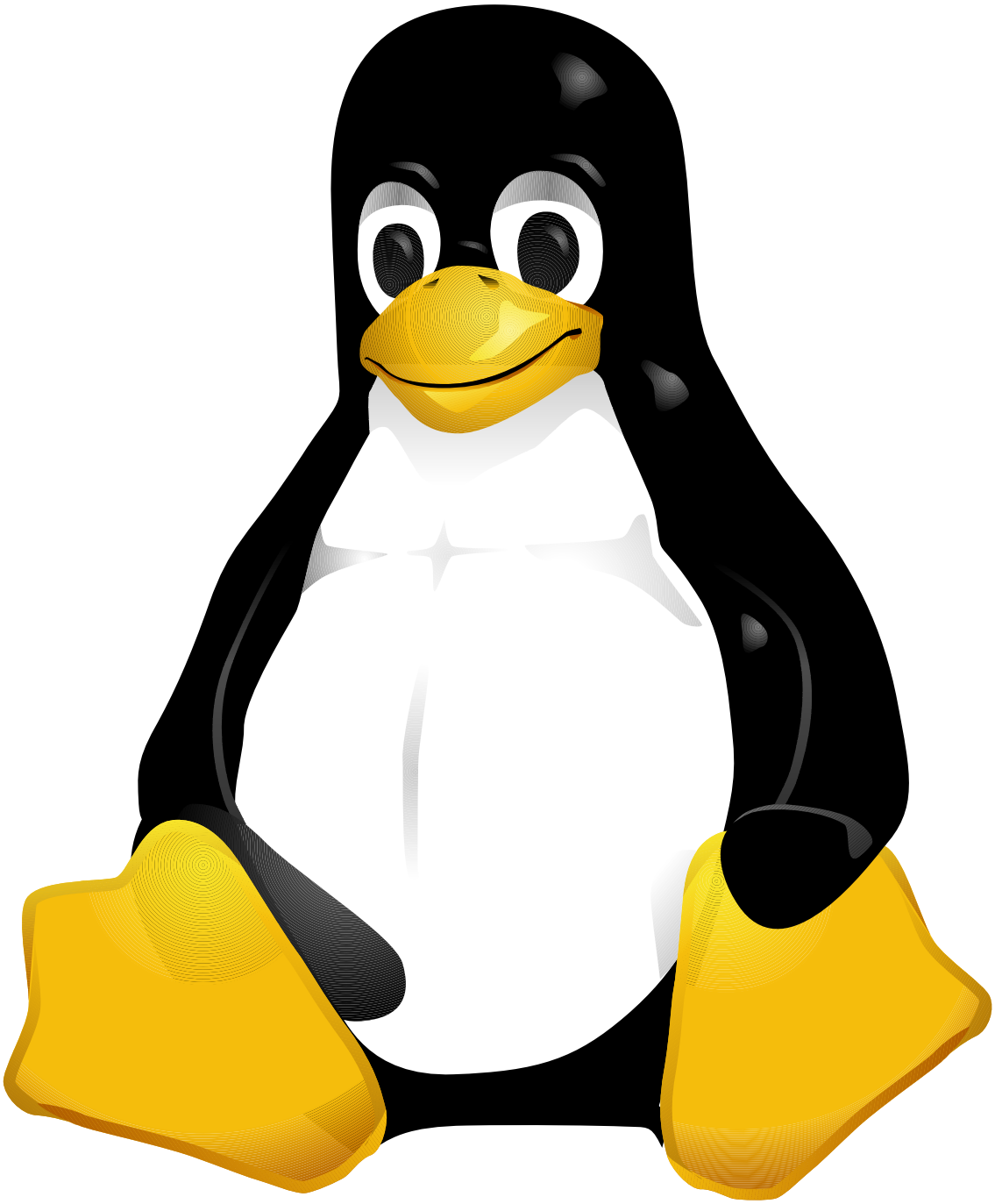
Technische Praxis der Computersysteme

Teil 2-1

Systemadministration unter Linux/Unix

Roland Steinbauer, Oliver Fasching, Andreas Nemeth,
Martin Piskernig, Florian Wisser

Version 1.01, Februar 2003



Inhaltsverzeichnis

Literatur

[1] Evi Nemeth, Garth Snyder, Scott Seebass, Trent R. Hein. *Unix System Administration Handbook.* 3rd Edition. Prentice Hall, New Jersey. 2000.

[2] Matt Welsh, Lar Kaufman. *Running Linux.* 3rd Edition. O'Reilly & Associates Inc. Sebastopol, CA. 1999.

[3] Aeleen Frisch. *Essential System Administration.* 3rd Edition. O'Reilly & Associates Inc. Sebastopol, CA. 2002.

[4] Cameron Newham, Bill Rosenblatt. *Learning the Bash Shell.* 2nd Edition. O'Reilly & Associates Inc. Sebastopol, CA. 1998.

[5] Lars Wirzenius, Joanna Oja, Stephen Stafford. *The Linux System Administrator's Guide.* Version 7.0. `http://www.linuxdoc.org/LDP/sag`. 2001.

[6] Steve Frampton. *Linux Administration Made Easy.* `http://linuxdoc.org/LDP/lame/LAME/linux-admin-made-easy`. 1999.

[7] Matt Welsh et al. *Linux Installation and Getting Started.* `http://linuxdoc.org/LDP/gs`. 1998.

[8] Larry Wall. *Programming Perl.* 3rd Edition. O'Reilly, UK. 2000.

Vorwort

Die vorliegende Version 1.0 ist eine wesentlich überarbeitete Neuauflage des Skriptums vom Sommersemester 2001. Teil 2-1 beinhaltet eine Einführung in die Grundlagen der Systemadministration unter Linux/Unix mit Ausnahme der Netzwerkfunktionen, die im Teil 2-2 behandelt werden.

Gegenüber dem Vorjahr wurde die Kapitlezusammenstellung teilweise neu arrangiert und vor allem die Kapitel 1, 3, 4, 8 und 10 neugestaltet und das Kapitel 12 sowie die Grundlagen der Sicherheit von Computersystemen vom Teil 2-2 in den Teil 2-1 vorverlagert. Desweiteren habe ich mich bemüht, die gesamte Präsentation homogener zu gestalten und die stilistischen Eigenheiten der verschiedenen Kapitel (deren ursprüngliche Entstehung verschiedenen Autoren zu verdanken ist) auszugleichen. Schließlich habe ich versucht, der schnellen Entwicklung des Gebiets Rechnung zu tragen und die nötigen Aktualisierungen vorgenommen.

Großartig unterstützt wurde ich in all diesen Belangen von Oliver Fasching, der auch die Verwaltung des LaTeX-Quellcodes übernommen hat. Weiters gilt mein besonderer Dank den zahlreichen StudentInnen und meinen Tutoren, die mich auf viele Unklarheiten und Fehler in der Version 0.95 aufmerksam gemacht und damit wesentlich zur Verbesserung des Textes beigetragen haben.
Wie immer gilt mein ganz besonderer Dank den Mitarbeiterinnen des Sekretariats für ihre freundliche Unterstützung.

Roland Steinbauer, April 2002

Vorwort zur Version 0.95 (Sommersemester 2001)

Das vorliegende Skriptum dient zur Begleitung des 1. Blocks der Vorlesung „(Technische) Praxis der Computersysteme, Teil 2“ im Sommersemester 2001 am Institut für Mathematik der Universität Wien, die als Fortsetzung der Vorlesung vom vergangenen Wintersemester konzipiert ist. Es deckt die (meisten) grundlegenden Themen der Unix/Linux Systemadministration – mit Ausnahme aller Netzwerkfunktionen – ab: Letztere bleiben dem 2. Block und einem weiteren Teil des Skriptums vorbehalten.

Als Vorkenntnisse für ein erfolgreiches Lesen sind die grundlegende Vertrautheit im Umgang mit einem Unix-System, der Unix-Shell und die Beherrschung eines Texteditors (bevorzugt `vi`) zu nennen; etwa in dem Umfang, wie sie im ersten Teil der Vorlesung bzw. des Skriptums vermittelt werden.

Grundlage des „Werks“ sind die Vortragsfolien und -vorbereitungen der letztjährigen Vorlesung mit gleichem Titel, die ich ebenso am Institut für Mathematik gehalten habe. Schon damals wurde ich tatkräftig von Florian Wisser als Tutor unterstützt; Reinhard Danzinger, Andreas Nemeth und Martin Piskernig haben als Hörer aktiv ins Geschehen eingegriffen und zu einigen Themenbereichen selbst Vorträge ausgearbeitet. Nun wurde das vorjährige „Skriptums-Gerippe“ unter geteilter Autorschaft zu einem fortlaufenden Text erweitert, der viele der ursprünglichen Folien als Zusammenfassung und Referenzpunkte enthält, aber auch zum eigenständigen Lesen und Lernen einlädt.

Als Quellen haben wir – neben unserer Erfahrung als Systemadministratoren und den im Text angeführten Seiten im Internet – vor allem die Bücher [1] und [2] herangezogen.

Der Umfang des Textes ist vor allem durch das Einbeziehen vieler praktischer Aspekte (etwa in Kapitel 6 und 11) überraschend angewachsen, sodass das ganze Projekt beinahe den vorgegebenen zeitlichen Rahmen gesprengt hätte. Die sehr zügige Endredaktion hat mich davon abgehalten, an Formulierungen zu feilen

und den einen oder anderen Aspekte noch stärker zu betonen, weshalb die Versionsnummer nochmals unter 1.0 geblieben ist.

Besonders zu Dank verpflichtet bin ich neben meinen Koautoren meinen (Ex)-TutorInnen Mark Heinzle, Barbara Pocza und Matthias Zeichmann sowie unserem Sekretariat, das mich mit besten Kräften bei der (rechtzeitigen) Fertigstellung des Skriptums unterstützt hat.

Wie immer liegt die Verantwortung für alle Ungereimtheiten und Fehler bei mir und ebenso natürlich wie im Wintersemester ergeht die Einladung an alle Interessierten, an einer verbesserten Neuauflage für das nächste Mal mitzuarbeiten. Weitere Informationen zur Vorlesung, den Übungen und zum Thema finden sich unter `http://www.mat.univie.ac.at/praxis`.

Roland Steinbauer, April 2001

1 Einleitung

In diesem ersten Kapitel geben wir einen Überblick über Grundlagen und Aufgaben der Systemadministration. Wir besprechen kurz allgemeine Strategien und Methoden, die „politischen“ Aspekte der Systemadministration und widmen uns dann ihren praktischen Auswirkungen. Wir besprechen kurz die häufigsten Aufgaben in der Systemadministration und diskutieren Aspekte des Rootzugangs zu Unix-Systemen.

