Ergebnisse eines Programmlaufs in einem weiteren Programm weiterverarbeitet. Diese Schnittstelle ist in Form von "Pipes" (|) realisiert. Bekannt ist sicherlich das System-Kommando **ls**, welches dazu dient sich die Dateien in einem Verzeichnis auflisten zu lassen. Wenn man sich aber in einem Verzeichnis mit einer sehr hohen Anzahl von Dateien befindet und die Liste durchblättern möchte, kann man die Ausgabe, z.B. an das Kommando **more** weiterleiten (z.B. durch `ls -la | more`).

Weiterhin möchte man evtl. die Zahl der Dateien ermitteln, welches durch einfaches Zählen leicht etwas umständlich wird. Um die Zahl von Zeichen, Wörtern oder Zeilen in einer Datei zu ermitteln, steht das Kommando **wc** (Word Count) zur Verfügung. Ein Weg besteht also darin, die Ausgabe von `ls -l` in eine Datei zu schreiben und mittels `wc -l` die Zahl der Zeilen ermitteln zu lassen. Der Umweg über eine Datei lässt sich jedoch mittels einer Pipe umgehen: `ls -l | wc -l` . Unix/Linux benutzt an dieser Stelle das Zeichen "|" (Pipe). Man verkünpfte nun einfach die beiden Kommandos mittels dieses Zeichens zu einer Zeile und erhält nun die Anzahl der Dateien im aktuellen Verzeichnis. Nur ein kleiner Haken an dieser Stelle: **ls** gibt als erste Zeile keinen Dateinamen aus, sondern eine Zeile, in der Informationen zum entsprechenden Verzeichnis angezeigt werden, man muss hier also vom Ergebnis eine Zeile subtrahieren, um auf das exakte Ergebnis zu kommen.

Statt der Tastatur als Standardeingabe kann man alternativ aus einer beliebigen Datei lesen, die mit absolutem oder relativem Pfad angegeben wird: `sort`

`</etc/passwd`. Das Zeichen “<” erzeugt die gewünschte Funktionalität. Dies mag an dieser Stelle vielleicht etwas sinnlos erscheinen, da der **sort**-Befehl einen Dateinamen als Argument übernehmen kann. Es gibt jedoch Situationen, in denen die Quellenauswahl nur auf diese Weise realisiert werden kann.

Nicht alle Ergebnisse möchte man sich direkt am Bildschirm ansehen, sondern sie für spätere Bearbeitung oder aufgrund sehr grosser Datenmengen in eine Datei schreiben. Dieses geschieht durch den Einsatz von “>”. Sollen die Daten an eine bereits existierende Datei angehängt werden, ohne den bestehenden Inhalt zu überschreiben setzt man das “>” doppelt hintereinander: “>>”. Gerade für Sachen, die einen nicht interessieren, bietet sich die Umleitung in die Spezialdatei */dev/null* an. Möchte man z.B. testen, ob ein bestimmtes Verzeichnis existiert, genügt der Return-Code, die Ausgabe des Kommandos selbst, kann hingegen entsorgt werden: `ls -al /usr/bin > /dev/null`. Da der Ausgabe- und der Fehlerkanal verschiedene Deskriptoren haben, lassen sich mit `ls -al /usr/bin 2> /dev/null` auch nur die Fehlermeldungen wegdrücken. Das davor gegebene Beispiel müsste formal eigentlich `ls -al /usr/bin 1> /dev/null` lauten, wenn aber keine Kanalnummer angegeben ist, wird die Standardausgabe als Defaultwert angenommen. Sollen einfach beide Ausgaben gemeinsam in eine Datei geschrieben werden, können beide Kanäle mit `ls -al /usr/bin &> /dev/null` (im Fall der **bash**) oder mit `ls -al /usr/bin 2 >&1> /dev/null` gemeinsam umgelenkt werden.

3.2.5 Abkürzen von Befehlen

Der Alias-Mechanismus dient der Ersparnis von Tipparbeit, macht Kommandos leichter erinnerbar, verschönert Kommandoausgaben und kann auch zur Absicherung gegen Tippfehler verwendet werden. Ein Alias ist eine definierte Zeichenfolge, die für eine andere Zeichenfolge steht. Welche Aliase in der aktuellen Shell definiert sind, kann mittels **alias** abgefragt werden:

```
dirk@hermes:~> alias
alias +='pushd .'
alias -='popd'
alias ..='cd ..'
[... weitere Alias-Deklarationen ...]
```

Die Alias-Definitionen muss man nicht jedesmal eingeben, sondern kann sie in den Setup-Dateien unterbringen. Listen solcher Abkürzungen der Form **alias <Abkürzung> <Befehl (mit Optionen)>** trägt man am besten in die globale Datei */etc/profile* oder in benutzerlokale Dateien, wie *.profile* oder *.bashrc* ein. Der Befehl **alias** ohne Argumente zeigt alle bereits vergebenen Abkürzungen an. Mit dem Befehl **unalias** kann man Definitionen wieder aufheben.

3.2.6 Wildcards in Dateinamen

Viele Kommandos, zum Beispiel **ls** oder **cp** haben als Argumente die Namen von Dateien oder Verzeichnissen. Namen, die Wildcards enthalten, werden zu einer Liste von Namen expandiert. Wildcards sind nahe verwandt zu Regulären Ausdrücken, die von vielen Programmen verarbeitet werden können. Beispiele von Wildcards sind:

Wildcard	**Bedeutung**
?	beliebiges Zeichen
$*$	beliebige Zeichenkette, auch der Länge Null
$[abc]$	die einzelnen Zeichen a, b oder c
$[a-dg-i]$	die einzelnen Zeichen a, b, c, d, g, h oder i
$[bd]$	nicht die einzelnen Zeichen b oder d
$\{dies, das\}$	optionale Zeichenketten

Tabelle 3.2: Beispiele für Wildcards

3.2.7 Zeichen mit besonderer Bedeutung

Eine ganze Reihe von Zeichen haben für die Shell besondere Bedeutung, ein Teil hiervon wurde bereits in den vorherigen Abschnitten vorgestellt:

```
[SPACE], [TAB], [CR], $, *, [, ], ?, {, }, ~, -,
<, >,&, !, |, ;, (, ),\ , ’, ‘‘, ‘.
```

Um diese Zeichen benutzen zu können, ohne dass die Shell sie interpretiert, müssen diese quotiert werden. Ein einzelnes Sonderzeichen wird durch vorstellen eines Backslash (“\”) quotiert:

```
echo Bitte Drücken Sie eine Taste\!
```

Wenn man doppelte Anführungsstriche (“Text”), so werden von der Shell nur noch !, $, und ‘ als Sonderzeichen behandelt. Man kann also z.B. noch Umgebungsvariablen verwenden: `echo ‘‘$PRINTER’’` gibt den Wert der Variablen “PRINTER” aus. (vgl. Kap. 3.4, S. 38 Verwendet man einfache Anführungsstriche (’Text’) [3], so werden von der Shell nur noch ! und ‘ als Sonderzeichen behandelt.

3.2.8 Spezielle Escape-Sequenzen

Einige Zeichen haben in Zusammenhang mit dem Backslash in der Shell eine besondere Bedeutung und werden speziell interpretiert, bevor eine Ausgabe auf dem Bildschirm erscheint. Dies wird z.B. zum Aufbau der Prompt-Variablen[4] benutzt oder für besondere Ausgaben in der */etc/issue* oder */etc/motd*.

3.2.9 Jobkontrolle

Wenn ein Kommando abgesetzt wurde, wartet die Shell normalerweise bis dieses ordnungsgemäß beendet wurde. Anschließend gibt sie wieder den Prompt aus, um auf das nächste Kommando zu warten. Manche Kommandos können jedoch viel Zeit benötigen oder gar während der kompletten Arbeitssitzung laufen. Damit man nicht für jedes Programm, das gestartet werden soll, eine eigene Shell öffnen muss, können Programme, wie man sagt, im Hintergrund gestartet werden. Das bedeutet nichts anderes, als dass die Shell nicht erst auf die Beendigung des abgesetzten Programmes

[3] nicht ‘ das sogenannte Backtick

[4] man sehe sich die Ausgabe von `echo $PS1` an

Escape-Sequenz	Bedeutung
\h	repräsentiert den Hostnamen einer Maschine
\l	enthält die Angabe über das verwendete virtuelle Terminal Device, wie z.B. *tty2*
\n	liefert den Netzwerknamen der aktuellen Maschine
\r	zeigt das aktuelle Kernel-Release an
\u	meldet die Kennung des gerade aktiven Benutzers
\w	liefert das "working directory". Hier könnte analog auch das shellinterne Kommando **pwd** aufgerufen werden

Tabelle 3.3: Beispiele für Escape-Sequenzen

wartet, sondern sofort wieder einen Prompt ausgibt, um ggf. ein weiteres Kommando entgegenzunehmen. Man kann ein Kommando auch direkt im Hintergrund starten, indem man "&"[5] an das Kommando anhängt. Weitere Kommandos können dann eingegeben werden, während das erste ausgeführt wird:

```
dirk@hermes:~> kdvi lak.dvi &
```

Unter der grafischen Oberfläche bedeutet das Anhängen von "&" im Terminalfenster, dass das Fenster, aus dem eine Anwendung gestartet wurde, weiter benutzt werden kann.

Der Sinn der Bezeichnungen Vordergrund und Hintergrund ist unmittelbar eingängig. In technischer Hinsicht sind Vordergrund und Hintergrund zwei Begriffe, die sich im Zusammenhang mit der Shell nur auf ein bestimmtes Terminal beziehen können. Ist die sogenannte Prozess-Gruppen-ID eines Prozesses identisch mit der eines Terminals, so kann der Prozess von diesem Terminal Signale empfangen. Solche Prozesse laufen im Vordergrund. Hintergrund-Prozesse sind solche, deren Prozess-Gruppen-Id von der des Terminals verschieden sind. Sie sind daher auch immun gegen irgendwelche Signale, die an der Tastatur eingegeben werden.

Der Begriff des Jobs ist eine Abstraktion, welche von der Shell zur Verwaltung eingesetzt wird. Als Job wird jede Pipeline bezeichnet, aus wievielen Kommandos oder Prozessen auch immer sie bestehen mag. Dem Job wird von der **bash** eine Jobnummer zugewiesen, unter welcher er angesprochen werden kann. Diese Jobnummer ist nicht mit der Prozess-ID zu verwechseln!

Jede Shell implementiert Funktionen zur Jobkontrolle. Ein laufendes Kommando kann mit [STRG]-[Z] angehalten werden. Nach dem Anhalten meldet sich die Shell mit ihrem Prompt wieder, neue Kommandos können eingegeben werden. Das unterbrochene Kommando kann mit dem Befehl **fg** (Foreground) wieder aktiviert werden. Mit dem Befehl **bg** (Background) kann das Kommando im Hintergrund fortgesetzt werden, sofern das Kommando keine Eingabe über die Tastatur (stdin) erfordert. Befinden sich mehrere Prozesse im Hintergrund kann dem **fg** die Jobnummer folgen, wie es weiter unten detailliert beschrieben wird. Die komplette Liste der in einer Shell laufenden Jobs kann mittels des Kommandos **jobs** angezeigt werden:

[5] das sogenannte "kaufmännische Und" bzw. "ampersand" im englischen

```
dirk@randy2:~> jobs
[2]-  Running                 konqueror lak.pdf &
[3]+  Running                 kdvi SharedFiles/tex/lak/lak.dvi &
```

In den eckigen Klammern wird die zugeteilte Jobnummer angezeigt. Sie unterscheidet sich von der sogenannten Prozessnummer, die hinter der Jobnummer angegeben wird. Das Plus-Zeichen bei der Ausgabe des **jobs**-Kommandos markiert den zuletzt gestarteten Job, das Minus-Zeichen den als vorletztes gestarteten Job.

Es gibt eine Reihe von Möglichkeiten, sich auf einen bestimmten Job zu beziehen. Das Zeichen % leitet einen Jobnamen ein. Jobnummer n kann als %n angesprochen werden. Man kann sich auch auf einen Job beziehen, indem man dem % die ersten Buchstaben des Kommandos voranstellt, mit dem man den Job gestartet hat. Hat man z.B. Kommando gestartet, kann man sich darauf mittels %ko beziehen, falls kein weiterer laufender Job so beginnt. Auch eine Art von Wildcard ist erlaubt: %?ommando oder auch %?mmando bezieht sich ebenfalls auf den Job, der mittels **kommando** gestartet wurde. Wenn das angegebene Präfix oder Muster auf mehr als einen Job paßt, erfolgt eine Fehlermeldung. %% oder %+ bezieht sich immer auf den letzten Job. In den Begriffen der Shell ist das der zuletzt gestoppte Vordergrundprozess oder der zuletzt gestartete Hintergrundprozess. %- bezieht sich entsprechend auf den zuvorletzt gestarteten Job. Ein im Hintergrund laufendes oder unterbrochenes Programm benötigt unter Umständen die Tastatur bzw. den Bildschirm zur Ein- bzw. Ausgabe, deshalb kann die Shell dann nicht beendet werden, und es erscheint beim Versuch des Ausloggens die Fehlermeldung "There are suspended jobs".

3.2.10 Skripte/Batches

Mehrere Kommandos können in einer Textdatei zusammengefasst werden, zum Beispiel, um immer wiederkehrende Kommandofolgen nicht jedesmal explizit eintippen zu müssen. Ein solcher Stapel von sequenziell auszuführenden Kommandos wird auch als Batch bezeichnet, deshalb findet man häufig auch diese Bezeichnung. Von Kommandostapeln wird exzessiv bei den Runlevel-Skripten Gebrauch gemacht, die eine Maschine in einen bestimmten Zustand versetzen. Diese Textdateien heißen Shell-Skripte und stellen neue Kommandos dar. (Es ist sinnvoll, solche Skripte mit dem emacs zu editieren!) In Shell-Skripten sind mit "#" beginnende Zeilen Kommentare, diese Zeilen werden nicht ausgeführt. Beginnt die erste Zeile mit "#!" so wird zur Ausführung des Shell-Skripts die danach angegebene Shell gestartet: `#!/bin/bash` ruft die Bourne Again Shell auf. Shell-Skripte müssen mit dem Befehl `chmod u+x Skriptname` als ausführbare Kommandos gekennzeichnet werden.

Es ist in der Shell auch möglich - ähnlich wie in einer "richtigen" Programmiersprache - Funktionen zu deklarieren und zu benutzen. Mit dem Kommando **return** hat man die Möglichkeit, aus einer Funktion einen Wert zurückzugeben. Falls das Kommando **return** nicht eingesetzt wird, hat die Funktion als Rückgabewert den Standardoutput der eingesetzten Kommandos. Beispiel:

```
shadowfileexist() {
      if [ -f /etc/shadow ]
then
```

```
return 1 #shadow file exist
      fi
return 0 #does not exist
}
```

Beispiel:

```
count () {
      ls | wc -l # ls: Liste aller Dateien im Verzeichnis
      # wc: Word-Count, zählt Wörter
}
```

Die Funktion gibt die Anzahl der Dateien im aktuellen Verzeichnis zurück; aufgerufen wird diese Funktion wie ein Befehl, also einfach durch die Eingabe von **count**.

3.3 Filter

Filter sind Programme, die von der Standardeingabe lesen und zur Standardausgabe schreiben, insgesamt also filtern. Wichtige Filter sind:

- sed - Stream Editor
- awk - Muster-Such- und Verarbeitungssprache
- grep - Durchsucht Texte nach Mustern

Filter kann man kombinieren, z.B. durchsucht `cat *.txt | grep Unix | sed s/alt/neu/g` alle mit *.txt* endenden Dateien nach dem Wort Unix und ersetzt das Wort "alt" durch "neu". Die Ausgabe von **sed** erfolgt auf den Bildschirm.

3.4 Variablen

3.4.1 Umgebungsvariablen

Umgebungsvariablen enthalten Zeichenketten als Werte. Umgebungsvariablen werden entweder automatisch zum Beispiel vom Login-Prozess übernommen, durch System-Skripten oder vom Benutzer gesetzt. Standardvariablen, die automatisch gesetzt werden, sind "USER", "UID", "HOME". Das Aussehen des Prompts wird üblicherweise in der systemweiten Konfigurationsdatei */etc/profile* definiert. Benutzer können ihre eigenen Bedürfnisse durch das Setzen der "PRINTER" oder "PAGER" Variablen umsetzen. Dies erfolgt normalerweise in den benutzereigenen Dateien des Home-Verzeichnisses, wie *.profile* und in Abhängigkeit von der Shell in *.bashrc*, *.cshrc*, *.kshrc*.

Die Variablen werden von Programmen ausgewertet, die daraufhin ihr Verhalten danach ausrichten (sollten). Den Wert einer solchen Variablen kann man am besten mit: `echo $VARIABLENNAME` abfragen. Mittels **set** bzw. **export** läßt sich die komplette Liste aller in einer Shell definierten und an abgeleitete Programme weitergereichten Variablen anzeigen:

```
test2:~ # export
declare -x COLORTERM="gnome-terminal"
declare -x CPU="i686"
declare -x DISPLAY=":0"
declare -x GDMSESSION="Default"
declare -x GDM_LANG="en_US"
declare -x GNOMEDIR="/opt/gnome"
declare -x GNOME_PATH=":/opt/gnome:/usr"
declare -x GNOME_SESSION_NAME="Default"
declare -x HELP_BROWSER="gnome-help-browser"
declare -x HOME="/root"
declare -x HOST="test2"
declare -x HOSTNAME="test2"
declare -x HOSTTYPE="i386"
[... viele weitere Deklarationen ...]
declare -x USERNAME="dirk"
declare -x WINDOWID="31457868"
declare -x WINDOWMANAGER="/usr/X11R6/bin/kde"
declare -x XAUTHORITY="/home/dirk/.Xauthority"
declare -x XKEYSYMDB="/usr/X11R6/lib/X11/XKeysymDB"
declare -x XNLSPATH="/usr/X11R6/lib/X11/nls"
declare -x no_proxy="localhost"
```

Variable	**Funktion**
EDITOR	Enthält den Namen des bevorzugten Texteditors, Default ist **vi**.
SHELL	Enthält den Namen der Login-Shell.
DISPLAY	Enthält den Namen des Rechners und die Nummer des Bildschirms, auf dessen Bildschirm eine Anwendung gestartet werden soll.
LANG	Enthält die Beschreibung der Lokalisierung (Sprache), z.B. “de_DE” für Deutsch.
HOST	Enthält den Rechnernamen der Maschine.
HOME	Enthält den Pfad des Home-Verzeichnisses.
PRINTER	Enthält die Bezeichnung des Default-Druckers.
PATH	Enthält den Suchpfad für ausführbare Programme.

Tabelle 3.4: Wichtige Umgebungsvariablen der Shell

Der Suchpfad der Shell ist eine Liste von Verzeichnissen. Der Suchpfad wird durch die Umgebungsvariable “PATH” definiert - eine durch Doppelpunkte getrennte Liste. Beispiel eines Pfades:

```
test2:~ # echo $PATH
/usr/sbin:/bin:/usr/bin:/sbin:/usr/X11R6/bin:/opt/gnome/bin:.
```

Wird ein Programm aufgerufen, so werden alle in “PATH” definierten Verzeichnisse nach diesem Programm durchsucht. Der letzte Punkt im obigen Beispiel bewirkt, dass die Shell nach erfolglosem Suchen in den angegebenen Verzeichnissen auch noch im aktuellen Verzeichnis nachschaut. Dieses ist jedoch aus Sicherheitsgründen für den Benutzer "root" defaultmäßig abgeschaltet. Sonst könnte man versuchen, ein spezielles **ls** zu schreiben, welches der Systemadministrator z.B. beim Durchsuchen des */tmp*-Verzeichnisses ohne sein Wissen mit seinen privilegierten Rechten ausführt und dabei das System kompromittiert.

Die Reihenfolge der Verzeichnisse im Suchpfad ist wichtig: Im ersten Verzeichnis wird zuerst gesucht. Programme, die nicht in einem Verzeichnis des Suchpfades zu finden sind, können nur durch Angabe ihres Pfades aufgerufen werden.

3.4.2 Variablen zur Shellprogrammierung

Es ist möglich, in der Shell Variablenzuweisungen vorzunehmen, wobei ohne weitere Angaben von einer Stringvariablen ausgegangen wird. Die Syntax der Zuweisung ist sehr einfach: `VAR01=100` legt eine Variable des Namens *VAR01* an und weist dieser den Wert 100 zu. Als Konvention hat sich eingebürgert, dass längere Variablennamen meistens mit vollständig großen Buchstaben geschrieben werden, um besser zwischen den generell klein geschriebenen Programmnamen, bzw. Befehlen und den Skriptvariablen unterscheiden zu können. Der Wert einer Skriptvariablen wird durch das Zeichen $ dereferenziert, wie schon bei den Umgebungsvariablen demonstriert: `echo $VAR01`.

Mit Integervariablen (Zuweisung bei **declare** mit der Option "-i") kann auch gerechnet werden. Die einfache Addition ist direkt implementiert (s.o.), kompliziertere Ausdrücke lassen sich mittels **expr** berechnen.

Da die Standard-Shell keine arithmetischen oder logischen Ausdrücke auswerten kann, muss dazu ein externes Programm benutzt werden. Mit Hilfe des Shell-Kommandos **test** kann man den Zustand bestimmter Dateien oder Variablen erfragen. Die Bedingungen können durch Verknüpfung kombiniert werden.

Option	Funktion
-b Datei	Datei existiert und ist blockorientiertes Gerät
-c Datei	Datei existiert und ist zeichenorientiertes Gerät
-d Datei	Datei existiert und ist Verzeichnis
-f Datei	Datei existiert und ist reguläre Datei (kein Link etc.)
-g Datei	Datei existiert und hat das Gruppen-ID-Bit gesetzt
-h Datei	Datei existiert und ist Symboliclink
-k Datei	Datei existiert und hat das Sticky-Bit gesetzt
-p Datei	Datei existiert und ist Named Pipe
-r Datei	Datei existiert und ist lesbar
-s Datei	Datei existiert und ist nicht leer
-t [n]	Offene Dateideskriptor *n* gehört zu einem Terminal
-u Datei	Datei existiert und hat das Setuid-Bit gesetzt
-w Datei	Datei existiert und ist beschreibbar
-x Datei	Datei existiert und ist ausführbar

Tabelle 3.5: Optionen von **test**

Einige Beispiele für den Einsatz des Kommandos:

```
test -f /etc/issue
test -x /bin/ls
test $VAR01 -eq $VAR02
```

Vergleich	Funktion	Vergleich	Funktion
-n str1	Länge der Zeichenfolge str1 ist ungleich Null	n1 -eq n2	n1 ist gleich n2
-z str1	Länge der Zeichenfolge str1 ist gleich Null	n1 -ge n2	n1 ist größer oder gleich n2
s1 = s2	Zeichenfolgen s1 und s2 sind identisch	n1 -gt n2	n1 ist größer als n2
s1 != s2	Zeichenfolgen s1 und s2 sind nicht identisch	n1 -le n2	n1 ist kleiner oder gleich n2
Zeichenfolge	Zeichenfolge ist nicht Null	n1 -lt n2	n1 ist kleiner n2
		n1 -ne n2	n1 ist ungleich n2

Tabelle 3.6: Bedingungen für Zeichenfolgen und Ganzzahlvergleiche

Anstelle des Aufrufes von **test** kann man die Abfragen auch einfach in eckige Klammern [...] fassen. Hierzu existiert üblicherweise auf allen Systemen (mit einer tch als Shell) ein Link namens [auf das Programm **test**. Dementsprechend ist es auch zwingend erforderlich, nach der Klammer ein Leerzeichen zu schreiben. Das dient dazu, Bedingungen in **if**-Abfragen und ähnlichem lesbarer zu machen. Um dieses Konzept der Lesbarkeit zu unterstützen, sollte man diese öffnende Klammer auch wieder schließen. In der bash allerdings ist die [kein Link; hier muss sie wieder geschlossen werden. Die angegebenen Beispiele würden (egal in welcher Shell!) dann wie folgt dargestellt:

```
[ -f /etc/issue ]
[ -x /bin/ls ]
[ $VAR01 -eq $VAR02 ]
```

3.5 Kontrollstrukturen

Das Beispiel einer Schleifenkonstruktion wurde schon in der Einführung gegeben. Die Shell kennt darüberhinaus weitere Schleifen, wie z.B. die **for**-Anweisung und bedingte Abfragen, wie **if** / **then** und **case**. Mit Hilfe einer **if**-Abfrage lassen sich bedingte Verzweigungen in ein Shellskript realisieren:

```
if [ -f /tmp/bla ]; then
  echo "/tmp/bla is da!!"
else
  echo "/tmp/bla kenn wir nich!!"
fi
```

Eine etwas sinnvollere Anwendung ist vielleicht das Umwandeln einer Netzmaske in die Zahl der sie beschreibenden Bits[6]. Dieses kann in Startskripten für eine Netzwerk- oder VPN-Konfiguration vorkommen.

```
# compute prefix bit number from netmask
ipv4_nm2bn () {
 local netmask="$1"
  set ‘IFS="."; echo $netmask‘
   n=0
    for i in $1 $2 $3 $4 ; do
    if [ $i = 0 ] ; then break ; fi
if [ $i = 128 ] ; then n=‘expr $n + 1‘ ; break ; fi
if [ $i = 192 ] ; then n=‘expr $n + 2‘ ; break ; fi
if [ $i = 224 ] ; then n=‘expr $n + 3‘ ; break ; fi
if [ $i = 240 ] ; then n=‘expr $n + 4‘ ; break ; fi
if [ $i = 248 ] ; then n=‘expr $n + 5‘ ; break ; fi
if [ $i = 252 ] ; then n=‘expr $n + 6‘ ; break ; fi
if [ $i = 254 ] ; then n=‘expr $n + 7‘ ; break ; fi
if [ $i = 255 ] ; then n=‘expr $n + 8‘ ; continue ; fi done
echo $n
}
```

Auch die **case**-Anweisung ist vergleichbar in vielen anderen Programmiersprachen vorhanden. Sie dient, ähnlich wie die **if**-Anweisung, zur Fallunterscheidung. Allerdings wird hier nicht nur zwischen zwei Fällen unterschieden (Entweder / Oder), sondern es sind mehrere Fälle möglich. Man kann die **case**-Anweisung auch durch eine geschachtelte **if**-Anweisung austauschen, allerdings ist sie ein elegantes Mittel um den Code lesbar zu halten.

```
#!/bin/sh
case Wert in
     Muster1) Befehle1;;
     Muster2) Befehle2;;
     ...
esac
```

Wenn der Wert mit dem "Muster1" übereinstimmt, wird die entsprechende Befehlsgruppe (Befehle1) ausgeführt, bei Übereinstimmung mit "Muster2" werden die Kommandos der zweiten Befehlsgruppe (Befehle2) ausgeführt. Der letzte Befehl in jeder Gruppe muss mit ";;" gekennzeichnet werden. Das bedeutet für die Shell soviel wie "springe zum nächsten **esac**", so dass die anderen Bedingungen nicht mehr überprüft werden.

In den Mustern sind die gleichen Meta-Zeichen erlaubt wie sie bei der Auswahl von Dateinamen verwendet werden können. Wenn in einer Zeile mehrere Muster angegeben werden sollen, müssen sie durch ein Pipezeichen ("|" - das logische ODER) getrennt werden.

[6] So wird z.B. *255.255.252.0* durch "22" beschrieben, ein Klasse-A-Netz durch "8", Klasse-B durch "16", Klasse-C durch "24", ...

Eine **while**-Schleife führt den enthaltenen Körper in Abhängigkeit von einer Bedingung wiederholt aus:

```
declare -i VAR=0
while [ $VAR -lt 10 ] ; do
     echo $VAR
     VAR=`expr $VAR + 1`
done
```

Diese Schleife kann z.B. dazu genutzt werden, an ein Skript übergebene Argumente und Optionen zu parsen. Verknüpft ist das Beispiel mit einem Einsatz der **case**-Anweisung:

```
(( NARG=0 ))
TEST=( $@ )
while [ $NARG -le $# ] ; do
        case "${TEST[$NARG]}" in
        *help*)
                echo -n "$0 generates dxs filesystem from existing "
                echo -ne "linux installation\nPass nfsroot with the"
                echo -e "$0 -nfsroot <value> option\n"
                exit 0
        ;;
        *nfsroot*)
                NFSROOT=${TEST[$NARG+1]}
                (( NARG=$NARG+1 ))
        ;;
        *exp*)
                NET=${TEST[$NARG+1]}
                (( NARG=$NARG+1 ))
        ;;
        esac
(( NARG=$NARG+1 ))
done
```

Die **until**-Schleife ist das Gegenstück zur **while**-Schleife: Die ebenfalls aus vielen anderen Sprachen bekannte fußgesteuerte Schleife.

```
#!/bin/sh
until Bedingung
do
        Befehle
done
```

Mit einer **for**-Schleife können einer "Zählvariable"[7] der Reihe nach verschiedene Werte gegeben werden. Der Körper dieser Schleife wird so lange ausgeführt, bis der letzte verfügbare Wert zugewiesen wurde.

[7] in dieser Variable kann anstelle von Zahlen auch eine Liste von Dateinamen oder anderes stehen!

```
for i in 1 2 3 4 5 6 7 8 9 10 ; do
   echo $i
done
```

Es gibt weitere Kommandos zur Schleifensteuerung. Man benutzt **continue** um die restlichen Befehle in einer Schleife zu überspringen und mit dem nächsten Schleifendurchlauf anzufangen. Wenn der Parameter n angegeben wird, werden n Schleifenebenen übersprungen. Mit **break** kann man die innerste Ebene (bzw. n Schleifenebenen, wenn der Parameter n angegeben wird) verlassen ohne den Rest der Schleife auszuführen.

3.6 Weitere Interpretersprachen

Vielfach finden in der Shellprogrammierung die Tools **awk** und **sed** Anwendung. **awk** ist eine eigene "Sprache", **sed** steht für Stream Editor. Die Kombination aus verschiedenen Programmen und die Verknüpfung des Ausgabekanals des einen Kommandos mit dem Eingabekanal des nächsten machen die Shell sehr nützlich für die verschiedensten Administrationsaufgaben. Möchte man z.B. wissen, wie die Netzmaske und der Netzwerkname lauten, die auf der gegebenen Maschine eingestellt sind, gelingt dieses durch die Kombination von **grep** und **awk**:

```
NETNAME=‘route -n | grep eth0 | grep -v ‘‘UG" | awk ’{ print $1 }’‘
NETMASK=‘route -n | grep eth0 | grep -v ‘‘UG" | awk ’{ print $3 }’‘
```

Mit **sed** lassen sich Ersetzungen in Dateien vornehmen. Im Beispiel sind die zu ersetzenden Zeichenketten die jeweiligen *shadow*-Einträge von den Benutzern "root" und "dxs-admin". Die zu ersetzenden Strings stehen in den Shellvariablen $PWA und $PWR:

```
# setting passwords in shadow file
sed -e "s,root::,root:$PWR:," -e "s,dxs-admin::,dxs-admin:$PWA:," \
        $NFSROOT/dxs/etc.s/shadow >$NFSROOT/dxs/etc.s/shadow.new
```

Weiterhin interessant sind die interpretierten Sprachen Perl und Python. Diese setzt man besonders für komplexere Aufgaben, gerade auch im Bereich der Stringverarbeitung ein. Jedoch gehören sie nicht zum Standardumfang eines Minimalsystems, so dass man sich nicht in jedem Fall darauf verlassen kann, dass sie und alle notwendigen Erweiterungen auf einem System zur Verfügung stehen. Ein Kommandointerpreter wird hingegen immer benötigt und ist deshalb Teil einer jeden Minimalinstallation.

3.7 Aufgaben

3.7.1 Kommandozeile und Umleitung

1. Wie fasst man die beiden Standardausgabekanäle zusammen? Wie entsorgt man Ausgaben (auf irgendeinem Kanel) am besten rückstandsfrei?

2. Rufen Sie ein Kommando auf und zeigen Sie dessen Rückgabewert anschliessend an, jedoch keine seiner direkten Ausgaben (weder Fehler noch die Standardausgabe)!

3. Wie sorgt man dafür, dass der String (*dieses ist ein test*) als ein einziges Argument interpretiert wird?

4. Nennen Sie fünf wichtige Umgebungsvariablen!
   Welche verschiedenen Möglichkeiten bestehen, um an bereits eingegebene Kommanodos wieder heranzukommen?

5. Was passiert, wenn ich das Kommando `export PATH=/tmp` eingebe?

3.7.2 Skripting

1. Schreiben Sie ein Miniskript (Textdatei mit Kommandoabfolgen von Shell- und externen Befehlen), welches die Zerlegung der Kommandoeingabe in einzelne Token demonstriert (Die Shell zerlegt jedes Kommando in einzelne Blöcke zur weiteren Bearbeitung)! Demonstrieren Sie hier die Möglichkeit des Returncodes und setzen Sie ihn auf "42"! Wie bekomme ich die Zahl der übereichten Argumente ermittelt?

2. Man gebe die Datei */etc/passwd* sortiert aus und schreibe die Ausgabe in die Datei *passwd.sorted* Wieviele Zeilen enthält diese Datei?

3. Man zerlege die Datei */etc/passwd* und sortiere nach der Gruppen ID. Schreiben Sie das Ergebnis in die Datei *passwd.sort.keyID*.

4. Schreiben Sie ein Skript (bzw. Ergänzung zu bereits bestehenden), welches jeden Benutzer beim Login mit "Guten Morgen/Mahlzeit/Abend/Nacht! Willkommen auf *Rechnername*, Benutzer *Benutzername*! Sie sind eingeloggt an Konsole Nummer *Nummer*, mit Ihnen gleichzeitig angemeldet sind: (*Liste der gerade angemeldeten Benutzer*) Es ist *Wochentag ..., Datum*. Dabei sind die kursivgesetzten Teile als Variable aufzufassen und entsprechend zu ersetzen. In welche Datei müsste dieses Skript eingefügt werden, damit es bei jedem Nutzer/Login zur Ausführung kommt?

5. Schreiben Sie ein Shell-Skript, welches die Funktion des Kommandos **uptime** annähernd simuliert!

6. Schreiben Sie ein Skript, welches die */etc/passwd* auswertet: Wie hoch ist die Anzahl der UserID's <100, =>100 und die Zahl der GruppenID=>100. Übersetzen Sie die numerische UserID in den entsprechenden String!

7. Schreiben Sie ein Skript, welches die *.pinerc* (so sie noch nicht vorhanden ist) aus einer Defaultdatei in eine personalisierte Konfigurationsdatei im Home-Verzeichnis des entsprenden Users umsetzt. Mindestens sollte der vollst. Name des Nutzers, die zu verwendende Domain (ihre-domain.de), sein SMTP-Server, einen NNTP-Server und ein Default-FCC im entsprechendem Home-Verzeichnis vorhanden sein. (Die Datei soll dann "gesendete_email" heissen).

8. Bauen Sie eine einfache Schleife, die bis "10" zählt und den String "Hello World" mit Zeilennummer ausgibt.

9. Wie müßte ein Skript aussehen, welches folgende Parameter auf einer Linuxmaschine (z.B. ein Laptop -> Mobilgerät) per Übergabe aus der Kommandozeile einstellt:

   ```
   ./setipconfig -hw 00:00:b4:34:23:fd -ip 12.13.14.15 -dns ''134.76.60.21\
   134.76.10.46'' -gw 12.13.14.254 -nm 255.252.0.0 -if eth0
   ```

   Wobei dabei folgendes sei: hw=Hardwareadresse (MAC), dns=Liste der DomainNameServer, if=Interface, gw=Gateway, nm=Netzmaske

Kapitel 4

Die Linux-Verzeichnisstruktur und Filesysteme

4.1 Aufbau

Die Linux-Verzeichnisstruktur verteilt sich über eine einzige hierarchische Baumstruktur. Die Nahtstellen in dieser Hierarchie (Festplattenpartionen, Netz-Filesysteme, Übergang auf CD-Rom ...) lassen sich nur mit speziellen Tools (z.B. **mount**) sichtbar machen. Den "Baum" muss man sich umgedreht vorstellen: Die Wurzel (root, dargestellt durch /) befindet sich in der obersten Hierarchie, danach erfolgt die Verzweigung in weitere Ebenen. In diesem Baum kann man mit dem Kommando **cd** navigieren. Es ist immer möglich, zur Navigation und zum Aufruf von Programmen, den Pfad zu einem Verzeichnis oder für eine Datei absolut oder relativ anzugeben. Absolut bedeutet von root (/) aus gesehen und relativ ist immer in Bezug auf die aktuelle Position.

In dieser Baumstruktur kann man sich die Blätter als die Dateien und die Äste und Zweige als Verzeichnisse vorstellen. Es gibt keine Laufwerksbuchstaben, wie sie evtl. von anderen OS bekannt sind. So "wissen" Kernel (das eigentliche Betriebssystem) und die Programme immer, wo bestimmte Verzeichnisse (und damit die benötigten Dateien, wie Konfigurationsdateien, Bibliotheken, Programmodule ...) und damit bestimmte Resourcen zu finden sind.

Der Kernel und die Anwendungsprogramme sind auf diese Struktur angewiesen. Deshalb lassen sich Teilbereiche der Verzeichnisstruktur nicht beliebig ein- oder aushängen. Dies geschieht durch den Vorgang des Mountens bzw. Unmountens. Durch diesen Anmelde- und Abmeldevorgang kann das Kernel-Caching für Dateien und Dateiteile recht effektiv wirken.

4.2 Einhängen / Aushängen von Filesystemen (Mounten/Unmount)

Das Kommando zum Aufbau der Verzeichnisstruktur bzw. dem Einhängen von Filesystemen heisst **mount**; das Aushängen geschieht mit **umount**. Wegen der oben angesprochenen Problematik darf normalerweise nur der Superuser (User "root", UID

0, GID 0) dieses Kommando ausführen. Zusätzliche Filesysteme von weiteren Festplatten, CD-Rom, Floppy, ZIP ... werden an Verzeichnisse des Dateibaumes angehangen. Standen bereits Dateien in diesen Verzeichnissen, so werden diese für den Zeitraum des Mountens "verdeckt". Hier ist mit Vorsicht vorzugehen, damit keine systemwichtigen Daten (Devices, Programme, Konfigurationsdateien ...) auf diese Weise unsichtbar werden.

Beim Aushängen von Filesystemen ist zu beachten, dass diese nicht mehr in Benutzung sein dürfen. Schon wenn man sich in einem Verzeichnis des gemounteten Gerätes oder Netzwerkfilesystems befindet, ist dieses in Benutzung. CD-Laufwerke oder ZIP-Drives lassen sich im gemounteten Falle nicht aus dem Laufwerk entfernen, um einen undefinierten Zustand der Daten/Dateien zu vermeiden. Aus diesem Grund sollte man auch Disketten, Firewire- oder USB-Datenträger (USB-Stick, Firewire-Festplatte, Flash-Karten etc.) ausmounten oder auswerfen (mit **eject**), bevor man sie aus dem Laufwerk entfernt.

Beispiel zum Ein- und Ausmounten des CD-Laufwerkes (hier als Secondary Master am ATAPI-Bus): `mount -t iso9660 -o ro /dev/hdc /mnt/cdrom` Die Optionen -t und -o können gegebenenfalls weggelassen werden. Wenn ein entsprechender Eintrag in der Datei */etc/fstab* vorhanden ist, reicht ein einfaches `mount /mnt/cdrom` Zum Ausmounten: `umount /dev/hdc` oder `umount /mnt/cdrom`. Man kann das Ausmounten und Auswerfen eines Datenträgers durch das Kommando **eject** kombinieren: `eject dvd` wirft den Datenträger mit dem String "dvd" im Device-Namen soweit vorhanden aus. Eine Voraussetzung besteht auch hier - Das Gerät darf sich nicht mehr in irgendeiner Form in Benutzung befinden.

Linux ist für die Ablage von Dateien nicht auf ein bestimmtes Filesystem festgelegt, dieses muss aber bestimmte Eigenschaften mitbringen, welche sich z.B. aus dem Zugriffsrechtesystem von Linux/Unix ergeben.

Seit den Kernelversionen 2.4.X besteht die Möglichkeit mittels **mount** eine Art Link im Dateisystem zu etablieren: So kann man z.B. die beiden Verzeichnisse */tmp* und */var/tmp* auf eines zusammenfassen, ohne symbolische Links zu verwenden. Mit `mount -bind /tmp /var/tmp` macht man */tmp* zusätzlich auf */var/tmp* verfügbar. Dieses Verfahren arbeitet lokal; bei einem Transport über NFS[1] funktioniert diese Verknüpfung nicht.

4.3 Die Datei */etc/fstab*

In der */etc/fstab* sind alle wichtigen benötigten Bereiche des Filesystems aufgeführt, die während des Betriebes zu Verfügung stehen (sollen). Sie beschreibt quasi die Bastelanleitung der Linux-Verzeichnisstruktur.

```
# /etc/fstab
#  Device    Mountpoint FS-Type Options                    Dump Fsckorder
/dev/hda1    /          ext2    defaults                      1 1
/dev/hda3    /tmp       ext2    defaults                      1 2
/dev/hda2    swap       swap    defaults                      0 0
```

[1] mit dem Kernel-NFSD, da dieser sonst durch gleiche Inode-Nummern verwirrt werden könnte

```
/dev/fd0    /floppy    auto    noauto                   0 0
/dev/cdrom  /cdrom     iso9660 noauto,ro,noexec         0 0
none        /proc      proc    defaults                 0 0
/dev/hda4   /dos/c     vfat    user,noexec,nosuid,nodev 0 2
testlin:/export /mnt   nfs     rw,addr=144.41.13.150    0 0
#
# noauto = Do not try to mount at boot time
#
```

Für jeden Eintrag, der einem Teilbaum der Gesamtstruktur entspricht, muss eine eigene Zeile mit vier bis sechs Einträgen angelegt werden. Die Einträge werden durch "white spaces" (Leerzeichen oder Tabulator) voneinander getrennt. Der erste Eintrag bezeichnet die zu der Partition gehörende Gerätedatei im */dev* Verzeichnis oder den Pfad eines im Netzwerk liegenden Bereiches (z.B. zur Einbindung per NFS). Der Name dieser Datei wird mit absolutem Pfadnamen angegeben. Im Beispiel wird die erste Partition der ersten IDE-Festplatte */dev/hda1* in der ersten nichtkommentierten Zeile angegeben. Für spezielle Dateisysteme, wie das Prozeß- oder USB-Dateisystem steht anstelle einer Gerätedatei oder eines Netzwerkpfades das Schlüsselwort "none".

Der zweite Eintrag bezeichnet das Verzeichnis, an welches das Teilverzeichnissystem angehängt werden soll. Durch die Reihenfolge der Zeilen in der Datei */etc/fstab* muss sichergestellt sein, dass das Verzeichnis, auf dem ein Teilsystem aufgesetzt wird, auch tatsächlich zu diesem Zeitpunkt bereits existiert.

Die Art des Dateisystems, das zum Einbinden in der *fstab* angegeben werden kann, wird in der dritten Spalte eingetragen. Wenn hier "auto" eingetragen ist, wird versucht das Filesystem vor dem Einhängen zu erkennen. Diese Angabe wird üblicherweise dem Mount-Kommando mit der Option "-t" (Type) mitgegeben.

Das Kommando **mount** macht exzessiven Gebrauch von dieser Datei. Wenn es z.B. am Anfang des Bootvorganges einer Maschine mit der Option "-a" aufgerufen wird, werden alle Einträge der *fstab* automatisch in das Dateisystem eingebunden, die mit der Option "auto" (vierte Spalte) gekennzeichnet sind. Hinter der Option "defaults" verbergen sich eine ganze Reihe von Standardeinstellungen, wozu z.B. "auto" und "nouser" zählen. Wenn ein Dateisystem von jedem Systembenutzer eingebunden werden kann, durch die Option "user", entspricht "default" für nicht privilegierte Benutzer der Kombination "nosuid", "noexec", "nodev" und "rw". Es dürfen mehrere Optionen in der vierten Spalte in einer durch Kommata getrennten Liste angegeben werden. Die Optionen, die mit einem "no" beginnen, können auch ohne die Vorsilbe eingesetzt werden, womit sich ihre Bedeutung erwartungsgemäß umkehrt.

Die Tabelle 4.3 auf Seite 65 bietet eine Übersicht über die Optionen der vierten Spalte der fstab.

4.4 Filesysteme

4.4.1 Überblick

Die Linux-Verzeichnisstruktur kann sich durchaus auf sehr verschiedene Filesysteme verteilen, wobei zumindest für einen Teil der Verzeichnisse diese bestimmte Eigenschaften aufweisen müssen. Als Rootfilesystem (notwendige Basishierarchie) kommen z.B. EXT3 (zur Zeit eines der Standard-FS, ReiserFS (ein weiteres modernes Journaling-FS), UMSDOS (Filesystem auf Basis einer DOS bzw. Windows95/98/ME-Installation), MINIX (Mini-FS), NFS (Networkfilesystem, für Diskless Clients) in Frage.

Linux versteht (teilweise im Nur-Lese-Modus) weitere Filesysteme: DOS, VFAT, ISO9660, NTFS, HFS, Rom-FS, UFS, ...

Alle Dateisysteme werden auf ein übergeordnetes virtuelles Filesystem abgebildet, welches eine Reihe von Standarddateisystemfunktionen verfügt. Dieses übersetzt die Anforderungen seitens der Programme über den Kernel in die entpsrechenden Funktionen des gerade zur Verfügung stehenden Filesystems. Dabei kann es durchaus vorkommen, dass bestimmte Funktionen nicht zur Verfügung stehen und dann in irgendeiner mehr oder weniger passenden Form emuliert werden.