1.1 Aufgaben in der Systemadministration

Wir geben hier einen kleinen Überblick über die in der Systemadministration anfallenden Aufgaben und Tätigkeiten. Diese müssen nicht unbedingt alle von einer Person erledigt werden und in der Praxis werden oft verschiedene Aufgabenbereiche von verschiedenen Personen wahrgenommen. Trotzdem muss es für jedes System mindestens eine Person geben, die alle diese Tätigkeiten versteht, den Überblick bewahrt und sicherstellt, dass alle Aufgaben richtig und zeitgerecht erledigt werden. Die Liste der Aufgaben ist weder erschöpfend noch strikt nach Prioritäten geordnet; letztere variieren von System zu System.

Benutzerverwaltung: In einem Multiuser System müssen die Accounts der einzelnen Benutzer gewartet werden. Die Hauptaufgaben bestehen im Hinzufügen neuer und im Löschen von nicht mehr benötigten Benutzeraccounts. Diese Prozeduren können automatisiert werden aber gewisse administrative Entscheidungen sind naturgemäß unabhängig davon zu treffen. Kapitel 3 gibt einen Überblick über die technischen Aspekte der Benutzerverwaltung.

Software: Sowohl das Betriebssystem selbst, wie auch Anwenderprogramme müssen upgedatet, neue Anwendungen installiert werden. Sehr wichtig ist das *Testen* all dieser Programme (auf verschiedener Hardware, in verschiedenen Systemumgebungen, etc.). Wir besprechen das Installieren von Software und das Package-Management unter Linux im Kapitel 4.

Aufgaben in der Systemadministration

- Benutzerverwaltung: Accounts anlegen und löschen
- Software: Installation und Upgrade
- Hardware: Wechsel der HW, Einbau, Kaufentscheidung
- Backup: wichtig!!!
- Systemmonitoring: Sicherheit und Performance
- Troubleshooting: Problem- und Fehlersuche und -behebung
- Benutzerbetreuung: Anfragen (Politik!)

Folie 1

Hardware muss erneuert und verändert werden. Die Aufgaben des Administrators reichen vom Einbau neuer Hardware in vorhandene Rechner über den Umbau von Hardware bis zur Integration neuer Rechner in ein Netzwerksystem. Außerdem sind Administratoren sinnvollerweise in die Kaufentscheidungen miteinbezogen.

Backup: Eine der langweiligsten aber wichtigsten Routineaufgaben in der Systemadministration ist das Erstellen, Überprüfen und Aufbewahren von Backups. Hier sind auch wichtige administrative Entscheidungen zu treffen. Wir befassen uns mit Backups in Kapitel 9.

Systemmonitoring: Der Administrator ist für das Funktionieren des Gesamtsystems verantwortlich. Um dies zu gewährleisten sind eine Reihe von Routineaufgaben notwendig, die das Überprüfen von Systemlogfiles, das Lesen der Rootmail, und das Überwachen von Systemdiensten im Hinblick auf Sicherheit und Performance inkludieren. Wir besprechen Aspekte und Tools des Systemmonitorings in Kapitel 5, Logging in Abschnitt 12.3, System- und Netzwerksicherheit in den Kapiteln 12 und 18.

Troubleshooting: Software und Hardware ist fehlerhaft. Die Aufgabe des Systemadministrators ist die Analyse und das Isolieren von auftretenden Problemen, um dann die richtigen Maßnahmen ergreifen zu können, zB das Einschalten der richtigen Experten (Hardwareservice, etc.). Oft ist das Aufspüren und Isolieren des Problems die eigentlich schwierige Aufgabe, die eine gute Kenntnis des Systems und seiner Funktionsweisen voraussetzt.

Benutzerbetreuung: Obwohl selten in der Arbeitsplatzbeschreibung eines Administrators zu finden, nimmt diese Tätigkeit oft den Löwenanteil der Arbeitszeit in Anspruch. Benutzerbetreuung hat viele „politische “ Aspekte und ist oft Hauptauslöser von Frustration und Überlastung von Systemadministratoren; siehe dazu Folie 2.

1.2 Allgemeine Strategien und Methoden

Im Rahmen einer Firma oder Organisation (zB Universitätsinstitut, Schule) die große EDV-Systeme für Verwaltungs- und/oder Lehrzwecke und/oder andere produktive Aufgaben betreiben, kommt Systemadministratoren eine ganz besondere Verantwortung im Bezug auf das Funktionieren der gesamten Infrastruktur zu. Computersysteme besitzen aufgrund ihrer rasanten Entwicklung und der immer schneller und weiter fortschreitenden Vernetzung über eine beträchtliche Eigendynamik. Die organisatorischen und rechtlichen Rahmenbedingungen innerhalb vieler Firmen und Organisationen können nicht Schritt halten und es entstehen Graubereiche und Unklarheiten.
Systemadministration findet in diesem Spannungsfeld statt. Erfahrungsgemäß wenden Systemadministratoren viel Zeit und Energie in diesem nicht technischen sondern „politischen“ oder sozialen Arbeitsfeld auf. Wir geben hier einen kleinen Überblick und verweisen auf die ausführlichen Überlegungen und Erfahrungsberichte zu diesem Themenkomplex im Kapitel 27 von [1].

Viele erfahrene Administratoren empfehlen eindringlich die schriftliche Niederlegung von Regeln und Verfahrensweisen. Diese *Regeln* (Policy) sollten vom Management der Organisation genehmigt und auch in ihren juristischen Dimensionen abgesichert sein. Bereiche in denen diese Regeln große Wichtigkeit haben sind auf Folie 3 aufgezählt.
Niedergeschriebene *Verfahrensweisen* oder Methoden (Procedure) für gewisse Eventualitäten und/oder Routineaufgaben der Systemadministration in Form von Rezepten dokumentieren einerseits die übliche Praxis und stellen andererseits deren Einhaltung sicher; sie Reduzieren die Wahrscheinlichkeit von Fehlern und setzten nachvollziehbare Standards. Eine Liste von Aufgaben, die so standardisiert werden sollten, finden sich auf Folie 4.

„Politik“

Sysadmin im Spannungsfeld

- Verantwortung für Infrastruktur
- Eigendynamik von Computersystemen
- Organisationsstrukturen
- Security

empfohlen

- niedergeschriebene Regel
- Übereinstimmung mit Management
- überprüfen rechtlicher Aspekte

Folie 2

Policy

feste Regelungen für

- verfügbare Services
- Rechte und Pflichten der User
- Rechte und Pflichten der Sysadmins
- Gastaccountvergabe
- ...

Folie 3

Darüber hinaus gibt es Fragen und Aufgaben, die genau zwischen Regeln und Methoden zu entscheiden/regeln sind; zB wer bekommt einen Account am System und was passiert, wenn diese Person die Organisation wieder verlässt? Das „Ob“ wird von den Regeln bestimmt, das „Wie“ von der Verfahrensweise. Die Tatsache, dass das gesamte Regelwerk schriftlich niedergelegt ist und Teile (etwa eine Benutzungsordnung) sogar mit Unterschrift bestätigt sind, garantiert die Konsistenz und Verbindlichkeit der Standards.
Besondere Relevanz und Brisanz erlangt diese klare Strukturierung im Bereich der *Sicherheit* (Security) von Computersystemen. Da sich die Sicherheit des Systems im Allgemeinen indirekt proportional zur Bequemlichkeit der Benutzung und zur Zahl und Qualität der angebotenen Services verhält, ist hier ein besonderes Spannungsfeld gegeben (vgl. Kapitel 12).

Auch im Bereich der *Benutzerbetreuung* – der in der Praxis sehr zeitaufwendig und nervenaufreibend ist – ist eine niedergeschriebene Policy Gold wert. Es sollte genau geklärt werden, welche Services und welcher Support den Usern zusteht und was von den Systemadministratoren *nicht* erledigt wird/werden muss. Weiters sollte klargestellt werden, in welcher Form Useranfragen an die Administratoren herangetragen werden müssen, bzw. welche Information eine Anfrage mindestens beinhalten muss, um bearbeitet zu werden. Anfragen der Art: „Was habt ihr denn wieder gemacht; mein Computer geht schon wieder nicht!“ sind nicht nur ärgerlich, sondern tragen auch nicht zur effizienten Problembearbeitung bei. Anfragen sollten immer mindestens die folgende Informationen beinhalten müssen: Auf welchem Host tritt bei welchem Benutzer welches Problem auf? Wie heißt die genaue Fehlermeldung, falls eine auftritt? Ist das Problem reproduzierbar? Wie wichtig ist die sofortige Problembehebung? Es bietet sich zB an, nur Anfragen über ein standardisiertes Webformular entgegenzunehmen.

Zum Abschluss verweisen wir noch Explizit auf die Abschnitte 27.3 und 27.8 in [1] für eine statistische Untersuchung über die realen Arbeitsbedingungen von Systemadministratoren sowie für „War Stories and Ethics“.

Procedure

Verfahrensweisen in Rezeptform

- Dokumentation und Standards
- weniger Fehler, mehr Konsistenz

für folgende Situationen

- Anlegen neuer Benutzer
- Einbinden neuer Hosts
- Sicherheitskonfiguration
- Backup und Restore
- Upgrade und Installation von Software
- Security Alarm, Notfälle
- Recovery from Desaster
- ...

Folie 4

1.3 Rootzugang

Jedes Unix-System verfügt über einen Administratoraccount, den sogenannten *Rootaccount* (Login: root), der durch die UID 0 (User Identity; siehe Kapitel 3) definiert ist. Dieser ist mit besonderen Privilegien und Rechten am System ausgestattet (siehe Folie 6), die das Erledigen aller administrativen Tätigkeiten, die Konfiguration des Systems, das (De)Installieren von Systemsoftware und vieles andere mehr ermöglicht. Für root gelten keine Dateiberechtigungen, root kann Dateibesitzer und Prozessbesitzer ändern und (ohne Passwort) auf jedem Account einloggen. Root „kann im Wesentlichen alles“, außer dem offensichtlich Unmöglichen, d.h. jede gültige Operation an einer Datei oder einem Prozess vornehmen.