Die Filesysteme, die vom laufenden Linux-Kernel einer Maschine angesprochen werden können (incl. der aktuell geladenen Module für die FS-Unterstützung) kann man durch das Auslesen der Datei */proc/filesystems* ermitteln. Möchte man bei der "auto"-Option des Mountkommandos (z.B. `mount -t auto /dev/fd0 /floppy`, wenn man nicht genau weiss, ob die Diskette DOS-, Minix-, HFS- ... formatiert ist) weitere Dateisysteme unterstützen, deren Module evtl. noch nicht geladen sind, trägt man diese in der */etc/filesystems* ein. Die Module werden dann zum Test auf das entsprechende Format geladen und die Filesysteme in der Reihenfolge in dieser Datei durchprobiert.

4.4.2 EXT2 und EXT3-Filesysteme

Das Extended Secondary Filesystem hat jahrelang als Linux-Standard-Filesystem dominiert. Es ist schnell und effizient, hat aber den gravierenden Nachteil, dass wenn es nicht regulär abgeschlossen wird (z.B. durch **umount**), beim nächsten Mounten eine Konsistenzprüfung durchgeführt wird. Diese wird durch das Programm **fsck** angestossen. Es stellt sicher, dass unvollständige Einträge im Dateisystem repariert werden. Hierin liegt jedoch ein gravierender Nachteil dieses Konzepts: Bei den heute üblichen Festplattengrößen, kann dieser Vorgang durchaus Stunden in Anspruch nehmen, was für Produktionssysteme nicht akzeptabel ist. Während des Checks ist ausschliesslich ein lesender Zugriff auf das Dateisystem erlaubt.

Ein weiterer Nachteil war, dass Dateigrößen größer als 2 GByte nicht unterstützt waren. Als diese Nachteile immer stärker zu Tage traten, wurden Anstrengungen von eine Reihe von Seiten unternommen, diese auszugleichen. Fast zeitgleich wurden von verschiedenen Parteien Dateisysteme mit Journalfähigkeiten für Linux programmiert oder portiert. Diese Fähigkeit bedeutet, dass ähnlich zu transaktionsbasierten Datenbanken, vor jeder Änderung ein Protokolleintrag erfolgt und erst anschliessend diese ausgeführt wird. Dadurch wird sichergestellt, dass im Fall eines Crashes noch ausstehende Vorgänge anhand des Protokolls identifiziert und sauber abgeschlossen werden können.

4.4.3 ReiserFS, JFS und XFS

Alle drei Dateisysteme sind im Zuge der Suche nach einem EXT2-Ersatz für Linux entwickelt oder portiert worden. Das ReiserFS ist eine gesponsorte komplette Neuentwicklung, XFS und JFS Portierungen. Allen dreien gemeinsam ist die Journalfähigkeit.

4.4.4 Das Andrew Filesystem

Das Kürzel AFS steht für Andrew File System oder A distributed File System. AFS wurde ursprünglich von der Carnegie-Mellon University entwickelt. Kommerzialisiert wurde AFS später von Transarc. Diese Firma wurde 1994 von IBM gekauft. IBM ihrerseits stellte im Jahre 2000 AFS als Open Source zur Verfügung. Seit dieser Zeit wird AFS von der OpenAFS Community gepflegt. AFS wird nicht mehr offiziell von IBM weiterentwickelt. Das Andrew File System (AFS) ist ein Netzwerk-Dateisystem, welches Dateien netzwerk- und weltweit zur Verfügung stellen kann. Dieses Dateisystem arbeitet nach dem Client-Server-Prinzip: Daten werden auf dem Fileserver gespeichert, der Zugriff erfolgt von Seiten der Clients.

Die AFS-Unterstützung wird über ein Kernelmodul realisiert, welches alle benötigten Bibliotheksfunktionen zur Verfügung stellt. Zu diesem Modul werden Kernelprozesse gestartet, welche die Netzwerkinteraktion realisieren. Ausschnitt der Anzeige aller geladenen Module und AFS-Prozesse:

```
linux:~/test # lsmod
Module                  Size  Used by    Tainted: PF
[...]
nls_cp437               4316   1  (autoclean)
nls_iso8859-1           2812   1  (autoclean)
smbfs                  34384   1  (autoclean)
snd-pcm-oss            45888   1  (autoclean)
snd-mixer-oss          13560   0  (autoclean) [snd-pcm-oss]
videodev                5600   0  (autoclean)
libafs                442528   2
[...]
linux:~/test # ps aux | grep afs
root   2001  0.0  0.0   0   0 ?   SW   Aug11   0:08 [afs_rxlistener]
root   2003  0.0  0.0   0   0 ?   SW   Aug11   0:00 [afs_callback]
root   2005  0.0  0.0   0   0 ?   SW   Aug11   0:00 [afs_rxevent]
root   2007  0.0  0.0   0   0 ?   SW   Aug11   0:00 [afsd]
root   2009  0.0  0.0   0   0 ?   SW   Aug11   0:00 [afs_checkserver]
root   2011  0.0  0.0   0   0 ?   SW   Aug11   0:01 [afs_background]
root   2013  0.0  0.0   0   0 ?   SW   Aug11   0:01 [afs_background]
root   2015  0.0  0.0   0   0 ?   SW   Aug11   0:00 [afs_cachetrim]
```

AFS besitzt gegenüber anderen verteilten Dateisystem wie z.B. Samba oder NFS einige Vorteile. Es arbeitet plattformunabhänig und verwendet zur Beschleunigung des Zugriffs auf jedem Client ein Cache von bis zu 2 GByte. Dieser dient zur Entlastung des Netzwerkes, zur Beschleunigung von Zugriffen und zur Überbrückung von Schwankungen, da AFS auf die Verwendung in Wide Area Networks angelegt ist. Für eine grössere Ausfallsicherheit können Volumes über mehrere Fileserver repliziert werden.

Der Integrationsaufwand in das jeweilige Betriebssystem fällt jedoch höher aus, als für die meisten bereits genannten Filesysteme. Es existieren Implementationen für Windows[2], MacOS X und diverse UNIXe inklusive Linux. AFS kennt einen weltweit eindeutigen Namensraum. Egal wo und an welcher Maschine ein Benutzer arbeitet, der Pfad für einen Benutzer stellt sich immer identisch dar. Der Pfadname beginnt mit */afs* (dem sogenannten Top Level). Auf der nächsten Hierarchieebene (Second Level) kommt der sogenannte Cell-Name. Die Zellennamen für bereits bestehende AFS-Zellen finden sich auf: *www.central.org/dl/cellservdb/CellServDB*.

Für die darin enthaltenen Unterverzeichnisse darunter existieren ebenfalls Konventionen, die eingehalten werden sollten. vgl. hierzu Tabelle 4.4.4 auf S. 66

Jedem Benutzer kann ein eigenes Volume zugeordnet werden. Sie bilden die Basis des ganzen Konzeptes, ihre Lage ist für den Client völlig transparent. Die Volumes können während des Betriebes von Fileserver zu Filserver verschoben werden. Auf diesem Volume wird ein Quota definiert. AFS implementiert eine ausgefeiltere Kontrolle über die Zugriffsberechtigungen mit Access Control Lists (ACLs), die anstelle der Unix-Rechte-Bits auf Verzeichnisebene zum Einsatz kommen.

Jeder Benutzer kann eigene Gruppen erzeugen und verwalten. Wenn ein normaler Benutzer ohne Systemrechte eine Gruppe erzeugen will, muss er als Präfix seinen eigenen Namen verwenden. Dieses kann so aussehen: *user:group* Ausserdem sind die folgenden drei System-Gruppen schon definiert: *system:anyuser*, *system:authuser* und *system:administrators*, diese Gruppen sind in allen AFS Systemen vorhanden. *system:anyuser* sind alle Benutzer (mit oder ohne gültigem Token), *system:authuser* alle Benutzer mit gültigem Token und *system:administrators* Benutzer mit Administrator-Privilegien.

Eine weitere Eigenschaft von AFS ist, dass es einen eigenen Login für die Freigabe seines Inhaltes benötigt. NFS in den Versionen 2 und 3 arbeitet mit einer IP-basierten Freigabe, die wenig Einschränkungen gegen geplanten Missbrauch bietet. Durch die Verwendung von Kerberos 4 bzw. 5 Authentifizierung kann es deshalb bedenkenloser als andere Netzwerkdateisysteme zur weltweiten Verteilung eingesetzt werden.

Mit der Authentifizierung gegenüber AFS, beim Login über PAM oder durch das Kommando klog <afs-username> erhält der angemeldete Benutzer ein Token. Dieses Token gestattet den Zugrif auf das AFS-Filesystem. Das Token hat aber nur eine begrenzte Gültigkeitsdauer. Nach Ablauf der Gültigkeit ist der Zugrif auf die AFS-Dateien wieder gesperrt. Mit dem Befehl **tokens** kann man anschauen, welche Token gerade gehalten werden und wie lange die einzelnen Token noch gültig sind.

Die Syntax der AFS-Kommandos ist etwas ungewohnt, da im Gegensatz zu den Standard-UNIX Kommandos die AFS-Kommandos nicht nur einem Zweck dienen, sondern mehrere Kommandos zu sogenannten Command Suites zusammenfassen. Die Syntax ist bei allen AFS Kommandos identisch: command_suite operation_code -switch <value> -flag

Zur Anzeige der Rechte auf ein AFS-Verzeichnis (nur auf diese können AFS-Rechte gesetzt werden, auf einzelne Dateien nicht) verwendet man folgendes Kommando:

[2] für die "Consumerwindows" - Spielkonsolen Win95/98/ME - jedoch nur der Client

```
dirk@randy2:/afs/.uni-freiburg.de/www/ks/htdocs/systeme> fs la
Access list for . is
Normal rights:
  webadmins rlidwka
  system:administrators rlidwka
  www rl
  www.ks rl
  dsuchod rlidwka
```

Die Ausgabe zeigt, dass die Benutzer-IDs der eingeloggten Person an der Maschine nicht zwingend mit der AFS-Benutzer-ID übereinstimmen müssen. An der Linux-Maschine ist *dirk* eingeloggt, mit dem Kommando `klog dsuchod` und anschliessender Passwort-Eingabe hat sich der angemeldete Benutzer ein AFS-Token für den Zugriff auf die *dsuchod* freigeschalteten AFS-Bereiche geholt.

Durch Eingabe des Kommandos `fs quota` oder kürzer `fs q` im AFS-Baum bekommt man die Meldung, wieviel Prozent des eigenen Quotas aufgebraucht sind:

```
dirk@randy2:/afs/.uni-freiburg.de/www/ks/htdocs/> fs q
74% of quota used.
```

Eine ausführlichere Information mit Angabe des zur Verfügung stehenden und verbrauchten Speicherplatzes bekommt man mit:

```
dirk@randy2:/afs/.uni-freiburg.de/www/ks/htdocs/> fs lq
Volume Name                      Quota       Used %Used   Partition
www.ks                          400000     295334   74%         73%
```

Die Zugriffsberechtigungen von AFS beziehen sich immer auf ein ganzes Verzeichnis und alle darin enthaltenen Dateien. Sie beziehen sich nicht automatisch auf enthaltene Unterverzeichnisse. Wenn die Zugriffsberechtigung für eine einzelne Datei geändert werden soll, muss diese ein anderes Verzeichnis mit den gewünschten Zugriffsrechten verschoben werden. Die Zugriffsberechtigung wird dabei vom übergordneten Verzeichnis geerbt. AFS definiert insgesammt 7 Rechtearten, dabei beziehen sich vier Zugriffsrechte auf das Verzeichnis. Drei weitere Berechtigungen beziehen sich auf die Dateien innerhalb des Verzeichnisses

Beim Setzen des Rechtelevels *lookup* kann in das Verzeichnis gewechselt werden. Der Inhalt der Dateien kann nur dann auch gelesen werden, wenn ebenfalls die Berechtigung *read* zur Verfügung steht. Die wichtigsten Berechtigungen lassen sich zusammenfassen, wie in der Tabelle zur AFS Rechtegruppierung gezeigt.

Ein nettes Feature ist, dass Rechteinhaber im AFS selbst Verzeichnisse für andere Benutzer bzw. Gruppen freigeben können. Hierzu dient zuerst einmal das Kommando **pts**.

```
pts creategroup dsuchod:systeme
  group dsuchod:systeme2 has id -754
pts adduser user01 dsuchod:systeme
fs sa testdir dsuchod:systeme rliwka
```

Das Ergebnis des Gruppenanlegens ist eingerückt dargestellt. Anschliessend kann ein User (hier: *user01*) oder mehrere dieser Gruppe hinzugefügt werden. Mit dem AFS-Kommando `fs sa testdir dsuchod:systeme rliwka` wird dann ein Rechtemuster auf das Verzeichnis *testdir* für die Gruppe *dsuchod:systeme* eingetragen. Dieses Kommando muss entsprechend für alle gewünschten Verzeichnisse wiederholt werden. Dabei sollte man bedenken, dass auch übergeordnete Verzeichnisse zumindest ein Lookup-Recht für die Gruppe bekommen sollten. Das Anzeigen der Rechte im Verzeichnis *testdir* liefert dann folgendes:

```
dirk@linux02:/afs/.uni-freiburg.de/www/ks/htdocs/systeme> fs la
Access list for . is
Normal rights:
  dsuchod:systeme rliwka
  webadmins rlidwka
  system:administrators rlidwka
  www rl
  www.ks rl
  dsuchod rlidwka
```

4.4.5 Dateisystemüberprüfung

Bei jedem **mount** wird geprüft, ob das zu mountende Verzeichnis in Ordnung ist. Wird ein Verzeichnis nicht ordnungsgemäß (durch **umount**) ausgehängt, so wird beim ersten nächsten mounten automatisch ein Filesystem-Check durchgeführt. Außerdem findet ein solcher umfangreicher Check in gewissen Abständen (zum Beispiel aller 30 mounts) statt. Dieses läßt sich mit dem Kommando **tunefs** konfigurieren. Damit wird sichergestellt, dass die Informationen auf dem Dateisystem immer konsistent sind. Der Systemadministrator kann einen Filesystemcheck mit Hilfe des Programms **fsck** durchführen.

Um Inkonsistenzen im Dateisystem zu vermeiden, ist es erforderlich, den Rechner vor jedem Abschalten herunterzufahren. Dies erledigt der Systemverwalter mit dem Programm **shutdown**.

4.5 Dateiablagestandards

Es gibt nicht "das" klassische Konzept für eine Filehierarchie der freien Unixe, früher orientierte sich die Struktur am System V Unix Release 4. Seit einiger Zeit gibt es einen Filesystemhierarchiestandard (FSHS), der Mitte diesen Jahres in einer aktualisierten Version veröffentlicht wurde. Die meisten neueren Linuxdistributionen halten sich an diesen.

Zwei Hauptkriterien der Verteilung der Dateien spielen im FSHS eine Rolle: verteilbare vs. lokale und variable vs. statische Daten. Die Idee dieser Einordnung entstammt dem Wunsch, Teile des Filesystems zentral auf einem Server im Netzwerk zur Verfügung stellen zu können.

4.6 Dateiarten

Linux kennt (das jeweilige Filesystem mag einige Typen davon nicht unterstützen) sechs Arten von Dateien. s. Tabelle 4.7, S. 67

4.7 Zugriffsrechte

Das Sicherheitskonzept von Unix basiert stark auf den Zugriffsrechten für Dateien. Deshalb hat in der Unix/Linux-Welt jedes Verzeichnis und jede Datei IndexZugriffsrechte für einen Besitzer ("user", Kürzel **u**, z.B. die User ftp, mysql, testuser ...), für die gesamte Gruppe ("group", Kürzel **g**, z.B. bin, named, users) zu der dieser Besitzer gehört und schließlich für alle anderen, den "Rest der Welt" ("other", Kürzel **o**). Diese sind im Ganzen drei Benutzergruppen, die beim Setzen der Rechte auch zusammen angegeben werden können ("all", Kürzel **a**). Es ist z.B. bei der Einrichtung einer Homepage im eigenen Verzeichnis darauf zu achten, dass den HTML-Dateien die Leserechte für "other", bzw. für "all" gesetzt wurden und die darüberliegenden Verzeichnissebenen das Suchrecht für "all" besitzen.

Weiterhin gibt es drei Arten des Dateizugriffs: Schreiben, Lesen, Ausführen (bzw. Suchen in Verzeichnissen). Das Kommando zum Setzen der Zugriffsrechte ist **chmod**. Dabei gibt es zwei Möglichkeiten es aufzurufen:

```
mayer@hermes:~/kursunterlagen >chmod o+r kurs01.txt
```

Setzt für alle anderen Benutzer (other) , die nicht der eigenen Gruppe (z.B. "users") angehören, das Leserecht (read) auf die Datei kurs01.txt. Sollen die Rechte entfernt werden, geschieht dieses durch die Angabe des Minuszeichens anstelle des Plus, welches für das Hinzufügen der Rechte steht.

Eine andere Möglichkeit besteht darin die Zugriffsrechte über die sie repräsentierenden Oktalwerte zu setzen, wie sie in der untenstehenden Tabelle gezeigt werden.

Der Aufruf des Kommandos **chmod** sieht dann z.B. so aus:

```
meier@hermes:~/kursunterlagen >chmod 0700 kurs01.txt
```

Darüberhinausgehende Rechtesysteme lassen sich mit den Posix-ACL's (Access Control Lists) implementieren. Dafür muss jedoch eine Unterstützung seitens des Dateisystems (z.B. xfs) vorhanden sein.

4.8 Systembefehle zur Arbeit mit Dateien

Folgende Basisbefehle stehen unter Linux zur Verfügung. Die spitzen Klammern sollen bloß anzeigen, dass dieser Bestandteil des Befehls - von den Leerzeichen abgesehen - variiert, die tippt man in Wirklichkeit nicht mit. Diese Befehle sind immer nach dem Schema: Befehl, Leerzeichen, Name, ev. Leerzeichen, ev. Name aufgebaut. (Nach der Eingabe des Befehls diesen immer mit "Enter" abschicken!)

mkdir <Verzeichnisname> erstellt ein neues Verzeichnis (im Windows-Jargon: einen neuen Ordner), mit dem von euch gewählten Namen <Verzeichnisname>. "mkdir" steht für "make directory" Bsp.: `mkdir test`. Das neue Verzeichnis wird immer als Unterverzeichnis des aktuellen Verzeichnisses (= in dem man sich gerade befindet) eingerichtet.

cd führt zum Verzeichniswechsel, dahinter kann man den Namen desjenigen Verzeichnisses angeben, in das man wechseln möchtet (z.B. `cd test`). **cd** steht für "change directory" Wenn man keinen Verzeichnisnamen angibt, bringt einen dieser Befehl immer in das Home-Verzeichnis des angemeldeten Users. Denkt daran, dass man sich so immer nur von einer Ebene in die nächste vorhangeln kann (relative Pfadangabe). Um in das nächsthöhere Verzeichnis zurückzuwechseln, könnt man **cd ..** benutzen (beachtet das Leerzeichen vor den Punkten). Wenn man Verzeichnisebenen überspringen möchtet, muss man den ganzen Pfad eingeben (z.B. */test/texte* - ohne Leerzeichen). Ein solcher Pfad bezeichnet den genauen Standort im Verzeichnisbaum. Das Home-Verzeichnis kürzt man mit / ab. Das aktuelle Verzeichnis gibt man mit ./ an.

pwd Diese Eingabe liefert den vollständigen Pfad zu dem Verzeichnis, in dem man sich gerade befindet. pwd steht für "print working directory"

ls zeigt in Inhalt von Verzeichnissen an. **ls** steht für "list". Man könnte auch **ls <Verzeichnisname>** eingeben, ohne vorher cd <Verzeichnisname> zu sagen, aber immer nur von einer Ebene in die nächste, oder mit dem vollständigen Pfadnamen. Wenn man **ls -l** eingibt, erfahrt man sogar noch mehr über die Dateien als den Namen (z.B. die Größe etc.)

rmdir <Verzeichnisname> löscht leere Verzeichnisse. **rmdir** steht für "remove directory". Wieder kann man nur Unterverzeichnisse des aktuellen Verzeichnisses oder den ganzen Pfad angeben. Wenn sich in dem zu löschenden Verzeichnis noch etwas befindet, löscht der Computer sie nicht. Dann muss man erst mit

rm <Dateiname> die Dateien darin löschen. Mit **rm *** kann man sie alle auf einmal beseitigen. Bestimmte Dateitypen löscht man mit **rm *.<extension>**, also z.B. **rm *.jpg**, um alle Bilddateien des Typs "jpg" zu löschen.

mv <Dateiname> <Verzeichnisname> verschiebt eine Datei von einem Verzeichnis in ein anderes - natürlich nur, wenn die Datei in dem aktuellen Verzeichnis steht, oder man den gesamten Pfad angibt. **mv** funktioniert nicht über die Grenzen von Filesystemen hinweg, da nur Änderungen in den Inode-Einträgen erfolgen und die Datei nicht physikalisch "bewegt" wird. mv steht für "move". Wenn man den Namen einer Datei ändern möchtet, gibt man **mv <alterDateiname> <neuerDateiname>** ein.

cp <Dateiname> <Verzeichnisname> kopiert tatsächlich eine Datei. Diese Datei existiert dann zweimal (im Gegensatz zu mv). Natürlich kann man auch einfach **cp <alterDateiname> <neuerDateiname>** ausführen, wenn man eine Datei zweimal in einem Verzeichnis haben möchte.

4.9 Aufbau einiger wichtiger Verzeichnisse

Dateien werden unter Linux geordnet nach ihrer Funktion abgelegt. So ergeben sich viele Vereinfachungen: Konfigurationsdateien liegen in einem gemeinsamen Verzeichnis, Programmdateien verteilen sich je nach Funktion über wenige Verzeichnisse, was den Suchpfad vereinfacht und die Angabe des absoluten Pfades vermeidet. Bibliotheken und Module liegen wieder in speziellen Verzeichnissen ...

In einigen Hierachiestufen wiederholen sich bestimmte Verzeichnisse: “lib”, “bin” findet man z.B. in */*, */usr*, */usr/local*, */usr/X11R6*, */opt/kde[23]* ...

4.9.1 Konfigurationsdateien

Konfigurationsdateien liegen üblicherweise im Systemkonfigurationsverzeichnis */etc*. Für bestimmte Programme (noKDE, Gnome) findet man diese jedoch in den entsprechenden Unterverzeichnissen. Im folgenden werden jedoch nur einige der Dateien in */etc* näher erläutert.

4.9.1.1 Allgemein

Die Dateien */etc/passwd* und */etc/shadow* verwalten die User des lokalen Systems, speichern deren Passwörter enthalten evtl. zusätzliche Informationen und legen das Home-Verzeichnis sowie die Login-Shell fest. Die Datei hat folgendes Aussehen:

```
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/bin/bash
daemon:x:2:2:Daemon:/sbin:/bin/bash
lp:x:4:7:Printing daemon:/var/spool/lpd:/bin/bash
mail:x:8:12:Mailer daemon:/var/spool/clientmqueue:/bin/false
games:x:12:100:Games account:/var/games:/bin/bash
wwwrun:x:30:65534:WWW daemon apache:/var/lib/wwwrun:/bin/bash
named:x:44:44:Nameserver daemon:/var/named:/bin/bash
nobody:x:65534:65533:nobody:/var/lib/nobody:/bin/bash
ftp:x:40:49:FTP account:/srv/ftp:/bin/bash
man:x:13:62:Manual pages viewer:/var/cache/man:/bin/bash
news:x:9:13:News system:/etc/news:/bin/bash
uucp:x:10:14:Unix-to-Unix CoPy system:/etc/uucp:/bin/bash
sshd:x:71:65:SSH daemon:/var/lib/sshd:/bin/false
postfix:x:51:51:Postfix Daemon:/var/spool/postfix:/bin/false
at:x:25:25:Batch jobs daemon:/var/spool/atjobs:/bin/bash
ntp:x:74:65534:NTP daemon:/var/lib/ntp:/bin/false
dirk:x:500:100:Dirk von Suchodoletz:/home/dirk:/bin/bash
```

```
test:x:501:100:Test User:/home/randy:/bin/bash
gdm:x:100:100::/home/gdm:/bin/bash
```

Nacheinander werden folgende Informationen durch Doppelpunkte getrennt aufgelistet: Der Username (als String), der Platzhalter für das Passwort ("x"), die numerische User-ID, die numerische Gruppen-ID, eine Beschreibung des Benutzers (die kommasepariert weitere Informationen enthalten kann), das Arbeitsverezeichnis und die Standard-Shell. Diese Datei ist die zentrale Benutzerdatenbank, es sei denn es werden zentrale Authentifizierungs- und Konfigurationsdienste (wie z.B. LDAP) verwendet. Über die Reihenfolge der Einbindung zur Authentifizierung entscheidet die */etc/nsswitch.conf*.

Die */etc/shadow* ist die zur */etc/passwd* korrespondierende Datei. Sie verfügt über stark beschnittene Zugriffsrechte, damit nur der Systemadministrator in diese Datei hineinsehen kann. Zwar sind die Passwörter nicht als Klartextvariablen abgelegt, dennoch besteht die Möglichkeit, das gecryptete Passwort mittels Brute-Force-Attacke auf einem anderen Rechner zu knacken. Die */etc/shadow* sieht folgendermaßen aus:

```
root:WM/BNS9W8FgbU:11962:0:10000::::
bin:*:8902:0:10000::::
daemon:*:8902:0:10000::::
lp:*:8902:0:10000::::
mail:*:8902:0:10000::::
news:*:8902:0:10000::::
uucp:*:8902:0:10000::::
games:*:8902:0:10000::::
man:*:8902:0:10000::::
wwwrun:*:8902:0:10000::::
ftp:*:8902:0:10000::::
named:*:8902:0:10000::::
nobody:4qCm64P9en1is:11984:0:10000::::
sshd:!:11962:0:99999:7:::
postfix:!:11962:0:99999:7:::
at:!:11962:0:99999:7:::
ntp:!:11962:0:99999:7:::
dirk:MDS5u2KR2OpAw:11964:0:99999:7:::
test:5rwQu8thQ2d1w:11962:0:99999:7:::
gdm:!:11988:0:99999:7:::
```

/etc/login.defs ist eng mit dem **login**-Prozess verknüpft. Dieses liest daraus die Parameter mit denen beispielsweise eingestellt wird, wie oft ein fehlgeschlagenes Login wiederholt werden darf und ob und wo die Meldungen über solche Fehlschläge festgehalten werden. Hier können auch einige Umgebungsvariablen für alle Prozesse festgelegt werden, die in der vom **login** erzeugten Prozessfamilie gestartet werden.

/etc/issue(.net) wird beim Konsolen-Login und bei "telnet"-Verbindungen vor dem Login-Prompt angezeigt.

/etc/rc.config ist die zentrale Konfigurationsdatei der Suse-Linuxdistribution.

4.9.1.2 Shell

/etc/skel ist ein Skeleton-Verzeichnis, wo die Konfigurationsdateien für Userprogramme liegen, welche beim Anlegen des Accounts in das Verzeichnis des entsprechenden Benutzers eingespielt werden und so das Tool zur Benutzerverwaltung unterstützt wird.

/etc/securetty gibt die Ports (Terminals) an, von denen aus sich der Systemadministrator einloggen darf. Diese Datei wird vom **login**-Kommando gelesen und ausgewertet. Die Verwendung dieser Datei wird in der *login.defs* spezifiziert.

/etc/shells listet alle verfügbaren (zugelassenen und uneingeschränkten) Loginshells auf. Diese Datei wird vom **chsh**-Kommando ausgewertet. Dem Anwender wird damit die Möglichkeit gegeben, die in dieser Datei zeilenweise aufgelisteten Programme als Loginshell in der Datei **/etc/passwd** einzutragen, wenn es sich um einen lokalen Account handelt.

/etc/profile setzt wichtige Umgebungsvariablen für die Shell (Suchpfad, Manpath ...). Diese Datei wird immer beim Login aufgerufen.

/etc/motd Die "Message of the day" wird nach dem Login angezeigt. Es gibt einige Escapesequenzen der Shell, welche z.B. den Hostnamen, Uhrzeit, verwendete Konsole etc. anzeigen können.

4.9.1.3 Netzwerk

/etc/hosts enthält eine Liste von IP's und dazugehörigem Rechnernamen. Sie wird verwendet, wenn z.B. kein DNS zur Verfügung steht. Dies ist wärend des Hochfahrens der Maschine üblicherweise der Fall.

/etc/resolv.conf In dieser Datei erfolgt die Konfiguration des Clients für die Namensauflösung (DNS).

/etc/nscd.conf Hier wird die Konfiguration des Name Service Caching Dienstes hinterlegt.

/etc/nsswitch.conf Der Nameservice-Switch regelt, wie bestimmte Informationen, wie Benutzernamen, UserID, Passwörter ermittelt werden.

```
[...]
passwd:     compat ldap
group:      compat
hosts:      files dns
networks:   files dns
[...]
```

Die gezeigten Einstellungen im Ausschnitt aus der Datei legen fest, dass für die Passwortüberprüfung und UserID zuerst in den klassischen Dateien */etc/passwd* und */etc/shadow* nachgesehen werden soll, bevor ein LDAP-Server um diese Daten befragt wird. Für die Auflösung von Rechnernamen wird zuerst die Datei */etc/hosts* bemüht, bevor eine Anfrage an den DNS-Server gestellt wird.

4.10 Binärdateien (ausführbare Dateien)

(/usr)/bin **und** *(/usr)/sbin* sind die Verzeichnisse für die Basisprogrammdateien - wobei in "bin" die Programme, die von Usern ausgeführt werden können und in "sbin" die Programme die vom System oder dem Systemadministrator verwendet werden, liegen. In "usr/(s)bin" liegen alle ausführbaren Dateien, die nicht zur absoluten Basis- bzw. Bootausstattung gehören. Weitere "bin"-Verzeichnisse gibt es unter */usr/X11R6*, */usr/local*, */opt/kde(2/3)*, */opt/gnome(2)* je nach Softwareausstattung, Distribution und Konfiguration der Maschine. Die Shellumgebungsvariable PATH erleichtert das Ausführen von Programmen, da nicht der absolute Pfad angegeben werden muss. Ausführbare Programme müssen entweder für die entsprechende Hardware programmiert und übersetzt (kompiliert) worden sein oder als Skripten installierter Interpretersprachen vorliegen.

Die Standardausstattung im Verzeichnis */bin* sieht ungefähr so aus:

```
dirk@randy2:/bin> ls
arch         df             gzip          netstat        stty
ash          dmesg          hostname      nisdomainname  su sync
ash.static   dnsdomainname  initviocons   openvt         synctar
awk          domainname     ipg           pidof          tar
basename     echo           kill          ping           tcsh
bash         ed             ln            ping6          touch
cat          egrep          loadkeys      ps             true
chgrp        eject          logger        pwd            umount
chmod        false          login         rm             uname
chown        fgrep          ls            rmdir          usleep
chvt         fillup         mail          rpm            vi
cp           fuser          mkdir         sash           vim
cpio         gawk           mknod         sed            ypdomainname
csh          getkey         mktemp        setserial      zcat
date         grep           more          sh             zsh
dd           guessfstype    mount         sleep
deallocvt    gunzip         mv            sort
```

Diese Kommandos genügen zur einfachen Systemadministration. Programme, wie die Shell, diverse Filter und Parser, wie **sed**, **awk**, **grep**, Packprogramme **tar** und **gzip** sind hier zu finden. Im */usr/bin*-Verzeichnis sieht es je nach Zweck der Maschine um einiges voller aus, so dass sie hier nicht dargestellt werden soll.

4.11 Bibliotheken

Programmbibliotheken (shared libraries) und Programmteile, Skriptsprachenelemente, der Compiler etc. liegen üblicherweise in den Verzeichnissen:
(/usr)/lib, /usr/X11R6/lib, ... Der "Suchpfad" für Bibliotheken wird in einem Hashfile abgelegt */etc/ld.so.cache*. Die Erzeugung dieser Datei erfolgt über **ldconfig** wobei der Suchpfad für dieses Programm in der */etc/ld.so.conf* definiert wird. **ldconfig** wird automatisch beim Systemstart ausgeführt und sollte nach der Installation bzw. Update von dynamisch gelinkten Bibliotheken aufgerufen werden. Die Basisbibliotheken, die für die Standardprogramme aus */(s)bin* benötigt werden, finden sich unterhalb von */lib*

`YaST`	`libe2p.so.2`	`libpam_misc.so.0.77`
`cpp`	`libe2p.so.2.3`	`libpamc.so.0`
`devfsd`	`libext2fs.so.2`	`libpamc.so.0.77`
`ld-2.3.2.so`	`libext2fs.so.2.4`	`libpcprofile.so`
`ld-linux.so.2`	`libgcc_s.so.1`	`libpthread.so.0`
`libBrokenLocale.so.1`	`libhandle.so.1`	`libreadline.so.4`
`libNoVersion.so.1`	`libhandle.so.1.0.1`	`libreadline.so.4.3`
`libSegFault.so`	`libhistory.so.4`	`libresmgr.so`
`libacl.a`	`libhistory.so.4.3`	`libresmgr.so.0.1`
`libacl.la`	`liblvm-10.so`	`libresolv.so.2`
`libacl.so`	`liblvm-10.so.1`	`librt.so.1`
`libacl.so.1`	`liblvm-10.so.1.0`	`libscpm.so`
`libacl.so.1.1.0`	`libm.so.6`	`libscpm.so.0.9`
`libanl.so.1`	`libmemusage.so`	`libss.so.2`
`libattr.a`	`libncurses.so.5`	`libss.so.2.0`
`libattr.la`	`libncurses.so.5.3`	`libthread_db.so.1`
`libattr.so`	`libnsl.so.1`	`libutil.so.1`
`libattr.so.1`	`libnss_compat.so.2`	`libuuid.so.1`
`libattr.so.1.1.0`	`libnss_dns.so.2`	`libuuid.so.1.2`
`libc.so.6`	`libnss_files.so.2`	`libxcrypt.so.1`
`libcap.so`	`libnss_hesiod.so.2`	`libxcrypt.so.1.1.1`
`libcap.so.1`	`libnss_nis.so.2`	`libz.so.1`
`libcap.so.1.92`	`libnss_nisplus.so.2`	`libz.so.1.1.4`
`libcom_err.so.2`	`libnss_winbind.so`	`lsb`
`libcom_err.so.2.0`	`libnss_winbind.so.2`	`modules`
`libcrypt.so.1`	`libpam.so.0`	`scpm`
`libdb.so.2`	`libpam.so.0.77`	`security`
`libdl.so.2`	`libpam_misc.so.0`	

Unterhalb von */lib* finden sich (distributionsspezifisch) einige weitere wichtige Verzeichnisse, wie *modules* für die Kernelmodule, *security* für die PAM-Bibliotheken und Module.

4.12 Variable Daten

Unterhalb des Verzeichnisses */var* werden alle vom System wärend der Laufzeit generierten Daten, wie z.B. Logfiles, Caches, Locks ... abgelegt. Hier liegen auch bestimmte Laufzeitdaten, wie z.B. die Datenbankfiles von MySQL und LDAP oder die Tabellen des Nameservers.

Die SMTP-Programme (Mailagents) legen unterhalb von */var* üblicherweise ihre Queues und teilweise die Mail der Systembenutzer ab. Dies geschieht auch durch den Serverdienst für Usenet-News. Laufen viele und häufig benutzte Dienste, so wird in diesem Bereich Mail gespeichert ..., dann sollte für dieses Verzeichnis ausreichend Platz für dynamisches Wachstum eingeplant werden.

4.13 Das Temporärverzeichnis

Im Verzeichnis */tmp* legen einige Programmen temporäre Dateien zur Zwischenspeicherung von Laufzeitdaten oder zur Prozess-Interaktion an. Zum Erzeugen einer Datei in diesem Verzeichnis braucht ein Prozess keine besonderen Rechte. Das Löschen von Dateien ist nur den Prozessen des Dateieigentümers erlaubt, wenn das Stickybit für das */tmp* Verzeichnis gesetzt ist.

```
user@linux:~/skripten/grundlagen> ls -ld /tmp/
drwxrwxrwt   37 root     root         4096 2003-08-20 14:15 /tmp
```

Weil in */tmp* Dateien unvorhersehbarer Größe zu unberechenbaren Zeitpunkten von allen Benutzern eines Systems erzeugt werden können, legt man dieses Verzeichnis am besten in eine eigene Partition. Der Cron-Dienst sollte dazu eingesetzt werden, regelmäßig in bestimmten Abständen das Verzeichnis aufzuräumen und die Dateien zu entsorgen. Sonst kann es schnell passieren, dass normale Benutzer durch großzügiges Auslagern ihrer Überbestände den Temporärbereich überlaufen lassen.

4.14 Literatur

- S. Hetze, D. Hohndel, M. Müller, O. Kirch, *Anwenderhandbuch -LINUX-, Leitfaden zur Systemverwaltung*, LunetIX, 1997

- Fred Hantelmann, *LINUX für Durchstarter*, Springer, 1997

Beide Bücher geben eine gute Einführung in den Umgang mit Linux, das Login, die Shell (Kommandoeingabe), das Dateisystem und wichtige Systembefehle. Erklärungen zu Netzwerken erfolgen nicht oder nur sehr am Rande und sind in Anbetracht des Druckdatums nicht mehr aktuell.

4.15 Aufgaben

4.15.1 Filesystem - Aufteilung

1. 1) Wo liegen in der Linux-Verzeichnisstruktur üblicherweise die Bibliotheken und ausführbare Progamme? Nennen Sie wichtige Kommandos und ihre Lage im Dateisystem!

2. Warum verwendet man nicht das Konzept der Laufwerksbuchstaben, wie unter DOS/Windows bekannt?

3. Wo findet man Dateien, wie z.B. **kmail**, **mount** oder **ls**?

4. Wo befinden sich die Konfigurationsdateien zum Einrichten einer Modemverbindung? Wo liegen im Linux-Dateisystem üblicherweise die Konfigurationsdateien? Nennen Sie wichtige Konfigurationsdateien, die im Kurs vorgekommen sind!

5. Weshalb darf der Befehl `mv *.txt *.html` so nicht angewendet werden, was passiert da?

6. Woran erkennt man den Dateityp "Verzeichnis" bzw. "Link"?

7. Wie bestimmt man generell den Belegungsgrad der gemounteten Festplatten?

8. In welcher Datei sind die Benutzer und in welcher ihre Passwörter eingetragen? Wie verhält es sich bei einer zentralen Authentifizierung?

9. Wofür dienen die Kommandos **find** und **locate**? Worin bestehen die Unterschiede zwischen ihnen?

4.15.2 (Un-)Mounten

1. Geben Sie die Kommandos ein, um festzustellen, welches Device das CDROM ist (wichtig beim Mounten), wie gross die eingebaute Festplatte ist und wie diese partitioniert ist!

2. Mounte das Diskettenlaufwerk und anschliessend das CD-Rom nach */mnt*!

3. In welcher Konfigurationsdatei wird festgelegt welche Bereiche und Devices normale User mounten dürfen? Welche Dateisysteme werden beim Booten automatisch gemountet und welche nicht? Wie legt man das fest?

4. Welche Vereinfachungen gibt es zum vollständig ausgeschriebenen Mountkommando?

5. Mit welchem Dateisystem ist die Root Partition formatiert?

6. Welche Partitionen sind auf Ihrem Rechner wohin gemountet? Wieviel Speicherplatz ist auf Ihnen noch vorhanden.

7. Nenne Dateisysteme, welche als Root-Filesystem für eine Linuxmaschine in Frage kommen. Welche weiteren gibt es und warum können diese nicht für den genannten Zweck eingesetzt werden?

4.15.3 Speicherplatz auf der Festplatte

1. Bestimmen Sie die Menge an Speicherplatz die das Verzeichnis */usr/share* auf Ihrem Computer belegt! Wie erhalte ich eine Ausgabe in Mega- oder Gigabyte-Angabe?

2. Wie bestimme ich generell den Belegungsgrad meiner gemounteten Festplatten?

4.15.4 Rechtesystem

1. Wie lässt sich das Rechte-Muster -rw-r--r-- einer Datei erzeugen und was sagt es aus?

2. Mit welchem Kommando lässt sich der Dateibesitzer/die Gruppe ändern? Wie kann man die Rechte-Maske auch für Unterverzeichnisse geltend machen?

3. Wozu dient das SUID-Bit und wie kann es gesetzt werden?

Option	Bedeutung
async	erzwingt asynchrone IO-Operationen
auto	erlaubt das automatische Einbinden eines Eintrages
defaults	entspricht normalerweise der Kombination "suid", "rw"
gid=Wert	bei DOS/VFAT und HPFS Dateisystemen wird allen Dateien die angegebene Gruppen-ID zugeordnet
noauto	verhindert das automatische Einbinden der Partition
nodev	die zeichen- und blockorientierten Gerätedateien in dieser Partition werden nicht angesprochen
noexec	verbietet die Ausführung jedes (binären) Programms dieses Bereiches
norock	nur ISO9660, schaltet die Rock-Ridge Erweiterung ab
nosuid	unterdrückt die Wirkung der SUID und SGID Bits bei der Ausführung von Programmen
nouser	verbietet ausdrücklich die Benutzung von **mount** durch unprivilegierte Systembenutzer
remount	veranlaßt mount ein bereits aufgesetztes Dateisystem ab- und sofort wieder aufzusetzen. Auf diese Weise können die Parameter eines bereits aufgesetzten Dateisystems geändert werden
ro	read only, verbietet das Schreiben auf diese Partition
rw	read write, erlaubt das Lesen und das Schreiben (muss vom Filesystem unterstützt werden)
swap	kennzeichnet eine Swappartition
sync	die Metadaten (Superblock, Inode, Verzeichnisdaten) werden ungepuffert (synchron) auf das Speichermedium geschrieben
umask=Wert	läßt die Zugriffsrechte für Dateien und Verzeichnisse im DOS/VFAT oder HPFS Dateisystem durch die Maske Wert erscheinen. Der Wert wird als Oktalzahl eingegeben und interpretiert wie beim Shellkommando **umask** beschrieben
uid=Wert	mappen der UserID aller Dateien auf die angegebene UserID
user	kann das Einbinden von Dateisystemen durch normale Systembenutzer erlauben. Die Benutzer können dann die Kommandos **mount** und **umount** benutzen, denen sie entweder die Gerätedatei oder das Verzeichnis zum Aufsetzen als Parameter übergeben können

Tabelle 4.1: Auswahl einiger Optionen in der vierten *fstab*-Spalte

Name	**Beschreibung**
common	Systemunabhängige Daten
public	Öffentliche Daten
service	Koordination und Konfiguration der Zelle
sys_type	Systemspezifische Daten z.B. i586_linux
usr	Home-Verzeichnisse für Benutzer
wsadmin	Konfiguration von Clients

Kommando	**Beschreibung**
kpasswd	wird benutzt um das AFS-Passwort zu ändern
klog	Erwerben eines Tokens
tokens	Überblick der gültigen Tokens
pts	kann Benutzer und Gruppen anzeigen und verwalten
fs	kann ACLs und Quotas anzeigen/verwalten

Tabelle 4.2: Einige wichtige AFS-Kommandos

Kommando	**Beschreibung**
fs help	erklärt alle zur Verfügung stehenden Optionen
fs lq <dir>	zeigt definiertes Quota für ein Verzeichnis an
fs la <dir>	listet die gesetzten Rechte
fs sa <dir> <user> <rights>	setzt Rechte für ein Verzeichnis
fs q	Prozentangabe der Quotaauslastung

Tabelle 4.3: Setzen und Ansehen der Zugriffsrechte

Kürzel	**AFS-Recht**	**Beschreibung**
a	administer	die ACLs ändern
d	delete	Dateien und Verzeichnisse löschen oder verschieben
i	insert	Neue Dateien und Verzeichnisse anlegen
l	lookup	Inhalt des Verzeichnis kann aufgelistet werden und ein Wechsel in dieses Verzeichnis ist erlaubt
r	read	Inhalt der Dateien lesen. Dazu gehört z.B. auch `ls -l`
w	write	Ändern von Dateien
k	lock	Ermöglicht das Ausführen von Programmen die über Systemcalls Dateien sperren (locks)

Tabelle 4.4: AFS Rechteschema

Zusammenfassung	Beschreibung
all	alle Berechtigungen (rlidwka)
none	keine Berechtigung
read	nur lesen (rl)
write	lesen und schreiben (rlidwk), d.h. alle ausser a

Tabelle 4.5: AFS Rechtegruppierung

	Verteilbar	**Lokal**
Statisch	/usr, /opt	/etc, /boot
Variabel	/var/mail, /var/spool	/var/log, /var/run

Tabelle 4.6: Aufteilung des Dateisystems

Art	Beschreibung	Kennung
"normale" Datei	Klassische Dateien, wie ausführbare Programme, Porgrammbibliotheken, Konfigurations-, Druck-, Textdateien ...	-
Verzeichnisse	"Containerdatei" in der wieder alle sechs Dateitypen vorkommen können.	d
Links	Soft- oder Hardlinks. Softlinks sind Spezialdateien in denen ein Zeiger auf eine andere Datei steht. Diese können im Gegensatz zu Hardlinks auch über physikalische Grenzen von Datenträgern hinweg existieren. Hardlinks sind weitere Einträge in die Inode.	l
Sockets	Erlauben die Kommunikation zwischen Prozessen über das Filesystem analog zu TCP/IP.	s
FIFOs (named pipes)	Einfachere Version (als Sockets) zur Kommunikation zwischen Prozessen über das Filesystem.	p
Gerätedateien	Schnittstellen zum Kernel. Blockorientiert oder Zeichenorientiert.	b (block) od. c (character)

Tabelle 4.7: Dateiarten

Oktalwert	**Beschreibung**	**Benutzergruppe**
4000	Set-User-Id-Bit	
2000	Set-Group-Id-Bit	
1000	Sticky-Bit	
400	Lesezugriff	
200	Schreibzugriff	Dateibesitzer
100	Ausführungs- / Suchzugriff	
40	Lesezugriff	
20	Schreibzugriff	Benutzergruppe
10	Ausführungs- / Suchzugriff	
4	Lesezugriff	
2	Schreibzugriff	Alle anderen
1	Ausführungs- / Suchzugriff	

Tabelle 4.8: Die traditionellen Zugriffsrechte unter Linux

Kapitel 5

Systemüberwachung und Prozessmanagement

5.1 Einführung

In Unix-Systemen können mehrere Programme gleichzeitig gestartet werden. Ein Programm, das gerade ausgeführt wird, trägt die Bezeichnung Prozess. Das Betriebssystem verwaltet mehrere Prozesse mit einem Scheduler, der die Zuteilung der CPU an die Prozesse steuert,in dem er jedem ausführbereitem Prozess die CPU eine gewisse Zeit zuteilt und sie ihm nach Ablauf dieser Zeit wieder entzieht. Dieses Prinzip wird präemptives Multitasking genannt, im Unterschied zum kooperativen Multitasking unter älteren Windows-Versionen, in denen ein Prozess selbst entscheiden konnte, ob er die CPU-Zuteilung an einen anderen Prozess erlaubt. Alle Prozesse erhalten eine eindeutige Nummer (PID), angefangen wird mit der “1”, es geht bis “32767”, dann startet die Vergabe wieder von vorne, wobei bereits belegte Zahlen ausgelassen werden. PID steht für Process Identifier. Somit sind für das System Verwechslungen ausgeschlossen. Logischerweise können Prozesse über diese Zahl auch kontrolliert und gegebenenfalls beendet werden.