Aus dieser (notwendigen) „Allmacht“ des Rootaccounts ergibt sich sowohl eine große *Verantwortung*, wie auch ein hohes *Gefahrenpotential*. Root kann alle Dateien auf dem System lesen zB auch die Email aller Benutzer. Dies ist sinnvoll, um etwa nach Viren in Mailattachments zu scannen. Andererseits verbietet (nur) die Etikette und sein Verantwortungsbewusstsein root die Mail anderer zu lesen.
Natürlich birgt die Verwendung der Rootaccounts vor allem durch ungeübte oder gar böswillige Hände vielfältige Gefahren in sich. Root kann zB alle Dateien auf den Festplatten löschen oder gar die Platten formatieren. Daher gilt vor allem für Anfänger in der Systemadministration: „Sit on your hands before you press ENTER!“ Da aber natürlich auch erfahrenen Administratoren Fehler unterlaufen (können), sollte der Rootaccount sehr sparsam und wirklich *nur (!)* zur Systemadministration verwendet werden, also *nur dann, wenn es unbedingt erforderlich ist*. Es ist nicht nur schlechter Stil, sonder äußerst gefährlich, wenn der Administrator gewohnheitsmäßig statt (s)eines eigenen (normalen) Benutzeraccounts den Rootaccount verwendet.

Der Rootaccount

- Account des Systemadministrators
- definiert durch User Identity (UID) = 0
- „allmächtig“
- große Verantwortung
- gefährlich in ungeübten oder böswilligen Händen
- sparsame Verwendung; **nur** wenn wirklich nötig!!!

Folie 5

Rootrechte

jede gültige Operation an einer Datei oder einem Prozess vornehmen

- Dateiberechtigungen gelten für root nicht
- kann Dateibesitzer ändern
- Prozessbesitzer ändern
- Runlevel ändern
- Hostnamen setzen
- Netzwerkgeräte konfigurieren
- Systemzeit setzten
- Devicefiles erzeugen
- negative Nicewerte setzen
- ...

“Sit on your hands before you press ENTER!”

Folie 6

Neben der äußerst sparsamen Verwendung des Rootaccounts sollte ein Rootlogin weder auf der grafischen Oberfläche (manche Systeme (zB SuSE 7.2) warnen explizit beim Login davor) noch direkt auf einer Textkonsole erfolgen. Der Grund dafür ist, dass dann im Logfile `/var/log/messages` nur die Tatsache vermerkt wird, dass root eingeloggt hat; es ist aber nicht ersichtlich, *wer* als root eingeloggt hat. Stattdessen sollte (jeder) der Administrator(en) zunächst unter seinem eigenen (normalen) Benutzeraccount einloggen und dann mittels des Kommandos `su` (Switch User) auf dem Rootaccount einloggen. Dabei wird natürlich das Rootpasswort abgefragt. Im Logfile wird dann eingetragen, welcher Benutzer das `su`-Kommando ausgeführt hat (vgl. Folien 8, 9).
Bei Verwendung der Syntax `su -` gelangt man in ein Root-Loginshell, also mit dem in `/root/.bahsh_profile` festgelegten Environment (inklusive richtigem Suchpfad; vgl. Teil 1, Abschnitt 6.2). Dem `su`-Kommando kann als optionales Argument ein Username übergeben werden (der Default ist also root) um in den Account des entsprechenden Users einzuloggen. `su` fragt nach dem Passwort des Accounts außer, wenn root das Kommando ausführt; so kann root in jeden beliebigen Account einloggen. Auf vielen Systemen ist als Schutzmaßnahme die Verwendung des `su`-Kommandos auf die Benutzergruppe `wheel` beschränkt (mittels PAM-Konfiguration; siehe Abschnitt 12.2), sodass die Accounts aller Administratoren zu dieser Gruppe gehören müssen.
Logins als root über ein Netzwerk sollten (wenn überhaupt) nur mittels Secure Shell (`ssh`) erfolgen. Wird ein webbasiertes Konfigurationstool (Webmin, Linuxconf; siehe Kapitel 2) benutzt, sollte unbedingt eine SSL-Unterstützung (Secure Socket Layer) verwendet werden, die einen verschlüsselten Datentransfer erlaubt.

Oft wird der Rootaccount als Gruppenaccount verwendet, d.h. mehrere Personen (Administratoren) kennen das Rootpasswort. In diesem Fall ist es aus mehreren Gründen (Fehlersuche, etc.) wichtig zu wissen, wer als root eingeloggt war (für die praktischen Aspekte siehe voriger Absatz). Eine gute Methode, um in dieser Situation nicht den Überblick zu verlieren wer was getan hat, ist das Führen eines „Logbuchs“. Jeder Administrator muss vor dem Ausloggen als root in der Logbuchdatei (zB

`/root/LogBook`) eintragen, warum er als root eingeloggt war und was er getan hat. Eventueller Nachlässigkeit kann mittels eines Eintrags `vi /root/LogBook` in `/root/.bash_logout` abgeholfen werden.

Unter Unix ist es generell nicht möglich Rootrechte abgestuft zu vergeben, d.h. es gibt keinen Account, der etwa „halbe“ Rootrechte besitzt; das Unix-Sicherheitskonzept ist „binär“; entweder man darf „alles“ (root) oder „nichts“ (normaler Benutzer). Abhilfe schafft hier ein Tool namens `sudo`, das es erlaubt gewissen Benutzer(gruppen) das Ausführen gewisser Befehle als root zu gestatten, ohne das Rootpasswort abzufragen; es wird lediglich das Passwort des Benutzers abgefragt, um Missbrauch zu verhindern. So kann etwa einem Mitarbeiter die Möglichkeit eingeräumt werden, Backups zu machen oder Benutzeraccounts anzulegen, ohne dass er im Besitz des Rootpassworts sein muss. Neben einer sehr detailliert abgestuften Konfiguration ermöglicht `sudo` das Mitloggen aller ausgeführten Befehle, sodass immer ersichtlich ist, wer wann was getan hat. Für weitere Informationen siehe `http://www.sudo.ws`; leider ist das `sudo`-Paket nicht Bestandteil aller Linux-Distributionen.

Klarerweise kommt beim Schutz des Rootaccounts dem Rootpasswort ganz besondere Bedeutung zu. Alle Regeln für eine gute Passwortwahl (vgl. Teil 1, Abschnitt 3.1) sollten hier genau befolgt werden. Das Rootpasswort sollte in unregelmäßigen, unangekündigten Abständen geändert werden. Auf verschiedenen Systemen sollten verschiedene Rootpasswörter verwendet werden. Weiters sollte das Rootpasswort geändert werden, wenn immer der Verdacht besteht, dass die Sicherheit des System kompromittiert wurde und wenn einer der Administratoren, der im Besitz des Passworts ist, die Organisation verlässt.

Root Login

- **nur** wenn wirklich nötig
- **nicht** unter X Window
- **nicht** direkt auf der Konsole
- **nicht** über unsichere Netzwerkverbindung
- **richtig:**
  - lokal: `su`-Kommando
  - remote: `ssh`,
    SSL bei webbasierten Tools

Folie 7

Switch User (su)

```
[roland@pablo roland]$ whoami
roland
[roland@pablo roland]$ su
Password:
[root@pablo roland]# whoami;pwd
root
/home/roland
[root@pablo roli]# echo $PATH| grep sbin
[root@pablo roli]# exit
[roli@pablo roli]$ su -
Password:
[root@pablo root]# whoami;pwd
root
/root
[root@pablo root]# echo $PATH
/bin:/sbin:/usr/bin:/usr/sbin:
/usr/local/bin:/usr/local/sbin:
/usr/bin/X11:/usr/X11R6/bin:/root/bin
[root@pablo root]#
```

Folie 8

Rootlogin – Logeintrag

- falsch: auf Konsole oder unter X

```
$ tail /var/log/messages
Mar  9 11:06:20 pablo login(pam_unix)[1021]: session
                opened for user root by LOGIN(uid=0)
Mar  9 11:06:20 pablo  -- root[1021]: ROOT LOGIN
                ON tty4
```

- richtig: `su`

```
$ tail /var/log/messages
Mar  9 11:06:44 pablo su(pam_unix)[6341]: session
                opened for user root by roli(uid=500)
```

Folie 9

Schutz des Rootaccounts

- Rootpasswort
  - sicher wählen („Shocking Nonsense“)
  - in unregelmäßigen Abständen ändern
  - verschiedene Rootpasswörter auf verschiedenen Systemen
- immer zuerst als Benutzer anmelden dann `su` auf root
- unsicheres Remotelogin für root abstellen
- Führen eines „Logbuchs“
- eingeschränkte root-Rechte mit `sudo`

Folie 10

2 Administrationstools

In diesem Kapitel geben wir einen groben Überblick über die grundlegenden Methoden der Systemkonfiguration. Neben den wichtigsten Methoden – dem Editieren der entsprechenden Konfigurationsdateien und dem Erstellen und Modifizieren der Konfiguration mittels Shellscripts – stehen unter Linux verschiedene Administrationstools zur Verfügung. Wir behandeln kurz die distributionsunabhängigen Tools Linuxconf und Webmin, sowie einige distributionsspezifische Administrationsprogramme.

2.1 Systemkonfiguration

Generell ist unter Unix (beinahe) die gesamte Konfiguration des Systems in *Textdateien* gespeichert; der kleinste gemeinsame Nenner jeder Konfigurationsarbeit an Unix-Systemen das Editieren von ASCII-Dateien. Daher ist für Systemadministratoren die Beherrschung eines Editors ebenso unerlässlich, wie die gute Beherrschung (zumindest) einer Shell.
Eine besondere Rolle spielt hier der vi-Editor, der auf praktisch allen Unix Systemen zur Grundausstattung gehört und auch in jedem Notfall zur Verfügung steht. Wir empfehlen daher jedem angehenden Administrator unbedingt zumindest die Grundbedienung des vi zu erlernen. Ein guter Kompromiss ist hier der `gvim`-Editor, eine graphische vi-Version, die zwar das „vi-Feeling“ vermittelt aber auch über Menüs bedienbar ist (siehe auch Teil 1, Kapitel 5).

Weiters spielen *Shellscripts* eine entscheidende Rolle in der Systemkonfiguration; weitaus die meiste Konfigurationsarbeit an Unix-Systemen wird über Shellscripts erledigt. Das Lesen, Verstehen und Modifizieren vorhandener Scripts sowie das Schreiben eigener Shellscripts gehört daher zur alltäglichen Arbeit eines Systemadministrators. Am weitesten verbreitet ist die `sh`-Scriptsprache, die auch in der Linux-Standardshell `bash` implementiert ist. Wir erklären die Grundlagen der Bashprogrammierung in Kapitel 11.
Für größere Projekte empfiehlt sich auch die Scriptsprache *Perl*. Diese bietet mehr Flexibilität als die `sh`-Sprache und besitzt viele Charakteristika, die sich besonders für die Systemkonfiguration eignen. Eine Einführung in Perl findet sich zB in [8].