Nach dem Starten des (Linux-)Kernels werden eine Reihe von Programmen und Hintergrundsdiensten hochgefahren. Dieses lässt sich in definierten Abstufungen steuern. Vieles wird dabei von speziellen Skripten erledigt. Linux bietet vielfältige Möglichkeiten, sich über den Zustand des Systems zu informieren.

5.2 Systemstart/Runlevel

Es gibt unter Unix/Linux zwei Konzepte: Je nachdem, welches Konzept ein System verfolgt, läuft der bootstrap ein wenig anders ab. Die beiden Familien sind BSD und SystemV, sprich System5. Eines haben beide Familien beim Bootstrap gemeinsam: Die Datei */etc/inittab*. Diese Datei ist der Anfang des Unix-Lebens. Das erste Skript, das von */etc/inittab* aus ausgeführt wird, rangiert unter dem Namen sysinit (Abkürzung si). In diesem Eintrag steht meistens */sbin/init.d/boot*. In dem Sysinit-Skript werden Dinge wie **fsck** und das Einbinden von Filesystemen, Installation von Swap-Partitionen sowie weitere Basisaufgaben erledigt. Bis hierher verhalten sich alle Systeme gleich. Zurück zur */etc/inittab*:

In der */etc/inittab* wird die erste Möglichkeit geschaffen, dass sich jemand einloggen kann, um das System zu managen. Von hier aus werden auch alle anderen Konfigurationsdateien/Startupdateien (indirekt) angestoßen. In dieser Datei wird der Default-/Runlevel festgelegt, und je nach Runlevel auch Konfigurations-Shellskripte wie */etc/rc.d/rc.multi* angestoßen. In dieser und in einer handvoll anderer Dateien wird das gesamte System gestartet. Der Bereich /etc/rc.d ist typisch für BSD-artige Systeme. Die meisten Linuxdistributionen verwenden jedoch das "System V - Init".

SystemV hat eine andere Methode für die Start-Skripte. Hier wird intensiv von Softlinks Gebrauch gemacht. In SystemV gibt es pro Dienst (bzw. Daemon) genau ein Start-/Stop-Skript. Die Start-/Stop-Skripte in SystemV akzeptieren genau einen Parameter; dieser muss entweder "start", "stop", "restart" , "status" ... sein. Ruft man ein solches Skript mit "start" auf, so wird der Dienst hochgefahren.

Entsprechend wird er bei dem Parameter "stop" angehalten. All diese Dateien befinden sich in einem einzigen Verzeichnis, üblicherweise in *(/etc)/init.d* bzw. */etc/rc.d* (Vor der Festlegung der File System Hierarchy Standard fand man diese Unterverzeichnisse unterhalb */sbin*.)

Außer diesem Verzeichnis gibt es auch noch die Unterverzeichnisse:

- (/etc)/init.d/rc0.d
- (/etc)/init.d/rc1.d
- (/etc)/init.d/rc2.d
- (/etc)/init.d/rc3.d
- (/etc)/init.d/rc4.d
- (/etc)/init.d/rc5.d
- (/etc)/init.d/rc6.d

In den Verzeichnissen befinden sich Links der Form

- (/etc)/init.d/rc?.d/S??dienstname –> (/etc)/init.d/dienstname
- (/etc)/init.d/rc?.d/K??dienstname –> (/etc)/init.d/dienstname

Wobei nicht für jedes Skript unbedingt eine Start und eine Stopverknüpfung existieren müssen. Für das automatische Hochfahren von Diensten gibt es zwei Möglichkeiten der Steuerung: Entweder werden die o.g. Links nur bei Bedarf angelegt oder es existieren für jedes Skript die notwendigen Einträge. Dieses ist bei Suse-Linux der Fall. Die Steuerung ob ein Dienst gestartet werden soll oder nicht, findet hier in der zentralen Konfigurationsdatei */etc/rc.config* statt.

Möchte das System nun aus irgendeinem Grund in einen Runlevel kommen, so schaut es (auch wieder ein Skript) in dem entsprechenden Verzeichnis nach Dateien (die in Wirklichkeit Links nach (/etc)/init.d sind), die mit "S" beginnen, und führt diese in alphabetischer/numerischer Reihenfolge mit dem Parameter "start" aus. Konventionsgemäß kommt nach dem "S" eine zweistellige Zahl, mit der man festlegen kann, in welcher Reihenfolge ein Dienst gestartet werden soll. Soll ein Dienst in einem Runlevel nicht verfügbar sein, so gibt es einen Softlink auf das entsprechende Skript, der mit einem K beginnt. Diese Skripte (K*) werden mit dem Parameter "stop" ausgeführt - der Dienst wird dann gestopt.

Mit den neueren SuSE-Versionen (ab 7.3 bzw. 8.0) wird der Eintrag von Diensten in die entsprechenden Runlevel mit dem Programm **insserv** erledigt.

Suse-Linux verwendet nicht alle Runlevel, wobei die folgenden Runlevel standardmässig zur Verfügung stehen und nachstehende Bedeutung haben (dieses kann von anderen Linuxdistributionen abweichen):

Runlevel	**Funktion**
S	Single-User-Mode
0	Halt des Systems
1	Single-User-Mode ohne Netzwerk ohne X
2	Multiusermode ohne Netzwerk ohne X
3	Multiusermode mit Netzwerk ohne X
5	Multiusermode mit Netzwerk und grafischer Oberfläche
6	Reboot

Tabelle 5.1: Runlevel

Den aktuellen Runlevel kann man sich mit dem Befehl **runlevel** anzeigen lassen. Wobei nur der Init-Prozess dafür sorgt, dass die Programme, die in der */etc/inittab* angegeben sind, permanent laufen, wenn sie über eine gültige Konfiguration verfügen. Ist man erstmal in einem bestimmten Runlevel, so können bereits weitere Dienste von Hand nachgestartet oder gestoppt werden, hier reagiert das System nicht automatisch.

5.3 Dämonen

Im Unterschied zu anderweitigen Prozessen, die stets an ein Dialogstation (tty) und Benutzer gekoppelt sind und je nach Aufruf im Vorder- oder Hintergrund laufen, werden Dämonen meistens automatisch beim Hochfahren des Systems über die Runlevelskripte gestartet. Sie sind nicht an eine Konsole gebunden und schreiben deshalb ihre Meldungen üblicherweise über das Syslog oder in spezielle Logdateien und -verzeichnisse.

5.4 System- oder Resourcen-Auslastung

Prozesse können im System in verschiedenen Zuständen auftreten. Die naheliegensten sind "in Ausführung befindlich" und "auf die Zuteilung von CPU-Zeit wartend". Meistens wird aber noch feiner klassifiziert, wobei die Einteilung für verschiedene Unix-Systeme abweicht. Unter Linux gibt es folgende Einteilung:

- **R - runnable (ablaufbereit)** - Der Prozess wird gerade auf der CPU abgearbeitet oder steht zur Abarbeitung bereit.

- **S - sleeping (schlafend)** - Der Prozess wartet auf das Eintreten eines Ereignisses, zum Beispiel das Ablaufen eines Timers, das Ende einer Interaktion mit Festplatten, dem Nutzer oder einem anderen Prozess.
- **D - uninterruptibly sleeping (Schlaf ohne Unterbrechungsmöglichkeit)** - Der Prozess wartet auf einen bestimmten Hardwarezustand.
- **T - traced (gestoppt)** - Der Prozess befindet sich im Einzelschrittlauf (Tracing).
- **Z - zombie** - Der Prozess wurde beendet, der Elternprozess hat jedoch den Rückkehrstatus noch nicht geprüft.

Informationen über den Zustand der momentan im System ablaufenden Prozesse kann sich der Administrator auf verschiedene Weise beschaffen. Dazu fallen einem zuerst die folgenden Tools ein: **ps** und **top**. Sie zeigen laufende Prozesse und ihren Status an.

5.4.1 ps

Der Befehl **ps** liefert einen "snapshot" der laufenden Prozesse in der gerade aktiven Shell bis zum kompletten Abbild des Systemzustandes. Was die einzelnen Spalten bedeuten, steht in der Manpage. Jeder Prozess im Unix-System wird von einem anderen Prozess mit Hilfe eines Systemaufrufs gestartet. Dadurch entsteht eine Eltern-Kind-Beziehung zwischen den Prozessen. Diese Abhängigkeiten von Prozessen lassen sich mit dem Kommando **pstree** oder auch durch `ps auxf` darstellen. Mit einer bestimmten Option wird neben der Prozess-ID (PID) auch die Prozess-ID des Elternprozesses (PPID - parental PID) angezeigt. Der Vaterprozeß wird vom "Ableben" (der Beendigung) seines Kindprozesses durch einen speziellen Mechanismus, der Signal genannt wird, benachrichtigt.

Jeder Prozess hinterläßt bei der Beendigung einen sogenannten Exitstatus, den der Vaterprozeß auswerten kann. Der Exitstatus enthält Informationen darüber, ob und durch welches Signal der Prozess beendet wurde und einen vom Programmierer selbst bestimmten numerischen Wert, mit dem er dem Nutzer weitere Informationen über das Ende des Prozesses geben kann.

Solange der Vaterprozeß die Statusinformation seines beendeten Kindprozesses noch nicht abgeholt hat, existiert dieser in Form eines Zombieprozesses. Ein Zombieprozeß nimmt weder Rechenzeit noch Hauptspeicher in Anspruch, sondern existiert nur als Eintrag in der Prozesstabelle, die das Betriebssystem führt. Unter Umständen kann es dazu kommen, dass der Elternprozeß vor dem Kindprozeß beendet wird. Damit kann dieser die Statusinformation beim Beenden des Kindprozesses nicht mehr abfragen. In diesem Fall übernimmt **init** diese Statusmeldung.

Jeder Prozess besitzt einen Standardeingabe-, Standardausgabe- und Standardfehlerkanal. Diese sind normalerweise mit der Tastatur (Standardeingabe) bzw. dem Bildschirm (Standardausgabe, Standardfehler) verbunden. Wie diese Kanäle umgebogen werden können, so dass Fehlerausgaben zum Beispiel in eine Datei geschrieben werden, wird im Rahmen der Betrachtungen zur Shell-Kommandozeile behandelt. (s. 3.2.4, S. 32)

5.4.2 top

Dieses Kommando greift in regelmässigen Abständen auf das Procfilesystem zu und gibt eine gute (sortierte) Übersicht darüber, welche Task auf einem Rechner wieviel Resourcen (i/o , Memory , Prozessor) schluckt. Die Zeit zwischen den Updates kann bereits beim Aufruf über die Kommandozeile eingestellt werden. **top** wird interaktiv bedient. Das einfache Drücken einer Taste genügt also, um dem Programm mitzuteilen, was Sie wünschen. Die wichtigsten Tasten sind:

- h : zeigt die Hilfe mit allen Tastenkürzeln an.
- M : sortiert die Liste nach dem Speicherverbrauch der Programme.
- P : sortiert die Liste wieder nach der Prozessorbelastung der Programme .
- k : killt einen Prozess. Fragt nach der PID und dem gewünschten Signal.
- q : beendet top

Nun soll kurz auf ein paar der Spalten eingegangen werden, die **top** in seiner tabellarischen Ausgabe anzeigt:

PID	Logischerweise der PID des angezeigten Programms
USER	Der Benutzer, der das Programm aufgerufen hat
SIZE	Die Größe des Programms im Arbeitsspeicher
SHARE	Menge des Speichers, den das Programm mit anderen gemeinsam nutzt
%CPU	Anteil an der Prozessorauslastung durch das Programm
%MEM	Anteil an der Auslastung des Arbeitsspeichers durch das Programm

Tabelle 5.2: Spaltenüberschrift und deren Bedeutung

top kann in vielen Fällen ein hilfreiches Tool sein. Kommt Ihnen Ihr Computer z.B. viel zu langsam vor, so kann es sein, dass ein oder mehrere Programme die Ressourcen übermäßig beanspruchen. Diese Programme können Sie mit **top** leicht ausfindig machen und ggf. beenden. Desweiteren ist es sehr interessant zu sehen, was das System am meisten belastet.

5.4.3 uptime

uptime zeigt die Systemzeit, die Zeitspanne in der das System bereits läuft, die Zahl der Benutzer(Innen) und die durchschnittliche Systemlast der letzten Minute, fünf Minuten und 15 Minuten an:

```
dirk@shuttle:~ > uptime
 10:54pm  up  7:00,  0 users,  load average: 1.04, 1.11, 1.25
```

5.4.4 time

Manchmal kann es recht nützlich sein, die Zeit zu ermitteln, die einzelne Prozesse zur Abarbeitung benötigen: **time kommando** (z.B. um einen einfachen Bench zur Systemleistung zu haben, kann man die Kernelkompilationszeit mit **time make bzImage** ermitteln).

5.4.5 nice und renice

Mit **nice** kann man beim Start eines Prozesses bestimmen, wie kooperativ (freundlich) er mit der Systemresource "Prozessor" umgehen soll. Gerade für nicht-interaktive Prozesse (Langläufer) sollte man die Freundlichkeit erhöhen. Mit **renice** geht das auch im Nachhinein.

5.4.6 kill, killall -9

Frißt ein Prozess gar zu viele Resourcen, so bestraft man ihn am besten mit **kill** oder **kill -9** - um ihn zu beenden. Aber Achtung, wenn der Prozess gerade im IO steckt, so klappt das eventuell nicht. Das Kommando **killall programmname** wirkt auf alle Prozesse mit einem bestimmten Programmnamen.

Ein Signal wird durch einen numerischen Wert repräsentiert (kill -N). Jedes Programm kann sogenannte Signalhandler für die meisten Signale anbieten, in denen dann das Signal ignoriert oder in angemessener Weise darauf reagiert werden kann. Obwohl Signale allgemein vom Betriebssystem verwendet werden, um dem Programm schwerwiegende Fehler anzuzeigen (zum Beispiel Überschreitungen des Prozess-Speicherbereichs, Divisionen durch Null, ...), kann der Nutzer den von ihm selbst gestarteten Prozessen auch Signale schicken und die Prozesse damit steuern.

Signal	**Nummer**	**Beschreibung**
SIGHUP	01	Hangup (Auflegen bzw. erneutes Lesen der Konfiguration)
SIGINT	02	Interrupt (Unterbrechung z.B. durch Strg-[c])
SIGQUIT	03	Quit (Beenden)
SIGALL	04	illegale Instruktion
SIGTRAP	05	Trace Trap (-> Debugger)
...		
SIGKILL	09	Unbedingte Beendigung des Prozesses durch das Betriebssystem
...		
SIGSEGV	11	Prozess hat den Speicherschutz verletzt
...		
SIGTERM	15	Prozess soll Ausführung beenden

Tabelle 5.3: Prozess-Signale

Dabei legt der Programmierer fest, wie auf bestimmte Signale reagiert wird: SIGTERM soll zwar die Ausführung eines Prozesses beenden, kann auch ignoriert werden, sodass der Prozess dann nicht auf die Beendigungsaufforderung reagiert. Will man als Nutzer oder Administrator einen laufenden Prozess zerstören, so sollte zunächst das Signal SIGTERM benutzt werden. Viele Prozesse sichern daraufhin für ihre weitere Arbeit wichtige Daten und beenden sich dann selbst. Erst wenn auf SIGTERM hin keine Reaktion erfolgt, sollte SIGKILL verwendet werden.

SIGSEGV hat eher Informationscharakter: Ein Prozess hat den Speicherschutz verletzt und versucht, außerhalb des für ihn vorgesehenen Speicherbereiches zu schreiben. Dieses Signal ähnelt von der Bedeutung her der "Allgemeinen Schutzverletzung"[1] unter älteren Windows-Systemen. Man beachte jedoch, dass unter Unix-Systemen ein Prozess keine Systemdatenstrukturen beschädigen kann. Daher ist dieses Signal mehr als Information über den Grund des Programmabsturzes zu betrachten. Das System wird auch nach dem SIGSEGV eines Prozesses stabil bleiben.

5.5 Selbständige Prozesse

Normalerweise führt das Beenden eines Elternprozesses zur Terminierung aller von diesem Prozess abgeleiteten Kindprozesse (Kann man sich mit `ps auxf` anzeigen lassen). Im Einzelnen zieht ein **logout**, das Abmelden vom System, ein automatisches Löschen aller noch anhängigen Jobs nach sich, die wärend dieser Sitzung gestartet wurden.

Abhilfe schafft hier der Befehl **nohup Kommando &**. Denn **nohup** startet den Prozess in der Weise, dass er gegen ein Beenden der Shell immun ist und schreibt die evtl. anfallenden Ausgaben des Befehls in eine Datei (üblicherweise *nohup.out*). Ein Nebeneffekt der Verwendung von **nohup** ist, dass die Prozesse mit einem um fünf höheren Nicelevel gestartet werden.

5.6 Zeitsteuerung

Linux bietet etliche Programme für das Automatisieren von Aufgaben. Ein Beispiel ist **cron**. Dieser ist ein Systemdienst und sollte beim Booten automatisch gestartet und im Normalbetrieb nicht beendet werden. Cron ist für sich wiederholende Aufgaben zuständig, die automatisch zu bestimmten Zeiten stattfinden sollen.

Für die Verwaltung seiner Aufgaben liest **cron** die *crontab*-Datei. Das System und jeder User haben ihre eigenen *cron*-Dateien. Die des Systems befindet sich in der Datei */etc/crontab*. (Sie sollte möglichst nicht geändert werden!) Auch der Benutzer "root" sollte seine eigene Datei erzeugen.

Das Kommando **crontab /etc/crontab** wird eine crontab-Datei erzeugen, die eine Kopie der System crontab-Datei ist. Diese Datei kann nun mit **crontab -e** editiert werden. Beachten Sie, dass *crontab* sowohl der Name der Datei, als auch der Name des ausführbaren Programmes ist - ähnlich wie *passwd*. Die erzeugte Datei wird in ihrer Grundstruktur in etwa wie diese Konfiguration eines SuSE-Systems aussehen:

[1] dem berühmtesten Windows-Fenster, dem Blue-Screen

```
SHELL=/bin/sh****
#dhcpcd: if not then delete /var/run//dhcpcd-eth0.pid file

PATH=/usr/bin:/usr/sbin:/sbin:/bin:/usr/lib/news/bin
MAILTO=root
#
# check scripts in cron.hourly, cron.daily,
#  cron.weekly, and cron.monthly
#
-*/15 * * * *   root  test -x /usr/lib/cron/run-crons && /usr/lib/\
cron/run-crons >/dev/null 2>&1
59 *  * * *     root  rm -f /var/spool/cron/lastrun/cron.hourly
15 4  * * *     root  rm -f /var/spool/cron/lastrun/cron.daily
29 5  * * 6     root  rm -f /var/spool/cron/lastrun/cron.weekly
44 4  1 * *     root  rm -f /var/spool/cron/lastrun/cron.monthly

# reinitialize adsl connection
30 4  * * *     root (rcnetwork stop dsl0; sleep 5; rcnetwork start\
dsl0;)
```

Die Zeilen zur Steuerung bestimmter Abläufe haben ein spezielles Format. Fünf Zeitfelder, gefolgt von dem auszuführenden Programm. Die systemweite *crontab* besitzt ein weiteres Feld, das **cron** anweist, das Programm als spezieller User auszuführen (z. B. "root"). In einem User-crontab wird dieses Feld ignoriert. Die fünf Zeitfelder sind: Minuten, Stunden, Tag-des-Monats, Monat, Wochentag.

Zeit-/Datumsfeld	**Definitionsbereich**
Minute	0-59
Stunde	0-23
Tag-des-Monats	1-31
Monat	1-12 (oder Namen, siehe unten)
Wochentag	0-7 (0 oder 7 ist Sonntag oder Namen)

Tabelle 5.4: Definitionsbereiche der Zeit- und Datumsfelder

Ein Feld kann auch ein Stern (*) sein, was immer für "Erster-Letzter" steht. Zahlenbereiche sind erlaubt. Bereiche sind zwei Zahlen, getrennt durch einen Bindestrich. Die angegebenen Grenzen sind inklusive. Beispielsweise: 8-11 in "Stunde" bewirkt die Ausführung um 8, 9, 10, 11 Uhr. Listen sind ebenfalls erlaubt. Eine Liste ist eine Menge von Nummern (oder Bereichen), getrennt durch Kommata. Beispiele: 1,2,5,9 oder 0-4,8-2.

Schrittweiten können in Verbindung mit Bereichen genutzt werden. Hinter einem Bereich mit "/<Schrittweite>" angegeben, bestimmt die Schrittweite, ob Werte innerhalb des Bereiches übersprungen werden. Beispiel: "0-23/2" kann unter Stunden benutzt werden, um ein spezielles Kommando alle zwei Stunden auszuführen. Die Alternative wäre: "0,2,4,6,8,10,12,14,16,18,20,22". Schrittweiten sind auch nach Sternen (*) erlaubt, "alle zwei Stunden" lässt sich auch durch "*/2" beschreiben.

Namen können für “Monat” und “Wochentag” benutzt werden. Benutzen Sie die ersten drei Buchstaben des entsprechenden Tages oder Monats (Gross-/Kleinschreibung wird nicht beachtet). Bereiche oder Listen sind mit Namen nicht erlaubt.

5.7 Systeminformation- und Überwachung

5.7.1 System-Log

Der Syslog-Daemon ist vor allem bei der Fehlersuche und Sicherheitsprüfung ein wichtiger Helfer. Programme können an ihn Meldungen schicken (mit Hilfe spezieller Befehle), die dann - je nach Einstellung - in verschiedenen Dateien und/oder auf Konsolen auch auf anderen Rechnern geschrieben werden. Viele Daemons und der Kernel benutzen diese speziellen Befehle und geben ihre Ausgaben und Meldungen an den **syslogd** weiter, da sie sonst keine Ausgabemöglichkeit haben.

Typ		**Priorität**	
auth	Authentifizierung	debug	Debugmeldungen
cron	Meldungen des cron-Daemon	info	Informationen
daemon	Meldungen von Daemons	warn	Warnungen
kern	Kernelmitteilungen	crit	kritische Fehlermeldungen
syslog	Meldungen des Logsystems	...	weitere ...
mail	Meldungen des Mail-Subsystems		
...	weitere ...		

Tabelle 5.5: Unterteilung der Meldungen nach zwei Eigenschaften

Als einer der ersten Prozesse wird beim Hochfahren von Linux **syslog** gestartet. Dieser Dienst nimmt die System-, Fehler- und Warnmeldungen des Kernels und der einzelnen Dienste auf und schreibt sie je nach Konfiguration (*/etc/syslog.conf* an einen Logserver im Netz, in die Datei(en) */var/log/messages* bzw. */var/log/warn* (und weitere) und/oder auf die zehnte Textkonsole. Wenn viele Log-Informationen geschrieben werden, kann es unterhalb von */var/log* voll werden, was vielleicht zum Überlauf der entsprechenden Partition führt. Je nach Log-Einstellungen muss also dieser Bereich regelmäßig kontrolliert werden.

Jeder Autor eines Programmes, das Meldungen an den **syslogd** weitergibt, ist angehalten, sich an diese Konventionen zu halten; die Schlüsselwörter also im richtigen Sinne zu gebrauchen.

```
# /etc/syslog.conf - Configuration file for syslogd(8)
#
# einige Standardlogfiles
#
auth,authpriv.*                 /var/log/auth.log
*.*;auth,authpriv.none          -/var/log/syslog
cron.*                          /var/log/cron.log
```

```
daemon.*                        -/var/log/daemon.log
kern.*                          -/var/log/kern.log
user.*                          -/var/log/user.log
#
#
# alle DEBUG-Meldungen in diese Datei
#
*.=debug;\
        auth,authpriv.none;\
        news.none;mail.none     -/var/log/debug
#
#
# alle oben nicht gelogten Meldungen in diese Datei
#
*.=info;*.=notice;*.=warn;\
        auth,authpriv.none;\
        cron,daemon.none;\
        mail,news.none          -/var/log/messages
#
#
# alle Meldungen auf die Konsole 10 ...
#
daemon,mail.*;\
        news.=crit;news.=err;news.=notice;\
        *.=debug;*.=info;\
        *.=notice;*.=warn       /dev/tty10
#
#
# ... und die XKonsole
#
daemon.*,mail.*;\
        news.crit;news.err;news.notice;\
        *.=debug;*.=info;\
        *.=notice;*.=warn       |/dev/xconsole
```

5.7.2 Boot-Log

Für einen Administrator ist es sinnvoll zu wissen, was beim Booten vor sich ging. Diese Information erhält man aus den Logfile(s) */var/log/boot.msg* (in Abhängigkeit von der Distribution), */var/log/messages*, sofern der **syslogd** läuft und aus dem Befehl **dmesg**.

5.7.3 Belegung des Plattenspeichers

Auskunft über den verfügbaren Festplattenplatz erhält man mit dem Kommando **df** (DiskFree). Dieses zeigt die einzelnen gemounteten Bereiche des Filesystems auf. Mit

du (DiskUsage) lässt sich der belegte Platz in einem Verzeichnis bestimmen. Dieses kann jedoch je nach Grösse des Bereichs recht lange dauern. Die Kommandos **df**, **du** und **ls** kennen den gemeinsamen die Option "-h", die aus der Zahlenausgabe in Byte etwas "menschlich lesbares" (von human readable) generiert.

5.7.4 Offene Dateien und Netzwerkverbindungen

Das Kommando **lsof** (ListOpenFiles) zeigt an, welche Prozesse welche Dateien (zum Lesen oder Schreiben) geöffnet haben. Weiterhin werden offene Sockets (Netzwerkverbindung mit Portadresse), sowie verwendete Pipes und Filesockets angezeigt.

5.7.5 Das Kommando "netstat"

netstat hat sich auf das Anzeigen offener Sockets, sowohl des Filesystems, als auch von Netzwerkverbindungen spezialisiert. Die Ausgabe von `netstat -a` hat folgendes Aussehen, wobei die Protokolle TCP, UDP und unix (Socket im Filesystem) in dieser Reihenfolge sortiert aufgeführt werden.

```
dirk@shuttle:~/ > netstat -a |more
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address Foreign Address State
tcp        0      0 *:nfs         *:*             LISTEN
tcp        0      0 *:mysql       *:*             LISTEN
udp        0      0 *:who         *:*
udp        0      0 *:nfs         *:*
...
Active UNIX domain sockets (servers and established)
Proto RefCnt Flags    Type   State     I-Node Path
unix  9      [ ]      DGRAM            613    /dev/log
unix  2      [ ACC ]  STREAM LISTENING 3810   /tmp/.X11-unix/X0
unix  2      [ ACC ]  STREAM LISTENING 6332   /tmp/.ICE-unix/1051
...
```

Mit `netstat -M` kann man sich einen Überblick verschaffen, welche Netzwerkverbindungen gerade maskiert werden:

```
root@server:/ # netstat -M
IP masquerading entries
prot  expire  source          destination   ports
tcp   0:40.46 1234.test.local 210.24.30.216 3592 -> 2183 (64155)
tcp   0:45.64 1234.test.local 210.24.30.216 3593 -> 2187 (64156)
tcp   0:49.43 1234.test.local 210.24.30.216 3594 -> 2188 (64157)
tcp   1:03.12 1234.test.local 210.24.30.216 3596 -> 2195 (64159)
tcp   0:54.57 1234.test.local 210.24.30.216 3595 -> 2192 (64158)
...
```

Wird die Option "-n" beim Aufruf des Kommandos gesetzt, wird keine Namensauflösung durchgeführt, was die Anzeige stark beschleunigen kann, gerade bei IP- Nummern, die nicht im Nameserver bekannt sind.

netstat -r liefert die Kernel-Routingtabelle in ähnlicher Weise, wie sie auch vom Kommando **route** angezeigt wird. Wenn Sie netstat mit dem Flag -i aufrufen, gibt es die Statistiken für die gerade aktiven Netzwerk-Schnittstellen aus. Geben Sie außerdem das Flag -a mit an, werden alle im Kernel vorhandenen Schnittstellen ausgegeben, nicht nur die konfigurierten. An der Konsole produziert **netstat -i** in etwa folgendes:

```
root@server:/ # netstat -i
Kernel Interface table
Iface   MTU  Met  RX-OK  RX-ERR  RX-DRP  RX-OVR  TX-OK  TX-ERR  \
TX-DRP  TX-OVR  Flags
lo        0    0   3185       0       0       0  3185        0  \
     0       0   BLRU
eth0   1500    0 972633      17      20     120  628711    217  \
     0       0    BRU
```

Die Spalten MTU und Met geben die aktuelle MTU (Maximal Transfer Unit) und Metrik des Interface an. Die mit RX bzw. TX überschriebenen Spalten geben an, wie viele Pakete fehlerfrei empfangen bzw. gesendet wurden (RX-OK/TX-OK), wie viele beschädigt waren (RX-ERR/TX-ERR), wie viele wegeworfen werden mußten (RX-DRP/TX-DRP) und wie viele aufgrund eines Overruns verlorengingen (RX-OVR/TX-OVR). Die letzte Spalte zeigt wieder die Liste der Flags, die für die Schnittstelle gesetzt sind. Das sind einbuchstabige Versionen der langen Flagnamen, die **ifconfig** ausgibt. Siehe Tabelle 5.7.5 auf S. 80

Buchstabe	**Bedeutung**
B	BROADCAST
L	LOOPBACK
M	PROMISC
O	NOARP
P	POINTOPOINT
R	RUNNING
U	UP

Tabelle 5.6: Flags

5.8 Aufgaben

5.8.1 Runlevel

1. Welchen Sinn haben Runlevel?

2. Ab welchem Runlevel einer SuSE-Distribution habe ich üblicherweise eine a) Netzwerkverbindung, b) die grafische Oberfläche?

3. Starten Sie den Rechner manuell im Runlevel 1! Was hat dieser zu bedeuten? Können Sie nun andere Rechner im Netz erreichen? Wie erreicht man, dass die Maschine sofort in einen bestimmten (anderen als den Defaultrunlevel) startet?

4. Wechseln Sie nach dem Hochfahren von Linux auf Runlevel 1 und anschliessend auf Runlevel 3!

5. Laufen in einem bestimmten Runlevel immer die entsprechenden Programme oder kann dieses abweichen? Wie wird die Reihenfolge festgelegt, in der bestimmte Prozesse gestartet werden, welchen Sinn macht das?

6. Wie sorge ich dafür, dass bestimmte Skripte beim Wechsel des Runlevels ausgeführt werden? Wie regelt das Suse, um etwas Aufwand zu ersparen? Wie lege ich fest, dass ein bestimmter Daemon etc. gestartet oder nicht mehr gestartet wird?

7. Wie wird die Reihenfolge festgelegt, in der bestimmte Prozesse gestartet werden, welchen Sinn macht das?

8. Schreiben Sie ein Runlevelskript (die meisten Linux-Distributionen bieten hierzu Vorlagen), welches die LocateDB beim Start des Rechners aktualisiert. Dabei soll bei der Übergabe der Option "start" das Kommando "updatedb" ausgeführt werden und bei "status" zurückgemeldet werden, wieviele Stunden es her ist, dass dieses Update stattfand. Bei "stop" soll nur ausgegeben werden, dass nix passiert.

9. Wo würde man dieses Skript hinkopieren und wo würde man es am sinnvollsten in der Hierarchie einordnen? Weiterhin soll es nur in Runlevel 3 und 5 ausgeführt werden, wie erreicht man das?

10. Wie ändert man den "Default-Runlevel", so dass man im netzwerkfähigen Multiusermodus ohne X startet? In welcher Datei geschieht dieses? Wie sorgt man dafür, dass man nur mit 2 Konsolen startet. (Für kleine Maschienen mit wenig Speicher sinnvoll.)

11. Wenn man beim Booten bestimmte Befehle ausführen will, ohne ein Startskript zu schreiben, was kann man dann im einfachsten Fall tun? (In welche Datei sollte die Befehlsabfolge kopiert werden?)

12. Was kann man machen, wenn die grafische Ausgabe nicht funktioniert (wildes Blinken des Bildschirmes, kurzes Aufflackern des typischen grauen Hintergrundes von XFree86, ...) und man sich nicht einloggen kann?

13. Welche Möglichkeiten (grafisch bzw. kommandozeilenorientiert) bietet SuSE-Linux, um bestimmte Dienste einem Runlevel hinzuzufügen oder diese aus einem zu entfernen?

5.8.2 Prozesse

1. Wie bekomme ich heraus, welche Prozesse bestimmte Verzeichnisse noch benötigen (wenn sich z.B. das CD-Rom nicht ausmounten lässt)?

2. Welche Prozesse werden im Runlevel nach Beenden immer wieder neu gestartet und wo wird dieses konfiguriert?

3. Bestimmen Sie den Prozess, welcher die Datei */var/log/messages*, */tmp/.X11...* benutzt!

4. Sortieren Sie die Prozessliste nach Benutzername, Startzeit des Prozesses bzw. Speicherbelegung. Welches andere Kommando kommt dafür in Frage?

5. Welche Möglichkeiten gibt es z.B. den Dienst **dhcpd** zu beenden und sofort wieder neu zu starten? Wo muss bei SuSE was eingetragen sein, damit der **dhcpd** automatisch beim Booten der Maschine gestartet wird? In welche Kategorie von Programmen sind **dhcpd**, **named**, **smbd**, **proftpd**, ... einzuordnen?

6. Wie heisst das Kommando (unter Suse-Linux 8.X), um bestimmte Prozesse (bzw. deren Runlevelskripten) in den automatischen Bootvorgang (bzw. Wechsel der Runlevel) zu integrieren? Was macht dieses Programm genau?

Kapitel 6

Die grafische Oberfläche

6.1 Einführung

Die Basisinstallation eines Linux-Rechners muss nicht zwingend eine grafische Benutzeroberfläche (GUI) vorsehen. Aus Sicht des Linux-Einsteigers macht das die Installation und die ersten Schritte an der Maschine nicht gerade leichter, da heute jeder mit der Handhabung einer Maus und dem Konzept grafischer Oberflächen vertraut ist. Wie man gesehen hat, stellt ein textbasiertes System in vielen Anwendungen einschliesslich der Erstinstallation kein wirkliches Hindernis dar.

Linux "an sich", der Kernel, weiss nichts von Grafik - im Gegensatz zu weitverbreiteten anderen Betriebssystemen, die die grafische Oberfläche fest in ihrem Kern integriert haben. Bei UNIXen ist das nicht so, vielmehr läuft das X Window System völlig unabhängig vom Kernel; es ist sogar völlig unabhängig davon entstanden.

Linux, eigentlich jeder Distribution, verwendet, das X-Window-System zur Darstellung der grafischen Benutzeroberfläche. Die Treiber für die verschiedenen Grafikkarten werden vom XFree86-Team entwickelt [1] und stellen neben dem Kernel die zweite wesentliche Hardware-Software-Schnittstelle dar. XFree86 ist eine freie Implementation des X-Window-Systems, welches auf allen Unix-ähnlichen Betriebssystemen bis hin zu Mac-OS-X läuft. Die "86" im Namen läßt vielleicht vermuten, dass diese Software nur auf 3/4/5 86er-Prozessoren lauffähig ist. Ursprünglich wurde XFree86 auch dafür entwickelt, mittlerweile ist XFree86 aber auch auf anderen Prozessor-Architekturen lauffähig.

Anfang der 80er-Jahre kam man am MIT ("Massachusettes Institut of Technologies") darauf, dass es ja ein bisschen schade ist, wenn man ein echtes Mehrbenutzer- und Multitasking-System hat, darauf aber nur mit einer Konsole zugreifen kann. Die naheliegende Idee: Mehrere Anwendungen in unterschiedlichen, bewegbaren Fenstern auf dem gleichen Schirm anzeigen. So entstand X als akademisches Projekt; der Quellcode wurde von vielen kommerziellen UNIX-Anbietern aufgegriffen, fortentwickelt und in ihre UNIXe integriert.

X11 selbst entstand im Jahre 1986 als Ergebnis der beiden Vorgängerprojekte V und W. Im Laufe der Entwicklung wurden die grundlegenden Protokolle weiterentwickelt, blieben aber immer zu den älteren Versionen kompatibel. 1992 erschien dann die erste Version von X11 für PCs. Das XFree86-Projekt selbst startete mit ei-

[1] http://www.xfree86.org

nem Referenzserver für die PC-Plattform, der von Thomas Röll (heute Xi Graphics) geschrieben wurde und dem Projekt zur Verfügung gestellt wurde.

Ungeachtet dessen ist es inzwischen ohne weiteres möglich, mit jedem Standard-Linux in den Genuss einer grafischen Benutzeroberfiche zu kommen. Um eine Grafikkarte zu benutzen, benötigt man - wie bei anderen Betriebssystemen auch - einen Treiber. Diese Treiber stehen in Form sogenannter "X-Server" oder Hardwaretreibermodule zur Verfügung. Es werden im folgenden die Installation dieses Servers und einige ausgewählte Anwendungen vorgestellt.

6.2 X - Vorteile und Grenzen der Unix-GUI

Entsprechend der UNIX-Philosophie, realisiert X nicht alle Aufgaben integriert in ein einziges Programm ohne durchschaubare innere Struktur, sondern gliedert sich in einzelnen Komponenten:

- Der **X-Server** ist das Programm, das Tastatur- und Mauseingaben entgegennimmt und die Resultate auf dem Bildschirm anzeigt. Er stellt hierfür die passenden Gerätetreiber bereit. Die Events, wie die Benutzereingaben genannt werden, wertet der X-Server nicht selbst aus[2], sondern leitet diese an die betreffenden X-Clients weiter. Diese reichen wiederum ihre Resultate an den X-Server zurück.

- Die **X-Clients** sind praktisch alle Anwendungen (Browser, Textverarbeitung, Editor, PDF-Betrachter, ...), die die grafische Oberfläche benutzen wollen. Vom X-Server erhalten sie die Tastatur- und Mausevents, die sie betreffen, und melden ihm zurück, was auf dem Bildschirm erscheinen soll. Dazu wird das so genannte X-Protokoll verwendet.

- Der **Windowmanager** kümmert sich um die "Verwaltung" der Oberfläche, z.B. das Aussehen und die Funktionalität der Fensterrahmen und -menüs für die Anzeigen der X-Clients, Menüs auf dem Desktop, Minimieren und Maximieren der Fenster.

Das X-Protokoll realisiert durch die Trennung von Server und Client eine Schicht-Architektur, die einen wesentlichen Vorteil gegenüber anderen Systemen beinhaltet: Das X-Protokoll setzt auf dem Internet-Protokoll TCP/IP auf. Das bedeutet, dass Clients und Server auf unterschiedlichen Rechnern laufen können. In einem lokalen Netzwerk, in dem eine rechenintensive mathematische Software nur von einem einzigen Rechner verkraftet werden kann, weil alle anderen zu wenig Hauptspeicher haben, kann man sich einfach von seinem Arbeitsplatz aus dort einloggen, das Programm starten, und es erscheint auf dem eigenen Bildschirm, ohne dass man merkt, dass es auf einem ganz anderen Rechner läuft. Die einzelnen X-Clients müssen "nur" das X-Protokoll beherrschen, die Hardware kann ihnen relativ egal sein; das ist zwar heute allgemein so, aber zur Zeit der Einführung von X war diese Idee sehr fortschrittlich.

[2] bis auf bestimmte Ausnahmen, wie das Umschalten der Bildschirmauflösung und das "Abschiessen" (direkte Beenden ohne Umwege) des Servers

- Auf Systemen, die keine grafische Oberfläche brauchen (z.B. Webserver), kann sie einfach weggelassen werden, da sie ja vom Kernel unabhängig ist; das spart Speicher- und Prozessorressourcen.

- Das grafische System kann beliebig herauf- und heruntergefahren werden (z.B. zu Konfigurationszwecken), ohne das das System verändert oder angehalten werden muss.

- Die Textkonsolen arbeiten unabhängig von der grafischen Oberfläche; mit [Strg]-[Alt]-[F1] kann man auf die erste Textkonsole umschalten. Das ist z.B. sinnvoll, falls man schnell an der Systemkonfiguration was ändern will und sich somit auf einer Konsole als Systemadministrator anmeldet. Zum X-Bildschirm geht es standardmäßig mit [Alt]-[F7] zurück.

- Der Windowmanager ist beliebig wählbar; das macht das Erscheinungsbild individuell konfigurierbar.

Neben diesen schönen Eigenschaften merkt man X jedoch auch sein für Softwareverhältnisse hohes Alter an. Für viele moderne Anwendungen, wie Video- und Bildausgaben, ist X nicht sehr effizient und wird in der Darstellungsleistung von anderen Protokollen (wie z.B. Citrix-Metaframe für Windows) überholt.

- Das X-Protokoll kennt nur einfache Anweisungen wie: “Zeichne Linie von A nach B”; moderne Grafikkarten haben viele Möglichkeiten, um den Bildschirmaufbau durch die Hardware zu beschleunigen, diese können aufgrund dieser “Einfachheit” von X nicht genutzt werden und liegen brach. Solange X nicht durch ein neues Konzept abgelöst wird, wird sich daran auch nichts ändern.

- Die Programmierung von X ist durch seine Architektur recht umständlich. Mittlerweile gibt es allerdings so genannte GUI-Toolkits, die auf X aufsetzen und dem Programmierer das Leben recht einfach machen. Als “Altlast” gibt es aber viele alte Programme für X, von denen jedes eine andere Bedienphilosophie verfolgt und die zusammen einen kunterbunt aussehenden Zoo bilden. Dieses änderte sich zwar durch die Einführung von KDE und GNOME, jedoch verfolgen auch diese Desktop-Projekte durchaus eigene Strategien und Vorstellungen der Benutzerführung.

- X hat absolut nichts mit dem Drucksystem zu tun. Das ist zwar nicht unbedingt eine Designschwäche, hat aber zur Folge, dass es zwei paar Stiefel sind, z.B. eine Schriftart am Bildschirm anzuzeigen und sie später auszudrucken.

- X definiert lediglich ein Verfahren, um mehrere (Text-)fenster anzuzeigen. Von einer grafischen Oberfläche wird aber mehr verlangt. Den Anwender braucht dies aber wenig zu kümmern, denn im Zusammenspiel mit den modernen Desktop Environments von Linux liefert X eine Benutzeroberfläche, die kaum Wünsche offen lässt.

6.2.1 Komprimiertes X

Für schmale Bandbreiten, wie sie im WAN-Bereich mit ISDN oder DSL realisiert werden, ist X wegen seines Datenaufkommens eher suboptimal. Hier sollten entweder andere Lösungen wie VNC eingesetzt werden. Möchte man trotzdem X verwenden, hilft vielleicht die folgende Erweiterung des Protokolls weiter.

LBX steht für Low Bandwidth X.11 und führt Caching sowie Kompression in das X-Protokoll ein. Es ist seit Ende 1996 mit Verabschiedung von X11R6.3 eine offizielle Protokollerweiterung. Hierfür läuft auf der Remote-Seite ein Proxyserver. Bevor Applikationen ihre Daten über das Netz geschicken, werden sie vom Proxy komprimiert und gecacht und dann erst an das lokale Display geschickt vor dem der Benutzer sitzt. Um diese Erweiterung nutzen zu können, muss sie sowohl in den X-Server kompiliert sein, als auch das Proxy-Binary in der verwendeten Linux-Distribution vorhanden sein.

6.2.2 Spezielle X-Server und Remote-Displays

Überblick und Betrieb VNC erlaubt den Zugriff auf einen gemeinsamen Desktop von verschiedenen Rechnern, die sogar unterschiedliche Betriebssysteme verwenden können. VNC löst das Problem, indem es den kompletten Desktop des einen Rechners in einem Fenster auf dem zweiten Rechner darstellt. Ein VNC-Server unter Linux ist im Grund ein doppelter Server. Zum einen stellt er einen vollwertigen X-Server dar - nach dem Start kann man ihn in der Regel über *localhost:1* ansprechen. Angezeigt wird dabei aber nichts: Der Server läuft ohne Ausgabe. Zu sehen bekommt man den, auf dem neuen X-Server laufenden Desktop erst, wenn man einen VNC-Client (vncviewer) startet: Diesem gegenüber tritt der Server dann als VNC-Server auf und überträgt den Desktop-Inhalt als Bildinformation.

Wenn man zunächst die Voreinstellungen des VNC-Servers testen möchte, rufe man einfach das Skript **vncserver** auf. Dieses übernimmt den eigentlichen Start des Servers **Xvnc** mit Standard-Parametern. Dabei beachte man die Ausgabe des Server-Start-Skripts: Dort wird eine Display-Nummer (:1, :2, ...) angegeben, die für den späteren Zugriff auf den Server benötigt wird. Ein VNC-Server unter Linux ist im Grund ein doppelter Server - Er einen stellt er einen vollwertigen X-Server dar und öffnet einen weiteren Port für den VNC-Client-Zugriff. Deshalb belegt er unter Linux zwei TCP-Ports. Angezeigt wird auf dem X11-Port jedoch nichts, die Grafikausgabe erfolgt über den VNC-Port. Hierfür gilt folgende Zuordnung:

VNC-Display	TCP-Port	X11-Display	TCP-Port
:0 (1. Display, wenn kein X11 läuft)	5900	:0 (1. Display, Standard)	6000
:1 (1. Display, wenn X11 läuft)	5901	:1 (2. Display oder **Xnest**)	6001
:2 (weiterer VNC-Server)	5902	:2 (3. Display ...)	6002
...	5903	...	6003

Tabelle 6.1: Standard-Ports für VNC- und XFree86-Server

Aus Sicherheitsgründen legt man ein Passwort für den Zugriff fest: Beim Erststart wird die Abfrage von **vncserver** selbst übernommen, später kann dazu das Kommando **vncpasswd** verwendet werden. Das VNC-Passwort ist komplett unabhängig von irgendwelchen System- oder Accountpasswörtern. Es wird verschlüsselt in der Datei */.vnc/passwd* abgelegt.

Für einen ersten Test greife man lokal auf die laufende VNC-Sitzung zu: Dazu setzt man an der Konsole das Kommando `vncviewer localhost:1` ab, wobei ":1" eventuell durch die richtige Display-Nummer zu ersetzen ist. Dann erscheint nach korrekter Eingabe des gesetzten VNC-Passworts der neue Desktop in einem eigenen Fenster. Der VNC-Server führt nach dem Start das Skript */.vnc/xstartup* aus; diese Datei sollte also an die eigenen Bedürfnisse angepasst werden. Wenn **vncviewer** ohne Argument gestartet wird, dann folgt die Frage nach dem Server zusammen mit der Display-Nummer.

Im Gegensatz zu X11 kann VNC mehrere Clients auf einen Server zulassen. Der einfachste Weg, um das zu erreichen, ist, den Server in der Betriebsart "always shared" zu starten. Diese ist beim Kommando-Aufruf mit anzugeben, da sonst automatisch ein anderer Client beim Start des neuen beendet werden würde:

```
vncserver -geometry 1200x1000 -depth 16 -alwaysshared
```

Dieser Aufruf realisiert eine Auflösung des Serverdesktops von *1200x1000* Bildpunkten, eine Farbtiefe von 16 bit und die Möglichkeit, dass mehrere Clients sich gleichzeitig verbinden können. Die gewählte Bildauflösung kann flexibel eingestellt werden. Wenn man auf dem Zielsystem mit dem **vncclient** zwei nebeneinanderliegende Bildschirme von *1280x1024*er Auflösung füllen will, kann der Server auch mit *-geometry 2400x1000* gestartet werden.

Der VNC-Client steht zusätzlich als Java-Applet zur Verfügung, dass im Browser-Fenster ablaufen kann, womit man sehr einfach von fast überall eine Desktop-Sitzung auf einer entfernten Maschine einleiten bzw. wiederaufnehmen kann. Die Netzwerkanforderungen des klassischen VNC sind recht hoch, es gibt jedoch Implementierungen, die mit Kompression arbeiten und damit auch über sehr geringe Bandbreiten (wie ISDN-Verbindungen) noch akzeptabel funktionieren. Sind die Reaktionszeiten nicht akzeptabel, kann durch Verwendung eines sehr kleinen Desktops (etwa 600x400 Punkte) und nur 8 Bit Farbtiefe die notwendige Bandbreite stark reduziert.

Der ewige Desktop Im Gegensatz zu einer X11-Sitzung, die mit dem Abschalten des Remote-Displays automatisch geschlossen wird, kann eine VNC-Sitzung ewig "weiterleben". Auch wenn sich alle VNC-Clients abgemeldet haben, bleibt der VNC-Server aktiv, und alle darunter gestarteten Programme laufen weiter. So kommt man bei der erneuten Verbindung zu dem Zustand zurück, indem der Client beendet wurde. Trotzdem sollten aus Sicherheitsgründen alle offenen Dateien gesichert werden, bevor der Client geschlossen wird.

Anpassungen des Desktopinhaltes Das Standard-Benutzerinterface, welches VNC anbietet ist nicht besonders spannend. Es startet sehr schnell, da einfach nur ein

xterm und der alte Windowmanager **twm** geladen werden, die beide im Minimalumfang von XFree86 enthalten sind. Der VNC-Server führt nach dem Start das Skript */.vnc/xstartup* aus, welches nach der Installation:

```
#!/bin/sh

xrdb $HOME/.Xresources
xsetroot -solid grey
xterm -geometry 80x24+10+10 -ls -title "$VNCDESKTOP Desktop" &
twm &
```

die gezeigten Zeilen enthält. Diese Einträge können nun beliebig an die eigenen Bedürfnisse angepasst werden. Hier kann nun z.B. ein Start-Skript für eine der grafischen Desktops, wie GNOME oder KDE, oder einen Windowmanager, wie Windowmaker, eingetragen werden.

Absicherung durch SSH-Tunnel VNC kann zwar durch ein Passwort vor unberechtigtem Verbindungsaufbau geschützt werden, jedoch erfolgt die eigentliche Übertragung unverschlüsselt. Dies ist für die meisten Netze inakzeptabel, weshalb ein Tunnel verwendet werden sollte.

Hierzu wird der VNC-Server mit der zusätzlichen Option "-localhost" gestartet, welche verhindert, dass Verbindungen von anderen Rechnern aufgebaut werden dürfen. Damit nun ein Zugriff von einem entfernten Rechner erfolgen kann, wird der VNC-Port über die Secure Shell (**ssh**) getunnelt. Hierzu benötigt man die Port-Nummer des VNC-Servers, die sich leicht aus der Display-Nummer ermitteln läßt, indem man zu dieser 5900 addiert. (Display :1 ist entsprechend über Port 5901 erreichbar!) Wenn der VNC-Server *vncserv01* heisst, lautet das notwendige SSH-Kommando auf dem Client wie folgt:

```
ssh -L 5901:myserver:5901 myserver
vncviewer localhost:1
```

SSH fragt wie üblich nach dem Passwort. Anschliessend wird der **vnc-viewer** gestartet, der sich an das Tunnelende auf dem Client hängt. Der Viewer sieht dieses Tunnelende als lokal laufenden VNC-Server und spricht dazu Port 5901 an. Inzwischen geht das Ganze auch einfacher durch den Aufruf von:

```
vncviewer -via myserver :1
```

Weitere Lösungen Einen klassischeren Ansatz bietet das Programm **Xnest**. Es läuft unter X11 als Client in einem normalen Fenster, bietet selbst aber wieder Serverfähigkeiten an. Mit

```
Xnest :1 -query localhost
```

kann eine weitere XDMCP-Session auf der lokalen Maschine gestartet werden. Da bereits ein X-Server aktiv ist - nämlich der Desktop, an dem man bereits sitzt - muss der nächste freie Port gewählt werden. Dieses verläuft analog zu dem, wie es für VNC beschrieben wurde: Port 6000 ist der Standardport für den ersten X-Server, der mit ":0" bezeichnet wird. Die Addition der Display-Nummer ermittelt den jeweils gültigen Port, d.h. im beschriebenen Fall wird der zweite Server auf Port 6001 gestartet.

6.3 XFree86 Konfiguration

6.3.1 Mehrere Mäuse gleichzeitig

Manchmal ist es ganz nützlich mehr als eine Maus für XFree86 zu konfigurieren, z.B. wenn man ein Laptop hat, an welches sporadisch eine externe Maus zur bequemeren Benutzung angeschlossen wird. Damit mehrere Mäuse unter XFree86 verwendet werden können, lassen sich die zuvor definierten Eingabegräte in der Section "ServerLayout" aufführen und, wie hier für Mouse[1] und Mouse[2] zu sehen, parallel einbinden. Zu achten ist dabei auf den Parameter "SendCoreEvents". Im Beispiel werden hier eine PS/2-Maus und eine USB-Maus parallel betrieben, was sich für meinen Laptop als nützlich erwiesen hat, damit ich gleichzeitig auf den eingebauten Trackpoint und eine eventuell angeschlossene externe Maus zurückgreifen kann.

```
[ ... ]
Section "InputDevice"
    Driver "mouse"
    Identifier "PrimaryMouse"
    Option "Protocol" "PS/2"
    Option "Device" "/dev/psaux"
    Option "Emulate3Buttons"
    Option "Emulate3Timeout" "80"
EndSection

Section "InputDevice"
    Driver "mouse"
    Identifier "OptionalMouse"
    Option "Protocol" "imps/2"
    Option "Device" "/dev/input/mice"
    Option "ZAxisMapping" "4 5"
EndSection
[ ... ]
Section "ServerLayout"
    Identifier "Layout[all]"
    InputDevice "Keyboard" "CoreKeyboard"
    InputDevice "PrimaryMouse" "CorePointer"
    InputDevice "OptionalMouse" "SendCoreEvents"
    Option "Clone" "off"
    Option "Xinerama" "off"
    Screen "Screen[0]"
EndSection
[ ... ]
```

6.4 Desktop Environments

6.4.1 Überblick

Zum sinnvollen Betrieb einer grafischen Benutzeroberfläche unter Linux reicht es nicht aus, einen X-Server zu installieren. Neben den eigentlichen Anwendungen wird auch ein geeignetes Desktop Environment benötigt, welches u.a. einen Windowmanager mitbringt und weitere Features realisiert. Das X-Window-System selbst stellt nur ein geöffnetes Fenster für jedes Programm dar - ohne irgendwelche Rahmen, Buttons und Verzierungen.

Was macht ein modernes GUI (Graphical User Interface) aus? Funktionen wie Copy & Paste, Kontextmenüs, einheitliches Look & Feel, Drag & Drop, Desktop Panel - alles Dinge, die X selbst nicht bereitstellt. Dazu wurden die Desktop Environments oder Desktopumgebungen ins Leben gerufen. Desktopumgebungen bringen darüberhinaus eine ganze Ansammlung nützlicher Anwendungssoftware bereits mit (z.B. Web-Browser, Office-Paket, Spiele, Editor...), und bilden somit ein ganzes Software-Bündel. Zudem verwenden sie einheitliche Grafikbibliotheken (GUI-Toolkits), mit denen diese Anwendungen programmiert sind. Diese Toolkits machen das Aussehen für den Anwender gefälliger und nehmen dem Programmierer eine Menge Arbeit ab. Eine Desktopumgebung ist der Schlüssel zum Erfolg, um ein Betriebssystem einer breiten Masse an Anwendern zur Verfügung zu stellen, weil kaum ein Computeranwender gerne auf Komfort bei der Benutzung seines Computers verzichten will. Er möchte sich in erster Linie auf den Kern seiner Bedürfnisse konzentrieren (z.B. Briefe schreiben, im Internet surfen, Mails senden und empfangen). Solche Dinge mit einem Mausklick schnell und bequem zu machen, mit einem Klick Textpassagen aus dem Internet in die eigene Ausarbeitung zu kopieren, Icons auf der Oberfläche, die mit einem Klick eine häufig genutzte Anwendung starten, all das sind die Leistungungen eines Desktopsystems.

Die beiden wohl verbreitesten Desktopsysteme sind:

- KDE (Kool Desktop Environment), basierend auf dem GUI-Toolkit QT
- Gnome (GNU Network Object Model Environment), basierend auf dem GUI-Toolkit GTK

Beide Desktop Environments werden aber mit jeder größeren Distribution mitgeliefert und gleichen sich sehr im Funktionsumfang. Welches ist nun die geeignetere Oberfläche für einen bestimmten Einsatzzweck? Hier gilt der Grundsatz: ausprobieren! Die Geschmäcker sind verschieden und es gibt genug Alternativen, die nahezu die persönlichen Wünsche und Bedürfnisse eines Anwenders abdecken. Neben KDE und Gnome gibt es alternativ auch noch die Möglichkeit, einen "klassischen" Windowmanager wie FVWM oder IceWM zu verwenden - diese bieten zwar geringeren Komfort, schonen aber die Ressourcen, was z. B. auf einem Rechner, der überwiegend als Server eingesetzt wird, sinnvoll sein kann.

Jedes Standard-Linux-System verfügt inzwischen über eine große Anzahl verschiedener Windowmanager. Sie unterscheiden sich im Aussehen, in ihrer Handhabung und im Komfort. Einige dieser Windowmanager können auch mittels sogenannter "Themes" (verschiedene Erscheinungsbilder) während der Laufzeit im Aussehen angepasst werden. Man kann unter Linux deshalb verschiedene Windowmanager installieren und zwischen diesen wechseln. Ein Neustart der laufenden Anwendungen ist hierfür nicht notwendig.

6.4.2 Kurzdarstellung weiterer Benutzeroberflächen

Hier folgt ein Überblick über einige weniger verwendete Windowmanager, KDE und GNOME werden in eigenen Abschnitten behandelt.

6.4.2.1 Fvwm(2)

Einer der Klassiker unter den reinen Windowmanagern, aber nicht mehr wirklich aktuell und zeitgemäss ist der **fvwm**. Die aktuelle Version 2 wird auch häufig als "fvwm2" bezeichnet. Ein beliebter Ableger des **fvwm** ist **fvwm95**, welcher einem Redmonder Betriebssystem nachempfunden ist, so dass ein Umstieg leichter fallen kann. Am besten löst man ihn durch den **icewm** ab.

6.4.2.2 Windowmaker

Ein sehr beliebter Windowmanager ist der Windowmaker (Aufruf: **wmaker**), der dem Betriebssystem NeXTStep nachempfunden ist. Eine etwas ältere Adaption dieses Look&Feel bietet auch der " AfterStep".

6.4.2.3 Enlightenment

Enlightenment ist ein weiterer Windowmanager. Ziel der Entwicklung war es, einen möglichst weitgehend konfigurierbaren Windowmanager zu schaffen; dies betrifft sowohl das Aussehen wie auch seine Bedienung.

6.4.2.4 IceWM

Schmaler und schneller, aktuell weiterentwickelter Windowmanager mit einer ganzen Reihe von Themes.

6.4.3 GNOME

6.4.3.1 Einführung in GNOME

GNOME ist die Abkürzung für GNU Network Object Model Environment. GNOME ist somit Teil des im Jahr 1984 begonnenen GNU-Projekts, das die Entwicklung eines komplett frei verfügbaren Unix-basierten Betriebssystems zum Ziel hat.

Dieser Abschnitt über GNOME basiert auf dem offiziellen Benutzerhandbuch zu GNOME, das Copyright dafür liegt bei Red Hat Software und David A. Wheeler. Das Benutzerhandbuch zu GNOME unterliegt der GPL. Die deutsche Übersetzung[3] sowie

[3] http://www.gnome.org/users-guide/de/index.html

die englischsprachige Originalversion[4] findet man im Internet. Für dieses Skript wurden natürlich eingige Änderungen gegenüber dem ursprünglichen Text zu GNOME vorgenommen.

In diesem Abschnitt geht es um grundlegende Informationen, wie man die vielfältigen Funktionen und Möglichkeiten von GNOME nutzen kann. Auch wenn sich dieser Abschnitt in erster Linie an Benutzer wendet, die bislang noch nicht mit GNOME gearbeitet haben, sind die darin enthaltenen Informationen angesichts der raschen Weiterentwicklung von GNOME sicherlich auch für fortgeschrittene Anwender von Interesse.

Grundlagen Bei GNOME handelt es sich um eine komfortable grafische Benutzeroberfläche, mit deren Hilfe man als Nutzer den Computer auf einfache Weise konfigurieren und verwenden kann. GNOME besteht aus einem Panel (für das Starten von Anwendungen und Anzeigen von Statusmeldungen), einem Desktop (auf dem Daten und Anwendungen abgelegt werden können) und einer inzwischen sehr umfangreichen Auswahl (je nach Distribution) von mitgelieferten Hilfs- und Anwendungsprogrammen. Zudem stellt GNOME bestimmte Vorgaben (Konventionen) für die Gestaltung weiterer Anwendungsprogramme bereit, um diese nahtlos in die Benutzeroberfche integrieren zu können und deren problemloses Zusammenwirken (z.B. für den Datenaustausch) zu gewährleisten. Sollten Sie schon Erfahrungen mit anderen Betriebssystemen gesammelt haben, werden Sie rasch mit der grafischen Benutzeroberfläche von GNOME zurechtkommen.

Bei GNOME handelt es sich vollständig um Open-Source-Software (d.h. freie Software), deren Quellcode frei verfügbar ist und die von Hunderten von Programmierern auf der ganzen Welt weiterentwickelt wird. Weitere Informationen zum GNOME-Projekt erfährt man wie immer im Internet[5].

GNOME bietet seinen Benutzern eine Reihe von Vorteilen. So macht es einem GNOME einfach, Anwendungen zu konfigurieren und zu benutzen, ohne dabei auf ein Nur-Text-Interface zurückgreifen zu müssen.

Zudem kann man GNOME individuell konfigurieren und somit das Erscheinungsbild und die Funktionen des Desktops nach eigenen Wünschen anpassen. Der in GNOME integrierte Session-Manager speichert die persönlichen Einstellungen der Applikationen und der beendeten Sitzung. Somit steht beim nächsten Starten der Benutzeroberfläche automatisch wieder der individuell gestalteter Desktop zur Verfügung. GNOME unterstützt schon viele Sprachen und ist zudem für die Übersetzung der Benutzeroberfläche in zusätzliche Sprachen vorbereitet. Zudem unterstützt GNOME mehrere Protokolle für Drag and Drop (Ziehen und Ablegen), um dadurch den Datenaustausch mit Anwendungen zu erleichtern, die nicht mit GNOME kompatibel sind.

Darüber hinaus bietet GNOME auch Entwicklern eine Reihe von Vorteilen, die indirekt den Benutzern zugute kommen. So müssen Entwickler keine kostspieligen Softwarelizenzen erwerben, um ihre kommerziellen Anwendungen GNOME-kompatibel zu machen. GNOME wird nicht exklusiv von einem bestimmten Anbieter zur Verfügung

[4] http://www.gnome.org/users-guide/index.html

[5] Die zentrale Website http://www.gnome.org

gestellt - im Gegenteil, keine der Komponenten von GNOME unterliegt den Rechten eines bestimmten Unternehmens oder Einschränkungen hinsichtlich Änderungen oder Weitergabe. Für das Entwickeln von GNOME-kompatiblen Anwendungen können zudem verschiedene Programmiersprachen verwendet werden. Denn GNOME beruht auf der "Common Object Request Broker Architecture"(CORBA), die das nahtlose Zusammenwirken verschiedener Software-Komponenten ermöglicht, unabhängig davon, welche Programmiersprache für die Implementierung verwendet oder welche Plattform gewählt wurde. Und nicht zuletzt kann GNOME in Verbindung mit einer ganzen Reihe von Unix-basierten Betriebssystemen verwendet werden, zu denen auch Linux zählt.

6.4.4 KDE

6.4.4.1 Einführung in KDE

KDE ist die Abkürzung für K Desktop Environement. Das Projekt wurde 1996 von Matthias Ettrich gegründet. Das Ziel des KDE-Projektes ist es die Verbindung des UNIX-Betriebssystems mit dem Komfort einer Benutzeroberfläche. Oder anders gesagt: KDE will UNIX auf den Desktop bringen.

Dieser Abschnitt ist ein Auszug aus den Internetseiten des KDE-Projektes, genauer gesagt aus den FAQ[6], die unter der GNU Free Documentation License dort veröffentlicht sind. Das Ziel dieses Abschnittes ist die Vermittlung von grundsätzlichen Informationen über die Möglichkeiten der Nutzung von KDE.

Grundlagen Bei KDE handelt es sich um eine Benutzeroberfläche für alle Variationen von UNIX. Da die meisten KDE-Entwickler Linux benutzen, läuft KDE zuverlässig auf einer großen Anzahl von Systemen. Bei KDE handelt es sich nicht nur um einen Window Manager. KDE hat auch einen ausgefeilten Window Manager, den Kwin, ist aber in erster Linie eine ausgewachsene integrierte Desktop Umgebung. Darin enthalten sind ein Web-Browser, ein Dateimanager, ein Fenstermanager, ein Hilfesystem, ein Konfigurationssystem unzählbare Werkzeuge und Hilfsprogramme und eine weiter ansteigende Anzahl von Anwendungen.

KDE ist freie Software und steht unter der GNU General Public License (GPL). Alle KDE Bibliotheken sind unter der **LGPL (Lesser GPL)** verfügbar, die eine kommerzielle Softwareentwicklung für den KDE Desktop ermöglicht. Aber alle KDE Andwendungen sind unter der GPL lizensiert. KDE verwendet das "Qt(TM) C++ crossplatform toolkit" das auch unter der GPL veröffentlicht wurde.

Bei Qt(TM) handelt es sich um eine C basierte Klassenbibliothek zur Erstellung von Benutzereingaben. Es ist ein Produkt der Firma Trolltech, die letzte Version ist auf immer auf deren FTP-Server [7]. Außerdem ist Qt(TM) in den meisten aktuellen Linuxdistributionen enthalten.

KDE bietet dem Benutzer zur Systemkonfiguration das KDE-Kontrollcenter mit dem die Desktopumgebung leicht auf dessen Bedürfnisse angepasst werden kann, auch

[6] http://www.kde.org/documentation/faq/index.html
[7] ftpl.trolltech.com

sprach- und länderspezifische Einstellungen können hier vorgenommen werden. Als Dateiverwaltungswerkzeug ist der Konqueror in KDE enthalten, dieser kann u.a auch als Webbrowser und FTP-Client verwendet werden.

Es gibt verschiedene Möglichkeiten unter KDE Software zu entwickeln. Zum einen ist es möglich, Freeware mit Open Source unter der GPL zu entwickeln. Auch für die Entwicklung von kommerzieller Software sind Möglichkeiten vorhanden. Mithilfe der KDE Bibliotheken können kommerzielle Anwendungen unter Offenlegung der Quellen (commercial and open source) erstellt werden, dafür und für die Entwicklung von freier Software ist die von der Firma Trolltech herausgegebene Version Qt(TM) free edition vorgesehen. Für kommerzielle Anwendungen ohne Offenlegung der Quellen kann die Qt(TM) Professionell Edition verwendet werden.

6.5 Aufgaben

6.5.1 XFree86 - Der Grafikserver

1. Wie schaltet man von der Textkonsole zur Grafik und wie kommt man da wieder weg?

2. Wie kann man die Auflösung des X-Servers umschalten?

3. Wie kann man den X-Server beenden und wo kann man einstellen, wenn dieses verhindert werden soll?

4. Auf welcher Konsole(nnummer) landet üblicherweise die grafische Ausgabe? Wie kann man das ändern (in welcher Datei)? Wo landen dann weitere geöffnete Grafikkonsolen? Wie kann man zwischen den einzelnen Sessions umschalten?

5. Mit dem Programm **xhost** kann man Freigaben auf seinen X-Server erstellen. Man gebe seinen Server grosszügigst frei und lasse einen Nachbarn durch Ändern seiner Displayvariablen (wo steht die drin, wie zeigt man sie an, wie ändert man sie?) ein Programm auf seine Ausgabe umlenken.

6. Man experimentiere mit dem Programm **Xnest** herum, um unterschiedliche Anfragen (chooser, direkt, broadcast) zu realisieren.

7. Man probiere mal das Tool **xvidtune** aus und lasse sich die entsprechenden Modlines (Steuerzeilen für den Monitor) ausgeben!

8. Welche Datei muss man editieren, um bestimmten Hosts den Zugang zum Xserver zu gewähren, oder zu verwehren?

9. Was macht der Befehl `X:1 -bpp 16-query localhost`?

10. Was ist **Xnest**? Wie lautet die Syntax zum Aufbau einer Verbindung zu einem anderen System? Welche Ports werden hierfür benutzt?

11. Wofür ist die Datei *Xserver* zuständig? Wo befindet sich diese üblicherweise? Wie kann ich sie finden?

12. Welche Funktion hat der **xdm**? Welche Alternativen gibt es? Nenne zwei! Wo befinden sich die jeweiligen Konfigurationsdateien?

13. Weshalb macht **xvidtune** auf einem digitalen TFT-Display, bzw. für die Ausgabe auf einem Fernseher mit TV-Out wenig Sinn? (Weshalb wird man bei TFT-Displays üblicherweise nur eine Auflösung verwenden? Welche?)

14. Was leistet VNC? Worin liegt der Unterschied zum X11-Konzept?

6.5.2 Benutzeroberflächen

1. Welche grafischen Benutzeroberflächen stehen zur Verfügung? Worin unterscheiden sie sich?

2. Wo werden KDE bzw. Gnome üblicherweise installiert? Wo findet man die dazugehörigen Include-Dateien? Wo sind üblicherweise die Binärdateien des KDE/Gnome, bzw. die entsprechenden Bibliotheken abgelegt? (Wo muss man evtl. dafür sorgen, dass die Bibliotheken/Binaries gefunden werden?)

3. Welcher Windowmanager wird zusammen mit Gnome benutzt oder besser: installiert? Wie bekommt man das heraus?

4. Weshalb kann man nicht als normaler Benutzer den KDM (K-Displaymanager) konfigurieren? Wo liegt dessen Konfigurationsdatei?

5. Welches Programm zaubert die vielen (oder auch nicht so vielen) Icons auf den Desktop von KDE bzw. GNOME?

6. Welche Prozesse sind überlicherweise bei einer Gnome bzw. KDE-Session am Start? Welche Prozesse unter IceWM?

7. Wo müßte man die Language-Variable setzen, damit sie unter den verschiedenen grafischen Oberflächen und im Textmodus ausgewertet wird? Welche anderen Möglichkeiten bestehen, wenn man sich die **Xsession** bzw. den **gdm** ansieht?

Kapitel 7

Der Linux-Kernel und seine Bootloader

7.1 Überblick

Der Kernel ist das eigentliche Betriebssystem. Er abstrahiert die Hardware und stellt eine Vielzahl von Funktionen für die Systemsoftware bereit. Der Kernel enthält alle Hardwaretreiber und da es sich bei Linux um ein Unix-ähnliches System handelt, enthält der Kernel ebenfalls die für ein Netzwerk benötigte Funktionalität. Grafikkartentreiber im Kernel dienen dem Framebuffer, der nicht mit der grafischen Oberfläche X11 verwechselt werden sollte. X11 wiederum kann auf das Framebufferinterface aufsetzen, dieses ist unter der PC-Architektur jedoch eher die Ausnahme. Der Kernel-Framebuffer stellt meistens keine beschleunigte 2D oder 3D Architektur bereit und nutzt die Fähigkeiten der Grafikkarte damit nur selten aus. Die XFree86-Treiber, die erst beim Start des X-Servers benötigt werden, bilden eine Ausnahme, da sie nicht Teil des Kernels sind.

Alle Linux-Distributionen beinhalten einen fertig kompilierten und lauffähigen Kernel. Üblicherweise wird dieser für die niedrigste Architekturstufe auf der die Distribution läuft kompiliert, damit eine allgemeine Kompatibilität der Hardware sichergestellt ist. Die SuSE-Distribution hat neben anderen damit angefangen, mehrere spezifische Kernel zu bauen, die für besondere Einsätze, wie SMP und AMD Athlon gedacht sind.

Die eingesetzte Hardware entwickelt sich rasend schnell weiter. Dieses ist ein wesentlicher Grund, weshalb auch der Kernel einer ständigen Weiterentwicklung unterworfen ist. Um diese Weiterentwicklung zu ermöglichen, den Anwender aber vor den Problemen neuer Treiber zu bewahren, teilt sich die Kernelentwicklung von Linux in zwei Bäume: die Entwicklerkernel und die Produktivkernel. Zu welchem Baum ein Kernel gehört, erkennt der Benutzer an der Versionsnummer. Eine ungerade Zahl an der zweiten Stelle (zum Beispiel 2.3.95, 2.5.43) deutet auf einen Entwicklungskernel, eine gerade Zahl auf einen Produktivkernel (zum Beispiel 2.2.33, 2.4.21).

Die Produktivkernel sollten stabil laufen und werden für alle produktiven Umgebungen empfohlen. Wird eine neue Majorrelease eines Produktivkernels herausgegeben (d.h. die erste oder zweite Stelle der Versionsnummer hat sich geändert), so werden dieser Version keine neuen Treiber und Features mehr hinzugefügt. In den weiteren Versionen werden vor allem Fehler beseitigt. Die Entwicklerkernel dagegen dienen, wie der Name schon sagt, zur Entwicklung. Das heißt, dass hier neue Features oder Treiber eingebracht und getestet werden können. Die Entwicklerkernel laufen meist weniger stabil und sind schnelleren Änderungen unterworfen. Hier findet man jedoch vielleicht schon den Treiber für eine Hardware, die vom Produktivkernel noch nicht unterstützt wird. Ein klassisches Beispiel war der USB-Support im Entwicklerbaum der 2.3er Serie. Entwicklerkernel sollten jedoch nur benutzt werden, wenn man experimentierfreudig ist und auch ab und an einen Systemabsturz aushalten kann. Produktivkernel sind für den Einsatz auf Produktivsystemen getestet und sollten hoffentlich stabil laufen.

7.2 Die Modularisierung

Damit die gesamte im System installierte Hardware unterstützt wurde, mußte der Kernel in frühen Versionen mit allen entsprechenden Treibern konfiguriert und kompiliert werden. Seit Kernelversion 2.0 ist es möglich, fast alle Treiber auch dynamisch ladbar bereitzustellen. Diese dynamisch ladbaren Treiber heisen Module.

Die Module können von "root" mit Hilfe des Kommandos **insmod** oder **modprobe** geladen und mit dem Kommando **rmmod** wieder aus dem Kernel entfernt werden. Sehr komfortabel benimmt sich **modprobe**. Dieses Kommando lädt Module und erlaubt diesen das automatische Erkennen spezifischer Hardwareeinstellungen wie Interrupt oder Portadresse. Schlägt dieses "Autoprobing" fehl, kann der Administrator die Werte dem Module fest in der Datei */etc/modules.conf* zuordnen, die ebenfalls von **modprobe** durchsucht wird. Außerdem beachtet es sogenannte "module-dependencies", wenn diese mit dem Kommando `depmod -a` erzeugt wurden. Dieses Kommando erstellt eine Datei, die die Zusammenhänge zwischen den Modulen enthält.

7.3 Einstieg ins Selberbauen

Bei jeder Distribution liegt ein Kernel bei, der allen Anspüchen an Treiber usw. gerecht werden sollte. Durch den Modulsupport können fast alle Treiber dynamisch nachgeladen werden. Aus welchem Grund sollte man also einen eigenen Kernel bauen? Folgende Punkte sprechen für einen neuen Kernel:

- nur die benötigten Treiber und Grundfunktionen werden eingebunden
- spezielle Eigenschaften können nur direkt im Kernel konfiguriert werden
- der Kernel wird für den entsprechenden Prozessor optimiert
- neueste Kernel stehen immer als Sourcen, aber erst viel später fertig kompiliert zur Verfügung
- spezielle Bootmodelle, wie Diskless Clients lassen sich nicht mit dem Standardkernel betreiben

Aus diesen Günden soll hier kurz auf die Konfiguration und das Kompilieren eines eigenen Kernels eingegangen werden.

Den Kernel zu kompilieren ist etwas, was sich ungeheuerlich schwierig anhört, aber in Wirklichkeit recht einfach ist! Also... nur Mut, es kann nicht allzu viel schief gehen, wenn man sich vorher einige Sachen überlegt. Das Einzige, auf das man immer achten sollte, ist, dass man immer von einem funktionierenden (alten) Kernel booten kann. Das erreicht man, indem man den laufenden Kernel kopiert, die */etc/lilo.conf* editiert und **lilo** einmal startet. Dieses wird im Folgenden vorgeführt:

Man editiert die Datei */etc/lilo.conf* mit seinem Lieblingstexteditor und dupliziert den Image-Eintrag mit der folgenden Zeile, die sich auf den aktuellen Kernel bezieht (üblicherweise "image = /boot/vmlinuz") und benennt das Image nach z.B. "/boot/vmlinuz.bakup" und anschliessend das Label nach z.B. "backup" um. Anschliessend ruft man `lilo (-v)` erneunt auf und kann am Liloprompt im Falle eines Fehlers den alten Kernel starten.

7.4 Bezug des Kernels

Den aktuellen Kernel kann man von vielen FTP-Servern weltweit beziehen, ein guter Ausgangspunkt ist die Kernelseite im WWW: www.kernel.org. Derzeit akutell ist die Kernelserie 2.4, in vielen Fällen setzt man aber auch auf die ältere Serie 2.2, gerade für kritische Serveranwendungen, die auf bestimmte Hardwareunterstützungen verzichten können.

Üblicherweise lädt man den Kernel in das Verzeichnis */usr/src* herunter und packt den Kernel mit `tar -xpzf linux-2.X.Y.tar.gz -C linux-2.X.Y` aus. Vorher sollte man das Zielverzeichnis für den Kernel ("linux-2.X.Y") angelegt haben. Dann verändert man den Link *linux* in diesem Verzeichnis, so dass er auf unseren gerade entpackten Kernel zeigt.

Voraussetzungen für das Übersetzen des aktuellen Kernels, kann man der Datei *Changes* entnehmen, die im obersten Verzeichnis des ausgepackten Kernelbaumes liegt. Eingehendere Informationen zu einzelnen Kerneltreibern und Funktionen sowie weitergehende Hinweise findet man unterhalb von *Documentation*. So sind z.B. ein aktueller C-Compiler (gcc) und die aktuellen Bin- und Modutils (**depmod**, ...) Voraussetzung für eine erfolgreiche Kompilation.

7.5 Konfiguration des Kernels

Zur Konfiguration des Kernels stehen mehrere Möglichkeiten zur Verfügung: Die älteste Methode (jedoch auch die am wenigsten komfortable) wird mit **make config** aufgerufen. Es sind dann nacheinander die Einstellungen vorzunehmen, wobei in den meisten Fällen die Fragen mit **y(es)** oder **n(o)** beantwortet werden. Die Defaultoption ist durch einen grossen Buchstaben symbolisiert. Kann ein Modul erzeugt werden, welches dann später dynamisch in den Kernel nachgeladen werden kann, wird der Buchstabe **m(odule)** zusätzlich aufgeführt.

Die Konfiguration wird im Rootverzeichnis der Kernelsourcen in der Datei *.config* abgelegt. Wenn bereits eine ältere Konfiguration existiert, so wird diese beim Erstellen der neuen nach *.config.old* kopiert. Diese Datei kann als Grundlage für eine neue Konfiguration dienen. Man kopiert sie aus dem Verzeichnis der älteren Kernelsources in das aktuelle und ruft **make oldconfig** auf. Es werden dann nur noch Optionen abgefragt, die im alten Kernel noch nicht existierten oder veränderte Einstellungen haben.

Bei weitem komfortabler ist `make menuconfig`, welches als erstes ein nettes Textfrontend kompiliert, welches menugesteuert arbeitet. Unter der grafischen Oberfläche bietet sich auch `make xconfig` an. Mit einem speziellen Patch kann anhand des laufenden Kernels eine Konfiguration erstellen `make cloneconfig`.

7.6 Kompilation und Installation

Mit **make bzImage** wird der Kernel erzeugt. Das fertige komprimierte Image (**bzImage**) liegt dann im Verzeichnis *arch/i386/boot*. Soll dieses sofort installiert und dem Bootloader übergeben werden, führt man stattdessen `make bzlilo` aus.

Im ersten Fall muss man den Kernel in das Bootverzeichnis (üblicherweise */boot*) kopieren und den Bootloader per Hand aufrufen.

Die Module werden durch `make modules` erzeugt und mit `make modules_install` unterhalb des Verzeichnisses */lib/modules/[Kernelversion]* installiert. Gleichzeitig wird **depmod** aufgerufen und die Liste der Modulabhängigkeiten *modules.dep* im o.g. Verzeichnis angelegt.

7.7 Bootloader

7.7.1 Überblick

Bootloader werden verwendet, um den Kernel eines Betriebssystems von der Festplatte zu starten, ohne dass dieser sich im Master Boot Block befinden muss. Damit ist es möglich, verschiedene Kernel zu laden und diesen Startoptionen mitzugeben. Windows NT verwendet dazu z.B. den **ntloader**.

7.7.2 Der GRand Unified Boot Loader (GRUB)

GRUB ist der Bootloader des GNU-Projektes (deshalb heißt er offiziell auch "GNU GRUB"). GRUB besitzt im Gegensatz zu den meisten anderen Bootloadern wie LILO oder dem NT-Loader einen sehr großen Funktionsumfang und ist komfortabel über ein Shell-artiges Interface zu bedienen.

Eines der wichtigsten Eigenschaften ist die Multiboot-Fähigkeit. GRUB kann verschiedene UNIX-Derivate wie Linux, FreeBSD, NetBSD oder OpenBSD direkt starten, d.h. den Kernel und eventuelle Ramdisks in den Speicher laden. Kommerzielle Betriebssysteme wie die Microsoft Windows-Palette oder OS/2 lassen sich über sogenanntes "chain-loading" starten. Dabei lädt GRUB den Bootloader des jeweiligen Systems in den Speicher und übergibt ihm den Bootvorgang. Dieses Verfahren kann auch angewendet werden, um wiederum LILO oder GRUB selber zu starten. Zum anderen findet GRUB direkt alle Dateien, die zum Systemstart von Nöten sind; im

Gegensatz zu z.B. LILO, der dies mit Hilfe von Blocklisten ermölglicht. Auf diese Listen kann er verzichten, da GRUB in der Lage ist, diverse Dateisysteme direkt zu lesen. Unter anderem unterstützt GRUB EXT2 und EXT3, ReiserFS, FAT32 und BSD FFS. Das Booten von Partitionen, die größer als 8 GByte sind, stellt für GRUB ebenfalls kein Problem dar. Zusätzlich bietet GRUB eine Kommandozeile an, durch die es möglich wird, Konfigurationen zu booten, die noch gar nicht eingerichtet sind. Auch ist es möglich, nach Dateien zu suchen oder sich beispielsweise die Partitionstabelle anzuschauen. Dieses macht ihn vergleichbar mit Bootloadern, wie sie z.B. auf Alpha-Maschinen zu finden sind.

All diese Features machen GRUB zu einer Konkurrenz des lange dominierenden Linux Loader LILO. Ohne Bootloader kann selbstverständlich kein System gestartet werden. Deshalb erfolgt an dieser Stelle der selbstverständliche Hinweis, eine Bootdiskette des alten Bootloaders für den Notfall bereitzuhalten bzw. ein Rettungssystem parat zu haben. Viele Linux-Distributionen liefern GRUB inzwischen standardmäßig aus, so dass dieser als RPM- oder DEB-Paket installiert werden kann und alle notwendigen Verzeichnisse automatisch erzeugt werden.

GRUB sollte natürlich vor der endgültigen Installation im Master-Boot-Record bereits konfiguriert werden, da sich das System sonst nur auf Umwegen starten lässt. Die Datei, die für ein Bootmenü notwendig ist, muss *menu.lst* heißen und unterhalb von */boot/grub* angelegt werden. Für alle Einstellungen ist wichtig, dass GRUB immer bei "0" anfängt zu zählen, womit z.B. die erste Partition auf der primary-slave Festplatte (entspräche */dev/hdb1*) mit *(hd1,0)* bezeichnet würde. Die Titel der einzelnen Konfigurationen müssen eindeutig sein. Im Folgenden wird ein Beispiel-Listing gezeigt:

```
default 1
gfxmenu (hd0,1)/boot/message
timeout 8

title linux
    kernel (hd0,1)/boot/vmlinuz root=/dev/hda2 vga=0x317 splash=silent
    initrd (hd0,1)/boot/initrd

title linux.nopnp
    kernel (hd0,1)/boot/vmlinuz.nopnp root=/dev/hda2 vga=0x317
                splash=silent
    initrd (hd0,1)/boot/initrd.nopnp

title failsafe
    kernel (hd0,1)/boot/vmlinuz.shipped root=/dev/hda2 vga=0x317
                ide=nodma apm=off acpi=off vga=normal nosmp noapic
                maxcpus=0 3
    initrd (hd0,1)/boot/initrd.shipped

title win98
  root (hd1,0)
  makeactive
```

```
  chainloader +1

title winXP
  root (hd1,2)
  makeactive
  chainloader +1
```

Der erste Eintrag nach den Kommentaren ist *default=1*, der im gezeigten Beispiel festlegt, welcher Eintrag im Bootmenü vorselektiert angezeigt werden soll. Standardmäßig beginnt GRUB bei 0, im gezeigten Fall wird "linux.nopnp" markiert. Soll z.B. "win98" als Default gewählt werden, müsste der Wert entsprechend auf 3 eingestellt werden. Anschließend folgt *timeout=10*. Hierdurch wird eingestellt, nach wie viel Sekunden GRUB den markierten Eintrag automatisch starten soll. Wird dieser Eintrag nicht vorgesehen, wartet GRUB solange, bis ein Eintrag gewählt und mit Enter bestätigt wurde.

Da am Boot-Prompt über geeignete Kernel-Commandline-Options ein Linux-System so gestartet werden kann, dass kein Root-Passwort benötigt wird, gibt GRUB die Möglichkeit eine Sicherung einzubauen. Mit dem Eintrag *password=geheim* wird ein Passwort definiert, mit dem sich der Zugriff auf die GRUB-Konsole beschränken lässt. Dieses wird unverschlüsselt in der Konfigurationsdatei abgelegt, weshalb man deren Rechte so setzen muss, dass nur der Systemverwalter sie lesen kann.

Mit *gfximage* gibt man die Lage eines Hintergrundbildes für das Boot-Menü im Dateibaum an. Wie man an dieser Beispielkonfiguration erkennen kann, benutzt GRUB für Parameter, die an den Kernel übergeben werden sollen, keine separate append-Zeile wie LILO. Sämtliche Parameter wie *vga=0x317* werden direkt (mit einem Leerzeichen dazwischen) an den Aufruf des Kernels angehängt.

Ein nettes Feature ist die Eigenschaft von GRUB, Kernel von einem TFTP-Server zu laden, wie es Diskless Geräte mit einem geeigneten Boot-ROM können. Dieses setzt natürlich voraus, dass eine Netzwerkkarte in der Maschine eingebaut ist, die von GRUB unterstützt wird und mit der eine Verbindung zum TFTP-Server hergestellt werden kann. Folgendermaßen würde das Booten dann aussehen:

```
grub> ifconfig --adress=192.168.1.3 --server=192.168.1.1
grub> root (nd)
grub> kernel /vmlinuz root=/dev/hda3
grub> boot
```

Der erste Unterschied zum klassischen Bootvorgang ist, dass die Netzwerkkarte mittels **ifconfig** aktiviert werden muss. Dabei gibt address die eigene IP und server die IP des TFTP-Servers an. Außerdem gibt man bei root als Platte/Partition nd an. Natürlich kann GRUB die Netzwerk-Informationen auch von einem DHCP-, BOOTP- oder RARP-Server abholen.

GRUB bringt darüberhinaus nette Eigenschaften mit, die helfen können, "zerschossene" Systeme wieder zu retten. Wenn sich aus irgendeinem Grund der neue Kernel nicht mehr booten läßt und kein geeigneter Menüeintrag für den alten Kernel vorhanden ist, läßt sich dieser unter Umständen trotzdem booten: Wenn das Boot-Menu erscheint, ist die Taste [c] dran. Dadurch kommt man auf die GRUB-Kommandozeile und kann dort durch Tippen der Befehle, die sonst in der Konfigura-

tionsdatei angegeben werden, den Rechner booten. Beachtet werden sollte, wie auch vom BIOS bekannt, die englische Tastaturbelegung.
Ein Beispiel: Wenn der Kernel im Verzeichnis */usr/src/linux/arch/i386/boot* auf der Partition */dev/hda3* liegt, würde das Ganze so aussehen:

1. Warten, bis Boot Menu erscheint und [c] drücken
2. root(hd0,2) - Partition, auf der der Kernel liegt, angeben
3. `kernel /usr/src/linux/arch/i386/boot/bzImage root=/dev/hda3` - Pfad zum Kernel, Dateiname ist hier *bzImage*. Danach die Optionen wie root-Partition und eventuelle weitere Einstellungen eintragen.
4. boot - Den spezifizierten Kernel booten

Manchmal kann es sehr nützlich sein, den Bootloader auf einer Diskette zu haben[1]. Dazu sind folgende Schritte auf dem Ausgangssystem oder einem anderen beliebigen Linux-Rechner mit installiertem Grub notwendig:

```
root@balisto:~#  fdformat /dev/fd0
root@balisto:~#  mke2fs /dev/fd0
root@balisto:~#  mount -t ext2 /dev/fd0 /misc/floppy
root@balisto:~#  cp -a /boot/grub /misc/floppy
root@balisto:~#  grub-install --root-directory=/misc/floppy '(fd0)'
root@balisto:~#  umount /misc/floppy
```

Mit diesem Schritt kopiert man zwar noch den alten Second-Stage-Bootloader der Altinstallation. Dieses bereitet aber keine Probleme, wenn man sowieso nur auf die Commandline der Grub-Shell zugreifen möchte. Wenn diese von der Diskette geladen wurde, kann es wie oben beschrieben weitergehen. Die Diskette könnte noch um einen Minimalkernel (mit Ramdisk) erweitert werden, solange es deren Speicherplatz hergibt.