Der Großteil der Systemkonfigurationsdateien in vielen Unix-Systemen ist im Verzeichnis `/etc/` (manchmal auch `/adm/` oder `/var/adm/`) und seinen Unterverzeichnissen gespeichert. Beispiel dafür ist zB die Datei `/etc/inittab`, die das Verhalten des Initprozesses bestimmt und die Initscripts in AT&T-artigen Systemen (die dem System V Standard folgen; unter ihnen die meisten Linux-Distributionen) in `/etc/init.d/`, die die meisten Systemdienste starten (siehe Teil 1, Kapitel 9).

Der genaue Pfad verschiedener Konfigurationsdateien variiert allerdings stark zwischen den verschiedenen Unix- und auch Linux-Varianten. Unter RedHat zB wird das Verzeichnis `/etc/sysconfig/` verwendet, um dort in einzelnen Dateien wichtige Parameter für einzelne Komponenten der Konfiguration des Gesamtsystems (zB Netzwerk, Systemzeit, ...) zu speichern.
Unter SuSE wird ein anderes Schema verwendet. Es gibt eine (große) globale Konfigurationsdatei `/etc/rc.config`, in der faktisch alle Parameter gespeichert werden. Ein eigener Dienst (rc.SuSEconfig) wertet diese Datei aus und schreibt die Parameter in die (vielen) eigentlichen Konfigurationsdateien, die dann von den entsprechenden Prozessen gelesen werden.

Systemkonfiguration

- gesamte Unix-Konfiguration in Textdateien gespeichert
  - Beherrschung eines Editors (`vi` !)
  - Beherrschung der Kommandozeile
- meiste Konfigurationsarbeit durch Scripts erledigt
  - Beherrschung der `sh`-Scriptsprache
  - Für größere Projekte: Perl

Folie 11

Konfigurationsdateien

- meist in `/etc/` und Unterverzeichnissen
  manchmal `/adm/`, `/var/adm/`, ...
- viele Unterschiede zwischen Unix-Varianten
  und auch den einzelnen Linux-Distributionen
- RedHat: einzelne Dateien in `/etc/sysconfig/`
  zB `network, keyboard, mouse`, ...
- SuSE: globale Datei `/etc/rc.config`
  Dienst `rc.SuSEconfig` erstellt einzelne Dateien

Folie 12

2.2 Administrationswerkzeuge

Neben der für den Systemadministrator unerlässlichen Beherrschung der entsprechenden Unix/Linux-Befehle an der Kommandozeile bietet Linux die Möglichkeit verschiedene (großteils graphische) Konfigurationsprogramme zu Hilfe zu nehmen. Diese erlauben menügesteuert Änderungen an der Systemkonfiguration vorzunehmen; im wesentliche schreiben sie die entsprechenden Konfigurationsdateien und (re)starten die jeweiligen Services und Daemonen. Obwohl es für den Administrator wichtig ist die einzelnen Konfigurationsdateien und zumindest ihre grundsätzliche Syntax zu kennen und sie per Hand editieren zu können, erleichtern es die nun vorzustellenden Programme dem Einsteiger einen Überblick zu bekommen und gewisse Konfigurationsarbeiten zu erlernen.

Es gibt verschiedene Administrationsprogramme, die für beliebige Linux-Distributionen verwendet werden können; zB Linuxconf und Webmin. Darüber hinaus besitzen manche Distributionen eigene Administrationstools, die auf die Verwendung mit ebendieser Distribution zugeschnitten sind. Paradebeispiel ist YaST („Yet another Setup Tool“) von SuSE.

Linuxconf
ist ein mächtiges Tool, das für (fast) alle Konfigurationsarbeiten am System herangezogen werden kann. Es wurde ursprünglich von *Jacques Gélinas* entwickelt; die Projekthomepage findet sich unter `http://www.solucorp.qc.ca/linuxconf/`. Linuxconf besitzt eine modulare Struktur, d.h. für jede Konfigurationsaufabe (zB Konfiguration eines bestimmten Serverdienstes) ein eigenes Interface, das die entsprechenden Tasks übernimmt. Dadurch ist Linuxconf sehr flexibel und leicht erweiterbar. Linuxconf ist Bestandteil vieler Distributionen; RedHat beabsichtigt aber, Linuxconf in Zukunft nicht mehr in die Distribution aufzunehmen.

Administrationstools

- distributionsunabhängig
  - Kommandozeile (natürlich...)
  - Linuxconf (Konsole, graphisch, Netzwerkinterface)
  - Webmin (Netzwerkbasiert)
- distributionsspezifisch
  - YaST, YaST2 (SuSE)
  - Lisa (Caldera)
  - (c)kontrol-panel (RedHat)
  - ...

Folie 13

Linuxconf

- mächtiges Konfigurationstool
- praktisch für das ganze System
- geschrieben und gewartet von Jacques Gélinas
- Homepage `http://www.solucorp.qc.ca/linuxconf`
- verwendet übersichtliche Baumstruktur

Folie 14

Linuxconf darf nur von root benutzt werden und kennt vier Betriebsarten: Das *Kommandozeileninterface* ist vor allem für die Verwendung in Scripts konzipiert. Auf der Textkonsole kann ein *„Zelleninterface“*, im X Window ein entsprechendes *graphisches Interface* verwendet werden. Schließlich stellt Linuxconf auch einen *netzwerkbasierten Modus* zur Verfügung. Damit ist es möglich, sich mit dem Webbrowser ihrer Wahl mit Linuxconf zu verbinden und nach Abfrage des Rootpassworts die Konfigurationsarbeit durchzuführen.

Praktisch sieht die Verwendung von Linuxconf (mit Ausnahme des Kommandozeileinterfaces) vor, zuerst durch die in einer baumartige Struktur aufbereiteten Konfigurationspunkte zu browsen und in den entsprechenden Menüs die Änderungen einzutragen. Dann kann man mittels „Preview“ sehen, welche Änderungen (schreiben von Konfigurationsdateien, Neustarten von Services) ausgeführt werden müssen, um die angestrebte Konfiguration zu erreichen. Schließlich kann man diese Tasks ausführen lassen. Linuxconf schreibt alle von ihm ausgeführten Änderungen in der Datei `/var/log/netconf.log` mit.
Das Linuxconf-Webinterface ist standardmäßig aus Sicherheitsgründen deaktiviert, kann aber mittels Linuxconf selbst aktiviert werden. Es muss allerdings auch der Internetdaemon `(x)inetd` am System laufen um das Webinterface verwenden zu können. Man kann sich dann mit einem beliebigen Webbrowser auf Port 98 (`http://myhost.mydomain:98`) verbinden. Fraglich bzw. eine Geschmacksfrage ist, ob ein Webbrowser das geeignete Tool zur Systemadministration ist; vor allem sicherheitsrelevante Fragen sind hier zu beachten. Es lässt sich zwar festlegen, von welchen Hosts aus das Interface benützt werden darf, um aber das Rootpasswort verschlüsselt über das Netz zu versenden, muss erst ein Webserver mit Unterstützung für SSL (Secure Socket Layer) aufgesetzt werden (Details unter `http://www.terminator.net/linuxconf/linuxconf-ssl.html`). Außerdem muss man beachten, dass die meisten Webbrowser Passwörter speichern, also ein unberechtigter Benutzer eventuell durch Betätigen des „Zurück“-Buttons des Browsers auf die Linuxconf-Seite gelangen kann.

Linuxconf verwenden (1)

- nur root !
- 4 Betriebsarten
  - Kommandozeile (Scripts!)
  - Charakterzellen (interaktiv, textbasiert)
  - X-Window-basiert (interaktiv, GUI)
  - Netz-basiert (interaktiv, remote)
- Logfile `/var/log/netconf.log`

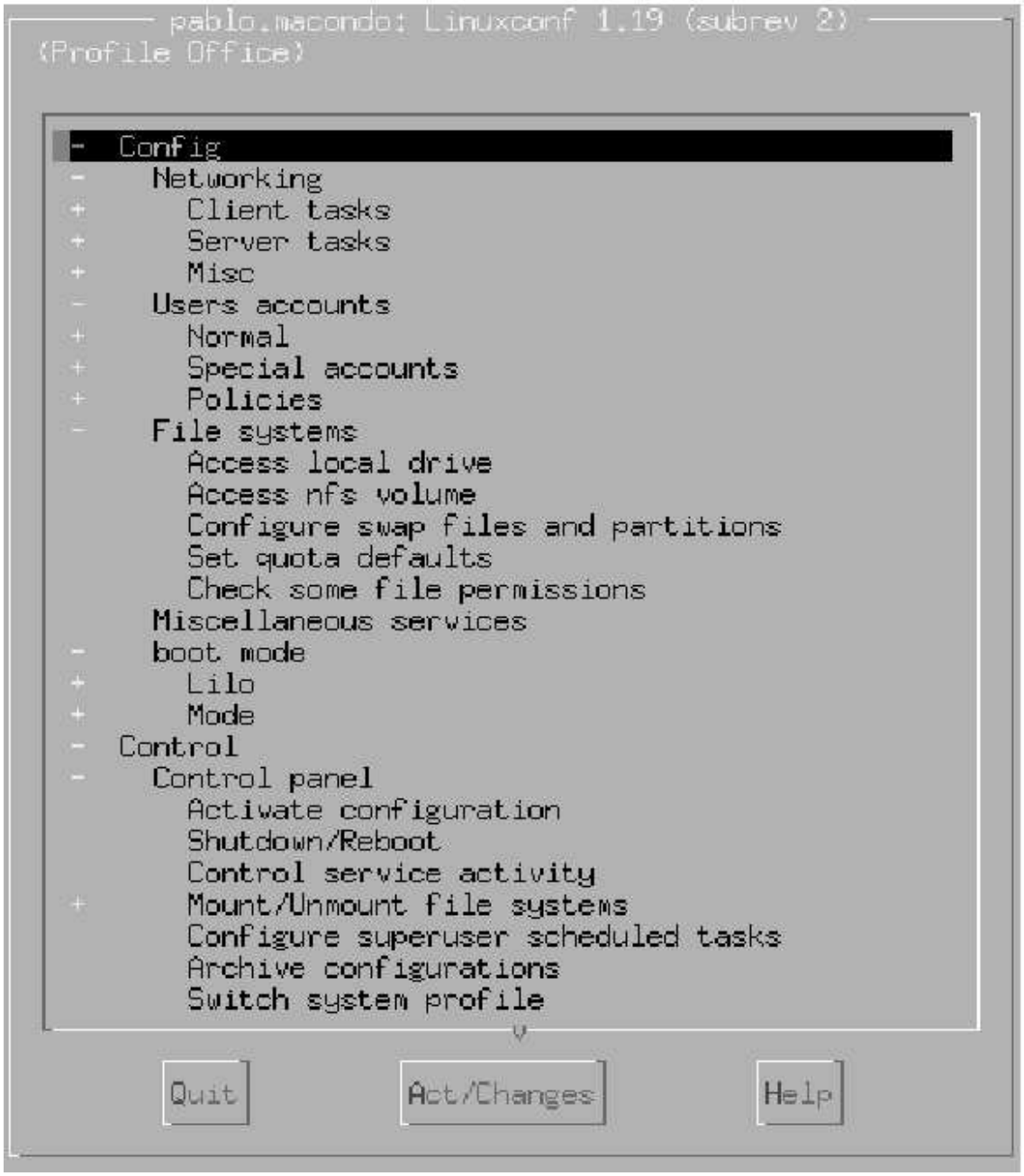
Folie 15

Linuxconf verwenden (2)