7.7.3 Der Linux-Loader (lilo)

Der Linux Boot Loader ist das klassische Werkzeug zum Starten eines Linux-Systems und anderer Betriebssysteme. LILO wird entweder in den Master-Boot-Record (MBR) oder an den Anfang einer Partition innerhalb der ersten 1024 Zylinder der ersten oder zweiten Festplatte geschrieben. Diese Restriktionen werden nicht von Linux, sondern vom BIOS gesetzt.

In diesem Beispiel liegt die root-Partition (hier */dev/hda3*) des Linux-Systems in der dritten Partition der ersten Festplatte. Weitere Systeme (hier Win98 und WindowsXP) liegen auf der ersten bzw. dritten Partition der zweiten Platte. In der Datei */etc/lilo.conf* wird der LILO konfiguriert. Hier werden die Einstellungen vorgenommen, um die Windows-Partition und die Linuxpartition (mit 2 verschiedenen Kerneln) zu booten.

[1] z.B. wenn man eine Festplatte dupliziert hat und nun die neue Festplatte in einem anderen Rechner booten möchte ...

Eine Lilo-Bootdiskette für ein bereits installiertes System - diese Diskette ist aber bei weitem nicht so flexibel, wie eine Grub-Diskette, da sie nur für eine bestimmte festgelegte Konfiguration eingesetzt werden kann - läßt sich auf die nachstehend beschriebene Weise erstellen:

```
root@balisto:~# fdformat /dev/fd0H1440 # Lay tracks on new diskette
root@balisto:~# mkfs -t minix /dev/fd0 1440 # Create minix file system
                                              on floppy
root@balisto:~# mount /dev/fd0 /mnt/floppy # Mount floppy
root@balisto:~# cp -p /boot/chain.b /mnt/floppy # Copy chain loader
root@balisto:~# cp -p /boot/boot* /mnt/floppy
root@balisto:~# lilo -v -C /etc/lilo.floppy # Install lilo and the map
                                              onto floppy
root@balisto:~# umount /mnt/floppy
```

Dabei sollte die Datei *lilo.floppy* z.B. wie folgt aussehen:

```
boot=/dev/fd0        - Diskette, wo der Lilo-Bootsektor hinsoll
map=/mnt/floppy/map  - Stelle der Festplatte, wo die Bootkernel liegen
prompt               - LILO Boot-Prompt anzeigen
linear               - Speziell für SCSI-Konfigurationen
timeout=50           - Timeout zum erreichen des Lilo-Prompts
image=/boot/vmlinuz  - Diesen Linux-Kernel booten
label=linux          - Label für die Identifikation dieses Kernels
root=/dev/hdb2       - Rootverzeichnis
initrd=/boot/initrd  - Ramdisk, welche mit dem Kernel geladen wird
read-only            - Root-Partition read-only mounten
```

7.7.4 Das Syslinux-Paket

Das Syslinux-Paket umfasst eine ganze Gruppe von Bootloadern. Diese erlauben von speziellen Geräten, wie von Diskette oder CD-Rom zu booten. Ein eigenes Paket kümmert sich um das Booten über ein Netzwerk in Zusammenhang mit PXE.

PXE-Linux stellt einen Second Stage Boot Loader bereit, welche mit PXE zusammenarbeitet, um einen Linux-Kernel mit evtl. Optionen sowie einer Ramdisk mittels TFTP zu laden.

PXE-Linux verwendet einen etwas anderen Ansatz, um die notwendigen Kernel-Boot-Infor-mationen zu beziehen: Die Parameter werden aus einer Datei im Verzeichnis *pxelinux.cfg/* beschafft, deren Namen sich aus der hexadezimalen Repräsentation der IP-Adresse des Clients zusammensetzt, vom Server bezogen. Sollte eine solche Datei nicht existieren, versucht PXE-Linux sukzessive von rechts beginnend Zeichen zu reduzieren und erneut zu matchen. Sollte keine geeignete Datei auf diesem Wege gefunden werden, kann man generell eine Default-Konfigurationsdatei bereit stellen.

```
#pxelinux.cfg/default file
timeout 100
default linux
prompt 1
```

```
display boot.msg

F1 help

label linux
    kernel vmlinuz
    append vga=0x317 initrd=initrd splash=silent apic debug=2
    ipappend 1
label local
    localboot 0
```

Die Option *ipappend 1* sorgt dafür, dass analog zu **mknbi-linux** die IP-Konfiguration über die Kernel-Commandline übermittelt wird. Diese besitzt bis auf den Parameter für den Rechnernamen das identische Format[2], wie es auch Etherboot generiert:

```
... ip=10.8.4.102:10.8.4.1:10.8.4.254.255.255.252.0(:host02) other= ...
```

Das erlaubt innerhalb der später zu startenden Ramdisk-Umgebung eine fast einheitliche Interpretation und Weiterverarbeitung.

7.7.5 Andere Bootloader

Ein anderer Linux-Bootloader ist**chos** (Choose OS). Dieser wird nicht mehr weiterentwickelt, erfüllt jedoch immer noch die meisten Anforderungen. Konfiguriert wird **chos** über die */etc/chos.conf*; die Boot-Maps liegen, wie bei **lilo** auch unterhalb von "/boot".

Linux erfordert nicht zwingend, dass "sein" Bootloader der erste ist, der von einem Rechner gestartet wird. **lilo** und **chos** können auch als Chainloader fungieren und z.B. durch den NT-Loader oder **xfdisk** aktiviert werden.

7.8 Aufgaben

7.8.1 Kernel

1. Wozu kann es nützlich sein, den Kernel modular auszulegen?

2. In welchen Dateien wird das automatische Laden von Kernelmodulen konfiguriert? Welche ist dabei vom Systemadministrator zu ändern und welche wird automatisch beim Installieren der Kernelmodule angelegt?

3. Welche Optionen "versteht" das Kommmando **make** zur Konfiguration des Kernels? Wie passe ich am einfachsten die Konfiguration eines älteren Kernels auf meinen neuen an?

[2]Schlüsselwort "ip=" gefolgt von Client-IP, DHCP-Server-IP, Default-Gateway, Netzmaske und bei Etherboot dem Client-Hostname

4. Mit welchem Aufruf kann ich in einem frisch ausgepackten Kernelverzeichnis die Konfiguration eines älteren Kernels einfach anpassen? Welche Datei muss ich dafür kopieren?

5. Welches Tool erzeugt die Modulabhängigkeiten, wohin werden die Kernelmodule installiert? Wie lautet der entsprechende **make** Aufruf?

6. Sie haben eine alte Netzwerkkarte geerbt und wissen von dem Teil, dass es funktioniert, ein NE2000-Clone ist und PnP-konfigurierbar ist. Wie binden Sie das Teil in einen Linuxrechner ein? (Dasselbe Problem stellt sich, wenn Sie eine ISA-ISDN-Karte günstig erwerben.)

7.8.2 Booten

1. Überprüfen Sie die Bootreihenfolge im BIOS! Stellen Sie diese ein, so dass das Gerät zuerst von Floppy, Festplatte und vom CD-Laufwerk bootet!

2. Welche Möglichkeiten bestehen, um die Installationsumgebung zu booten?

3. Welche Bootloader kennen Sie? Welche Konfigurationsdateien sind für die Linuxbootloader anzupassen?

4. Erhöhen Sie die Wartezeit, bis vom Bootloader ein Betriebssystem gestartet wird!

5. Wozu brauche ich **rdev**? Brauche ich es zwingend, wenn ich einen Bootloader verwende oder kann ich mir anders behelfen?

6. Wie überprüft man, ob der Kernel sein Rootdevice (oberste Hierarchie des Filesystems) kennt, welches Rootdevice aktuell gestartet wurde und wie muss ich das Rootdevice Prompt des Bootloaders angeben, wenn es z.B. auf */dev/sda3* liegt?

Kapitel 8

Systemsicherheit

8.1 Generelle Überlegungen

Sichere Rechner bzw. Betriebssysteme kann es nicht geben. Diese werden von Menschen geschaffen und die Ideen zur Sicherheit werden meistens durch Bequemlichkeit, Unaufmerksamkeit oder Schlampigkeit wieder ausgehebelt. Weiterhin möchte kein Administrator Tag und Nacht vor einer Maschine hocken und nachsehen, was die Benutzer, Services (Dienstprogramme) und die verschiedenen Prozesse so treiben.

Generell gibt es zwei Sicherheitsbereiche, die man unterscheiden kann: Die (physikalische) Sicherheit der Maschine selbst und Angriffsmöglichkeiten aus dem Netzwerk bzw. Internet.

8.2 Sicherheit auf dem Rechner

8.2.1 Einleitung

Ein Angriff eines Hackers war bereits im ersten Schritt erfolgreich, wenn er sich unerlaubt als ganz normaler Benutzer Zugriff auf einer Maschine verschaffen konnte. Dann sind seine Rechte schon weit höher als wenn er noch von "aussen" versuchen muss, auf das Gerät einzudringen.

Deshalb sollte es immer Politik bei der Vergabe von Dateirechten geben, normalen Systembenutzern immer gerade soviele Rechte einzuräumen, wie sie zum Arbeiten benötigen. Deshalb laufen inzwischen auch viele Dienste, wie z.B. der Nameserver Bind (Daemon: **named**) oder der X-Displaymanager **gdm** unter einer eigenen BenutzerID und nicht mehr mit Rootrechten. So bedeutet ein erfolgreicher Einbruch in diese Dienste noch nicht eine automatische "Übernahme" der Maschine.

8.2.2 Passwörter

Das Passwort sollte nicht trivial sein! Worte, die in einem Lexikon zu finden sind, haben als Passwort keinen Wert. Knackprogramme benutzen genau solche Lexika, um bekannte Worte vorwärts, rückwärts und in Kombinationen durchzuprobieren.

Der Einwand "man habe keine schützenswerte Daten in seinem Account und müsse darum nicht so viel Aufwand treiben" ist vordergründig verständlich, aber dennoch nicht richtig. In einem Beispiel versteht man es besser: Möglicherweise hat ein Mieter in einem Mietshaus tatsächlich keine Wertgegenstände, die ihm schützenswert erscheinen, doch es ist nicht akzeptabel, dass er darum die Haustüre offen stehen läßt und damit das Eigentum der Mitbewohner gefährdet. Obendrein, wenn die Bude abbrennt, wird auch dieser Mieter gewahr, dass das Gebäude, sein Dach über dem Kopf, doch ein schützenswertes Objekt darstellt. Um im Bild zu bleiben, auch die Weitergabe von Schlüsseln (Passwörtern) an Dritte ist in diesem Rahmen nicht akzeptabel! Deshalb genügt es auch nicht, wenn das Rootpasswort besonders sicher ist. Bereits mit den Rechten eines normalen Benutzers hat man mehr Möglichkeiten des Einbruchs als mit gar keinem Account auf einer Maschine.

Die Konfiguration zur Passwortsicherheit (Gültigkeitsdauer von User-Passwörtern, Fehlversuche beim Login, Länge des Passwortes) werden in der Datei */etc/login.defs* eingestellt.

8.2.3 Der Admin-Account

Den Account des Administrators "root" nur so lange verwenden, wie unbedingt notwendig, z.B. für die Installation von Software, die allen Benutzern zur Verfügung gestellt werden soll. Keinesfalls als "root" die tägliche Arbeit machen. Dafür existieren die gewöhnlichen Benutzeraccounts, deren Rechte für die Standardaufgabenstellungen ausreichen. Administrative Aufgaben sollten niemals müde oder in Eile ausgeführt werden, da mit "root"-Rechten leicht irreversible Schäden angerichet werden können.

8.2.4 */etc/passwd* und */etc/shadow*

Ein System sollte das Passwort der Benutzer unbedingt nicht in */etc/passwd* speichern! Die Datei */etc/passwd* ist für die Welt lesbar und muss es auch sein, damit der eine oder andere Dienst funktioniert. Das Problem dabei ist, dass somit jeder mit einem Cracker bewaffnet auf Passwortsuche gehen kann, solange die Passwörter in dieser Datei stehen. Es leuchtet ein, dass ein Passwort, das in einer Datei steht, die nur "root" lesen kann, diese Hintertüre schließt. Also: Shadow-Passwort (in */etc/shadow*) benutzen!

8.2.5 Locken oder ausloggen

Aus demselben Grund wie zuvor, ist beim Verlassen einer Maschine darauf zu achten, dass man bei allen Konsolen ausgeloggt ist oder diese gelockt hat. Arbeiten mehrere Leute an der Konsole, dann sollte man so nett sein und die Sitzung beenden (und nicht nur den Bildschirm sperren).

8.2.6 Setuid und Verzeichnisse

Es ist darauf zu achten, dass normale Benutzer in System-Verzeichnissen wie */etc, /bin, /usr/bin, /sbin* usw. keine Schreibberechtigung besitzen. Sollten sie es doch haben, so können sie dort ein Programm unterbringen (z.B. Verschreiber zu **ls** oder **cd** (ls-al ld la ks xs vf ...)). Das Programm wartet, bis es einmal ausversehen von "root" aufgerufen wird ... und Voila! Es wird ein Programm z.B. mit dem Namen **gainr** nach */tmp, /var/tmp* oder sonst wo hin geschrieben, das nichts anderes als eine Shell mit setuid ROOT ist. Ruft man dann als normaler Benutzer **gainr** auf, so ist man plötzlich "root" - ganz ohne Passwort. Erste Konsequenz: Ein normaler Benutzer soll NUR Zugriff auf sein Home-Verzeichnis und auf */tmp* haben! Zweite Konsequenz: Die Variable PATH des Benutzers "root" muss restriktiv gehalten werden. Das aktuelle Arbeitsverzeichnis hat im PATH von "root" nichts verloren! Dritte Konsequenz: Die Liste der "setuid root" (-rwsr-xr-x) Programme regelmäßig mit der Frage nach Wachstum kontrollieren.

8.2.7 Setuid und Mounting

Mit dem Mounten ist es genauso, nur noch einfacher für den Hacker. Ein wechselbares Medium darf nie und nimmer setuid-fähig sein! Ansonsten stellt sich der Hacker zuhause ein Medium her mit einer Shell, die "root" gehört und das setuid-bit gesetzt hat (vergl. **gainr**. Dieses mountet er im Zielsystem (oder läßt es vom Admin mounten...) und ...Voila! Einfach aufrufen und fertig!

Das Mountkommando kann aus Sicherheitsgründen nur vom Superuser aufgerufen werden. Sollen normale User dazu in die Lage versetzt werden, ist die */etc/fstab* bzw. die Konfiguration des Automounters anzupassen. Dann sollte man darauf achten, dass für diese Bereiche keine Ausführungsrechte für Programme gesetzt werden (Option: noexec).

8.2.8 Browser, CGI / Java-Applet und Binaries, per Mail

Grundsätzlich sind alle Programme, die über einen grauen Kanal von außen kommen, als hoch gefährlich einzustufen. Ein Binary, das per Mail kommt und zur Ausführung gebracht wird, kann alles enthalten, wovor sich der Admin in schlimmsten Fieberträumen fürchtet. Auch ein Java-skript kann im Prinzip tun was es will, auch die Datensicherheit korrumpieren. Ein einfaches CGI-Skript einer Suchmaschine ist ein echtes Risiko. Wie in der CT 4'98 beschrieben, reicht es aus, ein Semikolon gefolgt von einem subversiven Befehl als Suchbegriff einzugeben und schon steht da anstelle eines grep-Parameters ein falsch abgesetzter **grep**-Befehl gefolgt von einem Befehl frei nach Hackers Wahl. Auf Grund der Vielzahl von Schwachstellen und Programmierfehlern sollten Web-Browser an sich für den User "root" tabu sein und nur unter der ID normaler User ausgeführt werden!!

8.2.9 Physikalischer Zugriff

Derjenige, der physikalischen Zugriff zu einer Maschine hat, ist eigentlich auch schon "root"! Das liegt daran, dass die Konsole zugreifbar ist. Um es nicht ganz so einfach zu machen, sollten bootfähige und wechselbare Laufwerke (z.B. floppy) von der Boot-Fähigkeit abgehalten werden. Der Durchschnitts-Hacker braucht aber diese Laufwerke ohnehin nicht. Dieser ist mit einer Reset-Taste oder einem Netzstecker, sowie einer Konsole vollkommen zufrieden. Der **lilo** ohne Passwortschutz erlaubt einfachste Angriffe! Abhilfe: Die Maschine unter Verschluss halten. Keiner Maschine, die nicht unter

Verschluss ist oder mit dem Internet verbunden ist, kann man vertrauen.

8.3 Datensicherheit

8.3.1 Einge Grundsätze

Die erste Hürde, die sich Angreifern stellt, ist üblicherweise der lokale Zugang auf ein System. Dazu benötigen sie meistens eine gültige UserID mit Passwort bzw. den Zugriff über einen auf der Maschine verfügbaren Netzwerkdienst, wie den Web-, NFS-, Nameserver etc. Dieser Zugriff kann durch das Ablauschen von Verbindungen, wie z.B. "telnet", "ftp", "pop3", ... erfolgen, weil diese keine verschlüsselten Passwörter verwenden. Die andere Möglichkeit ist, die Ausnutzung von Sicherheitslücken in verfügbaren Diensten.

Deshalb sollten beim Aufsetzen eines Servers immer überlegt werden, welche Dienste überhaupt notwendig sind und aus welchem Bereich der Zugriff erfolgen können soll. So wird z.B. der Webserver weltweit erreichbar sein, das Login, NFS, bzw. Samba für den Zugriff der Webadministratoren meistens jedoch nur aus einem beschränkten Netzwerkbereich erforderlich sein.

Darüberhinaus gilt: "Never touch a running system" ist out. Security-Updates müssen so bald als möglich eingespielt werden. Um immer auf dem Laufenden zu sein, empfiehlt sich das Lesen der Nachrichten von DFN-CERT. Deren Sicherheits-Infos werden auf zahlreichen Webseiten angeboten. Immer an die Backups denken! Backup-Strategien schon vor dem Systemausbau planen. Zudem dafür sorgen, dass die Backups nicht nur erstellt, sondern auch geprüft werden. Bei wichtigeren Systemen nicht nur ans Backup denken, sondern von vornherein ein Desaster-Recovery-Plan zurecht legen. Grössere Veränderungen an einer Maschine besser zuerst an einer baugleichen Maschine testen, sofern dies möglich ist. Nur die Software installieren, die auch benötigt wird. Weniger Software bedeutet weniger mögliche Sicherheitslöcher.

8.3.2 Gesicherte Verbindungen

Das Internet-Protokoll IP und die Transportprotokolle UDP und TCP bringen keinerlei Sicherheit mit. Die Pakete können auf dem Transportweg verfälscht und der Inhalt leicht abgelauscht werden. Hiergegen sind Vorkehrungen zu treffen, die an verschiedenen Stellen der Verbindung ansetzen können:

Host 1	**Sicherung**	**Host2**
Anwendung	S-HTTP, PGP, eCASH	Anwendung
Transport	SOCKS, SSL	Transport
Netzwerk	IPSec	Netzwerk
Datensicherung	PAP, CHAP, PPTP	Datensicherung

Tabelle 8.1: Absicherung auf verschiedenen Ebenen

Bekannt sind inzwischen sicherlich die Kryptosoftware zum Mailversand, PGP und der Telnetersatz SSH. Diese setzen an der Anwendungsschicht an. Die Absicherung der Transportschicht kann durch SSL (Secure Sockets Layer) erfolgen, was von verschlüsselten HTTP-Verbindungen bekannt sein sollte. IPSec setzt an der Netzwerkschicht an und soll zum Einen die Herkunft der Pakete sicherstellen und zum Anderen das Mitlesen der Daten verhindern und diese vor unerkannter Manipulation schützen.

8.3.3 Der Intenet-"Super"-Daemon (x)inetd

Viele Dienste, die nicht sehr häufig nachgefragt werden, wie z.B. "tftp", "bootp" und "ftp" müssen nicht permanent im Hintergrund laufen und die CPU, sowie den Arbeitsspeicher belasten. Sie werden auf Anforderung durch den **(x)inetd** gestartet und nach Beendigung der Verbindung wieder geschlossen.

Der **xinetd** stellt die weiterentwickelte Form des **inetd** dar und kann die oben genannte Forderung nach der Beschränkung des Zugriffs von sich aus erfüllen. In der Konfigurationsdatei des Dienstes der */etc/xinetd.conf* können Bereiche angegeben werden, von denen der Zugriff auf einen bestimmten Dienst erlaubt wird:

```
service ftp
{
        socket_type     = stream
        protocol        = tcp
        wait            = no
        user            = root
        server          = /usr/sbin/proftpd
        server_args     = -p 0
        only_from       = 172.16.0.0/16
}
```

Der **inetd** benötigt hierzu noch den TCP-Wrapper **tcpd**, welcher über die Dateien */etc/hosts.allow* und */etc/hosts.deny* gesteuert wird.

Generell gilt natürlich, dass alle Dienste, die nicht benötigt werden, auf jeden Fall deaktiviert werden sollten. Gerade diese bergen häufig die Gefahr, dass (weil man nicht an sie denkt) hierüber der erfolgreiche Angriff stattfindet.

8.3.4 xhost + und das unsichtbare Fenster

Wer "xhost +" absetzt, der serviert potentiellen Einbrechern die gewünschten Informationen auf einem goldenen Tablett! Das einzige, was der Hacker zu tun hat, ist ein unsichtbares Fenster auf dem Bildschirm zu platzieren. Dieses bekommt natürlich, weil es zu oberst liegt, alle Informationen von Maus und Keyboard exklusiv. Je nach Gutdünken des Hackers werden diese Informationen an die Fenster, an die sie eigentlich gerichtet sind, weitergeleitet oder eben auch nicht. Der Hacker kann alles mitprotokollieren oder das Keyboard/Maus toten Mann spielen lassen oder alles andere, was das Herz begehrt, mit dem Bildschirm anstellen. Ein Login-Bildschirm über Nacht auf's Display gezaubert, bringt am nächsten Morgen ein sicheres Passwort.

8.3.5 .rhosts

Es ist grundsätzlich abzulehnen, den eigenen Account gegenüber einem Benutzer zu öffnen, der behauptet, sich auf einer Maschine zu befinden, die ihrerseits behauptet, einen bestimmten Namen zu tragen. Man sieht schon jetzt wie fragil das Konstrukt ist... Jeder kann irgendetwas behaupten, aber darum ist demjenigen noch kein Account zu öffnen.

Das Konzept der R-Tools (Steuerung über die .rhosts im eigenen Home-Verzeichnis) wie sie einmal von Sun eingeführt wurden, ist in der heutigen Zeit obsolet geworden. Dazu gibt es SecureShell und -Copy und den SSH-Agent, wenn man vermeiden will, bei jedem Login auf einer anderen Maschine immer wieder ein Passwort eingeben zu müssen.

8.3.6 Überprüfung der Netzwerksicherheit eigener und anderer Rechner

Es gibt inzwischen eine ganze Reihe von Programmen, mit denen sich der Netzwerkverkehr überwachen und überprüfen lässt. Dieses sind zum Einen die Tools zum "Mitlauschen" des Datenverkehrs **tcpdump**, **ntop** und **ethereal**. Gerade das letzte Tool ist besonders zu empfehlen, da es über ein komfortables grafisches Frontend, gute Packetviewer und Filterfunktionen verfügt. **ntop** ist ein textbasiertes interaktives Programm, welches aktuelle Verbindungen und die dabei umgesetzten Datenmengen nach Verbindung und Protokollart aufzeigt. **tcpdump** arbeitet an der Kommandozeile und schneidet Pakete mit, welche an der Netzwerkschnittstelle angenommen werden. Bei allen Tools lassen sich auch Nicht-IP-Protokolle registrieren.

Die Untersuchung auf bestimmte Dienste, d.h. offene Ports von Rechnern oder in Netzwerken lässt sich mit dem Portsniffer **nmap** bewerkstelligen.

nmap -O 172.16.0.0/16	Feststellen des Betriebssystems im Class-B-Subnetz
nmap -sP 172.16.1.0/24	Ping-Scan (Erreichbarkeit von Rechnern Class-C-Subnetz
nmap -sT 172.16.1.1	Scan auf offene TCP-Ports der Maschine mit der IP 172.16.1.1
nmap -sU www.gwdg.de	Scan auf offene UDP-Ports der Maschine mit dem FQDN www.gwdg.de

Tabelle 8.2: Beispiele für den Einsatz von **nmap**

8.3.7 Firewall

Inzwischen gibt es drei Firewallimplementationen für den Linuxkernel: Das **ipfwadm** führte diese Funktionalität in den 2.0.er Kernel als ein wichtiges neues Feature ein. Die Firewalling-Fähigkeit wurde mit **ipchains** im Kernel der 2.2.er Serie stark verbessert und wird deshalb in einem eigenen Kapitel beschrieben. Der Kernel 2.4 bringt eine weiter verbesserte Firewall mit **netfilter** mit, ist aber abwärtskompatibel über Kernelmodule zu **ipfwadm** und **ipchains**.

8.4 SSH und SCP

8.4.1 Telnet- und FTP-Ersatz

Es ist immer damit zu rechnen, dass irgendwo hinter den nächsten beiden Ecken ein Hacker sein Laptop genau in das Netz-Segment eingeklinkt hat, in dem man sich befindet. Die Folge davon ist, dass alles, was man Klartext über das Netz schickt oder was man geschickt bekommt, mitgelesen werden kann. Das betrifft TCP genau so wie UDP! Der Hacker muss also nur warten, bis sich irgend ein Benutzer auf einer anderen Maschine einloggt (telnet (pine), ftp ...) und dann genau die folgenden Pakete dieser beiden Maschinen mitprotokollieren. In der Folge erhält er einen Account-Namen und ein Passwort, das dazu paßt. Abhilfe schafft man hier durch Verwendung der ssh (SecureShell) und SecureCopy scp zum Kopieren von Dateien über Rechnergrenzen hinweg. Damit baut man einen verschlüsselten Kanal zwischen zwei Rechnern auf:

```
dsuchod@server14:~ > ssh dirk@s1.goe.net
dsuchod@server14:~ > ssh -l dirk s1.goe.net
dsuchod@server14:~ > ssh -X -l dirk s1.goe.net xterm
```

Auch wenn man dann X-Anwendungen startet, benutzen diese den verschlüsselten Kanal (Hier spart man sich sogar gegenüber **telnet** das Setzen der DISPLAY-Variablen. Das Setzen von Umgebungsvariablen wird im Abschnitt zur Shell erörtert. Im letzten der Beispiele, die oben gezeigt wurden, hat man sich nicht mehr auf der Maschine eingeloggt, sondern nur noch ein Kommando abgesetzt, welches remote ausgeführt wird. Hierzu sollte mittels der Option '-X' das X11-Forward eingeschaltet werden, wenn dieses nicht defaultmäßig durch die jeweiligen Konfigurationsdateien */etc/ssh/sshd_config* für den Server bzw. */etc/ssh/ssh_config* für den Client geschieht.

8.4.2 Das Leben einfacher machen

Möchte man sich regelmässig auf verschiedenen Rechnern einloggen und nicht jedesmal sein Passwort für diese Verbindung angeben, kann man einen Schlüssel auf dem entsprechenden Rechner generieren und den Public-Teil in der Datei:
/.ssh/authorized_keys (im Home-Verzeichnis unterhalb des versteckten Verzeichnisses *.ssh*) speichern. Damit das Ganze dann auch funktioniert sind folgende Schritte notwendig:

```
maier@linux:~ $ ssh-keygen -t rsa1   # für SSH-Version 1
maier@linux:~ $ ssh-keygen -t rsa    # für SSH-Version 2
maier@linux:~ $ ssh-keygen -t dsa    # für SSH-Version 2
```

Damit werden eigene Schlüsselpaare erzeugt. (Für Protokol 2 gibt es RSA und DSA Keys, eigentlich reicht normalerweise einer von beiden.) Die Passphrase sollte nicht zu lang sein, denn sie wird doch das eine oder andere Mal abgefragt.

Es werden eine Reihe von Dateien mit den jeweiligen Schlüsselpaaren - für Protokoll 1 (RSA1) */.ssh/identity* und */.ssh/identity.pub*, für Protokoll 2 (RSA, DSA) */.ssh/id_{rsa,dsa}* und */.ssh/id_{rsa,dsa}.pub* - angelegt. Die **.pub*-Dateien werden nun auf den Zielhost kopiert und dort an die Datei */.ssh/authorized_keys* angehängt:

```
maier@linux:~ $ scp ~/.ssh/*.pub username@remote-host:~/.ssh
maier@linux:~ $ ssh username@remote-host
username@remote: ~> cat ~/.ssh/*.pub >> ~/.ssh/authorized_keys
username@remote: ~> rm ~/.ssh/*.pub
```

Viel eleganter geht es durch die Ausnutzung eines netten SSH-Features, welches im nächsten Abschnitt erklärt wird.

```
maier@linux:~ $ cat ~/.ssh/*.pub | ssh username@remote-host \
"cat >>~/.ssh/authorized_keys"
```

Ab diesem Augenblick sollte man jedoch bedenken, dass die Sicherheit aller Rechner, die in den *authorized_keys* gespeichert sind, eine größere Rolle spielt: Der Einbruch auf einer dieser Maschinen erlaubt dem Angreifer sofort ein unproblematisches Weiterverbinden auf weitere Rechner. Dieses kann durch den **ssh-agent** erschwert werden. Mit diesem Tool kann man darüberhinaus das Leben wiederum erleichtern: Der Agent fragt zu Beginn einer Sitzung die Passphrase für die gesammelten Schlüssel ab und authentifiziert jede bereits einmal aufgebaute Verbindung automatisch ohne weitere Nachfrage.

8.4.3 SSH-Tunnel und Dateitransfers

SSH kann einiges mehr als nur einen sicheren Ersatz für **telnet** und **rlogin**[1] zu bieten. Tunnel, z.B. Mail checken.

```
ssh -N -f -L 30110:mail.mydomain.example:110 user@mailserver
```

Man kann verschiedene SSH-Forwards in einer einzigen SSH-Sitzung zusammenfassen, indem man mehrfach die '-L'-Option verwendet.

Dateien über das Netz tarren:

```
tar cf - . | ssh remote-host  "cd $dir; tar xf -" # oder alternativ
tar cf - . | ssh xxx.xxx.xxx.xxx tar -C $dir -xf -
```

Über die einfache Eingabe-Pipe kann man z.B. auch:

```
cat /etc/profile |  ssh username@login2 "cd /etc; cat - >profile"
```

[1] Die sogenannten R-Tools sind inzwischen eigentlich Geschichte. **telnet** sollte man zwar nicht mehr für seine ursprünglichen Aufgaben nutzen, es aber nicht deinstallieren, da es ein nettes Werkzeug zum Testen von TCP-Services darstellt.

die Datei */etc/profile* direkt über das Netz kopieren (was natürlich sinnvollerweise mit **scp** geschieht, aber in einigen Fällen, wie z.B. dem oben genannten, ist es wegen des Anhängens an eine bestehende Datei nicht verwendbar). Man könnte auch direkt aus der Standard-Eingabe lesen:

```
cat - |  ssh username@login2 "cat - /tmp/eingabe-von-otherhost"
```

und das Ergebnis auf dem entfernten Rechner in eine Datei schreiben. Dann sollte aber keine Passwort-Abfrage dazwischenkommen.

8.5 Literatur

Einschlägige Mailing-Listen abonnieren und Webseiten untersuchen
http://internet-security.de
http://www.cert.org/
http://www.ers.ibm.com/
http://www.ccc.de/
THE DENIAL OF SERVICE PAGE Linux-Mailinglisten

Monografien gibt es inzwischen auch einige zum Thema ... z.B.:
Simson Garfinkel, Gene Spafford, *Practical Unix and Internet Security 2.Edition*, O'Reilly, 2000

8.6 Aufgaben

8.6.1 Secure Shell

1. Wie sieht das Kommando aus, wenn man sich auf eine entferne Maschine (Remote-Login) mit dem Rechnernamen *test.domain.local* verbinden will und man die Benutzer-ID "userid" auf der entfernten Maschine verwenden will und unter einer anderen ID auf der Ausgangsmaschine angemeldet ist?

2. Wie könnte der SCP-Kopierbefehl aussehen, wenn eine Datei namens *unterlagen.txt* vom Rechner *test.domain.local* unter der UserID "root" in das lokale Verzeichnis auf dem eigenen Rechner kopiert werden soll?

3. Welches Problem liegt vor, wenn ich folgende Meldung zu sehen bekomme?

```
user@linux:~/SharedFiles/kurs/klausur> ssh 10.30.4.44
8014: @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
8014: @    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
8014: @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
8014: IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
8014: Someone could be eavesdropping on you right now (man-in-the-middle attack)!
8014: It is also possible that the RSA host key has just been changed.
8014: The fingerprint for the RSA key sent by the remote host is
53:8c:4b:13:f6:df:2e:d1:e5:cf:21:12:a4:c2:ea:c5.
8014: Please contact your system administrator.
8014: Add correct host key in /home/user/.ssh/known_hosts to get rid of this message.
8014: Offending key in /home/user/.ssh/known_hosts:11
8014: RSA host key for 10.30.4.44 has changed and you have requested strict checking.
8014: Host key verification failed.
```

4. Wie kann ich mich auf anderen Maschinen per SSH unter Vermeidung der Eingabe eines Passwortes anmelden? Wann benötigt man dieses?

5. Wie tunnele ich eine POP3-Verbindung (Port 143) von meiner Maschine (z.B. mymachine.dyndns.org) auf den POP3-Server (z.B. pop3.mydomain.local), wo ich über einen SSH-Zugang verfüge?

Kapitel 9

Übersicht über wichtige Kommandos

9.1 Nachschlageteil

In diesem Teil des Skriptes werden alle (nach Meinung des Autors) mehr oder weniger wichtigen bzw. häufig benutzten Kommandos vorgestellt. Benutzer dieses Skriptes hatten vielfach nach einer solchen Liste gefragt, so dass im Folgenden versucht wird, auf diesen Wunsch einzugehen. Die Beschreibung des jeweiligen Befehls ist beileibe nicht vollständig, sondern soll nur einen ersten Anhaltspunkt liefern, wenn die Lösung zu einen Problem gesucht wird, oder unklar ist, wozu ein gegebenes Kommando gut sein soll.

9.2 Wichtige Programme in der Shell

9.2.1 Umsehen auf dem System

Kommando	**Aufgabe**
cal	zeigt Kalender für den laufenden Monat, das ganze Jahr kann durch `cal 2002` dargestellt werden
date	meldet aktuelle Systemzeit und Systemdatum, der Sysadmin kann mit diesem Kommando die Systemzeit neu setzen
finger	aktuell angemeldete Benutzer mit Herkunftsshell und gerade getätigtem Kommando, netzwerkfähig aber kaum noch anwendbar
free	zeigt Belegung des Arbeitsspeichers incl. Cache an
id	zeigt die eigene UserID und Gruppenzugehörigkeit an
last	zeigt alle zuletzt an der Maschine angemeldeten Benutzer an
uname	meldet die Art des Unix-Systems (Linux), mit `uname -a` sieht man mehr, z.B. auch die Kernelversion
uptime	meldet die Systemauslastung (drei Werte: Durchschnitt einer, fünf und fünfzehn Minuten) mit vergangener Zeit seit dem Neustart der Maschine
w	analog wie lokales **finger** mit Informationen zur Uptime
who	ähnlich wie lokales **finger**
whoami	meldet den eigenen Accountnamen

Hat man sich auf einem Linux-System über lokale Konsole oder über das Netzwerk mittels **ssh** angemeldet, kann man sich einige Standardinformationen über das System anzeigen lassen. Die Zahlendarstellung von **free** kann mit den Optionen "-b" auf Byte, "-k" auf KiloByte (das ist der Default) und "-m" auf MegaByte eingestellt werden.

```
dirk@hermes:~> free -m
             total       used       free     shared    buffers     cached
Mem:           629        535         93          0        174         97
-/+ buffers/cache:        264        365
Swap:          516         24        492
```

Die Kommandos **finger**, **w** und **who** liefern einen recht ähnlichen Output (in unterschiedlichem Format) über gerade am System angemeldete Benutzer.

```
dirk@dozent:~/text/lak> w
 12:50pm  up 34 days, 14:57,  1 user,  load average: 0.02, 0.04, 0.00
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
root     tty1     -                17Feb03 34days  0.43s  0.32s  -bash
dirk@randy2:~/text/lak> who
root     tty1     Feb 17 21:57
dirk@randy2:~/text/lak> finger
Login      Name                 Tty      Idle  Login Time   Where
root       root                 1         34d  Feb 17 21:57
```

Als die Welt des Internets noch in Ordnung war :-), konnte man sich mittels **finger @rechnername.domain** alle auf entfernten Rechnern eingeloggten Benutzer anzeigen lassen. Der Finger-Daemon war nicht, wie heute üblich, standardmäßig abgeschaltet. Das Kommando `last | less` wirft einen Blick in die Geschichte der angemeldeten Benutzer. Da diese Liste durchaus länger als ein Bildschirm werden kann, schickt man die Ausgabe von **last** gleich an einen Pager, in diesem Fall an **less**.

```
dirk@test2:~/text/lak> last|less
root     pts/7        pd9e29571.dip.t- Sat Mar 22 16:45 - 22:29  (05:44)
dirk     pts/9        pd9e29571.dip.t- Sat Mar 22 16:27 - 22:29  (06:02)
dirk     pts/7        pd9e295cb.dip.t- Sat Mar 22 12:18 - 16:40  (04:22)
root     pts/7        java1.ruf.uni-fr Fri Mar 21 16:13 - 16:17  (00:04)
root     pts/7        java1.ruf.uni-fr Fri Mar 21 15:58 - 15:59  (00:00)
dirk     pts/7        dozent.lp2.ruf.u Thu Mar 20 10:49 - 13:00  (02:11)
dirk     :0                            Wed Mar 19 11:21   gone no logout
dirk     pts/0        suenne.stud.uni- Fri Mar 14 15:12 - 16:57  (01:44)
dirk     tty2                          Mon Mar 10 19:42 - 19:42  (00:00)
dirk     :0                            Mon Mar 10 14:51 - 13:32 (2+22:40)
dirk     pts/11       np9.ruf.uni-frei Wed Mar  5 19:30 - 19:30  (00:00)
dirk     pts/8        p5083ffa6.dip.t- Sun Mar  2 16:42 - 22:54  (06:11)
dirk     pts/0        acbd9832.ipt.aol Sat Feb 22 19:10 - 21:23  (02:13)
dirk     pts/0        acbd7ced.ipt.aol Thu Feb 20 00:04 - 02:16  (02:12)
root     pts/2        randy3.ruf.uni-f Wed Feb 19 10:51 - 11:08  (00:16)
dsuchod  pts/2        randy2.rz.uni-fr Tue Feb 18 13:29 - 13:30  (00:00)
```

```
dirk     :0                            Mon Feb 17 21:59 - 14:51(20+16:52)
root     tty1                          Mon Feb 17 21:57   still logged in
reboot   system boot  2.4.19-4GB       Mon Feb 17 21:55        (34+14:59)
dirk     pts/14                        Mon Feb 17 20:51 - down   (01:02)
dirk     pts/13                        Mon Feb 17 19:36 - down   (02:18)
dsuchod  pts/13       localhost        Mon Feb 17 19:31 - 19:33  (00:01)
dirk     pts/12                        Mon Feb 17 19:31 - down   (02:23)
[...]
dirk     :0           console          Mon Jan  6 19:38 - 11:21 (7+15:43)
reboot   system boot  2.4.19-4GB       Mon Jan  6 19:37        (20+21:34)

wtmp begins Mon Jan  6 18:01:01 2003
```

Auch hier finden sich viele der Informationen wieder, welche die oben gezeigten Kommandos generieren. Die Daten werden in zwei Dateien hinterlegt: */var/run/utmp* liegen die kurzfristigen Informationen zu gerade angemeldeten Benutzern und in */var/log/wtmp* liegen die Daten, die z.B. **last** auswertet.

9.2.2 Shelleigene Standardkommandos

Kommando	**Aufgabe**
echo	Ausgabe von Variableninhalten, z.B. `echo $PATH` meldet die aktuelle Liste des Suchpfades für ausführbare Programme
export	zeigt alle definierten Variablen an, die an Subshells weitergereicht werden, Untermenge von **set**
help	liefert eine Liste der in der aktuellen Shell zur Verfügung stehenden, eingebauten Kommandos
history	gibt eine Liste der am Prompt abgesetzten Kommandos an
pwd	meldet das aktuelle Arbeitsverzeichnis
set	zeigt alle in der aktuellen Shell definierten Variablen an
which	meldet die Lage von ausführbaren Programmen. Dazu wird im Bereich der in "PATH" angebenen Verzeichnisse gesucht

9.2.3 Shelleigene Strukturen und Schleifen

Die Unix-Shell kann durchaus mit einer Interpreter-Programmiersprache, wie Perl oder Python verglichen werden. Sie kennt Variablen, Kontrollstrukturen und Schleifen, die bereits fest eingebaut sind.

Kommando	**Aufgabe**
break	bricht eine Schleife ab
case	Switchanweisung in der Shell
continue	bricht nur den aktuellen Schleifendurchlauf ab
declare	erlaubt Nicht-String-Variable in der Shell zu definieren, z.B. `declare -i n` erzeugt die Ganzzahlvariable *n*
echo	Ausgabe von Textzeilen, z.B. `echo -n “Das ist ein Test!”` Das “-n” schaltet das sonst obligatorische Newline ab
for	einfache Schleife zur Abarbeitung von Listen, z.B. `for i in * ; do echo $i; done` ist ein einfacher Ersatz für **ls**
else	Alternative beim Nichtzutreffen von **if**
exit	Rückgabewert einer Funktion oder des Shellskriptes
if	Bedingung, z.B. `if [ $i -gt 10 ]; then echo “i ist größer als 10”; else “i ist kleiner oder gleich 10”`
return	beendet eine Subroutine und kann einen Wert zwischen *0* und *255* zurückliefern
while	einfache Schleife, die auf eine Abbruchbedingung prüft. `while [ $i -lt 100 ]; do echo $i; i=$i+1; done` läuft solange die Variable *i* kleiner als *100* ist

9.2.4 Operationen auf Dateien

Kommando	**Aufgabe**
cp	Kopieren von Dateien und Verzeichnissen (Quelle bleibt erhalten)
chmod	ändert Zugriffsrechte: Schreiben, Lesen, Ausführen bzw. Directorylisting auf Dateien oder Verzeichnissen
chown	ändert die Besitzer einer Datei (Benutzer und Gruppe)
file	ermittelt die Art einer Datei (Text, Bibliothek, ausführbares Programm, PNG-Bild, TeX, ...)
find	Suchen nach Dateien, z.B. alle Dateien unterhalb von */bin* anzeigen, die größer als 100 kByte sind: `find “/bin” -size +100k`
ln	Anlegen von Links (eine Art Verweis) auf Dateien, z.B. mit `ln -s quelle ziel` wird ein Verweis von “ziel” auf “quelle” angelegt
locate	Arbeitet erst nachdem mittels **updatedb** eine Hashtable mit allen Dateien erzeugt wurde. bf updatedb bedient sich dabei wiederum von **find** ...
ls	Anzeigen von Verzeichnisinhalten
mkdir	legt Verzeichnisse an
mv	verschiebt/umbenennt Dateien, wobei die Quelle anschließend nicht mehr verfügbar ist
rm	löscht Dateien oder Verzeichnisse
rmdir	löscht leere Verzeichnisse. Nichtleere Verzeichnisse lassen sich mittels `rm -r` entsorgen, wobei gerade als Sysadmin etwas Vorsicht geboten ist :-))
touch	legt eine leere Datei an oder trägt die aktuelle Zeit auf eine Datei ein `touch datei`

Das Kommando **ls** kennt die Option "-h", um eine besser lesbare Darstellung in Mega-, GigaByte-Angaben zu erhalten.

9.2.5 Verzeichnisstruktur und Filesysteme

Kommando	Aufgabe
df	Diskfree zeigt freien Speicherplatz auf verschiedenen Datenträgern nach Mountpoints sortiert
du	Diskusage 'Speicherbelegung in einem (Unter-)verzeichnis
eject	Auswerfen (und Einziehen) von "Removable Devices", z.B. `eject /dev/hdd` wirft das Medium aus, welches am Secondary Slave IDE hängt
mount	Einhängen von neuen Partitionen, Netzwerkshares etc. in das Dateisystem der Maschine
mkfs	Formatieren einer Festplattenpartition
tune2fs	Einstellungen für EXT2/3-basierte Dateisysteme, z.B. Häufigkeit von erzwungenen Dateisystemüberprüfung beim Mounten
umount	Aushängen der mit **mount** eingehangene Dateisystemteile

Die Kommandos **df** und **du** kennen analog zu **ls** die Option "-h" für die besser lesbare Zahlendarstellung.

9.2.6 Texteditoren

Texteditoren gibt es für Linux richtig viele. Einige davon sind etliche Jahre alt, andere sind erst in jüngster Zeit programmiert worden. Wenn es um die Administration von Systemen geht, benötigt man, gerade wenn man sich entfernt über das Netzwerk eingeloggt hat, Editoren, die in der Kommandozeile arbeiten. **joe** ist ein einfacher word-star-kompatibler Editor. Die Hilfe innerhalb des Editors kann mit [STRG]-KH ein und ausgeschaltet werden. Der Editor **pico** ist noch etwas rudimentärer; er wird mit dem textbasierten Mailprogramm **pine** mitgeliefert wird. Er eignet sich für Benutzer, die mit dem besagten Mailprogramm arbeiten. Seine Beliebtheit hat inzwischen für eine Weiterentwicklung gesorgt; unter dem Namen **nano** kann er aufgerufen werden (wenn die Weiterentwicklung auf dem System installiert ist. **vi** kann man als den Standard-Unix-Editor bezeichnen. Ihn findet man auf allen klassischen Unixsystemen vor. Nichtbenutzer sollten zumindest wissen, wie man ihn wieder verläßt :-)) (mit [ESC],dann [:] und dann [q].

Unter den diversen grafischen Benutzeroberflächen existieren jeweils etliche Editoren mit mehr oder weniger gelungener Benutzerschnittstelle. Mit **xemacs** und **gvim** stehen jeweils grafisch orientierte Vertreter ihren textorientierten Vorlagen gegenüber. KDE kennt daneben weitere, wie **kedit**, **kate**, **kwrite**. Ähnlich sieht es unter Gnome aus. Andere Editoren verwenden wiederum andere Grafikbibliotheken, so dass sich jeder Anwender den passenden Editor heraussuchen können sollte.

Die Bedienung der einzelnen Editoren weicht dabei stark voneinander ab, wobei die Ähnlichkeit jeweils zwischen der grafischen und der Textausgabe eines Editors, wie **vi** am größten sind. Etliche Editoren eignen sich gut für die Programmierung in den diversen Sprachen, der Shell oder LaTeX, da sie über ein gutes Syntax-Highlighting verfügen.

9.2.7 Operation auf Textdateien

In vielen Fällen will man für regelmäßig auftretende Aufgaben oder für die gleichzeitige Anwendung auf viele Dateien, nicht jedesmal einen interaktiven Editor öffnen müssen. Für diese Tasks bietet sich die Shellprogrammierung gemeinsam mit den nachstehend genannten Standard-Tools für die Bearbeitung von Textdateien an.

Viele Kommandos lassen sich dabei mittels Pipes ("|") hintereinanderschalten. Gute Beispiele für die verschiedenen Situationen und deren Beherrschung bieten z.B. die Runlevel-Skripte unterhalb von */etc/init.d* oder die Programme zur Steuerung der grafischen Benutzeroberfläche unterhalb von */etc/X11/xdm*.

<table>
<tr><th>Kommando</th><th>Aufgabe</th></tr>
<tr><td>awk</td><td>Eigene Programmiersprache für zeilenorientierte Textbearbeitung, z.B. zum Auftrennen von Zeilen: <code>cat /etc/passwd|awk -F : '{print $1 " " $2 " " $3 " " $4}'</code></td></tr>
<tr><td>cat</td><td>Einfache Ausgabe von Textdateien ohne interaktive Steuerungsmöglichkeit, deshalb besser zur Shellprogrammierung geeignet</td></tr>
<tr><td>grep</td><td>Durchsuchen von Zeichenketten nach Mustern auch Regular Expressions</td></tr>
<tr><td>less</td><td>Komfortabler Pager zum interaktiven Blättern innerhalb von Textdateien. Wird auch in Verbindung mit man eingesetzt. Verlassen deshalb in jedem Fall mit der Taste "q" (statt [CTRL]-C)</td></tr>
<tr><td>more</td><td>Einfacher Bruder von less mit weniger Möglichkeiten</td></tr>
<tr><td>sed</td><td>Streameditor zum zeilenweisen Bearbeiten von Strings, z.B. zum Austauschen des Trennzeichens ":" gegen ein Leerzeichen: <code>cat /etc/passwd|sed -e "s,:, ,g"</code></td></tr>
<tr><td>sort</td><td>Sortieren von Ausgaben eines Kommandos oder einer Datei</td></tr>
<tr><td>wc</td><td>WordCount zum Zählen von Zeilen und Wörtern</td></tr>
</table>

9.2.8 Textsatzsystem und Darstellung

Dieses Skript wurde mit Hilfe von LaTeX erstellt, welches einen ziemlich anderen Ansatz als bekannte Textverarbeitungsprogramme wählt. Der zu formatierende Text wird in einem einfachen Texteditor nach Wahl erstellt (am besten emacs) und mit Formatanweisungen versehen. Dieses Konzept kann vielleicht mit HTML verglichen werden. Anschliessend wird dieser "Rohtext" mit dem Kommando **latex** "kompiliert" und in ein Device-unabhängiges[1] Format umgewandelt. Die ".dvi"-Dateien kann man sich mit DVI-Viewern (z.B. xdvi) ansehen. Die im Folgenden genannten Befelhe gelten natürlich für jede Datei unabhängig vom Textsatzsystem LaTeX.

[1] device independent

Kommando	Aufgabe
dvips	Umwandlung von DVI-Dateien nach Postskript
latex	Erweiterter Textsatzkompiler, der auf Donald Knuth' TeX basiert und DVI-Dateien generiert
makeindex	Generieren einer Indexdatei, welche nach erneutem Lauf in die von **latex** generierte DVI-Datei eingebunden wird
pdf2ps	Umwandlung von PDF zu Postskript, wie es zum Drucken notwendig werden kann
ps2pdf	Umwandlung von Postskript in PDF, nutzt jedoch bei weitem die Fähigkeiten von PDF nicht aus
xdvi	zum Betrachten einer .dvi Datei
gv	zum Betrachten einer .ps oder .pdf Datei

Standardmäßig nehmen alle Programme immer die Inputdatei als Argument. Wenn nichts anderes spezifiziert wird, behält die Outputdatei den Originalnamen bei, nur die Endung wird entsprechend neu gesetzt, z.B.

```
dirk@hermes:~tex/lak> latex lak.tex
dirk@hermes:~tex/lak> makeindex lak.idx
dirk@hermes:~tex/lak> latex lak.tex
dirk@hermes:~tex/lak> dvips lak.dvi
dirk@hermes:~tex/lak> ps2pdf lak.ps
```

9.3 Systemprogramme

9.3.1 Prozess-Steuerung, Runlevel

Kommando	Aufgabe
insserv	SuSE-spezifisch zum Ein- und Austragen von Runlevelskripten
free	zeigt freien Systemspeicher an. Da Caches für Plattenblöcke und der Verzeichnisstruktur angelegt werden, scheint dieser meistens lächerlich niedrig, was aber selten ein ernstes Problem darstellt
kill	Abschiessen von Prozessen über ihre PID
killall	Abschiessen von Prozessen nach ihrem Namen, z.B. `killall mozilla`
lsof	Anzeigen geöffneter Filehandles
netstat	Anzeigen offener Netzwerk- und Dateisystemsockets
nice	lässt ein Programm mit veränderter Priorität laufen
ps	Anzeigen der Liste aktuell laufender Prozesse
pstree	Anzeigen der Liste der von einem Prozess ausgehenden Kindprozesse
top	Interaktives Tool für das Prozessmanagement

9.3.2 Dämonen

Dämonen sind im Hintergrund laufende Prozesse, die bestimmte Standardaufgaben des Systems übernehmen und überlicherweise nicht vom Administrator aus einer Shell heraus, sondern durch meistens gleichbenannte Runlevel-Skripte gestartet werden. Ihre dämonische Eigenschaft wird meistens durch das an den Protokollnamen angehängte *d* kenntlich gemacht. So wird aus dem Server für den Dienst "DHCP" der **dhcpd**. Das muss aber nicht immer so sein, wie das Beispiel "DNS" zeigt, wo der Dämon **named** heißt.

Kommando	**Aufgabe / Beschreibung**	**Konfiguration / Datenbereich**
atftpd	Erweiterter Trivial-FTP-Server, der auch Anfragen von PXE-Clients bedienen kann	keine direkte, mittels */etc/sysconfig/atftpd*
dhcpd	Der Server für dasDynamic Host Configuration Protocol	*/etc/dhcpd.conf*, */var/lib/dhcp/...*
httpd	Der Webserver. Meistens kommt hierbei der "Apache" zum Einsatz	*/etc/httpd/...* , aber stark distributionsabhängig
inetd	Internet-Super-Daemon - stößt andere auf der jeweiligen Maschine eher seltener benutzte Dienste, wie **telnetd**, **in.ftpd**, **in.tftpd**, ... an und öffnet hierfür die entsprechenden Netzwerkports	*/etc/inetd*, evtl. haben einzelne Dienste Datenverzeichnisse
mysqld	Der Server für die MySQL-Datenbank	*/var/lib/mysql/...*
named	Serverprozess des Domain Name Systems	*/etc/named.conf*, */var/(lib/)named/...*
nmbd	Server für das Windows-Name-System (WINS); kann mit dem DNS zusammenarbeiten und gehört zur Samba-Suite gemeinsam mit dem **smbd**	*/etc/samba/...*, */var/lib/samba/...*
proftpd	Einer der möglichen FTP-Server	*/etc/proftpd.conf*, stark distributionsabhängig
rpc.mountd	RPC-Dienst für das Network File System, arbeitet mit dem **rpc.nfsd** und evtl. einem geeigneten **lockd** zusammen	*/etc/exports*
rpc.nfsd	RPC-Dienst für das Network File System	*/etc/exports*
slapd	LDAP-Server	*/etc/openldap/slapd.conf*, */var/lib/(open)ldap*
smbd	Samba-Server für SMB-Dienste in Windowsnetzwerken	*/etc/samba/...*, */var/lib/samba/...*
wuftpd	Ein anderer FTP-Server	*/etc/wuftpd.conf*, stark distributionsabhängig
xinetd	Die erweiterte Ausgabe des Internet-Super-Daemon mit besserer (Netzwerk-) Zugriffskontrolle	*/etc/xinetd.conf*, analog zum **inetd**

Die Tabelle kann nur eine unvollständige Auflistung einiger häufig zu findender Serverprogramme bieten. Die oben genannten findet man relativ häufig in der Prozessliste von Servermaschinen.

9.4 Nützliche Tools

9.4.1 Packprogramme

Möchte man viele zusammenhängende Dateien kopieren oder weitergeben, sollte man sie als eine einzige Datei behandeln können. Mit dem Kommando **tar** (Tape Archiver) können Dateien oder ganze Verzeichnisse zu einem großen Block zusammengefasst werden. Am Zielort können die in den Block eingepackten Dateien dann wieder ausgepackt werden. Auch die Verzeichnisstruktur und die Zugriffsrechte werden gespeichert, d.h. die Dateien liegen nach dem Auspacken in denselben Verzeichnissen mit ihren ursprünglichen Rechten. Mit **tar** wird noch kein Platz gespart. Die TAR-Datei lässt sich jedoch mit einem Komprimierungsprogramm in ihrem Umfang reduzieren.

Kommando	**Aufgabe / Beschreibung**
compress	Altes Unix-Packprogramm mit historischen Kompressionsraten
bzip2	Pack- und Entpackprogramm mit den derzeit besten Kompressionsraten für Standarddateien, kommandozeilenkompatibel zu **gzip**
gzip	Das Standardpack- und Entpackprogramm, welches auch kompatibel ist zu WinZIP und Konsorten
tar	Tape Archiver, Komprimiert nicht, aber darf in diesem Zusammenhang nicht fehlen, da die anderen Packprogramme zum Teil nicht auf Listen von Dateien und Verzeichnissen losgelassen werden können
unarj	Auspacken von ARJ-gepackten Dateien (ARJ war längere Zeit ein Standard unter DOS)
unzip	Auspacken von PKZIP-Archiven
unrar	Auspacken von RAR-gepackten Archiven (ein Standardprogramm unter Windows)

Das Programm **gzip** sollte auf jedem UNIX-System vorhanden sein. Eine komprimierte Datei erhält die Endung ".gz"; ein komprimiertes Tarfile heißt also z.B. *tarfile.tar.gz*. Leistungsfähiger ist das ebenfalls weit verbreitete Programm **bzip2**. Mit **bzip2** komprimierte Dateien erhalten die Endung ".bz2". Die Dekomprimierung erfolgt entsprechend des Packers mit `gzip -d` oder **gunzip** bzw. mit `bzip2 -d`.

Die Optionen von **tar** müssen nicht wie üblich von einem "-" eingeleitet werden. **tar** kennt drei Modi: Einpacken, Auspacken und Inhalt anzeigen. Den Modus wählt man mit einer der folgenden Optionen aus: "c" (create - Erstellen/Einpacken eines tarfiles), "x" (extract - Auspacken eines tarfiles, wobei vorhandene gleichnamige Dateien überschrieben werden) und "t" (type - zeigt den Inhalt eines tarfiles an, ohne dieses auszupacken). **tar** wurde früher vor allem zur Datensicherung (Backup) auf Magnetbändern eingesetzt, woher der Name rührt. Will man das Archiv in einer Datei speichern oder eine Archivdatei auspacken, muss man die Option "f" verwenden und den Dateinamen dahinter angeben. Zusatzinformationen über die Aktivitäten des Programms liefert die Option “v”. Der Tape Archiver kennt die Möglichkeit, das Packprogramm aus sich heraus zu invozieren, dieses geschieht mit den Schaltern “z” oder “i” für **gzip** und “j” für **bzip2**.

9.4.2 Zugriff auf (DOS)-Disketten

Will man Dateien oder Programme auf Disketten abspeichern und transportieren, kann man die “mtools” verwenden. Diese stellen einen eingeschränkten Befehlssatz zum Arbeiten mit MS-DOS-formatierten Disketten zur Verfügung, daher das vorangestellte “m” vor den Kommandonamen. Ein wichtiger Unterschied zu DOS besteht im Verzeichnisdelimiter: Da der Backslash (”\”) in der Unix-Shell eine besondere Bedeutung hat, wird er durch den “/” ersetzt. Ähnliches kennt man vielleicht auch von der Benutzung der Samba-Tools.

MTool	**Funktion**	**Beispiel**
mcopy	Kopieren vom Laufwerk in ein UNIX-Verzeichnis oder umgekehrt	`mcopy lak.ps a:` kopiert die Datei lak.ps auf Diskette.
mdel	Löschen einer DOS-Datei	`mdel a:lak.pdf` löscht die Datei *lak.pdf* von der Diskette.
mdir	Anzeigen eines DOS-Verzeichnes	`mdir a:/psfiles` zeigt den Inhalt von *a:psfiles* an.
mmd	Anlegen eines DOS-Verzeichnisses auf der Diskette	`mmd a:/testdir`
mrd	Löschen eines (leeren) DOS-Verzeichnisses	`mrd a:/testdir`
mren	Umbenennen eines (existierenden) DOS-Verzeichnisses.	`mrd a:/testdir a:/dirtest`
mtype	Zeigt den Inhalt einer DOS-(Text-)Datei an.	`mtype a:/test.txt`

9.4.3 Netzwerk

Da der Zugriff auf das Netzwerk durchaus als sensibel eingestuft wird, lassen sich die meisten der nachfolgenden Kommandos nur sinnvoll mit Root-Rechten ausführen.

Kommando	Aufgabe / Beschreibung
arp	Programm zur Anzeige und Manipulation der Kernel-Arp-Tabelle
dig	Abfrage von Nameserverinformationen, Nachfolger von bf nslookup
ifconfig	Anzeigen der Konfiguration von Netzwerkinterfaces, genügt für die meisten anfallenden Standardaufgaben
ifuser	Über welches Interface verläßt ein Datenpaket an eine bestimmte IP-Adresse den Rechner
ip	Nachfolger von **ifconfig** mit erweiterten Fähigkeiten aber deutlich abweichender Bedienung. Übernimmt auch die Aufgaben von **route**
nslookup	Traditionelles Programm zum Test von Nameservern
ping	Einfaches Programm zum Test auf Erreichbarkeit von Rechnern
route	Manipulieren der nicht automatisch mit **ifconfig** angelegten Netzwerkrouten
tc	Traffic Control. Mit diesem Programm lassen sich Queues für das Management und die Priorisierung von Netzwerkdatenströmen einstellen
whois	Programm zur Anzeige des Personen- oder Firmennamens der zu einer IP oder Domäne gehört

9.4.4 Netzwerküberwachung

Kommando	Aufgabe / Beschreibung
arpwatch	Meldet neu im Netzwerk auftauchende MAC-Adressen. Eignet sich für die einfache Überwachung von Ethernets auf ungeschickt ins lokale Subnetz gehängte Maschinen
ethereal	Sehr komfortables Analysewerkzeug für Netzwerkdatenströme mit komfortabler grafischer Benutzeroberfläche
fping	Testen auf das Antwortverhalten von Rechnergruppen. Besonders für die Verwendung in Skripten geeignet
nmap	ist ein mächtiger Portscanner zum Aufspüren von offenen Netzwerkports auf (entfernten) Maschinen
netstat	Zeigt offene Netzwerkverbindungen und Ports an
ntop	Komfortables textorientiertes Traffic-Analyse-Tool mit weitgehenden Auswertungsmöglichkeiten
tcpdump	Einfacher Packet-Sniffer für die Kommandozeile, der die Paket-Header anzeigt
telnet	Einfache Überprüfung auf Erreichbarkeit von TCP-basierten Diensten, z.B. `telnet localhost 80` testet, ob der Webserver antwortet

Beispiel: `tcpdump -i eth1 host 1.2.3.4 and not port ssh`
Mit einer ganzen Reihe von TCP-basierten Netzwerkdiensten, kann man sich mittels **telnet** interaktiv “unterhalten” und diese so auf Erreichbarkeit überprüfen. Im folgenden Beispiel wird die “Unterhaltung” mit einem Pop3-Server dargestellt.

```
dirk@hermes:/home/dirk/SharedFiles/tex/lak> telnet pop3.test.local 110
```

```
Trying 10.16.20.86...
Connected to pop3.test.local.
Escape character is '^]'.
+OK ready  <7889.1060520932@pop3.test.local>
user dirk
+OK Password required for dirk.
pass geheim
+OK dirk has 117 visible messages (1 hidden) in 2844396 octets.
quit
+OK Pop server at pop3.test.local signing off.
Connection closed by foreign host.
```

Dieses Beispiel zeigt die "Unterhaltung" mit einem Pop3-Server.

9.5 Grafische Oberflächen

9.5.1 X-Programme

Es wird kaum möglich sein, alle Programme, die unter der grafischen Oberfläche des X11 laufen aufzuzählen. Deshalb folgen einige wichtige Standardprogramme, die nicht direkt nur einer der beiden wichtigen grafischen Benutzeroberflächen, wie KDE oder GNOME zuzuordnen sind.

Kommando	**Aufgabe / Beschreibung**
Xnest	Eingebetteter X-Server, der als normales Fenster unter X11 erscheint
Xvnc	Spezieller X-Server für VNC
acroread	Der offizielle Adobe PDF-Viewer, nicht besonders schön, da Motif-GUI, aber als Plugin für die diversen Browser ebenfalls verfügbar
mozilla	Der Netscape-Nachfolger und Open-Source-Browser, der fast alles kann, was ein Standardbrowser hinbekommen sollte
gv	Ghostview - Standardbetrachter für Postskript-Dateien unter Linux
netscape(6)	Der alte bzw. neue "kommerzielle" Browser von Netscape. Netscape 6 setzt stark auf Mozilla auf und bastelt Zusatztools hinzu, auf die man zum Teil gut verzichten kann
openoffice	Die Office-Suite von Sun Microsystems, die sich durchaus mit der Redmonter Konkurrenz in etlichen Punkten messen kann. Das Programm sieht vom Look&Feel noch sehr windows-like aus, es verwendet Fenster in Fenstern; in der Form X11-unüblich
xdvi	Ein alter direkt X11-basierter DVI-File Viewer

9.5.2 GNOME

Kommando	Aufgabe / Beschreibung
abiword	Textverarbeitung des Gnome-Projektes, welche Dateien auch im TEX-Format speichern kann
dia	Einfaches Malprogramm für Ablaufdiagramme mit etlichen Postscript-Bildern für Netzwerkgrafiken etc.
evolution	Ein Mail- und Kalenderwerkzeug; ähnlich wie M$-Outlook
galeon	Auf der Gecko-Engine basierender, schlanker Webbrowser
gimp	Mächtiges Grafikwerkzeug mit Funktionalität ähnlich Photoshop
ggv	Der Gnome-Betrachter für Postskript-Dateien
gnumeric	Tabellenkalkulation des Gnome-Projektes
nautilus	Ein Filemanager

9.5.3 KDE

Das KDE-Projekt (inzwischen in der dritten Version) widmet sich seit Ende der 90er Jahre der Entwicklung einer einheitlichen ansprechenden Benutzeroberfläche. Im folgenden werden einige wichtige Programme aufgelistet, wobei sie jedoch nur einen winzigen Ausschnitt der verfügbaren Werkzeuge repräsentieren.

Kommando	Aufgabe / Beschreibung
k3b	CD-Erstell-, Kopier- und Brennprogramm
kaddressbook	Wie der Name schon sagt ...
kbear	Downloadmanager
kdvi	Betrachter für DVI-Files, die z.B. durch TEX erzeugt werden
kghostview	Ghostscript-Frontend zum Ansehen von Postskript und PDF-Dokumenten
kmail	Standard-Email-Programm unter KDE, welches alle wichtigen Funktionen einer Mailapplikation beherrscht
koncd	CD-Erstell-, Kopier- und Brennprogramm
konqueror	mächtiger Datei- und Webbrowser mit vielen eingebauten Zusatzfunktionen
kspread	Tabellenkalkulation der K-Office-Suite
kword	Textverarbeitungsprogramm aus der K-Office-Suite
kwrite	KDE-Texteditor
pixie	Bildbetrachter mit Thumbnail-Funktion etc.

9.6 Software

9.6.1 Installation und Management

Das Standardporgramm zum Paketmanagement ist üblicherweise der RedHat-Packet-Manager **rpm**. Es gibt andere Ansätze, die bei Debian mit **dpkg** oder **aptget** verfolgt werden. Gentoo hat ebenfalls eigene Tools für diese Tasks am Start.

9.6.2 Entwicklung

Zur Programmentwicklung unter Linux gehören eine ganze Reihe von Werkzeugen. Die Editoren wurden gesondert abgehandelt. Es gibt integrierte Entwicklungsumgebungen, z.B. **kdevelop**, die viele der genannten Programme unter einer einheitlichen Oberfläche zusammenfassen.

Kommando	Aufgabe / Beschreibung
autoconf	Zusammen mit **automake**, um Software zur Kompilation auf einer Zielplattform geeignet einzurichten
automake	Generieren von Makefiles für die Zielplattform. Wenn man `./configure --prefix=/usr --nextoption ...` aufruft, steckt genau dieses dahinter
cvs	Concurrent Versions System (CVS) ist ein Tool zur Versionskontrolle
diff	Erstellt eine Differenzdatei von zwei unterschiedlichen Dateien oder Dateibäumen
gcc	Der GNU C-Compiler. Eines der wichtigsten Programme überhaupt
ld	Der Linker (dynamisches Binden von Programmen und Bibliotheken)
ldconfig	Einrichten des schnellen Hashes */etc/ld.so.cache* für das schnelle Auffinden dynamisch gelinkter Bibliotheken. Es wird normalerweise nach dem Systemstart automatisch aufgerufen, sollte aber auch nach der Installation neuer Bibliotheken angestoßen werden
ltrace	Verfolgen von Bibliotheksaufrufen zum Debugging von Programmen
make	Aufwändige spezielle Skriptsprache zur Analyse der Makefiles um bequem ganze Softwarepakete zu bauen
nm	zeigt in einem Programm oder einer Bibliothek enthaltene Funktionen an
patch	fügt durch **diff** erstellte Differenzdateien in eine Datei oder ein Verzeichnis ein
strace	Verfolgen von Systemaufrufen zum Debugging von Programmen, analog zu **ltrace**
strings	zeigt in einem Programm oder einer Bibliothek enthaltene Textstrings an

Kapitel 10

Installation

10.1 Einsatzbereiche

Linux ist in einem weiten Anwendungsbereich einsetzbar. Je nach Anwendungsziel und Einsatzgebiet stellt es unterschiedliche Forderungen an Hardwareausstattung und Festplattenplatz.

10.1.1 Router/Gateway

Die mit dem Kernel mitgelieferte umfangreiche Firewall und Routingfunktionalität macht einen Linuxrechner zu einem geeigneten Router für kleinere und mittlere Netzwerke. Darüberhinaus können solche Systeme ausgefeilte Sicherheitskonzepte realisieren.

Je nach Anwendungsbereich ist die Hardware zu dimensionieren: Ein älterer 586er mit 32MB RAM und 1 GByte Festplattenplatz reicht für ein kleines Firmen- oder Privatnetz aus, um mehrere Rechner hinter einer maskierenden Firewall an einem Modem-, ISDN- oder ADSL-Anschluss als Internetgateway zu betreiben. Dieser Rechner kann dabei gleichzeitig noch Aufgaben als DNS- und DHCP-Server übernehmen. Möchte man ein ausführliches Logging einschalten, so sollte eine eigene Festplatte oder ein Plattenbereich dafür vorgesehen werden. Ergänzt werden kann ein solches System durch einen Web- und FTP-Proxy, wie z.B. **squid**, um den Webzugriff zu beschleunigen und die eventuell nicht sehr schnelle Internet-Anbindung von Traffic zu entlasten. Filter auf Viren in Mails und Werbebannern lassen sich ebenso auf einer solchen Maschine unterbringen, womit jedoch die Hardwareanforderungen steigen.

10.1.2 Desktopsystem

Linux verwandelt einen PC in eine leistungsfähige Workstation. Als Arbeitsumgebung für inzwischen einige Officelösungen oder auch für SAP und als Entwicklungsumgebung zum Schreiben von Software ist Linux hervorragend geeignet. Beispiele: Netscape oder Mozilla, Star-Office oder Applixware und verschiedene Java-Implementationen. Die beiden integrierten graphischen Desktops KDE und GNOME bieten inzwischen einen sehr hohen Komfort der Arbeitsoberfläche. Je nach eingesetzter Software liegen die CPU und RAM-Anforderungen bei mindestens 300 Mhz für ein Pentium II-System und 128 - 256 MByte. Als Festplattenplatz sollte man 4 - 10 GByte nicht mehr unterschreiten, was in der Zeit von 100 GB-Festplatten keine übertriebene Forderung mehr ist.

10.1.3 Preiswerter Parallelrechner

Pools gleichartiger Rechnerkonfigurationen lassen sich per Boot-Diskette oder -ROM kurzfristig und ohne Änderungen an den bestehenden Betriebssystemen in einen Parallelrechner verwandeln. Inzwischen existiert eine ganze Reihe von Cluster-Software, wie Mosix oder PVM, die Linuxmaschinen in einen leistungsfähigen Rechnerverbund zusammenschalten können. Linux-Cluster finden sich durchaus schon recht weit oben auf der Liste der Supercomputer.

10.1.4 Kostengünstiges X-Terminal

Wenn man nur das Grundsystem (ohne C-Compiler, TeX, usw.) plus XFree86 installiert oder die Maschine per Boot-ROM direkt aus dem Netz bootet, so erhält man ein X-Terminal zu einem unschlagbarem Preis. Da die Leistungsanforderungen recht gering sind, kann man bereits mit älteren 586ern mit 24 - 32 MByte RAM unter der Voraussetzung einer vernünftigen Grafikkonfiguration recht gut arbeiten. So lässt sich auch ältere Hardware noch nutzbringend weiterverwenden.

10.1.5 Linux Diskless Client

Aus Gründen der Kostenersparnis erfreuen sich in den vergangenen Jahren Systeme einer wachsenden Beliebtheit, die Administratoren bei den Routinearbeiten der Einrichtung und des Betriebs von Rechnernetzen unterstützen. Einen entscheidenden Ansatz zur Vereinheitlichung und Automatisierung des Rechnerbetriebes bilden Thin-Clients. Dieser Rechnertyp versucht, durch Reduktion der Hardware und durch Zentralisierung im Betrieb, Kosten in mehreren Dimensionen zu reduzieren. Der Ansatz arbeitet im Bootvorgang analog zu X-Terminals, erlaubt dann aber auch direktes Arbeiten auf der Maschine mit dem vollen Komfort eines Linux-Desktops.

10.1.6 Fileserver

Linux arbeitet mit den meisten gängigen Betriebssystemen gut zusammen und eignet sich deshalb hervorragend als Fileserver. Unter Unix kann das NFS benutzt werden, für Windows steht Samba[1] zur Verfügung, die Anbindung an die Macintosh-Welt erfolgt mit dem Netatalk-Paket. Die Grafikkonfiguration solcher Systeme kann minimal ausfallen, die Festspeicher sind je nach Einsatzgebiet zu planen. Linux verfügt mit LVM und Soft-RAID über vielfältige Möglichkeiten Festplatten optimal zu nutzen.

[1] Protokoll zur Kommunikation mit der Windowswelt. Dieses Paket ermöglicht Windowsshares zur Verfügung zu stellen oder selbst einzumounten. Es kann darüberhinaus einen Windows-Name-Service bereit stellen.

10.1.7 WWW-Server

Mittlerweile gibt es ein reichhaltiges Angebot an Programmen, mit denen sich WWW-Server unter Linux realisieren lassen. Der "apache-httpd" ist der inzwischen meisteingesetzte Server. Mit Perl und PHP und der SQL-Datenbank MySQL stehen weitere leistungsfähige Werkzeuge zum Nulltarif zur Verfügung.

10.1.8 Allgemeine Server-Funktionen

Nahezu jede Server-Funktion läßt sich mit Linux realisieren. Sei es als E-Mail HUB[2], FTP[3]-, DHCP[4]-, DNS-Server[5], LDAP-Server[6]. Warum soll man für Maschinen, die irgendwo im Hintergrund ihre Dienste verrichten und häufig noch nicht einmal einen Monitor haben, teuere und unsichere Betriebssysteme einsetzen, die stark mit einer grafischen Benutzeroberfläche verheiratet sind und über eher mangelhafte Eigenschaften zur Wartung übers Netzwerk verfügen? Für die meisten Netzwerkdienste haben sich Standards herausgebildet und durchgesetzt, an die sich sogar Monopolisten zum größten Teil halten. Ansonsten wäre es nicht zu einem Wachstum dieser Größenordnung im Internet gekommen. Die oben genannten Services gibt es alle als kostenlose Software unter der GPL.

10.2 Vorbereitung der Installation

Nach den Vorüberlegungen, welche Aufgaben das neu zu installierende System übernehmen soll, muss nun bestimmt werden, wie Linux auf die Festplatte gespielt werden soll. Das wird im nächsten Abschnitt behandelt.

Weiterhin braucht man einige Informationen über den Rechner, die Software und Installationsquellen:

10.2.1 Hardware

Prozessortyp: Intel und kompatible (i386, i486, Pentium(II,III,4), K6, Athlon, 680x0 , ...), Sun Sparc Architektur, (DEC) Alpha, 64 bit Intel (Ithanium) oder AMD (Opteron)

Controllertyp: (E)IDE (Chipset des Mainboards), SCSI (Typ und Hardware), DiskOnchip, Flashdevices, ...

Partitionen: Welche auf welcher Festplatte?

[2] HUB (engl.) meint Verteiler, d.h. dass die Maschine, die das Versenden oder Weiterleiten ausgehender Mails mittels SMTP (Simple Mail Transport Protocol) realisiert und ankommende Mails zwischenspeichert, so dass sie von den BenutzerInnen per IMAP oder POP3 (Post Office Protocol) angeholt werden kann. Gleichzeitig lassen sich Filtermechanismen für Spam und Viren installieren.

[3] File Transfer Protocol

[4] Dynamic Host Control Protocol

[5] Domain Name System, die Zuordnung von Namen, d.h. Rechnernamen und Domänennamen (zusammen ergeben sie den sog. Full Qualified Domain Name (FQDN)

[6] Leightweight Directory Access Protocol, z.B. zur Benutzerverwaltung in grossen Rechnernetzen

Maus: Welche Art? (PS/2, serielle, USB, ...)

Grafikkarte: Hersteller, Typ, Chipsatz, Speicher, weitere Daten

Monitor: Hersteller, Typ, maximale Frequenzen, weitere Daten

Lokaler Drucker: PostScript (PS) , PCL, kompatibel mit ...

CD-/DVD-ROM: Atapi, SCSI, proprietär zu ...

Scanner: Hersteller, Typ , Anschluss, weitere Daten

Netzwerkkarte: Hersteller (des Netzwerkchips) und Typ

Sonstige: Weitere Hardwaredaten (Sound-, TV-Karte ...)

10.2.2 Evtl. Netzkonfiguration

Bezugnehmend auf ein typisches ethernet-basiertes LAN:

- IP-Adresse des Rechners
- Name des Rechners
- Domainname
- Domainadresse
- Subnet Maske
- Subnet Name
- Subnet Nummer
- Primary Nameserver
- Secondary Nameserver
- (Third Nameserver)
- Default-Gateway

Viele Distributionen versuchen bei der Installation alle wesentlichen Netzwerkparameter per DHCP zu beschaffen. Befindet sich ein entsprechender Server im Netz und ist die zu installierende Maschine auf diesem Server eingetragen, braucht man sich normalerweise um diese Daten nicht zu kümmern. Soll jedoch die Netzwerkkonfiguration später einmal fest auf der Maschine eingetragen werden, werden die Daten später immer noch benötigt.

10.2.3 Weitere Informationen

- Welche Distribution, woher bekommt man die Files?
- Wer gibt ggf. Hilfestellungen, wann?
- Welches Profil (Server, Multimedia, X-Terminal, SW-Entwicklung, ...)
- Wer übernimmt die Verwaltung der Maschine?
- Für welchen Benutzerkreis ist die Maschine gedacht?
- In welchem Kontext soll die Maschine laufen (Intra- oder Internet)? usw... (siehe Vorüberlegungen)

10.3 Installationsquellen

Als Installationsquellen kommen im Allgemeinen folgende Medien zum Einsatz:

- CD-ROM oder DVD der entsprechenden Linuxdistribution
- Eine Partition auf einer angeschlossenen, gemounteten Festplatte
- Ein per NFS gemountetes Verzeichnis
- Ein Installationsserver (NFS oder FTP) [s. Kap.10.3.1 S.136]
- Disketten, wird heute eigentlich nicht mehr angeboten.
  (Höchstens Ein-Disketten-Mini-Linuxe)

Sofern die Hardware nicht in der Lage ist, von CD oder DVD zu booten, oder man nicht von CD installieren möchte, muss zu Begin einer Installation das Erstellen der Boot- (und Module-/Root-) -Disketten erfolgen. Die Root-Diskette wird zuweilen auch Rescue-Disk oder Supplement-Disk genannt. Entweder erstellt man sich diese Diskette mit den angebotenen DOS-Tools (**rawrite.exe**), oder man erstellt sie sich auf einem laufenden Linux-System mit dem Befehl **dd** (diskdump).

Zunächst mountet man sich das Quellmedium (hier exemplarisch die CD-ROM) in den Dateibaum: **mount /media/cdrom**. Nun beginnt die Suche nach dem passenden Disk-Image. Hat man erst die passenden Dateien gefunden, eine leere Diskette eingelegt und mit **mount /media/floppy** gemountet, so schreibt man das Diskettenimage (sei der Imagename */mnt/images/eide*) mit `dd if=/mnt/images/eide of=/dev/fd0` auf die Diskette. Ebenso verfährt man mit der Rescue-, Modul- oder Root-Disk. Mit diesen Disketten wird dann gebootet und den Installationsanweisungen gefolgt. Ist man Besitzer eines bootfähigen CD-ROM-Laufwerkes (und eines Mainboards, welches eine entsprechende BIOS-Einstellung kennt), so kann der Schritt mit den Bootdisketten entfallen, sofern die CD-ROM bootbar ist (bei der SuSE-Distributionen die CD 1 und 2 bzW. die erste DVD).

Inzwischen gibt es vollständige Distributionen, die sich ausschliesslich von CD betreiben lassen. Ein sehr schönes Beispiel ist das Knoppix-CD-Linux[7]. Diese CD lässt sich sowohl für Demonstrationen der Leistungsfähigkeit von Linux, als auch als Rettungssystem einsetzen. Diese Distribution wird nun schon fast traditionell auf dem Linux-Tag als Veranstaltungs-CD verteilt.

10.3.1 Evtl. Installationsserver

Besonders in großen LANs (lokalen Netzwerken) mit häufigen Installationen bietet sich die Nutzung eines Installationsserver an:

- Internet-Adresse des Servers (NFS-Server oder FTP-Server)
  Gute Adressen sind hierfür :
  1. www.isoimage.org
  2. ftp.gwdg.de:/pub/linux/install/redhat/redhat-???/i386
  3. ftp.gwdg.de:/pub/linux/suse/8.2/i386_de
  4. ftp-suse.uni-freiburg.de:/pub/suse/i386/8.2/
  5. ftp.uni-hohenheim.de/pub/mirror/ftp.redhat.com/pub/redhat/ /redhat-???/i386/
- Und für Kernel-Quellcode :
  1. ftp.gwdg.de:/pub/linux/kernel/...
  2. www.kernel.org

10.4 Software, die nicht der Distribution beiliegt

Softwareinstallation ist ein weites Thema. Im Rahmen dieses Kurses kann unmöglich auf alle Aspekte eingegangen werden. Es ist nicht zwingend, die Pakete in den beschriebenen Verzeichnissen zu halten oder zu kompilieren, es ist jedoch gute Sitte und erhält die Ordnung sowie die "Wiederfindbarkeit"! Es ist aber genausogut möglich, Software im Home-Verzeichnis zu installieren.

Eine gute Quelle für aktuelle Informationen zur verfügbaren Linuxsoftware kann man unter www.freshmeat.net finden. Viele Projekte sind auf dem Server: www.sourceforge.net gehostet und können von dort heruntergeladen werden.

10.5 Aufgaben

10.5.1 Linux als Betriebssystem

1. Wozu gibt es verschiedene Nutzer auf einem Linux-/Unixrechner?

[7] verfügbar über http://www.knoppix.net

2. Was ist ein Multitasking-Betriebssystem? Nennen Sie ein Beispiel für ein Nicht-Multitasking-OS (Operating System)!

3. Warum darf Linux im Gegensatz zu den OSs eines bekannten, unter Monopolverdacht stehenden, Softwareherstellers auf beliebig vielen Maschinen vom selben Datenträger installiert werden?

10.5.2 Bestimmung der Rechnerkonfiguration

1. Welche Hardware-Informationen (Daten) benötige ich für eine typische Linuxinstallation?

2. Wie erhält man in einem PCIRechner Informationen über den Mainboardchipset, evtl. Einsteckkarten und die AGP-Grafikkarte?

3. Ermitteln Sie den CPU-Typ und die Menge des installierten Speichers, allgemein und unter Linux!

4. Wie bekomme ich Informationen zur vorliegenden Hardware heraus, auf der das installierte Linux gerade läuft oder auf der Linux installiert werden soll. Nennen Sie mindestens drei verschiedene Moeglichkeiten!

10.5.3 Einrichten der Festplatte

1. Wie gross ist Ihre Festplatte? Wieviel Festplattenplatz benötigen Sie für die Swappartition und wieviel für eine durchschnittliche Linuxinstallation mit X11 und einem aktuellen KDE?

2. Bestimmen Sie die Partitionierung Ihrer Festplatte! Welches Programm wird üblicherweise dafür verwendet (auch unter DOS/Windows)?

3. Setzen Sie die Kennung Ihrer Swap-Partition auf den Typ des Linux-EXT2-Filesystems. Booten Sie und sehen Sie nach, ob noch Swap verwendet wird. Machen Sie dieses wieder rückgängig.

4. Wie gross ist die Root- und die Swappartition Ihrer Installation? Wieviel Platz habe ich gerade noch unter Linux, wie bekomme ich heraus, wo welche Partitionen gemountet sind?

5. Wieviel freien Festplattenplatz haben Sie noch in Ihrer Rootpartition?

6. Setzen Sie das “activate” Flag für Ihre Bootpartition, wann kann dieses notwendig werden?

10.5.4 Rettungssystem / -diskette

1. Erstellen Sie eine Rettungsdiskette!
   a) Mittels Yast(2).
   b) Indem Sie den Kernel direkt auf die Diskette schreiben:
   ”dd if=/boot/vmlinuz of=/dev/fd0” Wird hierbei ein Bootloader benutzt?

2. Welchen Nachteil hat das Verfahren aus Aufgabe 1b)?

3. Welche Alternativen kennen Sie noch zur Rettung nicht mehr bootender Systeme?

Kapitel 12

Programmentwicklung

12.1 Intro

Linux ist die ideale Entwicklungsumgebung für vielerlei Arten von Softwareprojekten. Es stehen fast alle gängigen Programmiersprachen in einer frei verfügbaren Implementierung zur Verfügung und die Auswahl der Entwicklungswerkzeuge hat sich in den letzten Jahren stark vergrößert.

Bei größeren Software-Projekten oder Texten, wie diesem hier vorliegenden, kommt man mit einer manuellen Verwaltung durch Absprachen und regelmäßigem versuchten Abgleich der verschiedenen Versionen bald nicht mehr weit, wie auch die Autoren dieses Textes im mutigen Selbstversuch herausfanden! Deshalb widmet sich ein Abschnitt der automatischen Versionsverwaltung. Diese kann durch verschiedene Werkzeuge erfolgen, vorgestellt wird hier das verbreitete und für diesen Text verwendete "CVS".

12.2 CVS zur Projektverwaltung

12.2.1 Die Idee und das Konzept

Programmquellen[1] und deren Dokumentationen sowie Texte aller Art in geeigneter Form[2] unterliegen einer hohen Änderungsfrequenz. Gleichzeitig werden häufig viele Dateien in verschiedenen Versionen parallel von mehreren Projektbeteiligten verändert. Das Programmpaket CVS hilft dabei für alle den Überblick zu behalten.

Für die Verwaltung größerer Projekte, seien es Texte oder Programmcode gibt es eine Reihe von Anforderungen. So sollen Änderungen, z.B. Löschungen von Textabschnitten, rückgängig gemacht werden können. Oder es soll eine bestimmte Fassung, die zu einem bestimmten Datum aktuell war, rekonstruiert werden. Oder es soll der Änderungsverlauf nachvollziehbar sein, warum also wann von wem welche Änderungen durchgeführt wurden.

Es sind also verschiedene Fassungen oder Versionen einer Datei und die Zusammenhänge zwischen ihnen zu berücksichtigen. Hierzu können natürlich laufend Kopien

[1]sogenannte sources (engl.)

[2]proprietäre und in normalen Editoren unlesbare Texte, wie das MS-Word-Format fallen nicht in diese Kategorie

der entsprechenden Dateien erstellt werden. Ein besserer Ansatz ist jedoch der Einsatz eines Versionsverwaltungssystems, welches für solche Aufgaben entworfen wurde. Dieses macht sich die inkrementelle Natur des Vorgangs zunutze, indem es nur die Änderungen zwischen den Versionen archiviert. Versionen werden über sogenannte Versionsnummern identifiziert.

Die Situation wird verschärft, wenn gleichzeitig mehrere Entwickler an einem Projekt arbeiten, das aus vielen, vielleicht Hunderten von Dateien besteht. Einerseits ist dafür das unabhängige Arbeiten zu gewährleisten, andererseits dürfen Änderungen nicht verloren gehen.

Um zu verstehen, was dies mit der Unterstützung der Zusammenarbeit zu tun hat, wirft man am besten zunächst einen Blick auf die Mechanismen, mit denen CVS es ermöglicht, dass mehrere Personen zusammen an einem Projekt arbeiten. Zuvor einige Anmerkungen zu einem Mechanismus, der von anderen Versionsverwaltern bekannt sein mag, den jedoch CVS nicht bietet oder zumindest nicht empfielt: das Dateisperren. Vielleicht ist das Entwicklungsmodell Sperren-Ändern-Freigeben bereits bekannt, bei dem ein Entwickler zuerst den exklusiven Schreibzugriff auf die zu bearbeitende Datei (eine Sperre) erhalten muss, anschliessend die Veränderungen vornimmt und dann die Sperre wieder freigibt, damit andere Projektbeteiligte wieder auf diese Datei zugreifen können. Wenn jemand anders bereits eine Sperre für diese Datei gesetzt hat, so muss er diese zuerst wieder freigeben, bevor man selbst eine Sperre setzen und Veränderungen anbringen kann.

Dieses System bietet sich an, wenn alle Projektbeteiligten sich kennen und sehr eng zusammenarbeiten und daher wissen, wer was zu einem bestimmten Zeitpunkt machen möchte. Zugriffskonflikte lassen sich bei diesem Verfahren auf einer direkten Ebene klären, die nicht vom Versionsverwaltungssystem realisiert werden muss. Handelt es sich bei einer Projektgruppe um eine größere Anzahl von Entwicklern, die dazu noch geografisch weit verteilt arbeiten, kann das notwendige Sperren und Entsprerren schon wesentliche Teile der zur Verfügung stehenden Arbeitszeit auffressen.