- browse Linuxconf-Baum
- mache Änderungen
- Vorschau auf Änderungen
- Aktiviere Änderungen
- verlassen

Folie 16

Linuxconf-Zelleninterface



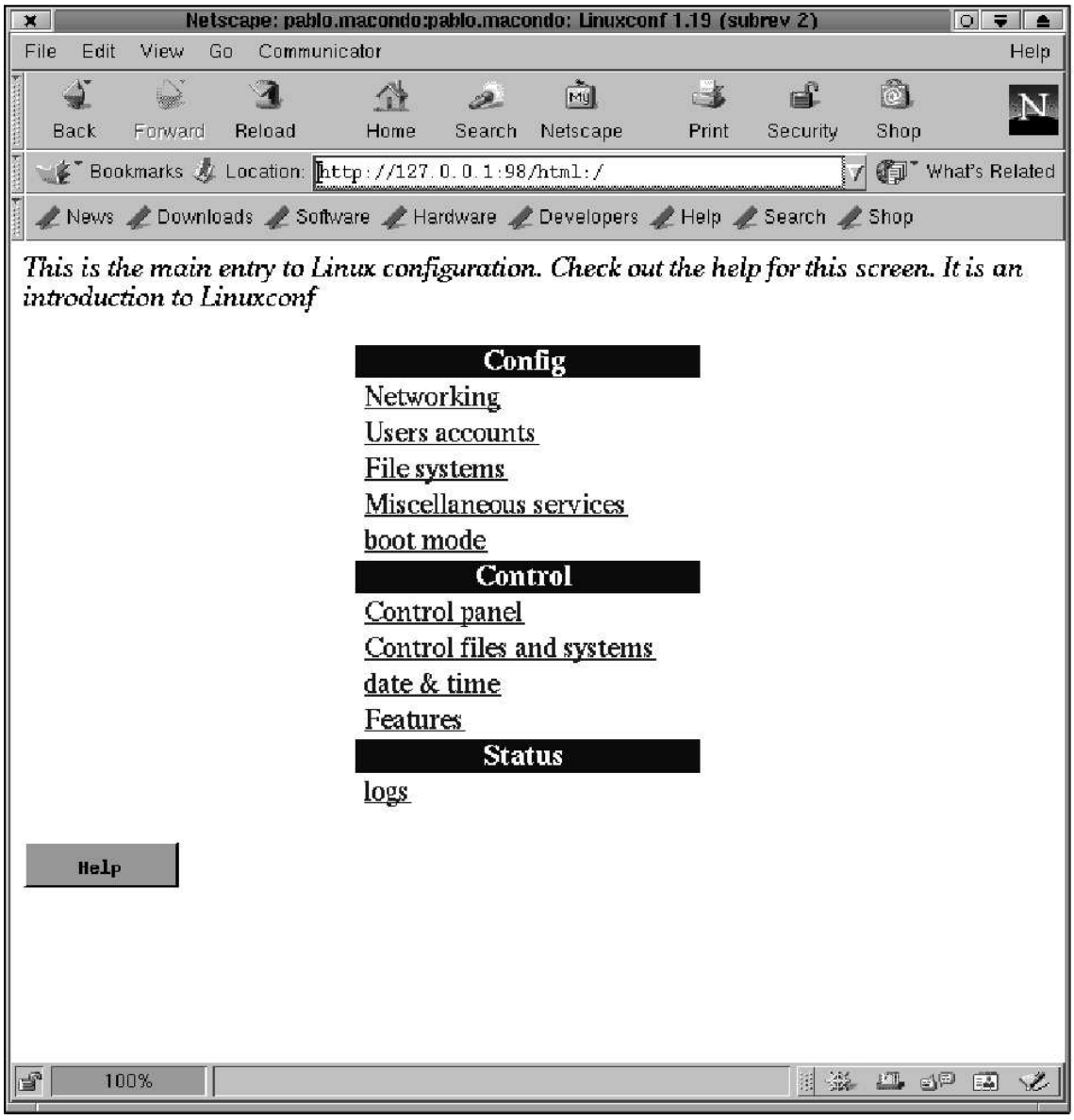
Linuxconf-GUI

Linuxconf-Netzwerkinterface

- standardmäßig deaktiviert
- einschalten:
  Config; Network; Misc; Linuxconf Net Access
  Internetdaemon (`(x)inetd`) muss laufen
- erreichbar am Port 98 (`http://myhost.mydomain:98`)
- Sicherheit!!!
  Hostaccess, verschlüsseltes Passwort, Browser speichert Passwort

Folie 17

Linuxconf-Webinterface



Webmin
ist ein relativ neues, netzwerkbasiertes Tool zur Systemadministration. Es verbindet einen modularen Aufbau mit dem Konzept einer plattformübergreifenden graphischen Administrationsoberfläche. Webmin unterstützt alle wichtigen Unix-Versionen, ist aber zB in der aktuellen RedHat-Distribution nicht enthalten. Rpm-Pakete kann man aber von der Webmin Homepage `http://www.webmin.com` herunterladen und sind problemlos installierbar. Für die Benutzung von Webmin gilt Ähnliches wie für das Netzwerkinterface von Linuxconf: der Server ist mit einem beliebigen Webbrowser auf dem Port 10000 (also unter `http://myhost.mydomain:10000`) erreichbar. Eine SSL-Unterstützung muss „per Hand“ eingebaut werden (siehe dazu `http://www.webmin.com/webmin/ssl.html`).

Neben diesen distributionsunabhängigen, bzw. distributionsübergreifenden Konfigurationstools stellen einige Distributoren eigene, auf genau diese Distribution zugeschnittene Werkzeuge zur Verfügung. Beispiele hierfür sind folgende

Distributionsspezifische Tools

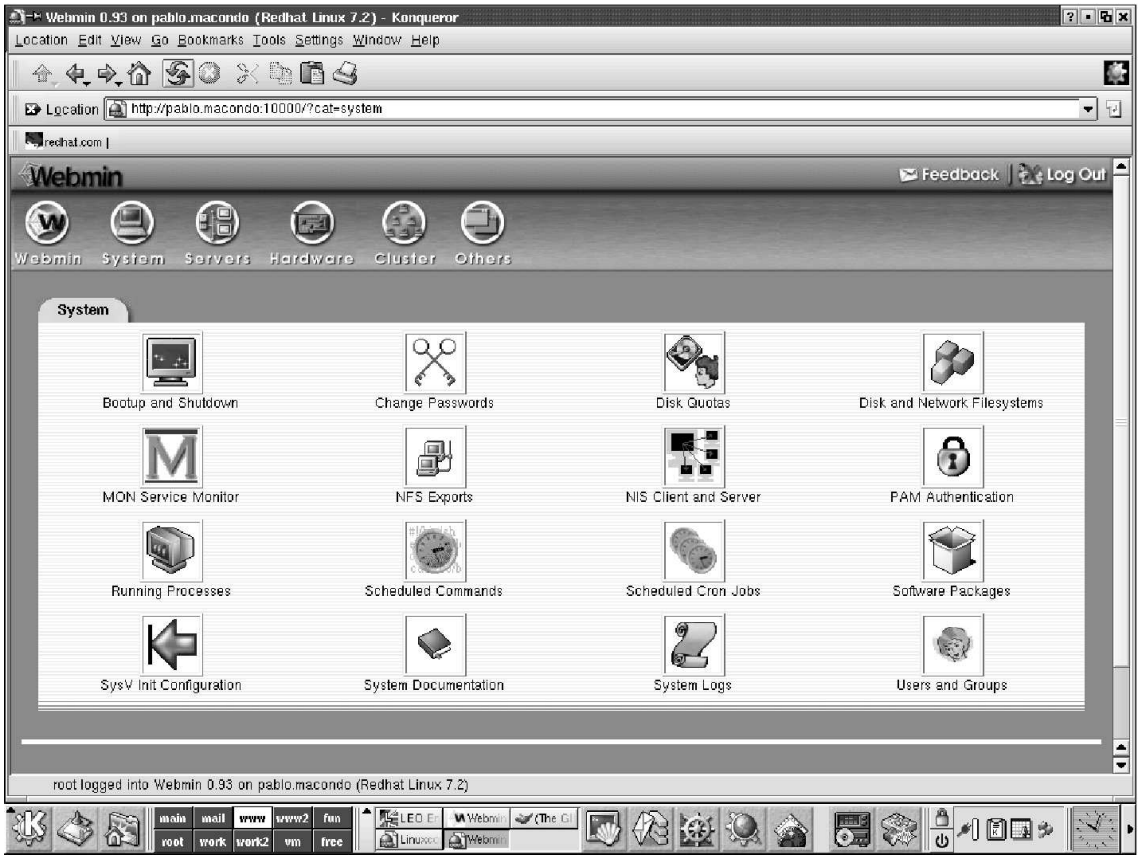
- *YaST* („Yet another Setup Tool“) bzw. YaST2 ist das Administrationswerkzeug von SuSE, das eine einfache Installation und Administration dieser Distributionen ermöglicht. YaST kann in anderen Distributionen nicht verwendet werden und ist nicht (einfach) erweiterbar. Andererseits ist es schwierig, SuSE-Distributionen ohne YaST zu konfigurieren; YaST schreibt das globale Konfigurationsfile `/etc/rc.config` und erzeugt mittels Aufrufs von `rc.SuSEconfig` daraus die entsprechenden Konfigurationsdateien für die Dienste. Händisch editierte Dateien werden dabei überschrieben, sodass jede Konfigurationsarbeit außerhalb von YaST sehr erschwert wird.