CVS realiseiert einen weitergehenden Ansatz um dieses Problem in den Griff zu bekommen. Die Entwickler haben nun nicht mehr die vorrangige Aufgabe ihre Arbeit eng zu koordinieren. Dieses übernimmt CVS durch die Integration der Veränderungen und den Überblick zu vorgenommenen Veränderungen. Beschrieben werden kann dieses Verfahrung durch das Kopieren-Modifizieren-Zusammenfassen-Modell, das wie folgt funktioniert:

- Entwickler 1 fordert eine Arbeitskopie von CVS an (ein Verzeichnisbaum, der alle Dateien eines Projektes enthält). Dies wird auch "Checking out" einer Arbeitskopie genannt, wie das Ausleihen eines Buches aus einer Bibliothek.

- Entwickler 1 arbeitet unbeschwert an seiner lokalen Arbeitskopie. Gleichzeitig können andere Entwickler ebenfalls an ihren eigenen Arbeitskopien werkeln. Da alle Kopien unabhängig voneinander sind, können zu diesem Zeitpunkt keine Konflikte auftreten: Es ist so, als hätten alle Entwickler ihre eigene Kopie des gleichen Buches aus der Bibliothek entliehen und sie alle schreiben, unabhängig voneinander, Kommentare an dessen Ränder oder überarbeiten bestimmte Seiten gänzlich neu.

- Entwickler 1 schliesst seine Veränderungen vorerst ab und sendet diese verknüpft mit einem Kommentar, der beschreibt, was der Zweck der Veränderungen war, an den CVS-Server ("commit"). Dieses läßt sich damit vergleichen, die Bibliothek darüber zu informieren, welche Veränderungen gemacht wurden und aus welchen Gründen. Die Bibliothek lässt diese wiederum in eine Hauptkopie einfließen, wo sie damit für alle Zeit aufgezeichnet werden.

- In der Zwischenzeit können andere Entwickler CVS dazu veranlassen, die Bibliothek abzufragen, um festzustellen, ob die Hauptkopie in jüngster Zeit Veränderungen ausgesetzt war. Wenn dieses zutrifft aktualisiert CVS automatisch die jeweils lokale Arbeitskopie.

Aus der Sicht von CVS sind alle Entwickler eines Projektes gleich. Zu entscheiden, wann ein "commit" oder eine Aktualisierung durchgeführt wird, obliegt einem selbst. Dieses ist eher eine Sache der persönlichen Einschätzung oder der Projektregeln, nie jedoch eine Festlegung seitens CVS. Eine übliche Strategie bei Programmierprojekten sieht vor, immer dann eine Aktualisierung vorzunehmen, bevor die Arbeit an größeren Veränderungen begonnen wird. Ein "commit" sollte erst dann erfolgen, wenn die Veränderungen vollständig und getestet sind, so dass die Hauptkopie sich möglichst zu jedem Zeitpunkt in einem funktionsfähigen Zustand befindet.

Es kann hierbei natürlich vorkommen, dass die Entwickler 1 und 2 in ihren jeweiligen Arbeitskopien unterschiedliche Veränderungen an der gleichen Datei an identischer Textstelle vornehmen und beide ihre Veränderungen mittels "commit" aufs Archiv übertragen wollen. In diesem Augenblick meldet CVS einen Konflikt, sobald Entwickler 1 oder 2, versucht seine Version abzuschicken. CVS setzt Konfliktmarkierungen, leicht zu erkennende Marken im Text, an die in Konflikt stehenden Stellen im Text seiner Kopie. Diese Stellen beinhalten beide Veränderungen und sind derart angeordnet, dass sie leicht verglichen werden können. Der jeweilige Entwickler muss sich nun manuell die Geschichte nochmal ansehen. Er muss eine Version bauen, die den Konflikt auflöst.

Was ist nun mit der Hauptkopie? In der offiziellen CVS-Terminologie wird diese das Archiv (Repository) eines Projektes genannt. Das Archiv ist schlicht nur ein Datei-/Verzeichnisbaum, der auf einem Server gespeichert ist. Ohne zu stark in die Tiefe der Struktur zu gehen, wird im Folgenden beschrieben, was ein Archiv leisten muss, um den Anforderungen des Checkout-Commit-Aktualisieren-Zyklus gerecht zu werden. Hierzu stelle man sich folgendes Szenario vor:

- Zwei Projektbeteiligte, 1 und 2, führen gleichzeitig einen "checkout" des gleichen Projektes aus. Das Projekt befindet sich noch am Ausgangspunkt - es wurden noch von niemandem Veränderungen per Commit an das Archiv geschickt, sodass sich noch alle Dateien in ihrem ursprünglichen Zustand befinden.

- Entwickler 1 beginnt anschliessend mit seiner Arbeit und führt schon bald einen ersten "commit" seiner Änderungen aus. In der Zwischenzeit ist Entwickler 2 immer noch mit anderen Dingen beschäftigt.

- Entwickler 1 arbeitet weiter und denkt, dass sich ein lohnender Zwischenstand ergeben hat. Er führt einen zweiten "commit " aus. Das Archiv enthält nun die

Originaldateien, gefolgt vom zweiten Satz der Veränderungen von Entwickler 1, gefolgt von diesem Satz an Veränderungen. Entwickler 2 hat bisher kein Bit an seinem Satz von Dateien geändert.

- Zusätzlich stößt ein weiterer Entwickler 3 zum Projekt und generiert für sich eine lokale Arbeitskopie aus dem Archiv. Die Kopie von Entwickler 3 enthält damit bereits zwei Sätze von Veränderungen von Entwickler 1, da diese schon im Archiv enthalten waren. als 3 sein "checkout" unternahm.

- Entwickler 1 ist immer noch am Programmieren und vollendet seinen dritten Satz an Modifikationen. Diese werden wiederum "commit"ed.

- Irgendwann beschliesst Entwickler 3 an die Arbeit zu gehen. Er kümmert sich nicht darum, eine Aktualisierung seiner Arbeitskopie durchzuführen, da er vielleicht nicht davon ausgeht, dass in der Zwischenzeit wesentliche Änderungen erfolgt sind. Er fängt an, Dateien zu bearbeiten, von denen einige dabei sein könnten, an denen 1 bereits Modifikationen vornahm. Kurz darauf führt Entwickler 2 seinen ersten "commit" dieser Veränderungen aus.

An diesem Punkt können nun zwei Dinge passieren. Wenn keine der von Entwickler 2 bearbeiteten Dateien von 1 bearbeitet wurde, dann ist der "commit" erfolgreich. Wenn CVS jedoch merkt, dass einige der Dateien von 2 verglichen mit den aktuellen Dateien des Archivs veraltet sind und diese auch von 2 in seiner Arbeitskopie verändert wurden, informiert CVS 2 darüber, dass er eine Aktualisierung durchführen muss, bevor ein Commit durchgeführt werden kann.

Wenn Entwickler 2 die Aktualisierung durchführt, fügt die Versionsverwaltung alle Veränderungen von 1 in 2's lokale Kopien der Dateien ein. Einige von 1's Veränderungen können mit 2's noch nicht abgeschickten Veränderungen in Konflikt geraten, manche nicht. Die Teile, welche nicht in Konflikt stehen, werden einfach ohne weitere Komplikationen in 2's Kopie eingefügt; die in Konflikt stehenden müssen zuerst von 2 bereinigt werden, bevor der Commit durchgeführt werden kann.

Wenn Entwickler 3 nun eine Aktualisierung durchführt, bekommt er mehrere Sätze an Veränderungen aus dem Archiv: den dritten Commit von 1, den ersten von 2 und vielleicht den zweiten von 2 (wenn 2 die Konflikte aufgelöst hatte).

Damit die Versionsverwaltung Veränderungen in der richtigen Reihenfolge an die Entwickler verteilen kann, deren Arbeitskopien unter Umständen unterschiedlich stark veraltet sind, muss das Archiv alle Commits seit Projektbeginn aufzeichnen. In der Praxis speichert das CVS-Archiv diese als aufeinander folgende Diffs. Daher kann CVS auch noch für sehr alte Arbeitskopien den Unterschied zwischen den Dateien der Arbeitskopien und dem aktuellen Stand des Archivs bestimmen und dadurch die Arbeitskopie effizient wieder auf den aktuellen Stand bringen. Für Entwickler ist es dadurch einfach, die Historie des Projektes einzusehen und zu jedem Zeitpunkt sogar sehr alte Arbeitskopien wieder zum Leben zu erwecken.

Obwohl das Archiv genau genommen das gleiche Resultat mit anderen Methoden erreichen könnte, ist das Abspeichern der Differenzdateien eine einfache und intuitive Methode, die notwendige Funktionalität zu implementieren. Dieser Prozess hat den zusätzlichen Vorteil, dass CVS durch die korrekte Anwendung von patch jeden vorangegangenen Zustand des Verzeichnisbaumes wiederherstellen und damit jede Arbeitskopie von einem Zustand in einen anderen überführen kann. Es erlaubt jedem, einen Checkout des Projektes in einem womöglich vergangenen Zustand zu machen. Es kann ebenso die Unterschiede im diff-Format zwischen zwei Zuständen des Projektes aufzeigen, ohne dabei irgendeine Arbeitskopie zu beeinflussen.

Daher sind genau diese Funktionen, die den vernünftigen Zugriff auf die Historie eines Projektes zulassen, auch dafür nützlich, es einer dezentralen, unkoordinierten Entwicklergruppe zu ermöglichen, an einem Projekt zusammenzuarbeiten.

12.2.2 CVS Begriffswelt und Kommandos

Im Folgenden werden immer wieder Bezeichnungen auftauchen, die einen bestimmten Vorgang bei der Benutzung oder die Aktionen von CVS beschreiben. Die CVS-Sub-Kommandos, soweit mit einzelnen Begriffen assoziiert, sind hinter diesen in Klammern angegeben.

- Aktualisierung (*up*) - Auch "update" genannt überträgt die Modifikationen anderer Entwicklern vom Archiv in die Arbeitskopie. In diesem Schritt wird gleichzeitig überprüft, ob die eigene Arbeitskopie noch nicht durch ein "commit" auf das Archiv übertragene Modifikationen beinhaltet.

- Arbeitskopie - Die Kopie einer Datei, die sich in Benutzung befindet, d.h. auf der aktuell Veränderungen vorgenommen werden. Dabei ist es erlaubt, dass mehrere Arbeitskopien einer Datei oder eines Projektes existieren. Jeder Entwickler verfügt üblicherweise über eine eigene Kopie.

- Archiv - Dieses bezeichnet die Hauptkopie eines Projektes. Hierin hinterlegt CVS die vollständige Revisionshistorie. Jedes Projekt verfügt über ein dediziertes Archiv.

- Checkout (*co* - Hierdurch holt sich der Entwickler eine Arbeitskopie vom Archiv ab. Dabei ist die angeforderte Kopie eine Momentaufnahme des gerade aktuellen Archivstandes eines Projektes. Um die eigenen Weiterentwicklungen und Änderungen anderen zugänglich zu machen, müssen diese mittels "commit" ins Archiv übertragen werden. Damit man die durch andere vorgenommenen Änderungen im Projekt zu sehen bekommt, aktualisiert man seine lokale Kopie durch ein "update".

- Commit (*commit*)- Wird auch als "Check-In", d.h. Bekanntgeben, der eigenen Veränderungen auf das zentrale Archiv bezeichnet.

- Importieren (*import*) - Kommen neue Dateien hinzu, die bisher nicht im Archiv erhalten sind, müssen diese "importiert" werden. Damit werden sie in die CVS-Archivliste des Projektes aufgenommen.

- Konflikt - Nicht alle Veränderungen lassen sich reibungslos auf das Archiv zurückübertragen. Es gibt Situationen, in denen zwei Entwickler versuchen Veränderungen in der gleichen Datei an gleicher Stelle per "commit" zu übertragen.

CVS stellt ein Problem fest und informiert die Entwickler. Eine Auflösung des Konfliktes muss manuell erfolgen.

- Log - Ein Nachrichtenteil (Logfile) welcher beim "commit" an eine Revision angehängt wird. Änderungen lassen sich auf diese Weise beschreiben, so dass andere Entwickler verfolgen können, was in dem Projekt passiert ist.

- Revision - Sie beschreibt alle Veränderungen an einer Menge von Dateien (die auch aus einer einzigen Datei bestehen kann), die durch einen "commit" abgeschlossen wurde. Eine Revision entspricht damit einer Momentaufnahme eines Projektes.

Wie bei vielen anderen Unix-Kommandos auch handelt es sich bei **cvs** um ein einziges Kommando, welches durch verschiedene Argumente seine Funktionalität ändert[3]. Es realisiert so die Aufgaben, wie "import", "commit", "diff", "update" und etliche weitere. Der generelle Aufbau eines CVS-Kommandos sieht so aus:
`cvs [` *cvs_options* `]` *command* `[` *command_options* `] [` *command_args* `]`

cvs_options	Einige Optionen, die alle Sub-Kommandos beeinflussen
command	Einer der vielen Sub-Befehle (wie "up", "commit", ...), die teilweise abgekürzt werden. Nur `cvs -H` (Hilfe für *command*) und `cvs -v` (Version) kommen ohne *command* aus
command_options	Optionen, die sich auf *command* beziehen
command_args	Argumente für *command*

CVS bringt bei seinen Arbeiten auf einzelnen Dateien verschiedene Statusmeldungen: Am Anfang der Zeile vor jeder Datei ein einzelner Buchstabe. Dabei steht *U* für "Update", *M* für "Modified" und *C* für Conflict. Durch das *M* wird angezeigt, dass man selbst als Letzter die Datei angefasst hat und etwaige Änderungen allen anderen noch nicht bekannt sind. *N* meldet beim Import, dass die entsprechende Datei neu im Archiv neu angelegt wurde. Ein *T* vor dem Dateinamen zeigt an, dass eine Markierung (ein sogenanntes "Tag") auf dieser Datei angebracht wurde.

12.2.3 Einige Vorbereitungen

Nicht alle Dateitypen lassen sich in CVS gleich effektiv organisieren oder anders ausgedrückt: Besonders reine Textdateien eignen sich für die Verwaltung im CVS. Dazu zählen alle Quelldateien beliebiger Programmiersprachen, Texte in HTML-, XML-, Tex@TEX - oder ähnlichen Formaten, jedoch keine scheinbar chaotisch aufgebauten binären Formate, wie Microsoft Word oder Excel. Dieses geht natürlich auch, aber ob die Fähigkeiten von CVS hier Sinn machen, darf bezweifelt werden. Ebenfalls müssen nicht alle Dateien verwaltet werden, die im Zuge eines Projektes so anfallen: Es macht wenig Sinn mit CVS binäre Zwischenformen, wie Object-Dateien und kompilierte Programme oder bei LATEX-Dokumenten DVI- und Postscript-Ausgaben zu verwalten.

CVS kann sowohl lokal als auch netzwerktransparent arbeiten. Im letzteren Fall ist jedoch eine Anmeldung beim CVS-Server erforderlich. Man meldet sich auf dem

[3] Es gibt eine ganze Reihe weiterer Unix-Kommandos, wie **ip**, **tc**, ..., die mit Sub-Befehlen arbeiten.

Server durch: `cvs -d:pserver:user@cvs-server.domain.local:/srv/cvs login` an. Bei erfolgreicher Anmeldung wird im Homeverzeichnis eine Datei *.cvspass* angelegt, die eine Kombination aus Server, Pfad und Passwort speichert, so dass für alle weiteren CVS-Kommandos keine weitere Authentifizierung notwendig wird. In dieser Kommandozeile bestimmt die Option "-d" das CVS-Repository[4], welches sich aus einer Kombination aus Benutzer- und Rechnername gefolgt von einem absoluten Dateipfad nach dem Doppelpunkt zusammensetzt. Das Argument "login" authentifiziert den angegebenen Benutzer "user" beim CVS-Server (im Beispiel: *cvs-server.domain.local*).

Wenn noch kein CVS-Unterverzeichnis im Projektbaum exisitert, wie das bei der Erstanlage meist der Fall sein wird, ist das CVS-Repository anzugeben. Für diesen Zweck existiert die Umgebungsvariable "$CVSROOT" oder der Kommandozeilenschalter "-d":

```
user@linux:~> CVSROOT=":pserver:user@cvs-server.domain.local:/srv/cvs"
user@linux:~> export $CVSROOT
```

Später wird die Umgebungsvariable nicht mehr benötigt oder kann sogar störend wirken, wenn verschiedene Projekte auf verschiedenen Servern verwaltet werden. Eine parallele Versionverwaltung stellt für CVS kein Problem dar, da die Datei *.cvspass* mehrere Einträge aufnehmen kann und alle notwendigen Basisinformationen im CVS-Unterverzeichnis gespeichert werden.

12.2.4 Anlegen eines neuen Projektes

Nicht jeder wird gleich ein neues Projekt anlegen wollen, trotzdem soll diese Beschreibung schrittweise aufeinander aufbauen. Das Concurrent Version System weiss zu diesem Zeitpunkt noch nichts von seinem Glück und muss deshalb zuerst mit dem Satz von Dateien gefüllt werden, der das Projekt ausmacht. Hierbei ist zu überlegen welche Dateien dazu zählen und welche davon nur temporären Charakter haben und von jedem Programmierer bei Bedarf leicht selbst erzeugt werden können.

Ein bestehendes Projekt in der Versionsverwaltung erstmalig anzulegen, geschieht mit dem Kommando `cvs import Argumentliste`. Dazu lege man zuerst eine Sicherheitskopie des Projektes an, wechsele in das entsprechende Verzeichnis und lösche alle nicht per CVS zu verwaltenden Dateien. Im Beispiel geschieht das für das Einchecken dieses Skriptes in CVS.

```
user@linux:~> cp -a ldap-projekt ldap-projekt.bak
user@linux:~> cd ldap-projekt
user@linux:~> rm *.{aux,dvi,toc,pdf,log,ilg,idx,ind,old,~}
user@linux:~> cvs import -m "TeX-Skript in einzelnen Kapiteln" ldap-projekt
autor start
N ldap-projekt/00kap.tex
N ldap-projekt/allestex
N ldap-projekt/davedap-new.eps
```

[4] auch als CVS-Root bezeichnet, das Archivverzeichnis ohne jeweiliges Projektverzeichnis des CVS auf dem Server

```
N ldap-projekt/davedap-user.eps
N ldap-projekt/dn-rdn.eps
N ldap-projekt/einleitung.tex
N ldap-projekt/gq-browser.eps
N ldap-projekt/header.tex
N ldap-projekt/java-ldapbrowser.eps
N ldap-projekt/ldap-browser.tex
N ldap-projekt/ldap-einf.tex
N ldap-projekt/ldap-linux.tex
N ldap-projekt/lehrstuhl-logo.eps
N ldap-projekt/linux-einf.tex
N ldap-projekt/php-html.tex
N ldap-projekt/php-ldap.tex
N ldap-projekt/prog-entw.tex
N ldap-projekt/relat-vs-hier.eps
N ldap-projekt/template-system.tex
N ldap-projekt/yala-login.eps
N ldap-projekt/yala-useradm.eps
N ldap-projekt/zentral.tex

No conflicts created by this import
```

Wenn die in der letzten Zeile gezeigte Meldung erfolgt, ist das Projekt erfolgreich auf den CVS-Server übertragen worden. Man besitzt mit dem soeben transferierten Verzeichnis jedoch noch keine CVS-offizielle lokale Arbeitskopie. Diese muss in einem weiteren Schritt "ausgecheckt" werden. Dazu lösche man am besten das Projektverzeichnis, nachdem man sich der Sicherheitskopie vergewissert hat.

12.2.5 Eine Arbeitskopie erstellen

Nicht jeder am Projekt Beteiligte wird selbst etwas initial im Concurrent Version System hinterlegen, jedoch benötigt jeder Beteiligte eine lokale Arbeitskopie, auch derjenige, der im soeben beschriebenen Schritt das CVS-Archiv angelegt hat. Das "check-out", das Erstellen der lokalen Arbeitskopie, geschieht mittels `cvs checkout projectname`. Im Beispiel bleibend:

```
user@linux:~>
dirk@hermes:~> cvs checkout ldap-projekt
cvs server: Updating ldap-projekt
U ldap-projekt/00kap.tex
U ldap-projekt/allestex
U ldap-projekt/davedap-new.eps
[ ... ]
user@linux:~>
```

Dieser Schritt legt im aktuellen Verzeichnis ein Unterverzeichnis mit dem Projektnamen an und kopiert in dieses alle aktuellen Archivdateien. In diesem Verzeichnis wird ein Unterverzeichnis *CVS* erzeugt, in dem CVS Informationen zur Versionskontrolle speichtert. Genauer gesagt, existiert nun in jedem Unterverzeichnis des Projektes ein CVS-Unterverzeichnis. Dieses impliziert, dass man tunlichst vermeiden sollte ein Unterverzeichnis mit diesem Namen zu verwenden, was aber nicht allzu schwer fallen sollte! Die Informationen, die sich in diesem Unterverzeichnis finden, kommen einem inzwischen sicherlich bekannt vor:

```
user@linux:~/ldap-projekt> cd CVS; ls
Entries  Repository  Root
user@linux:~/ldap-projekt/CVS> cat Entries
/00kap.tex/1.1.1.1/Sat Nov  1 19:21:12 2003//
/allestex/1.1.1.1/Sat Nov  1 19:21:12 2003//
/davedap-new.eps/1.1.1.1/Sat Nov  1 19:21:32 2003//
/davedap-user.eps/1.1.1.1/Sat Nov  1 19:22:02 2003//
/dn-rdn.eps/1.1.1.1/Sat Nov  1 19:22:20 2003//
/einleitung.tex/1.1.1.1/Sat Nov  1 19:22:20 2003//
/gq-browser.eps/1.1.1.1/Sat Nov  1 19:22:44 2003//
/header.tex/1.1.1.1/Sat Nov  1 19:22:44 2003//
/java-ldapbrowser.eps/1.1.1.1/Sat Nov  1 19:22:58 2003//
/ldap-browser.tex/1.1.1.1/Sat Nov  1 19:22:58 2003//
/ldap-einf.tex/1.1.1.1/Sat Nov  1 19:23:00 2003//
/ldap-linux.tex/1.1.1.1/Sat Nov  1 19:23:02 2003//
/lehrstuhl-logo.eps/1.1.1.1/Sat Nov  1 19:23:34 2003//
/linux-einf.tex/1.1.1.1/Sat Nov  1 19:23:38 2003//
/php-html.tex/1.1.1.1/Sat Nov  1 19:23:38 2003//
/php-ldap.tex/1.1.1.1/Sat Nov  1 19:23:38 2003//
/prog-entw.tex/1.1.1.1/Sat Nov  1 19:23:38 2003//
/relat-vs-hier.eps/1.1.1.1/Sat Nov  1 19:24:32 2003//
/template-system.tex/1.1.1.1/Sat Nov  1 19:24:32 2003//
/yala-login.eps/1.1.1.1/Sat Nov  1 19:25:00 2003//
/yala-useradm.eps/1.1.1.1/Sat Nov  1 19:25:28 2003//
/zentral.tex/1.1.1.1/Sat Nov  1 19:25:28 2003//
D
user@linux:~/ldap-projekte/CVS> cat Repository
ldap-projekte
user@linux:~/ldap-projekte/CVS> cat Root
:pserver:user@cvs-server.domain.local:/srv/cvs
```

Die Datei *Repository* verweist auf ein Projekt innerhalb des Archivs, welches unterhalb des in *Root* angebenen Verzeichnis gespeichert ist. *Entries* enthält die Liste aller Archivdateien. Jede Zeile beschreibt hierbei eine Datei. Es finden sich hier nur Einträge für die Dateien und Unterverzeichnisse des nächst übergeordneten Verzeichnisses. Jeder Eintrag dieser Datei besitzt das Format "/Dateiname/Revisionsnummer/Zeitstempel//". Mit den Zeitstempeln werden Uhrzeit und Datum der letzten Aktualisierung der Dateien in der Arbeitskopie hinterlegt. Dieses geschieht in universeller Zeit. So kann CVS einfach feststellen, ob eine Datei seit dem letzten "update", "checkout" oder "commit" angefasst wurde. Wenn sich der Zeitstempel des eigenen Dateisystems von dem in der

CVS/Entries unterscheidet, kann CVS davon ausgehen, dass die Datei wahrscheinlich geändert wurde. Die Revisionsnummern sind willkürlich und werden nicht ausserhalb von CVS sichtbar. Es gibt einen weiteren, vom genannten unabhängigen, Typ der Kennzeichnung: "Tags", der in einem Abschnitt weiter unten behandelt wird.

12.2.6 Laufende Veränderungen

Nun kann man davon ausgehen, dass ein Projekt nicht statisch vor sich hindämmert, sondern laufenden Veränderungen unterworfen ist. Sonst wäre der Aufwand zur Versionsverwaltung vielleicht etwas überkandidelt und allenfalls eine Fingerübung. Bevor man beginnt eigene Änderungen in der lokalen Arbeitskopie vorzunehmen, überprüft man, ob sich nicht vielleicht von anderer Seite bereits welche ergeben haben. Diese Aufgabe übernimmt `cvs update` oder kürzer `cvs up`:

```
user@linux:~/ldap-projekt> cvs up
cvs server: Updating .
```

Hat sich nichts geändert, sieht es genauso wie im Beispiel aus, sonst wird eine Liste der aktualisierten Dateien angezeigt:

```
user@linux:~/ldap-projekt> cvs up
cvs server: Updating .
M prog-entw.tex
M einleitung.tex
M php-ldap.tex
```

Möchte man einen detaillierteren Blick auf die Veränderungen werfen, kann man einen kompletten Report im "diff"-Format anfordern. **diff** vergleicht die möglicherweise modifizierten Dateien der Arbeitskopie mit den entsprechenden Gegenstücken im Archiv und zeigt jegliche Unterschiede auf.

```
user@linux:~/ldap-projekt> cvs diff
cvs server: Diffing .
Index: prog-entw.tex
===================================================================
RCS file: /home/src/ldap-projekt/prog-entw.tex,v
retrieving revision 1.1.1.1
diff -r1.1.1.1 prog-entw.tex
3c3
< \chapter{Programmentwicklung und Projektverwaltung}
---
> \chapter{Programmentwicklung}
8c8,18
< Softwareprojekten.
---
[ ... ]
```

Nun legt man selbst los und nimmt Veränderungen am Projekt vor. Nachdem man diese abgeschlossen hat oder für einen gewissen Zeitraum unterbrechen will, meldet man seine Heldentaten wieder ans CVS zurück: `cvs commit` gleicht die Arbeitskopie in umgekehrter Richtung mit dem Archiv ab.

```
user@linux:~/ldap-projekt> cvs commit
# (--> editor for log comments invoked)
Checking in prog-entw.tex;
/home/src/ldap-projekt/prog-entw.tex,v  <--  prog-entw.tex
new revision: 1.2; previous revision: 1.1
done
```

Im Beispiel wird automatisch der in der Umgebungsvariablen "$EDITOR" angegebene Editor gestartet, um Kommentare für das Logfile anzugeben. Möchte man Kommentare direkt ohne den Editor-Umweg angeben, hat man dazu die Option '-m "Kommentar" ' zur Verfügung. Im Beispiel wurden automatisch alle Dateien "committed". Einzelne Dateien können jedoch auch als Liste am Ende des Kommandos angegeben werden. Im Beispiel verlief das Update ohne Probleme. Die veränderte Datei *prog-entw.tex* aus dem Repository konnte ohne Konflikte mit der Arbeitskopie verschmolzen werden. Dieses Verfahren hat sich in der Praxis als relativ zuverlässig erwiesen. Man sollte natürlich an dieser Stelle Vorsicht walten lassen und nach Merging-Meldungen anschliessend überprüfen, ob sich die neu entstandenen Dateien noch sinnvoll verwendbar sind. Verläuft das Verschmelzen allerdings nicht reibungslos, wird man von CVS durch Konfliktmeldungen darauf aufmerksam gemacht.

Für das Versionsverwaltungs-Tool bedeutet das Löschen oder Hinzufügen von Dateien nichts anderes, als eine Änderung am bestehenden Datenbestand. Das hat auch einen einleuchtenden Grund. Wie schon erwähnt, ist es ja unter dem Concurrent Version System möglich, auf beliebige Versionen einzelner Dateien zurückzugreifen. Das heißt, man kann z.B. nach Belieben eine lauffähige Variante seines Programms auschecken, ohne auf aktuellere, aber möglicherweise fehlerbehaftete Teile Rücksicht nehmen zu müssen. Nun kann es ja durchaus vorkommen, dass in dieser Version Dateien vorhanden sind, die in der weiteren Entwicklung wieder verschwunden sind. Sie werden aber weiterhin mitverwaltet und existieren immer noch im Repository, auch wenn sie beim jetzigen Checkout nicht mehr auftauchen würden. Ebenso weiss die Versionsverwaltung nichts von neuen Dateien, die lediglich im Arbeitsverzeichnis, aber noch nicht im Repository auftauchen:

```
user@linux:~/ldap-projekt> cvs add test.txt
cvs server: scheduling file ‘test.txt’ for addition
cvs server: use ’cvs commit’ to add this file permanently
```

Hat man also eine neue Datei erstellt, die sich noch nicht auf dem CVS-Server befindet, so wird diese durch `cvs add neue.datei` hinzugefügt und anschliessend mittels `cvs commit` auf dem gewohnten Weg eingecheckt.

In jedem steckt ein gewisser Grad an Neugier - beim einen mehr, beim anderen weniger. Hier beschäftigt die Frage, was zwischen verschiedenen Revisionen passiert ist. Was macht den Unterschied zwischen 1.6 und 1.7 aus? Für diesen Zweck steht der Sub-Befehl "log" zur Verfügung:

```
user@linux:~/ldap-projekt> cvs log prog-entw.tex
```

```
RCS file: /home/src/ldap-projekt/prog-entw.tex,v
Working file: prog-entw.tex
head: 1.2
branch:
locks: strict
access list:
symbolic names:
        start: 1.1.1.1
        dirk: 1.1.1
keyword substitution: kv
total revisions: 3;     selected revisions: 3
description:
----------------------------
revision 1.2
date: 2003/11/01 19:32:34;  author: dsuchod;  state: Exp;  lines: +370 -3
*** empty log message ***
----------------------------
revision 1.1
date: 2003/11/01 19:23:39;  author: dsuchod;  state: Exp;
branches:  1.1.1;
Initial revision
----------------------------
revision 1.1.1.1
date: 2003/11/01 19:23:39;  author: dsuchod;  state: Exp;  lines: +0 -0
Skript fuer das Softwarepraktikum - LDAP-Projekte (PHP, Perl, Java, ...)
=============================================================================
```

Liest man die ausgeworfenen Zeilen von unten nach oben durch, wird einem das Prinzip schnell klar. Der Kommentar aus der letzten Zeile dürfte vom Anlegen des Repository bekannt sein. Vielleicht nur so viel: Bei Revision 1.2 folgt in der nächsten Zeile zunächst der genaue Zeitpunkt der Veränderung sowie der Autor der Datei. In der folgenden Zeile steht der Log-Kommentar, der beim "commit"-Vorgang im Editor (oder mittels '-m "Kommentar"') eingegeben wurde.

12.2.7 Löschen von Dateien und Verzeichnissen

Das Entfernen von Dateien funktioniert ähnlich, wie das Hinzufügen. Die Reihenfolge ist identisch: Zuerst wird die Datei lokal in der Arbeitskopie gelöscht, anschliessend durch `cvs remove datei` aus dem CVS-Archiv.

```
user@linux:~/ldap-projekt> rm for-removal.txt
user@linux:~/ldap-projekt> cvs remove for-removal.txt
cvs remove: scheduling 'for-removal.txt' for removal
cvs remove: use 'cvs commit' to remove this file permanently
user@linux:~/ldap-projekt> cvs ci -m "removed newfile.c" newfile.c
Removing for-removal.txt;
/home/src/ldap-projekt/for-removal.txt,v <- for-removal.txt
```

```
new revision: delete; previous revision: 1.1
done
```

Wichtig ist dabei, dass bei dem zweiten und dritten Kommando die Datei *forremoval.txt* explizit genannt wird, obwohl diese in der Arbeitskopie bereits entfernt wurde. Natürlich muss man das beim "commit" nicht zwingend, solange es nicht stört, wenn weitere Dateien einbezogen werden. Etwas anders sieht es bei Verzeichnissen aus, da diese nicht unter der Versionskontrolle von CVS stehen. Stattdessen nutzt man Besonderheiten des Concurrent Version System aus. So lassen sich leere Verzeichnisse speziell behandeln: Soll ein Verzeichnis aus einem Projekt gelöscht werden, entfernt man zuerst alle Dateien aus diesem und führt dann im übergeordneten Verzeichnis ein "update" mit der "-p" Option aus. Diese sorgt dafür, dass leere Verzeichnisse reduziert, also aus der Arbeitskopie entfernt werden. Ist dieses erfolgt, kann das Verzeichnis als gelöscht betrachtet werden. Anzumerken ist, dass CVS bei einem normalen "update" keine neuen Verzeichnisse aus dem Archiv in die Arbeitskopie überträgt. Deshalb sollte in regelmäßigen Abständen mit der "-p" Option überprüft werden, ob neue Verzeichnisse im Archiv hinzugekommen sind.

Das Umbenennen von Dateien und Verzeichnissen empfindet man vielleicht als etwas aufwändig: Dateien werden gelöscht und unter neuem Namen dem Archiv wieder hinzugefügt[5]. Ähnliches gilt mit den genannten Besonderheiten ebenfalls für Verzeichnisse.

12.2.8 Markieren von Zwischenständen

Es wurde bereits erwähnt, dass ein wichtiger Vorteil einer Versionsverwaltung darin liegt, jederzeit Zugriff auf den älteren Stand einer Entwicklung zu haben. So kann man sehr einfach, beispielsweise sobald man einen stabilen Zustand seiner Software erreicht hat, das komplette Paket mit einem Tag markieren und ist nun immer in der Lage, genau diese Version wieder auszuchecken.

```
user@linux:~/ldap-projekt> cvs tag "version-0-2"
cvs server: Tagging .
T 00kap.tex
T allestex
T davedap-new.eps
T ...
```

An strategisch wichtigen Punkten in der Entwicklung eines Projektes macht es sinn, nicht nur einzelne Dateien, sondern wie im Beispiel ein gesamtes Verzeichnis zu markieren. Hier lautet die Kennzeichnung "version-0-2"[6]. Ab diesem Augenblick ist es möglich, über die Angabe dieses Tags alle obigen Dateien im jetzigen Zustand zu erhalten. Der Befehl "status" mit der Option "-v" zeigt einem ausführlich die entsprechenden Markierungen einer Datei.

[5] siehe die jeweiligen Beschreibungen zu den Sub-Befehlen `cvs remove ...; cvs add ...`

[6] Es bestehen einige Einschränkungen zur Benennung, so dürfen z.B. "." und ":" nicht in der Bezeichnung auftauchen, wobei `cvs tag ...` anmerkt, wenn ungültige Zeichen verwendet wurden.

```
user@linux:~/ldap-projekt> cvs status -v prog-entw.tex
===================================================================
File: prog-entw.tex     Status: Up-to-date

   Working revision:    1.2
   Repository revision: 1.2     /home/src/ldap-projekt/prog-entw.tex,v
   Sticky Tag:          (none)
   Sticky Date:         (none)
   Sticky Options:      (none)

   Existing Tags:
        version-0-2                     (revision: 1.2)
        start                           (revision: 1.1.1.1)
        dirk                            (branch: 1.1.1)
```

Möchte man beispielsweise die Version von *prog-entw.tex* bearbeiten, die mit "version-0-2" markiert worden ist, kann man folgendes eingeben:

```
user@linux:~/ldap-projekt> cvs update -r version-0-2 prog-entw.tex
cvs update:
U prog-entw.tex
```

Ebenso gelingt es, den Stand eines Projektes vom Vortag wiederherzustellen:

```
user@linux:~/ldap-projekt> cvs checkout -D yesterday project
```

12.3 Betrieb eines CVS-Servers

12.3.1 Installation

In den vorangegangenen Abschnitten wurde davon ausgegangen, dass irgendwo ein netter Mensch einen CVS-Server aufgesetzt hat, den man einfach nur mit seinen Projekten befüllen musste. Davon ist jedoch nicht immer auszugehen, weshalb sich der folgende Abschnitt mit der Installation und dem Aufsetzen eines solchen Dienstes befasst.

```
root@debian:# apt-get update
root@debian:# apt-get install cvs
```

12.3.2 Archive anlegen

Sobald die CVS-Programme auf dem Server installiert sind, kann man sie direkt als Client benutzen, um auf entfernte Archive zuzugreifen, was bereits beschrieben wurde. Wenn man jedoch Revisionen auf dieser Maschine anderen zur Verfügung stellen will, muss ein Archiv angelegt werden. Dieses geschieht wiederum mit dem bereits hinlänglich bekannten "cvs"-Kommando:

```
root@linux:~ # cvs -d /export/cvs/repository init
```

Hierbei ist die Angabe */export/cvs/repository* ein Pfad zu der (beliebigen) Stelle, an der das Archiv erzeugt werden soll. Hierfür sind die notwendigen Schreibrechte vonnöten, die eventuell "root"-Privilegien erfordern. Es mag ein wenig gegen die Intuition sein, dass der Ort des neuen Archivs vor dem "init" Sub-Befehl anstatt danach angegeben wird, doch durch die Verwendung der "-d"-Option wird die Konsistenz mit den anderen CVS-Kommandos gewährleistet. Der Befehl gibt nach seiner Ausführung keine Meldung zurück.

12.3.3 Netzwerkverbindungen

In den meisten Beispielen wurde davon ausgegangen, dass ein CVS-Archiv über ein Netzwerk auf einer entfernten Server angesprochen wurde. Wenn ein entfernter CVS-Client die ":pserver:"-Methode verwendet, um sich mit einem Archiv zu verbinden, kontaktiert der Client einen festgelegten Port auf dem Server. Default ist hier Port-Nummer 2401. Dieser kann natürlich geändert werden, dann ist jedoch beim Client-Aufruf die Angabe des abweichenden Ports erforderlich, ein Verfahren, das von den meisten Netzwerkprogrammen bekannt sein dürfte.

Der Versionsserver wartet nicht wirklich auf Verbindungen zu diesem Port. Dieses übernimmt der Internet-Super-Daemon **(x)inetd**, der CVS startet, sobald er eine Verbindungsanfrage eintrifft. Hierzu ist die Konfigurationsdatei des **(x)inetd** entsprechend anzupassen. Damit sind eingehende Verbindungen sichergestellt. Nun möchte man meistens spezielle CVS-Passwörter - getrennt von den normalen Benutzerpasswörtern - anlegen wollen. So lassen sich Zugriffe ausschliesslich auf das Archiv realisieren.

12.3.4 Authentifizierung

Die CVS Passwortdatei befindet sich im Archiv in *CVSROOT/passwd*. Sie wird nicht standardmäßig beim Aufruf von `cvs init` erzeugt, da das Versionsverwaltungssystem nicht weiss in welchem Modus es später betrieben wird. Auch wenn die Passwortdatei erzeugt wurde, kann CVS nicht wissen, welche Benutzernamen und Passwörter angelegt werden sollen. Diese müssen manuell angelegt werden. Hier ist eine Beispiel-Datei für *CVSROOT/passwd*:

```
user:bKaASD5ezmhOo
anonymous:XPiZZe45smnk
test:bLa12IOg3JJ:pubcvs
```

Das Format ist einfach und verwendet folgende Syntax:

```
<user>:<password crypt>:<optional system user>
```

Der zusätzliche Doppelpunkt, gefolgt von einem optionalen Systembenutzernamen, sagt CVS, dass Verbindungen, die mit "user" authentiziert sind, unter der abweichenden Systembenutzerkennung "optional system user" laufen sollen - anders gesagt, wäre diese CVS-Sitzung nur dazu in der Lage, Dinge im Archiv zu tun, die jemand tun könnte, der als "system user" angemeldet wäre.

Wenn kein Systembenutzername angegeben ist, muss "user" einem tatsächlich existierenden Benutzernamen auf dem System entsprechen. Die Sitzung wird mit den Zugriffsrechten dieses Benutzers gestartet. In beiden Fällen sollte das verschlüsselte Passwort nicht gleich dem tatsächlichen Login-Passwort des Benutzers sein. Es sollte ein unabhängiges Passwort sein, das nur für CVS pserver-Verbindungen verwendet wird. Das Passwort wird dabei mit denselben Algorithmen wie die Standard-Unix-Systempasswörter in */etc/passwd* verschlüsselt.

12.4 Debugging

ltrace dient zum Verfolgen von Bibliotheksaufrufen zum Debugging von Programmen strace kann zum Verfolgen von Systemaufrufen und zum Debugging von Programmen verwendet werden. An einen bereits laufenden Prozess kann sich **strace** mit der Kommandooption '-p PID' und der angehängten Nummer des zu überwachenden Prozesses anhängen. Voraussetzung hierfür sind jedoch Systemadministrator-Privilegien.

Kapitel 13

Benutzerverwaltung unter Linux

13.1 Einführung

In sehr grossen Netzen mit vielen gleichartigen Maschinen und vielen Benutzern, die sich an fast allen Maschinen anmelden können, ist die klassische dezentrale Verwaltung von Benutzern und Resourcen nicht mehr leistbar. Hierfür ist es notwendig alle Maschinen, die Dienste für normale Benutzer[1] bereitstellen, geeignet in zentrale Strukturen einzubinden.

Im lokalen Netzwerk soll üblicherweise allen Benutzern eine überall gleich konfigurierte Umgebung zur Verfügung stehen. Die Benutzer finden ein einheitliches Home-Verzeichnis vor und verwenden überall den gleichen Account, für den das gleiche Passwort eingestellt ist. Hierzu muss mann die "lebenswichtigen" Daten wie Account-Informationen und Home-Verzeichnisse der Benutzer zwischen allen Hosts synchronisieren.

13.2 Der Klassiker zentraler Benutzerverwaltung

Fast schon historisch ist das seit Beginn der 90er Jahren dienende Network Information System (NIS), Standardinformationen in großen Unix-Netzwerken zentral zu verwalten. Neben den Account-Informationen lassen sich zusätzliche Dateien abgleichen, wie die */etc/hosts*, */etc/services* oder */etc/groups*.

Die Namensauflösung erledigt heutzutage nicht mehr das NIS, sondern übernimmt das Domain Name System. Einen zentralen DNS-Server für das lokale LAN aufzusetzen, gehört inzwischen zum Standardrepertoire eines Linux-Administrators. Die DNS-Client-Einrichtung läßt sich relativ problemlos den DHCP-Clients[2] überlassen.

NIS ist traditionell noch stark verankert, da es lange Zeit konkurrenzlos existierte. Deshalb lassen sich zentrale NIS-Daten bereits mit der Kernel-Command-Line übermitteln. Die NIS-Domain wird wie der Rechnername direkt im */proc*-Verzeichnis des Kernels gespeichert und läßt sich mittels `cat /proc/sys/kernel/domainname` oder

[1] keine speziellen Systembenutzer, wie z.B. "named" für DNS oder "gdm" für das grafische Login via Displaymanager

[2] **dhclient** welches **dhclient-script** triggert vom ISC oder **dhcpcd** und **pump** welche direkt auf den Konfigurationsdateien arbeiten

einfach mit dem Kommando **domainname** auslesen. Der Systemadministrator kann dieses Kommando ebenfalls verwenden, um den Domainnamen festzulegen.

Obwohl es der Name nahe legt, sollte man den NIS-Namen nicht mit dem Domainnamen des DNS verwechseln und nicht nur aus Sicherheitsgründen einen anderen String wählen. Der Zugewinn an Sicherheit darf dabei jedoch nicht überschätzt werden, da jeder nochmale angemeldete Benutzer auf einem im NIS befindlichen Rechner in der Lage ist, die NIS-Informationen auszulesen. Ein einfaches `ypcat passwd` liefert alle Einträge der durch NIS erweiterten */etc/passwd* mit den dazugehörenden verschlüsselten Passwörtern. Weitere einfache Sicherheitsmaßnahmen sind die Beschränkung des Zugriffs in der */etc/ypserv.conf*. Die Sicherheitsgewinne sind jedoch nicht sehr hoch, wenn auch potenziellen Angreifern das Leben etwas erschwert wird.

Lange Zeit standen im Sinne der GPL für Unix und Linux nur die Remote-Authentifizierungsmechanismen des NIS zur Verfügung . NIS wurde deshalb in den zentralen Authentifizierungsrouten der C-Bibliothek direkt verankert. Mit der Einführung des Name Service Switches, der mittels */etc/nsswitch.conf* eingestellt wird, wurde eine zusätzliche Abstraktionsschicht eingeführt, die andere Authentifizierungsverfahren bequem einbinden läßt.

Die einfache Datenbank-Zugriffseinrichtungen von NIS kann man verwenden, um Informationen, wie sie die Dateien passwd und shadow enthalten, an alle Hosts im Netzwerk zu verteilen. NIS basiert wie NFS auf Remote Procedure Calls (RPC). Deshalb wird zusätzlich der Portmap-Dienst (**portmap**) benötigt, der noch vor dem NFS- und NIS-Server zu starten ist. NIS besteht aus einem Server, einer Bibliothek für die Client-Seite und verschiedenen administrativen Tools.