Webmin

- webbasiertes Konfigurationstool
- unterstützt neben Linux alle wichtigen Unix-Versionen
- Open Source (BSD Open Source License)
- modulare Struktur
- plattformunabhängige Konfigurationsumgebung

Folie 18

Webmin


Webmin 0.93 on pablo.macondo (Redhat Linux 7.2) - Konqueror
Location Edit View Go Bookmarks Tools Settings Window Help
Location http://pablo.macondo:10000/?cat=system
redhat.com |
Webmin
Feedback | Log Out
Webmin System Servers Hardware Cluster Others
System
Bootup and Shutdown
Change Passwords
Disk Quotas
Disk and Network Filesystems
MON Service Monitor
NFS Exports
NIS Client and Server
PAM Authentication
Running Processes
Scheduled Commands
Scheduled Cron Jobs
Software Packages
SysV Init Configuration
System Documentation
System Logs
Users and Groups
root logged into Webmin 0.93 on pablo.macondo (Redhat Linux 7.2)
main mail www www2 fun
root work work2 vm free

- *Lisa* ist das Administrationstool von Caldera Open Linux und ebenfalls nur für diese Distribution verwendbar.

- Debian verwendet für mehrere Aufgaben kleine spezialisierte Tools, zB `modconf`, `tzconfig`, `rconf`.

- *Control-panel* ist ein graphisches Administrationstool für RedHat-Linux, kann aber auch unter Mandrake, Turbo Linux und einigen anderen Distributionen verwendet werden. Seit RedHat-7.2 wird es allerdings nicht mehr unterstützt und es gibt nur nur mehr seine KDE-Version `kontrol-panel`. Das k(c)ontrol-panel stellt einen graphischen Launcher für verschiedene Tools zur Verfügung. Vor dem Start des entsprechenden Tools wird das Rootpasswort abgefragt.

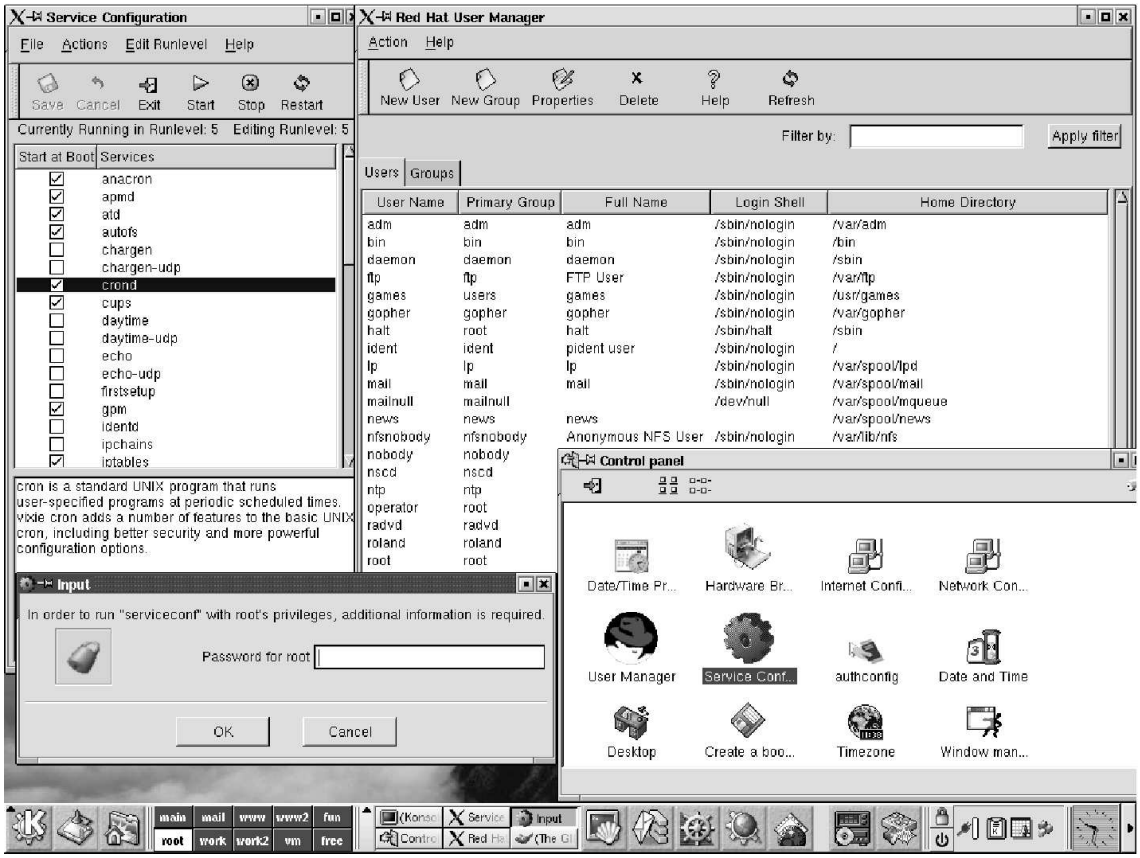
Darüber hinaus integrieren manche Distributionen verschiedene grafische Tools zur Systemadministration in die Desktopsysteme KDE und Gnome, die meist über das Startmenü und das Untermenü „System“ aufrufbar sind.
Ein Beispiel ist etwa unter RedHat das KDE Tool zu Benutzerverwaltung `kuser`. Unter SuSE ist Yast2 bzw. seine Module über KDE- oder Gnome Menüs erreichbar.

Distributionsspezifische Tools

- YaST, YaST2 (SuSE)
- Lisa (Caldera Open Linux)
- (c)kontrol-panel (RedHat, Mandrake, ...)
  - Systemdienste (Runlevel Editor)
  - Zeit
  - Netzwerk
  - Modem oder ISDN Anschluss
  - Usermanager
  - Authentifizierung

Folie 19

Kontrol-panel



3 Benutzerverwaltung

In einem Multiuser-Betriebssystem verfügt jeder Benutzer über einen von anderen Usern getrennten Bereich, wo er persönliche Daten und Einstellungen für die von ihm benutzten Programme ablegen kann. Dieser Bereich (das Heimat- oder Homeverzeichnis) zusammen mit Informationen über den Benutzer, wie etwa Benutzername, Passwort, Gruppenzugehörigkeit, Loginshell etc., die in globalen Konfigurationsdateien gespeichert sind, definiert einen „Benutzer-Account“. In diesem Kapitel beschreiben wir, wie Accounts angelegt, verwaltet und gelöscht werden.

3.1 Accountinformationen

Wichtige Informationen zu jedem Account befinden sich in den drei Dateien `/etc/passwd`, `/etc/group` und (meistens) `/etc/shadow`, deren Inhalt und Syntax wir nun im Detail erklären.

Die *Passwortdatei* `/etc/passwd` enthält grundlegende Informationen zu allen Accounts. Jede Zeile in der Datei definiert einen Account und ist in sieben durch Doppelpunkte separierte Felder unterteilt. Im 1. Feld ist der Benutzername (Accountname oder Loginname) gespeichert, das 2. Feld ist das das Passwortfeld. In den weiteren Feldern sind UID, GID, die GECOS-Information, das Homedirectory und die Loginshell angegeben.

Benutzernamen unter Unix haben üblicherweise maximal 8 Buchstaben und sind durchgehend klein geschrieben; prinzipiell sind Benutzernamen zwar case-sensitive, aber verschiedene Programme (zB der Mailserver Sendmail) setzen Kleinschreibung der Accountnamen voraus. Die Accountnamen sollten leicht zu merken sein und einfach mit dem Namen des Benutzers assoziiert werden können (Vorname, Nachnamen, Kombination aus beiden, etc.) und sollten (um Konflikte mit älterer Software zu vermeiden) nur aus alphanumerischen Zeichen bestehen.

Acounts

Was ist ein Account?

- Daten des Benutzers
  - Texte, Tabellen, Bilder, Mail, ...
  - persönliche Konfigurationen
    zB der Desktopsysteme, Netscape, Mailreader, ...
- Informationen über den Benutzer
  - Loginname, Passwort, Gruppenzugehörigkeit, ...
  - gespeichert in globalen Konfigurationsdateien
    - `etc/passwd`
    - `/etc/group`
    - `/etc/shadow`

Folie 20

Das *Passwortfeld* dient zur Speicherung des *verschlüsselten* Passworts. Oft wird allerdings aus Sicherheitsgründen das Passwort nicht in der Passwortdatei, sondern in `/etc/shadow` gespeichert (siehe unten). Unix verwendet standardmäßig DES-Verschlüsselung, die allerdings nur eine Passwortlänge von (unverschlüsselt) 8 Zeichen erlaubt; längere Passwörter werden zwar akzeptiert; es sind aber nur die ersten 8 Stellen signifikant. Verschlüsselten DES-Passwörter sind unabhängig von der Länge des unverschlüsselten Passworts 13 Zeichen lang. Die meisten Linux-Distributionen unterstützen auch MD5-verschlüsselte Passwörter. Hier ist die unverschlüsselte Länge mit 31 Zeichen limitiert, die verschlüsselten beginnen immer mit der Zeichenfolge 1. Da die Länge der Passwörter signifikant die Möglichkeit erfolgreicher Wörterbuchattacken beeinflusst, sollten unter Linux MD5 Passwörter verwendet werden. Einzige Ausnahme sind Systeme, wo Kompatibilität mit älteren Unix-Varianten gewahrt bleiben muss (zB NIS mit Linux und DEC Rechnern), die nur DES-Verschlüsselung unterstützen. Passwörter können mittels des Kommandos `passwd` geändert werden. Alle Benutzer außer root werden dabei aber nach dem alten Passwort des entsprechenden Accounts gefragt. Root kann also (als einziger) neue Passwörter setzen, ohne die alten zu kennen; die unverschlüsselten Passwörter kennt selbst root nicht.
Systemintern wird jeder Account durch eine eindeutige Zahl (32 Bit Integer), die *User Identity (UID)* repräsentiert, die im 3. Feld der Passwortdatei angegeben ist. Unter Linux können derzeit 65535 UIDs vergeben werden (limitiert durch die Länge des UID-Feldes im ext2 Filesystem; diese Zahl wird aber zukünftig noch steigen). Das System kennt nicht nur die „normalen“ Accounts (*Useraccounts*), die Benutzern die normale Verwendung des Systems ermöglichen, sondern sogenannte *Systemaccounts*. Der wichtigste von ihnen ist der *Rootaccount*, der Account der Systemadministrators. Dieser hat in allen Unix-Varianten die UID 0, sein Eintrag befindet sich in der ersten Zeile von `/etc/passwd`. Darüber hinaus laufen einige Systemdienstprogramme und Daemonen unter eigenen Accounts (auch privilegierte Pseudobenutzer genannt), um etwa Daten in einem reservierten Bereich ablegen zu können, oder damit spezifizierte Rechte an die Prozesse vergeben werden können. Diese Systemaccounts verfügen über niedere UIDs (unter Linux unter 100)

und können (und sollen auch) nicht zum normalen Login am System verwendet werden. Die „normalen“ Benutzeraccounts verfügen über höhere UIDs, unter RedHat ab 401.
Die Accounts am System können in *Gruppen* zusammengefasst werden, damit bestimmte Rechte (etwa im Dateisystem) an mehrere Benutzer gleichzeitig vergeben werden können. Die Information über die Gruppenzugehörigkeiten der Benutzer sind in der Datei `/etc/group` (siehe unten) festgelegt. Die Gruppen werden analog zu den Accounts systemintern über eindeutige Nummern verwaltet, die sogenannte *Gruppen Identity (GID)*. Jeder Account am System muss mindestens einer Gruppe zugeordnet sein, seiner Primary oder Login Gruppe, die im 4. Feld der Passwortdatei eingetragen ist. In vielen Linux-Distributionen ist es Standard, für jeden Benutzer eine eigene Logingruppe zu verwenden, die denselben Namen wie der Accounts selbst hat und die GID gleich der UID des Accounts ist.
Das 5. Feld der Passwortdatei ist das sogenannte GECOS (oder GCOS) Feld. (Der Name ist historisch und rührt daher, dass dieses Feld in den Bell Labs dazu verwendet wurde, batch Jobs auf den GECOS Mainframe zu überragen.) Die Syntax des GECOS Feldes ist nicht in allen Unix-Varianten gleich. Üblicherweise wird der erste Eintrag der durch Beistriche separierten Liste als voller Name des Benutzers interpretiert, dann folgen Büroadresse, Bürotelefonnummer und Heimtelefonnummer. Das `finger`-Kommando wertete (unter anderem) das GECOS Feld der Passwortdatei aus, die Einträge können mit dem `chfn`-Befehl (passwortgesichert) verändert werden.

Die Passwortdatei `/etc/passwd`

jede Zeile ein Account, 7 doppelpunkt-separierte Felder

1. Benutzername (Account- oder Loginname)
2. Passwortfeld (DES oder MD5 verschlüsselt), oft Shadowpasswörter
3. UID (User Identity): 32 Bit Integer, max. 65535
4. GID (Group Identity): definiert Logingruppe
5. GECOS: voller Name, Adresse und Telefon
6. Homedirectory
7. Loginshell (`/etc/shells`)

Folie 21

Beispieldatei /etc/passwd

```
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
mail:x:8:12:mail:/var/spool/mail:
ftp:x:14:50:FTP User:/pusers/ftp:
nobody:x:99:99:Nobody:/:
roland:x:401:401:Roland Steinbauer,Zi 407,,:/home/roland:/bin/bash
oliver:x:402:402:Oliver Fasching:/pusers/oliver:/bin/bash
flo:x:403:511:Florian Wisser:/pusers/flo:/bin/bash
test:x:500:8076::/home/test:/bin/bash
```

Folie 22

Das *Homedirectory* des Benutzers ist im 6. Feld der Passwortdatei gespeichert und heißt meistens `/home/username`. Ist das Homedirectory nicht vorhanden wird der Benutzer unter Linux auf der Textkonsole mit Homedirectory `/` eingeloggt, im graphischen Modus ist dann kein Login möglich.
Im 7. Feld schließlich wird die *Loginshell* festgelegt, d.h. jene Shell, in die der Benutzer unmittelbar nach dem Login gelangt. Unter Linux kann die Loginshell eines Benutzers (passwortgeschützt) mittels `chsh`-Befehl geändert werden. Erlaubt sind alle Shells, die in `/etc/shells` aufgelistet sind.

Die Passwortdatei muss für alle Benutzer am System lesbar sein, damit die hier gespeicherten wichtigen Informationen allen Benutzern, bzw. den von ihnen verwendeten Programmen zugänglich sind. Aus diesem Grund (vor allem um Wörterbuchattacken zu erschweren) werden in modernen Unix-Systemen die verschlüsselten Passwörter selbst in einer anderen Datei und nicht in `/etc/passwd` gespeichert. Besitzt ein Angreifer einmal ein verschlüsseltes Passwort, so kann er eines der verbreiteten Cracker-Programme verwenden, die eines ganzen Wörterbuches verschlüsseln und die so erhaltenen Strings mit den Passwörtern vergleichen. Sind die Passwörter schlecht oder zu kurz gewählt ist ein Knacken in wenigen Minuten bis Stunden möglich (einen entsprechend schnellen Rechner vorausgesetzt).

Daher wird auf modernen Unix-Systemen das Passwort zusammen mit einigen anderen passwortspezifischen Daten in der *Shadowdatei* `/etc/shadow` gespeichert; diese ist nur für root lesbar. Im 2. Feld der Passwortdatei ist dann ein „x“ eingetragen und man ist in der paradoxen Situation, dass in der Passwortdatei (fast) alles, außer dem Passwort selbst steht.
Die Syntax der Shadowdatei ist ähnlich der Syntax von `/etc/passwd`; jede Zeile beinhaltet die Einträge zu einem Account, die neun Felder sind durch Doppelpunkte getrennt. Der erste Eintrag ist der Benutzername und stellt so die Verbindung zwischen den entsprechenden Zeilen in der Passwort- und der Shadowdatei her.

Die Shadowdatei `/etc/shadow`

je Account eine Zeile
9 doppelpunkt-separierte Felder

1. Benutzername
2. verschlüsseltes Passwort
3. Letzter Passwortwechsel (Tage seit 1.1.1970)
4. Min. Tage zwischen Wortwechsel
5. Max. Tage zwischen Passwortwechsel
6. Anzahl Tage Warnung vor Passwortablauf
7. Anzahl Tage Accountablauf nach Passwortablauf
8. Accountablauf
9. Flags (unbenutzt)

Folie 23

Beispieldatei /etc/shadow

```
root:$1$D1233U6v$37fJ9VUFuLc6jX3df82MQ1:11635:0:99999:7:::
bin:*:11635:0:99999:7:::
daemon:*:11635:0:99999:7:::
shutdown:*:11635:0:99999:7:::
mail:*:11635:0:99999:7:::
ftp:*:11635:0:99999:7:::
nobody:*:11635:0:99999:7:::
roland:$1$RRO2LY5n$ertgHt0lnUhZYl0PdJ2uW1:11637:0:99999:7:::
oliver:$1$3HcA/1AK$1ZJs6DRHZRweqMTEJ8QKC0:11632:0:99999:7:::
flo:4liq2wkTRWbhJ:11619:0:99999:7:::134547972
test:$1$2759ACul$ozJrregreD1OAaxaQP8g0:11686:0:99999:7:::
```

Folie 24

Im 2. Feld der Shadowdatei steht das (verschlüsselte) Passwort; für die Verschlüsselung gilt dasselbe wie für Passwörter in `/etc/passwd`. Die Bedeutung der weiteren Felder, die alle optional sind, wird auf Folie 23 erklärt.

Um von einem System ohne Shadowpasswörter (also wo die Passwörter (noch) in `/etc/passwd` gespeichert sind) auf eines mit Shadowpasswörtern umzusteigen, gibt es das Tool `pwconv`, das aus `/etc/passwd` die entsprechende `/etc/shadow` erzeugt. Die umgekehrte Richtung bewältigt man mit `pwunconv`.

In größeren Systemen (zB Universitätsinstitut) entsteht rasch eine gewisse Notwendigkeit, Benutzer in *Gruppen* (zB Professoren, Assistenten, Studenten, Sekretariat, ...) einzuteilen. So kann man ein effektives Rechtekonzept entwickeln, um zB allen Sekretärinnen die Möglichkeit zu bieten, dieselbe institutsweite Email-Adresse zu verwenden. Studenten sollte es jedoch verboten sein, die Daten der Professoren (zB Prüfungstexte) zu lesen. Information über die Gruppen am System werden in `/etc/group` gespeichert.
Die Syntax dieser Datei ist ähnlich der Syntax von Passwort- und Shadowdatei. Jede Zeile definiert eine Gruppe, die vier Felder sind durch Doppelpunkte getrennt. Der erste Eintrag ist der Name der Gruppe; unter Linux wird meistens pro Benutzer eine eigene Gruppe angelegt (siehe oben). Es können aber auch darüber hinaus Gruppen definiert werden. Für die Gruppennamen gilt Ähnliches wie für die Loginnamen. Im 3. Feld wird die GID der entsprechenden Gruppe und im 4. Feld als kommaseparierte Liste die Mitglieder (resp. ihre Benutzernamen) der Gruppe eingetragen.
Ist ein User Mitglied mehrerer Gruppen, so bestimmt die Gesamtheit seiner Gruppenzugehörigkeiten (Groupset) seine Gruppenrechte im System. Der Befehl `groups` zeigt an welchen Gruppen ein Benutzer angehört. Neuerstellte Dateien werden immer mit Gruppenbesitzer gleich der Logingruppe verwendet, es sei denn, der Benutzer loggt mittels `newgrp my2ndgrp` in die Gruppe my2ndgrp ein; er bleibt als Benutzer eingeloggt, neue Dateien sind haben dann my2ndgrp als Gruppenbesitzer.

Mittels `newgrp` kann man (ohne Passwort) nur in Gruppen einloggen, deren Mitglied man ist. Es gibt prinzipiell die Möglichkeit den Login in Gruppen (wo der Benutzer nicht Mitglied ist) mit einem Passwort zu schützen. Das 2. Feld in `/etc/group` (oder `/etc/gshadow` analog zu `/etc/shadow`) enthält dann das verschlüsselte Gruppenpasswort. Dieser Mechanismus wird aber sehr selten verwendet; meist ist im 2. Feld von `/etc/group` ein „x“ eingetragen und der Login in Gruppen ohne Mitglied zu sein ist nicht möglich.