NIS hält seine Datenbank-Informationen in sogenannten Maps, wie die Zuordnungen genannt werden. Diese speichern die Key-Value-Paare in Hashtables. Die Zuordnungen hält der zentrale Host vor, auf dem der NIS-Server läuft. Clients können die Informationen dann von diesem Server über verschiedene RPC-Calls abrufen.

Die Maps selbst werden normalerweise aus Master-Textdateien wie */etc/passwd* und */etc/shadow* generiert. Dieses wird durch ein Makefile festgelegt, welches mit dem NIS-Server-Paket mitinstalliert wird. In diesem Makefile kann der Adminitrator festlegen, welche Maps generiert werden sollen.

Der Serverprozess heisst **ypserv** und wird durch ein Runlevelskript gesteuert. Dieses übernimmt das Setzen der NIS-Domain. Die durch das Makefile generierten Datenbanken finden sich je nach Distribution unterhalb von */var/lib/yp* oder */var/yp*. Das tatsächliche Erzeugen der Maps geschieht durch den Aufruf von make in diesem Verzeichnis.

Aus manchen Dateien werden mehrere Maps generiert, jeweils eine pro Suchschlüssel. Zum Beispiel kann die Datei */etc/hosts* nach einem Hostnamen ebenso wie nach einer IP-Adresse durchsucht werden. Eine klassische Suchabfrage geschieht z.B. mittels *ypcat hosts*.

Diese Einrichtung stammt noch aus Zeiten, in denen man NIS statt DNS zur Namensauflösung der Rechner verwendete. Entsprechend werden zwei NIS-Maps daraus erzeugt: *hosts.byname* und *hosts.byaddr*. Die folgende Aufstellung zeigt übliche Maps sowie Dateien, aus denen sie generiert werden:

```
Master-Datei Mapfile(s)
/etc/hosts hosts.byname, hosts.byaddr
```

```
/etc/networks networks.byname, networks.byaddr
/etc/passwd passwd.byname, passwd.byuid
/etc/group group.byname, group.bygid
/etc/services services.byname, services.bynumber
/etc/rpc rpc.byname, rpc.bynumber
/etc/protocols protocols.byname, protocols.bynumber
/etc/aliases mail.aliases
```

Da NIS ein recht lange bekanntes Konzept zur zentralisierten Rechnersteuerung ist, sind die entsprechenden DHCP-Variablen bereits fest definiert. Für die Client-Konfiguration wird eine Liste von NIS-Servern (”option nis-servers IP1, IP2;”) übermittelt, welche die Grundlage für die Konfigurationsdatei *yp.conf* liefern.

```
# Syntax:
# ypserver <Name_of_ypserver>
ypserver 192.168.100.20
```

Das Kommando `domainname name_der_nis_domain` trägt den NIS-Domain-String, der in der DHCP-Variablen ”option-domain ”dxs;” enthalten ein. Alternativ kann dieser Eintrag auch direkt nach */proc/sys/kernel/domainname* geschrieben werden.

Nachdem der Server gestartet wurde und die Client-Konfigurationsdatei entsprechend angelegt wurde, kann man durch das Absetzen des Kommandos *ypcat passwd* überprüfen, ob der Dienst funktioniert. Ebenso kann das Kommando `rpcinfo -p` aufgerufen werden:

```
s01: 01:22:13 /home/wtest $ /usr/sbin/rpcinfo -p
   Program Vers Proto   Port
    100000    2   tcp    111  portmapper
    100000    2   udp    111  portmapper
    100007    2   udp    802  ypbind
    100007    1   udp    802  ypbind
    100007    2   tcp    807  ypbind
    100007    1   tcp    807  ypbind
```

Auf der Kommandozeile erfolgt die Ausgabe der Passwortliste, die aus den Authentifizierungsdateien des Servers zusammengeführt wurde, andernfalls eine Fehlermeldung. Server und Client dürfen dabei durchaus auf einer einzigen Maschine laufen.

Der Administrator muß bei jeder Änderung der Orginaldateien (z.B. der Datei */etc/hosts*) “per Hand” die entsprechenden NIS-Mappen updaten (durch Starten von make im Verzeichnis “/var/yp/<YP-Domainname>”). Das betrifft alle Zuordnungen, deren Orginaldateien nur durch ”root” veränderbar sind. Die einzige Ausnahme bildet die Änderung des Paßwortes. Da diese Änderung vom

Nutzer vorgenommen wird, muß das Update der NIS-Mappe auf dem Server automatisch ablaufen. Der RPC-Daemon **rpc.yppasswdd** ist auf Serverseite und das Programm **yppasswd** auf Clientseite für die Änderung eines Paßwortes in einer NIS-Domain zuständig. NIS ist ähnlich wie NFS, was die Sicherheit anbetrifft nicht mehr auf dem aktuellen Stand: Die Daten aus */etc/passwd* und */etc/shadow* werden in einer gemeinsamen Map zusammengeführt und es läßt sich nicht verhindern, dass normale

Benutzer die Map auslesen und versuchen, mit Brute-Force-Attacken Passwörter zu ermitteln. Dieses wird auf modernen Systemen durch die Trennung in beide genannten Dateien verhindert. Die etwas sichereren Weiterentwicklungen zu NIS+ sind weniger verbreitet und ebenfalls nicht in der Lage, das Sicherheitskonzept entscheidend zu verbessern.

NIS verwendet keine verschlüsselten Verbindungen zum Übermitteln seiner Daten. Die Verwendung von NIS empfielt sich aus Sicht der Autoren nur, wenn bereits eine komplette Infrastruktur dieses Verfahren verwendet und Clientmaschinen ohne großen Aufwand in diese aufgenommen werden sollen. Die Daten zur Einrichtung der Clients lassen sich mit den entsprechenden bereits fest definierten DHCP-Optionen übertragen.

13.3 Pluggable Authentication Modules

13.3.1 Die Fuktionsweise

Die meistgenutzte Form der Authentifizierung an einem Standalone Linux-Rechner erfolgt immer noch durch die Benutzername- und Passworteingabe über die Tastatur. Die Überprüfung auf die Korrektheit der Eingaben werden von der C-Bibliothek übernommen. Für den geschilderten einfachen Fall stellt dieses noch keine besonderen Anforderungen an diese, da einfach Einträge aus zwei Dateien, */etc/passwd* und */etc/shadow* ausgelesen werden und mit den Eingaben verglichen werden müssen. Da Linux aber ein netzwerkfähiges multiuser und -prozessfähiges Betriebssystem ist, existieren eine ganze Reihe verschiedene Services, die in der einen oder anderen Weise Zugang auf das entsprechendem System erlauben.

So bietet neben dem klassischen Login, der X11-Displaymanager, der SSH-Dämon lokal oder entfernt den Zugriff auf ein System. Für viele andere Dienste wie Concurrent Version System (CVS), das File Transfer Protokoll (FTP) oder der Webserver ist es häufig sinnvoll neben den diensteigenen Implementierungen auch eine Authentifizierung gegen eine Standardsystemschnittstelle zu besitzen. Alle Systemzugangsdienste müssen nun normalerweise vor Erlauben des Zugriffs auf den entsprechenden Dienst oder der Gewährung von Systemzutritt feststellen, wer der Benutzer ist und ob diesem Benutzer Zugang oder Service gewährt werden darf.

Die Feststellung des Benutzers erfolgt üblicherweise in der eingangs genannten Form. Jede Anwendung oder Dienst muss hierfür die entsprechenden Prozeduren implementieren. Neben dem hohen Aufwand einer Mehrfachumsetzung des identischen Procederes steigt auch die Fehleranfälligkeit der Programmierung. Ändert sich der Authenifizierungmechanismus tritt das nächste Problem auf, da nun normalerweise ein Austausch der Applikation oder die Änderung des Quellcodes anstehen. Soll nun statt der klassischen Benutzername-Passwort-Kombination eine Chipkarte mit PIN eingesetzt werden, wären jeweiliges Neuübersetzen und Installieren der betroffenen Applikationen unumgänglich. Steht eine Ergänzung des Verfahrens durch eine zeitgesteuerte Komponente oder die Erweiterung der Authentifizierung um ein Einmal-Passwort an, müssten wieder alle betroffenen Programme angefasst werden.

Die Anforderungen an die Benutzerverwaltung unterscheiden sich je nach eingesetztem Client: Maschinen welche im Server-based Umfeld eingesetzt werden stellen lediglich die Benutzerschnittstelle zum Arbeiten auf dem Server dar und benötigen keine besonderen Mechanismen zur Authentifizierung von Benutzern. Dieses stellt sich auf Arbeitsstationen anders dar: Hier soll der Benutzer sich direkt an der Maschine anmelden können und seinen gewohnten Desktop wiederfinden, egal vor welchem Rechner er gerade arbeitet. Das setzt jedoch das Vorhandensein der Authentifizierungsinfrastruktur auf jeder Maschine voraus. Hierfür wurde ürsprünglich NIS entwickelt und in die klassische Unix-Authentifizierungsstruktur eingebaut, welches lange Zeit den Standard einer verteilten Benutzerauthentifizierung und -verwaltung im Unix-Umfeld bildete.

Mit den heute üblichen heterogenen Umgebungen, deren vielfältigen Authentifizierungsanforderungen (z.B. die Anmeldung an einem Radius-Server für Modem- oder ISDN-Einwahl, Freischaltung und Sperrung bestimmter Dienste für einzelne Benutzergruppen) und den gestiegenen Sicherheitsforderungen konnte NIS nicht mehr mithalten. Neben den inzwischen recht umfänglichen Daten, die die Verwaltung eines Benutzers erfodert, kommen weitere Wünsche, wie die nach einer zentralisierten Organisation von Rechner-Pools hinzu. Betreiber grosser Rechnernetzwerke mit heterogener Ausstattung werden darüberhinaus kaum von ihren Benutzern verlangen können, dass diese sich für jedes von ihnen benutzte System eine eigene Kennung und ein eigenes Passwort merken. Weiterhin wäre es zudem schnell problematisch, wenn zum Beispiel mittels Secure Copy (SCP) oder Remote Sync die Benutzer- und Shadowdateien auf eventuell viele hundert Systeme verteilt werden müssten. Eine zeitnahe Reaktion auf eine Passwortänderung eines könnte im Extremfall einen Sturm an Netzwerkaktivität auslösen, der nur für die Eventualität vorsorgt, dass sich der Benutzer an einer weiteren Maschine anmelden muss.

Die Nachteile der klassischen Lösung lassen sich wie folgt zusammenfassen:

1. Es bedeutet sehr viel Arbeit, da jede Anwendung Identifikation und Authentifizierung des sich anmeldenden Nutzers selbst übernimmt.

2. Für jeden neuen Authentifizierungssupport muss jedes Programm angepasst werden.

3. Die gesamte Prozedur ist ziemlich fehleranfällig und bei jedem auftretenden Fehler muss jedes Programm geändert werden.

4. Die Anwendungsentwicklung kümmert sich selten um die Entwicklungen im Bereich der Sicherheitstechniken.

5. Der Sourcecode muss verfügbar sein, um alternative Authentifizierungsmethoden in eine Anwendung einbauen zu können.

PAM realisiert deshalb einen völlig neuen Ansatz. Die einzelnen Applikationen benötigen selbst keine Routinen mehr zur Authentifizierung. Stattdessen benutzen die Programme Funktionen aus einer Bibliothek, die PAM zur Verfügung stellt. Die Funktionen benutzen die passenden PAM-Module und liefern das Ergebnis an die Applikation zurück. Da die PAM-Bibliothek ihrerseites keine Informationen darüber besitzt, auf welche Weise Benutzereingaben eintreffen, muss sich hierum die Applikation kümmern. Das ist für diese normalerweise kein Problem, da sie Funktionen, z.B. zum Lesen von der Rechnerkonsole, aus einer Netzwerkverbindung oder vom Displaymanager-Login, üblicherweise implementiert hat.

Zusätzlich wurde PAM gegenüber den klassischen Ansatz im Funktionsumfang erweitert. Neben dem "Authentication Management" werden nun auch die Aufgaben "Account Management", "Session Management" und "Password Management" von PAM übernommen. Jedes PAM-Modul muss mindestens einen, kann aber auch mehrere dieser Jobs abdecken. Diese Aufteilung findet sich in den entsprechenden Konfigurationsdateien wieder.

Ein entscheidendes Problem für einige Anwendungen hat PAM jedoch: Es ist rein passiv und muss immer von einer Applikation aufgerufen werden. Wenn ein Benutzer automatisch nach dem Durchziehen seiner Chipkarte oder dem Auflegen des Fingers auf einem biometrischen Leser eingeloggt werden soll, muss das regelmäßige Pollen und Anstossen der Authentifizierung die Applikation übernehmen. PAM kann nicht von sich aus Authentifizierungsgeräte pollen und externe Programme anstossen.

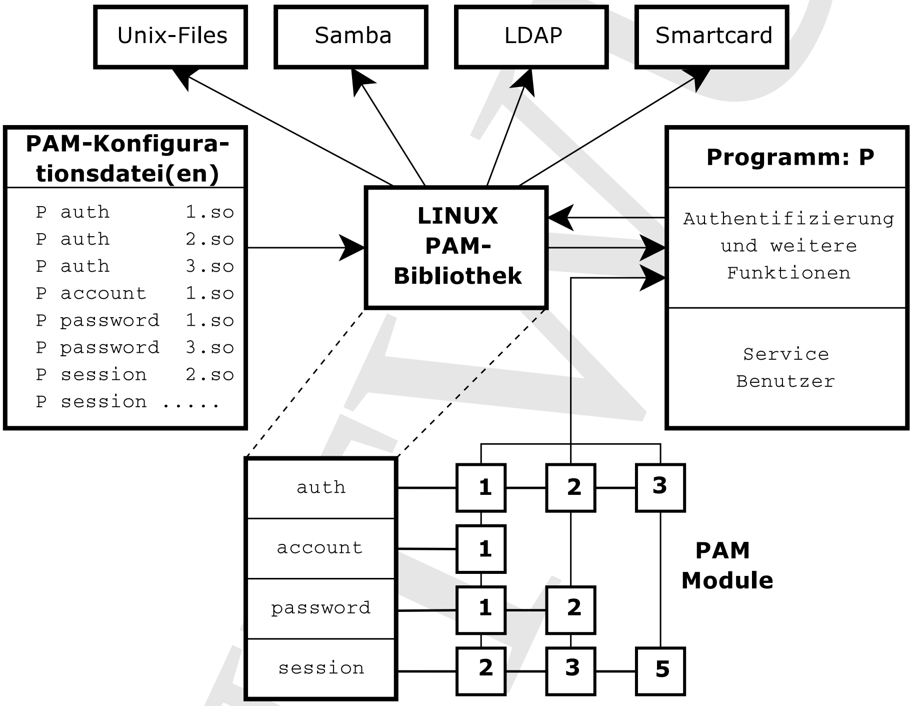

Abbildung 13.1: Die Funktionsweise von PAM

Die schematische Darstellung zeigt, wie der Kommunikationsfluss zwischen den Applikationen und PAM funktioniert. Die PAM-Bibliothek wird zur Laufzeit des entpsrechenden Programmes oder Dienstes geladen. Das PAM-Modul kommuniziert zum einen mit der PAM-Bibliothek, um seine Parameter auszulesen, zum anderen mit der Applikation, um an die Benutzerdaten wie Accountname und Passwort zu gelangen.

13.3.2 Dateien und Verzeichnisse

Die PAM-Bibliothek wird normalerweise als eigenes Paket in der jeweiligen Linux-Distribution ausgeliefert. Spezielle Module, die weitere Bibliotheken benötigen, wie das LDAP-Modul finden sich in separaten Paketen, die bei Bedarf zusätzlich installiert werden müssen.

Bei der Installation von PAM werden drei Verzeichnisse angelegt: */lib/security* nimmt die Bibliotheksdateien auf. Dieses Verzeichnis enthält alle aktuell auf einem System installierten Standard-PAM-Module:

```
wtest@s01:~ >ls /lib/security
pam_access.so      pam_lastlog.so      pam_shells.so
pam_afs.krb.so     pam_ldap.so         pam_smbpass.so
pam_afs.krb.so.1   pam_limits.so       pam_stress.so
pam_afs.so         pam_listfile.so     pam_tally.so
pam_afs.so.1       pam_localuser.so    pam_time.so
pam_chroot.so      pam_mail.so         pam_unix.so
pam_cracklib.so    pam_make.so         pam_unix2.so
pam_debug.so       pam_mkhomedir.so    pam_unix_acct.so
pam_deny.so        pam_motd.so         pam_unix_auth.so
pam_devperm.so     pam_nologin.so      pam_unix_passwd.so
pam_env.so         pam_passwdqc.so     pam_unix_session.so
pam_filter         pam_permit.so       pam_userdb.so
pam_filter.so      pam_pwcheck.so      pam_warn.so
pam_ftp.so         pam_resmgr.so       pam_wheel.so
pam_group.so       pam_rhosts_auth.so  pam_winbind.so
pam_homecheck.so   pam_rootok.so       pam_xauth.so
pam_issue.so       pam_securetty.so
```

Einige Module verfügen über eigene Konfigurationsdateien, die ihr generelles Verhalten unabhängig von der aufrufenden Applikation steuern. Diese sind in einem gleichnamigen Verzeichnis diesmal unterhalb */etc* abgelegt:

```
wtest@s01:~ >ls /etc/security
access.conf  group.conf   pam_env.conf      pam_unix2.conf
chroot.conf  limits.conf  pam_pwcheck.conf  time.conf
```

Für die Konfiguration der PAM-Bibliotheken zu den einzelnen Diensten gibt es zwei Möglichkeiten. Entweder sind alle Informationen in einer gemeinsamen Datei abgelegt, der */etc/pam.conf*. Oder es befindet sich für alle entsprechenden Dienste und Applikationen eine eigene Konfigurationsdatei unterhalb von */etc/pam.d*, was wegen der Fülle der Dienste bei den meisten Distributionen Standard ist. Letzteres hat den Vorteil, dass nicht mit jedem Programm welches nachinstalliert wird, die */etc/pam.conf* geändert werden muss. Es wird stattdessen nur eine zusätzliche Datei in */etc/pam.d* benötigt.

```
wtest@s01:~ >ls /etc/pam.d
chage cups          login  ppp     shadow sudo    xdm          zebra
chfn  gdm           other  rpasswd sshd   useradd xlock
chsh  gdm-autologin passwd samba   su     webmin  xscreensaver
```

Die Menge der Dateien in diesem Verzeichnis leitet sich aus der Zahl der PAM-fähigen Services ab, die auf der Maschine installiert sind. Für die meisten Dienste existiert aus einem weiteren Grund es eine eigene Konfigurationsdatei: Login sollte feststellen, ob der Systemadministrator sich von einem sicheren Terminal aus anmeldet. Bei der Benutzung von SSH macht dieses keinen Sinn, besonders wo in der SSH-Konfiguration festgelegt wird, ob sich "root" überhaupt per SSH einloggen darf.

Beim ihrem Start legt jede Programm fest, welche Konfigurationsdatei verwendet werden soll. Sinnvollerweise hat diese Datei den gleichen Namen wie der Dienst selbst. Leider stimmt das jedoch nicht immer, so dass man Netzwerkdienste findet, bei denen das abschliessende "d" fehlt, wie bei "ppp" oder die abweichend bezeichnet sind, wie "samba". Als Rückgriff für fehlende Konfigurationsdateien verwendet PAM die Datei */etc/pam.d/other*. Da jedoch der Systemadministrator nicht weiss, welcher Dienst eventuell diese Datei benutzt, sollte sie restriktiv eingestellt sein und eine Meldung im Syslog bei Verwendung hinterlassen.

Generell haben alle Konfigurationsdateien einen ähnlichen strukturellen Aufbau.

```
# (dienst)  modultyp    modulsteuerung  modulpfad    argumente
#
( gdm )     auth        sufficient      pam_unix2.so   nullok
( gdm )     auth        required        pam_ldap.so    use_first_pass
( gdm )     account     required        pam_unix2.so
( gdm )     password    required        pam_unix2.so
( gdm )     session     required        pam_unix2.so
( gdm )     session     required        pam_env.so
( gdm )     session     required        pam_devperm.so
```

Der "dienst"-Eintrag entfällt, wenn für jeden Dienst eine eigene Datei angelegt ist. Der "modultyp" bestimmt über die Management-Funktion, die ein Eintrag erfüllt. Es sind vier Modultypen definiert:

1. auth - Module in dieser Kategorie dienen der Benutzeridentifizierung. Dieses kann durch klassische Benutzername/Passwort Abfragen, aber auch durch biometrische Verfahren, Smartcard mit PIN oder ähnliches erfolgen. Voraussetzung sind entsprechend vorhandene Schnittstellen der angeschlossenen Geräte, wie Smartcard-Leser. In diesen Bereich fallen auch spezielle Module, die Benutzerkennung und Passwort abgreifen, um sie für einen authentifizierten Mount-Prozess des Home-Verzeichnisses z.B. von einem Samba-Server einzusetzen oder damit ein AFS-Token zu holen.

2. account - Diese Module verwalten den Zugriff auf Accounts nach erfolgter Anmeldeprozedur. Auf diese Weise kann der Zugriff auf einen Dienst abhängig von der Uhrzeit oder dem Wochentag gesteuert werden.

3. password - Diese Module werden dazu eingesetzt, um Passwort- oder Token-Aktualisierungen zu steuern. Verwendet das **passwd**-Kommando die PAM-Bibliotheken, lässt sich festlegen, welche Passwörter das Programm ändert und akzeptiert. Gleichzeitig kann eine Passwortänderung netzwerktransparant realisiert werden.

4. session - Module diesen Typs verwalten Einstellungen für die Sitzung des Benutzers. Hiermit kann eine Abrechnung über die im System verbrachte Zeit

eingerichtet werden. Ebenso fällt in diese Kategorie das Setzen von Limits oder Zugriffsberechtigungen auf Unix-Devices sowie das Mounten von Verzeichnissen. Wird für letzteres ein Passwort benötigt, sollte dieser Vorgang zum ”auth”-Modul verlagert werden.

In der Spalte ”modulsteuerung” wird festgelegt, wie PAM sich verhält, wenn ein Modul mit Erfolg- oder Fehlermeldung beendet wird. Hierfür sind vier Standardbedingungen definiert, die bei Bedarf jedoch verfeinert aufgegliedert werden können. Dadurch lassen sich PAM-Module stapeln. Diese Stapel, bei PAM als ”stacking” bezeichnet, werden in der Reihenfolge der Auflistung abgearbeitet. Unter bestimmten Bedingungen können weiter unten stehende Module auch nicht erreicht werden. Von diesem Vorgang der Abarbeitung bekommt das aufrufende Programm nur das zusammengefasste Endergebnis mitgeteilt. Dieses enthält einen Statusbericht, ob ein Erfolg ”success” oder Misserfolg ”fail” eingetreten ist.

1. required - Dieses Moduls muss mit dem Status ”success” beendet werden, damit das zusammengefasste Ergebnis aller Module dieses Typs erfolgreich sein werden kann. Ein Misserfolg dieses Moduls zeigt sich erst, nachdem alle Module dieser Kategorie durchlaufen wurden.

2. requisite - verhält sich ähnlich wie ”required”. Im Unterschied wird die Kontrolle sofort an die Anwendung zurückgegeben und der Rückgabewert ist der des ersten fehlgeschlagenen Moduls mit ”required”- oder ”requisite”-Steuerung.

3. sufficient - Liefert dieses Modul die Statusmeldung ”success” zurück, genügt dieses für PAM zur positiven Gesamtmeldung, solange keine vorher eingetragenen Module mit ”required” oder ”requisite” fehlgeschlagen sind. Anschliessend werden keine weiteren Module dieses Typs mehr aufgerufen. Das kann jedoch zum Problem werden, wenn z.B. ein folgendes Modul mit dem gültigen Passwort ein Mount ausführen soll. Liefert ein Modul mit ”sufficient”-Steuerung keine Erfolgsmeldung, werden die nachfolgenden Module aufgerufen. Der Misserfolg eines ”sufficient” Moduls bedeutet nicht das Fehlschlagen der Gesamtfunktion.

4. optional - Bei diesem Modul entscheiden Erfolg oder Misserfolg nicht über den Gesamtrückgabewert der Funktion. Ausnahmsweise geschieht dieses nur, wenn alle anderen Module im Stapel keinen definitiven Erfolg oder Misserfolg melden. Das Modul sollte mit Vorsicht benutzt werden, wie später in einem Beispiel gezeigt wird.

Je nach Art des Programmes oder Dienst ruft dieses bei Bedarf einen der vier Modul-Typen auf. Die PAM-Bibliothek geht nun alle Module, die in der Konfigurationsdatei zur dieser Kategorie vermerkt sind, nacheinander durch. Nach Erreichen des letzten Moduls oder dem vorzeitigen erfolgreichen Abbruch bei ”sufficient” liefert sie den Gesamtstatus an die aufrufende Applikation zurück.

Der Modulpfad in der nächsten Dateispalte legt fest, wo nach der entsprechenden PAM-Bibliothek gesucht wird. Standardmässig werden Module in */lib/security* erwartet. In diesem Fall reicht die Angabe des Modul-Namens. Andernfalls gilt das klassische Unix-Procedere - Pfadnamen mit ”/” beginnend werden als absolute Pfade interpretiert. Fehlt "/" wird ein Pfad relativ zu */lib/security* angenommen. So lassen sich selbst hinzugefügte Module von den standardmässig installierten unterscheiden.

Die letzte Spalte "Argumente" ist optional und enthält nur bei einigen Modulen einen Eintrag. Argumente werden als Liste angegeben, die durch Leerzeichen getrennt werden. Es gibt einfache Flags, wie "nullok" oder "use_first_pass" und Zuweisungen, z.B. "strict=false". Kommen Leerzeichen in einem Argument vor, ist Argument in eckigen Klammern anzugeben. Sollte wiederum eine eckige Klammer im Argument vorkommen, kann diese durch Quoting mit Backslashes eingetragen werden. Generell sind Argumente nicht vorgegeben, sondern vom jeweiligen Programmierer des Moduls vorgesehen. Es gibt von fast allen Modulen verstandene Argumente: "debug" liefert Diagnosemeldungen an den Systemlog-Dienst, "no_warn" unterdrückt diese. Authentifizierungs- und Passwortmodule kennen darüberhinaus "use_first_pass", welches veranlasst zu versuchen das Passwort des vorhergehenden auth Moduls zu verwenden. Schlägt es fehl, meldet das Modul den Status "fail" zurück. Ähnlich wirkt für Authentifizierungsmodule "try_first_pass". Das Module versucht das Passwort des vorhergehenden Moduls zu verwenden. Wenn dies fehlschlägt, fordert es den Benutzer zur Passworteingabe auf.

Falsch eingetragene Argumente in einer Zeile werden ignoriert, falsch formatierte Zeilen der Konfigurationsdatei führen jedoch dazu, dass das betroffene Modul eine Fehlermeldung ("fail") als Statuswert zurückgibt.

13.3.3 Ein Modul für jeden Zweck

Inzwischen gibt es für PAM fast alle erdenklichen Module, für einige Aufgaben existiert auch mehr als eine Implementation. Ein großer Teil davon wird bereits mit dem Standardpaket mitgeliefert. Spezifische Module, z.B. für die LDAP-Authentifizierung und die Beschaffung von AFS-Tokens wärend des Login-Vorganges werden einzeln oder mit ihrem entsprechenden Softwarepaket installiert.

- pam_access - stellt eine Art "Access Control Lists" zur Verfügung. Das Modul erwartet eine Konfigurationsdatei */etc/security/access.conf*, die aber mittels übergebenem Argument auch an anderer Stelle gefunden werden kann. In dieser Datei lassen sich Listen der Art "permission : users : origins" pflegen.

- pam_afs - ist ein Authentifizierungsmodul. Es kann für zwei Szenarien verwendet werden. Entweder findet die Authentifizerung des Benutzers durch ein anderes Modul statt, dann übernimmt "pam_afs" lediglich das Holen des Tokens vom AFS-Server. Oder "pam_afs" übernimmt beide Aufgaben. Eine ausführliche Beschreibung zu AFS findet sich im Kapitel zu den Dateisystemen.

- pam_chroot - mit diesem Modul werden sowohl Account-, Session- und Authentifizierungskomponenten abgedeckt. Es dient dazu einen normalen Benutzer beim Login zur Erhöhung der Systemsicherheit in einen "Change-Rooted"-Bereich zu versetzen.

- pam_cracklib - stellt Routinen zum Festlegen von Passwort-Richtlinien in der Kategorie "Password" zur Verfügung. Es erfordert die zusätzliche Installation der "Cracklib" und des dazugehörenden Wörterbuchs.

- pam_deny - verhindert generell den Zugriff durch eine entsprechende Rückmeldung an die aufrufende Applikation und kann somit als Default verwendet werden, wenn PAM auf "other" zurückfällt, weil für einen Dienst keine Konfiguration existiert.

- pam_devperm - kann dazu verwendet werden, die Zugriffsrechte auf die Benutzerhomeverzeichnisse und einige ".*" Dateien zu überprüfen und Warnungen ausgeben, wenn die Rechte zu großzügig gesetzt sind.

- pam_env - kann zur Steuerung von Umgebungsvariablen verwendet werden. Es erfordert hierzu eine Konfigurationsdatei */etc/security/pam_env.conf*, die wieder per Modul-Argument auch woanders gefunden werden kann. Diese Datei kann festlegen welche DEFAULTs für eine Variable gelten sollen oder wie diese überschrieben (OVERRIDE) werden soll.

- pam_homecheck - implementiert ähnliche Funktionalität, wie "pam_devperm".

- pam_lastlog - kümmert sich als Modul der Session-Kategorie um die Datei */var/log/lastlog* und trägt dort den Benutzer ein. Es kann, wenn das nicht schon die Applikation erledigt, auch anzeigen, wann der Nutzer das letzte Mal angemeldet war.

- pam_ldap - ermöglicht die Benutzerauthentifizierung gegen eine LDAP-Datenbank. Je nachdem, wie dieses Modul kompiliert wurde, bezieht es seine LDAP-Konfiguration aus */etc/(openldap/)ldap.conf*. Mehr zur LDAP-Authentifizierung läßt sich im Kapitel zu LDAP nachlesen.

- pam_limits - erlaubt dem Systemadministrator festzulegen, wieviel System-Ressourcen wie Speicher und CPU-Zeit dem Benutzer maximal zur Verfügung gestellt werden. Diese Einschränkungen gelten nicht für User mit der ID 0. Die Konfigurationsdatei für dieses Modul ist */etc/security/limits.conf*.

- pam_mkhomedir - legt wie der Name schon vermuten läßt, bei Bedarf ein Home-Verzeichnis für den Benutzer an.

- pam_mktemp - kann als Teil des Session- oder Accountmanagements für angemelete Benutzer ein persönliches Temporärverzeichnis festlegen, wofür die Umgebungsvariablen TMPDIR und TMP gesetzt werden.

- pam_nologin - realisiert eine Überprüfung auf die Datei *pam_nologin*. Es stellt ausschliesslich Authentifizierungsfunktionalität zur Verfügung. Wenn diese existiert, erhalten normale Benutzer bis auf den Systemadministrator den Inhalt dieser Datei angezeigt und dürfen sich nicht anmelden.

- pam_permit - liefert immer ein fröliches "sucess", ein Modul welches nur unter sehr speziellen Bedingungen zum Einsatz kommen sollte.

- pam_pwcheck - stellt Hilfsfunktionen in der "password"-Kategorie für die Passwortänderung zur Verfügung und verwendet hierzu Einstellungen aus der */etc/login.defs*. Hier kann mittels Modul-Argument z.B. festgelegt werden, nach welchem Crypt-Verfahren ("bigcrypt", "blowfish", ...) das Passwort verschlüsselt wird.

- pam_rootok - ist ein Modul der Authentifizierungskategorie. Es kann für bestimmte Dienste, wie "su" ein Admin-Login ohne Passwort erlauben. Auf diese Weise kann der Root-Benutzer zu einem normalen User werden, ohne dessen Passwort wissen zu müssen.

- pam_securetty - stellt fest, von welchem Terminal (TTY) sich der Systemadministrator anmelden darf. Die Identifikation erfolgt über die Benutzer-ID 0. Dieses Modul überprüft, ob die Konfigurationsdatei */etc/securetty* als reine Textdatei vorliegt und nicht für jeden schreibbar ist. Die Datei enthält eine Liste von Devices von denen sich "root" aus anmelden kann. Dieses Modul bedient ausschliesslich die Auth-Kategorie.

- pam_unix - ist das Standard-Unix-Authentifizierungsmodul, welches ebenfalls Komponenten für "Session", "Password" und "Account" zur Verfügung stellt. Es benutzt hierfür die Standardaufrufe der Systembibliotheken. Üblicherweise werden diese Informationen aus */etc/passwd* und */etc/shadow* beschafft.

- pam_unix2 - realisiert eine Erweiterung des Standardmoduls, um z.B. auch NIS(+) zur Authentifizierung benutzen zu können.

- pam_warn - hat die Aufgabe Log-Output zu generieren und an den Syslog weiterzugeben. Es arbeitet in den Kategorien "Authentication" und "Password".

- pam_winbind - kommt mit der Samba-Suite mit und erlaubt eine Authentifizierung gegen Windows-Systeme.

Zusätzlich zu diesen mit dem Default-Paket oder in Zusatzpaketen verbreiteten PAM-Bibliotheken gibt es noch eine ganze Reihe weiterer Module, die für bestimmte Einsatzszenarien geschrieben wurden. Einige davon befinden sich noch im Beta-Stadium und sollten, nur mit einiger Vorsicht eingesetzt werden. Die Authentifizierung am Rechner ist eine der sicherheitsrelevanten Stellen, die Fingespitzengefühl erfordert. "pam_storepw" ist ein Passwort-Caching-Modul. Es arbeitet in der Auth-Kategorie und kann im Stapel mit anderen Authentifizierungsmodulen Benutzername und Passwort abgreifen, um sie in einem nur für "root" lesbaren Verzeichnis unterhalb von */var/run/pw* zu speichern. Das kann dann dazu benutzt werden von einem Automounter-getriggerten Skript Passwort-geschützte Verzeichnisse zu mounten. Auf diese Weise kann z.B. ein Samba-Home-Verzeichnis eingebunden werden. Dieses ist unter Sicherheitsgesichtspunkten aber eher als Notlösung einzustufen, da Passworte im Klartext ins Dateisystem geschrieben werden. Hier hilft eventuell "pam_mount" weiter und löst vielleicht ein Teil der Probleme, die zum Einsatz von "pam_storepw" geführt haben könnten. Es ist in der Lage verschiedene Arten passwortgeschützter Dateisysteme, wie auch lokale verschlüsselte Verzeichnisse einzubinden. Weiterhin gibt es eine Reihe zum Teil auch kommerzieller Module, die gegen MySQL, Smartcards mit RSA-Verschlüsselung und weitere Quellen authentifizieren können.

13.3.4 Eine Beispielkonfiguration für den GDM

Die nachstehende Datei */etc/pam.d/gdm* zeigt ein typisches Beispiel einer PAM-Konfiguration. Es werden vier Authentifizierungsmodule hintereinandergeschaltet:

Das erste Modul überprüft, wenn die UserID ungleich Null ist, ob eine Datei */etc/nologin* existiert. Es unterbindet in diesem Fall eine weitere Anmeldung, da das Fehlschlagen eines "required"-Moduls zum Gesamtergebnis "fail" beiträgt, welches PAM an die Applikation zurückmeldet. In der nächsten Zeile wird versucht den sich anmeldenden Benutzer gegen die Standard-Unix-Dateien zu authentifizieren. Gelingt dieses, werden keine weiteren Module mehr abgearbeitet. Schlägt dieser Schritt fehl, wird einfach das Modul verwendet. Klappt die Anmeldung des Benutzers gegen den LDAP-Server, wird das letzte Modul aufgerufen. Dieses macht keine Authentifizierung im klassischen Sinne, sondern schreibt Benutzername und Passwort in eine Datei.

```
#%PAM-1.0
auth     required       pam_nologin.so
auth     sufficient     pam_unix2.so
auth     required       pam_ldap.so     use_first_pass
auth     required       pam_storepw.so  use_first_pass
account  required       pam_unix2.so
password required       pam_unix2.so    #strict=false
session  required       pam_unix2.so    debug # trace or none
session  required       pam_devperm.so
session  required       pam_resmgr.so
```

Diese Konfiguration eignet sich für Umgebungen, in denen es bis auf den Administrator keine lokal eingetragenen Normalbenutzer gibt. Zuerst bei der Authentifizierung überprüft, ob eine Authentifizierung gegenüber den klassischen Unix-Dateien Erfolg hat. Dieses trifft nur bei "root" ein. Da der Root-Benutzer nicht im LDAP gespeichert ist und kein zentrales Home-Verzeichnis hat, darf er auch nicht mehr bei den auf "pam_unix2" folgenden Modulen vorbeikommen, welches durch "sufficient" unterbunden wird. Entscheidend in dieser Konfiguration ist, dass "pam_unix" nicht so konfiguriert ist, dass es selbst gegen LDAP authentifizieren kann. Dieses geschieht über die Konfigurationsdatei */etc/security/pam_unix2.conf*. Die Alternative, so dass LDAP gleich benutzt wird, ist hinter den Kommentarzeichen angegeben.

```
auth:      nullok    # auth:      use_ldap nullok
account:             # account:   use_ldap
password:  nullok    # password:  use_ldap nullok
session:   none      # session:   none
```

Das "pam_unix2" kommt zusätzlich als Account, Passwortänderungs und Session-Modul zum Einsatz. Die Accounting-Funktion dieses Moduls prüft anhand der Felder in der Shadow-Datei, ob der Account noch gültig ist oder ob eventuell das Passwort abgelaufen ist. In diesem Fall kann es die Authentifizierung verschieben, bis der Benutzer sein Passwort aktualisiert hat oder auch eine Warnung an den Benutzer ausgeben, dass er sein Passwort ändern möchte. Als Session-Modul zeichnet es einfach nur den Benutzernamen und den Diensttyp über den Syslog-Dienst auf. Das Modul "pam_env.so" setzt Umgebungsvariablen, die in */etc/security/pam_env.conf* definiert wurden. "pam_mail" ist als "optional" angegeben, darf also ohne Konsequenzen fehlschlagen. Es dient lediglich dazu, dem Benutzer mitzuteilen, ob er neue Mail für ihn vorliegt. Da dieses jedoch nur für lokal erreichbare Mailfolder klappt, kommt dieses Modul eher selten sinnvoll zum Einsatz.

13.3.5 Sicherheit und Fallstricke

PAM "bewacht" die verschiedenen Eingänge zu einem Rechner und ist deshalb ein wesentlicher Baustein der Sicherheitsrichtlinien. Eine Fehlkonfiguration kann dazu führen, dass sich Benutzer auch ohne ausreichende Authentifizierung anmelden können, wie im folgenden Beispiel gezeigt:

```
auth     required   pam_nologin.so
auth     optional   pam_unix2.so     set_secrpc
auth     optional   pam_ldap.so      use_first_pass
auth     optional   pam_storepw.so   use_first_pass
[ ... ]
```

Hier werden wieder vier Module "gestackt": Das erste Modul überprüft auf die Existenz der */etc/nologin*. Gibt es diese Datei nicht, liefert es den Status "success". Selbst wenn nun alle drei folgenden Module fehlschlagen, wird das Gesamtergebnis "success" nicht mehr angezweifelt und ein Benutzer kann sich auch mit falschem Passwort anmelden. Noch radikaler wäre der Einsatz des "pam_permit"-Moduls, welches immer einen Erfolg der Anmeldung garantiert.

Umgekehrt kann eine Fehlkonfiguration auch komplett verhindern, dass sich irgendjemand an der Maschine anmelden kann:

```
auth     required   pam_nologin.so
auth     required   pam_unix2.so     set_secrpc
auth     required   pam_ldap.so      use_first_pass
auth     required   pam_storepw.so   use_first_pass
[ ... ]
```

Die Kontrolleinstellung "required" sorgt dafür, dass ein Benutzer sowohl in den Unix-Dateien *passwd* und *shadow* als auch im LDAP bekannt sind. In verteilten Authentifizierungsarchitekturen liegt genau dieses nicht vor: Der Systemadministrator ist immer nur lokal eingetragen, normale Benutzer sinnvollerweise nie. Die Aussperrung von der Maschine ist perfekt. Deshalb veranstaltet man die Tests am besten auf einem unkritischen Dienst und überträgt die erfolgreiche Kombination erst dann auf alle anderen. So könnten die Experimente auf dem SSH-Login laufen, aber der Konsolen-Login vorerst noch mit der alten aber laufenden Konfiguration gesteuert werden. Im Notfall kann Linux über die Kernel-Command-Line ohne jegliche Authentifizierung gestartet werden, in dem der Init-Prozess gegen die Shell ausgetauscht wird: `init=/bin/sh`. Anschliessend muss das Root-Dateisystem noch mit `mount -n -o remount,rw` in den schreibbaren Zustand versetzt werden und eventuell ungünstige PAM-Konfigurationen lassen sich wieder rückgängig machen.

Vielfach kann es vorkommen, dass man gerade PAM frisch konfiguriert hat und es sofort ausprobieren möchte. Wenn dabei zentrale Benutzerdatenbanken wie LDAP ins Spiel kommen, kann es passieren, dass der Name Service Caching Daemon (**nscd**) einen Benutzer noch als unbekannt gespeichert hat. Dann schlägt trotz korrekter Konfiguration eine Anmeldung fehl. Ein einfacher Neustart dieses Dienstes kann das Problem beheben.

Je nach Konstruktion eines Systems ist auch auf den "su"-Mechanismus ein Auge zu werfen. In grossem Umgebungen traut man vielleicht den lokalen Administratoren insoweit nicht, als dass sie in die Home-Verzeichnisse beliebiger Nutzer hineinsehen dürfen. Dieses wird durch eine zusätzlich notwendige Authentifizierung wie für AFS oder Samba effektiv verhindert. Dann macht auch die Einstellung des "su"-Mechanismus keinen Sinn, dass sie sich ohne Angaben eines Passworts in normale zentral verwaltete Benutzer "verwandeln" dürfen, da das Home-Verzeichnis dann nicht automatisch zur Verfügung steht.

Für alle PAM-bezüglichen Dateien und Verzeichnisse muss unbedingt darauf geachtet werden, dass nur der Systemadministrator Schreibberechtigung hat und Besitzer der Verzeichnisse */lib/security*, */etc/security* und */etc/pam.d* darin befindlichen Dateien ist. Ausserdem sollten auch keine PAM-Module verwendet werden, die von Benutzern schreibbare Programmbibliotheken verwenden. Sonst könnte es Angreifern gelingen an verschiedenen Stellen des PAM-Mechanismus ihre Ziele zu finden. Ein Beispiel für die Platzierung von Trojanern bietet das "pam_storepw"-Modul, welches als Bestandteil oder Ersetzung eines anderen Moduls Passwörter abgreifen kann. Aus diesem Grunde sollten PAM-Module von Drittanbietern aus dem Internet mit einiger Vorsicht verwendet werden. Optimalerweise überprüft der Systemadministrator diese Module und ihren Quellcode auf korrekte Funktionsweise.

Weiterhin sollte ein Fallback-Mechanismus definiert für unkonfigurierte Dienste definiert werden.

```
auth       required       pam_warn.so
auth       required       pam_unix2.so
account    required       pam_warn.so
account    required       pam_unix2.so
password   required       pam_warn.so
password   required       pam_cracklib.so
password   required       pam_unix2.so       use_first_pass use_authtok
session    required       pam_warn.so
session    required       pam_unix2.so
```

Beim Passwortwechsel wird eine Prüfung des Passwortes mit der cracklib-library vorgenommen, ansonsten meldet sich jeder Modultyp mit einer Warnung beim Syslog und führt anschließend seine Funktion über das "pam_unix2"-Modul durch. Diese Art der other-Konfiguration erfüllt den Zweck, dass zum Einen eine Warnung ausgegeben wird, die darauf hinweist, dass der Dienst nicht konfiguriert ist, zum Anderen aber eine Anmeldung unter einer gültigen Unix-UID noch möglich ist. Eine paranoide Einstellung des Other-Fallbacks könnte so aussehen:

```
auth       required       pam_deny.so
auth       required       pam_warn.so
account    required       pam_deny.so
account    required       pam_warn.so
password   required       pam_deny.so
password   required       pam_warn.so
session    required       pam_deny.so
session    required       pam_warn.so
```

In dieser Einstellung wird die Benutzung unkonfigurierter Dienste komplett unterbunden und darüberhinaus eine Warnung ins Syslog geschrieben.

Wenn PAM für Netzwerkdienste verwendet werden soll, z.B. für die FTP-Authentifizierung, sollte beachtet werden, dass PAM selbst keinerlei Methoden zum Schutz der Übertragung von Passwörtern und Benutzernamen über das Netz realisiert. Dieses ist immer Aufgabe der Applikation selbst, wie z.B. SSH. Darum sollte PAM bei Diensten, die von sich aus keine Verschlüsselung bieten, wie das Login über einen Displaymanager nur zur lokalen Anmeldung benutzt werden.

Einige Dienste erfordern zusätzliche Einstellungen damit sie überhaupt PAM verwenden. Dieses kann beim SSH-Dämon irritieren, da hier der Administrator darauf achten muss, dass die PAM-Authentifizierung erlaubt wird. Dieses wird in der Datei */etc/ssh/sshd_config* eingeschaltet, indem "UsePAM yes" eingetragen wird.

siehe auch http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/

Index