Die Gruppendatei `/etc/group`

Jede Zeile eine Gruppe; 4 doppelpunkt-separierte Felder

1. Gruppenname
2. Gruppenpasswort (selten verwendet)
3. GID
4. Gruppenmitglieder als kommaseparierte Liste

Folie 25

Beispieldatei `/etc/group`

```
root:x:0:root
bin:x:1:root,bin,daemon
daemon:x:2:root,bin,daemon
sys:x:3:root,bin,adm
mail:x:12:mail
ftp:x:50:
roland:x:401:
diana:x:500:roland,mike,migro,diana,gue,michael,irina
cc:x:501:flo,susanne,helpdesk,roland,herman
test:x:8076:roland,test
```

Folie 26

Ändern/Abfragen der Accountinformation

- `passwd` Passwort ändern
- `chsh` Loginshell ändern
- `chfn` GECOS Feld ändern
- `finger` Benutzerinfo abfragen
- `groups` Gruppenzugehörigkeit abfragen
- `newgrp` Gruppenlogin

Folie 27

3.2 Accountverwaltung

Einige Konfigurationsarbeiten an Accounts, die jeder Benutzer selbst durchführen kann (Ändern von Passwort, Loginshell, GECOS-Einträgen) haben wir im vorigen Abschnitt behandelt. Hier besprechen wir nun die dem Administrator vorbehaltenen Konfigurationsänderungen, im besonderen, das Anlegen, Sperren und Löschen von Benutzeraccounts.

Der kleinste Nenner bei der Accountverwaltung zwischen allen Unix-Systemen ist das direkte Editieren der Konfigurationsdateien `etc/passwd, /etc/shadow` und `/etc/group`. Dies sollte jedoch nicht mit einem gewöhnlichen Editoraufruf, sondern mittels Aufruf von `vipw`, `vigr` bzw. `vipw -s` (auf manchen Systemen für `/etc/shadow`) geschehen, die die Dateien für alle anderen Prozesse während des Editiervorgangs sperren. Als Standard wird dabei der `vi` Editor verwendet; dieses Verhalten kann durch setzten der Umgebungsvariable `EDITOR` verändert werden. Diese Art der Konfiguration der Useraccounts ist allerdings sehr gefährlich und wird(/sollte) nur in Ausnahmefällen verwendet (werden). Ein zu viel weggelöschter Doppelpunkt kann zB die gesamte Passwortdatei invalidieren, sprich: Niemand – auch nicht root – kann mehr einloggen. Daher empfiehlt es sich, nach dem Abspeichern von `/etc/passwd` und `/etc/shadow` und vor dem Ausloggen aus der root-Shell auf einer parallelen Konsole zu testen, ob root noch einloggen kann.

Neben dieser „low-level“ Methode stellen alle Unix-Varianten ähnliche Kommandozeilenwerkzeuge zur Benutzerverwaltung zur Verfügung. Unter Linux weit verbreitet sind `useradd` und `userdel` zum Erstellen bzw. Löschen von Useraccounts, analog dazu `gropuadd` bzw. `groupdel` und zum Ändern der Konfiguration eines Accounts bzw. einer Gruppe `usermod` und `groupmod`. Wir besprechen diese Utilities später jeweils im Kontext ihrer Funktionen.

Accountverwaltung

- User: Ändern der Accountinformationen
  (Passwort, Loginshell, GECOS)
- root:
  - Anlegen
  - Konfigurieren
  - Sperren
  - Löschen

  von Accounts

Folie 28

Editieren der Konfigfiles

- kleinster gemeinsamer Nenner auf allen Unix-Systemen
- statt normalem Editoraufruf (Locking!)
  - `vipw`
  - `vigr`
- Gefährlich! Tippfehler können ganze Datei invalidieren
  vor root-Logout testen!!!

Folie 29

Utilities

- Kommandozeilenwerkzeuge
  - Anlegen: `useradd, passwd, groupadd`
  - Konfig: `usermod, groupmod`
  - Löschen: `userdel, groupdel`
  - Automatisierung durch Skripts
- Grafische Tools
  YaST, Linuxconf, Webmin, Kuser, ...

Folie 30

In größeren (Netzwerk-) Systemen wird der manuelle Einsatz dieser einfachen Tools schnell zu umständlich und die Praxis zeigt, dass man (zB in unserem fiktiven Institutsnetz) ziemlich bald ein angepasstes (adduser-)Script zum Anlegen von Accounts schreibt, das einen neuen Benutzer gleich in eine netzwerkweite Benutzerdatenbank einträgt und bequem aus einer Liste von Arbeitsgruppen seine Gruppenzugehörigkeit aussuchen lässt, oder etwa 100 Standardaccouts (für ein PC-Labor) mit Zufallspasswörtern inklusive der entsprechenden Anmeldeformulare erstellt.
Außerdem stellen fast alle Linux-Distributionen grafische Werkzeuge zur Benutzerverwaltung zur Verfügung, die sich allerdings schlecht für die oben angesprochen Automatisierung eignen, also nur für kleine Systeme oder das manuelle Nachbessern einiger weniger Konfigurationen in einem großen System sinnvoll zu verwenden sind (vgl. Kapitel 2).

Accounterstellung
Vor allem bei größeren Systemen ist es auf jeden Fall sinnvoll, neue Benutzer vor der Accounterstellung über die Gepflogenheiten des Systems zu informieren. Weiters ist es dringend zu empfehlen, eine Benutzungsordnung schriftlich auszuhändigen und eine Kopie vom neuen Benutzer unterschreiben zu lassen. Für weitere Überlegungen zu Benutzerordnung und Policy siehe Abschnitt 1.2.

Unabhängig vom verwendeten Tool müssen bei der Accounterstellung prinzipiell folgende Vorgänge erledigt werden: Zunächst ist ein Eintrag in der Passwortdatei nötig, der die entsprechenden Informationen über den Benutzer enthält. Weiters wird ein Eintrag im Gruppenfile und (eventuell) im Shadowfile benötigt. Ein wichtiger Punkt ist das setzten des Anfangspasswort. Hier werden oft folgende zwei Vorgehensweisen verwendet: Entweder der neue Benutzer gibt sein Anfangspasswort selbst ein (muss dazu also persönlich bei der Accounterstellung anwesend sein) oder der Administrator übermittelt dem neuen User ein (standardisiertes) Anfangspasswort, dass dieser aber beim ersten Login (per Konfiguration erzwungen) ändern muss. Schließlich muss das Homedirectory für den neuen Account erstellt und (eventuell) eine Anzahl von Standardkonfigurationsdateien (zB `~/.bashrc`) angelegt werden. Auf Mailservern müssen noch

Mailhomedirectory (enthält die Inbox des Users) und eventuell Mailaliases gesetzt werden.

Bei Verwendung der Kommandozeilenwerkzeuge geht man zweckmäßiger Weise gemäß Folie 32 vor. Zunächst wird mit `useradd username` der Eintrag in Passwort-, Gruppen- und Shadowdatei erzeugt und das Homedirectory angelegt. Außerdem kopiert `useradd` alle Dateien aus `/etc/skel/` ins neue Homedirectory; dort sollte der Administrator also alle benötigten Standardkonfigurationsdateien speichern. Unter RedHat legt `useradd` standardmäßig zu jedem neuen Benutzer eine Gruppe mit gleichem Namen und GID=UID an. Nun existiert der Account zwar, aber es ist kein Passwort gesetzt und somit noch kein Login möglich. Das Anfangspasswort muss root nun mittels `passwd username` setzen. Gegebenenfalls sind nun noch mittels `groupadd` eine neue Gruppe hinzuzufügen oder mittels `groupmod` der neue Benutzer einer bereits bestehenden Gruppe zuzuordnen. Für alle weiteren Details und vor allem die distributionsspezifischen Defaulteinstellungen siehe die Manpages der entsprechenden Befehle bzw. die lokale Dokumentation.

Account anlegen

- Benutzungsordnung!!!
- Einträge in Passwort-, Gruppen- und Shadowdatei
- Anfangspasswort setzen
- Homedirectories erstellen
- Anfangskonfigurationsdateien erstellen
- ...

Folie 31

Accounterstellung (Kommandozeile)

- `useradd` erzeugt
  - Einträge in Passwort-, Gruppen (Logingr.)- und Shadowdatei
  - Homedirectory
  - Anfangskonfigurationsdateien aus `/etc/skel/`
- `passwd` Anfangspasswort setzten
- `groupadd, groupmod` für weitere Gruppenzugehörigkeit

Folie 32

Accountkonfiguration
Neben den Konfigurationsarbeiten am Account, die der Benutzer selbst vornehmen kann, hat der Administrator die Möglichkeit alle Accountdaten zu verändern; dazu sind die Kommandozeilenwerkzeuge `usermod` und `groupmod` gedacht, die die entsprechenden Einträge in `/etc/passwd` und `/etc/shadow` bzw. in `/etc/group` modifizieren. Insbesondere können so Ablaufdatum des Accounts, Homedirectory und sogar der Name des Accounts geändert werden; letzteres ist allerdings nur dann möglich, wenn der entsprechende Benutzer keine Prozesse am System laufen hat.

Die Defaultwerte für die Usererstellung werden in der Datei `/etc/login.defs` festgelegt; eine Beispieldatei befindet sich auf Folie 33. Insbesondere werden hier die in `/etc/shadow` gespeicherten Einträge über erzwungene Passwortänderung und Vorwarnzeit festgelegt, wie auch die Mindestlänge des Passworts (für root nicht bindend) und die Bereiche für die automatisch fortlaufende UID und GID Erstellung.
Weitere Einstellungen zur Beschaffenheit des Passworts (etwa Check gegen zu einfaches Passwort) werden über das Pluggable Authentication Module (PAM) vorgenommen (siehe Kapitel 12.2 und `man passwd`).

/etc/login.defs

```
# Password aging controls:
#
PASS_MAX_DAYS   99999
PASS_MIN_DAYS   0
PASS_MIN_LEN    5
PASS_WARN_AGE   7
# Min/max values for automatic u/gid selection in useradd
D_MIN                   500
UID_MAX                 60000
GID_MIN                   500
GID_MAX                 60000
```

Folie 33

Accounts deaktivieren und löschen
Ist es aus gegebenen Gründen notwendig, Accounts zu *sperren* (Verstöße des Users gegen die Benutzungsordnung, Wartungsarbeiten, ...), gibt es mehrere Alternativen, zwischen denen root wählen kann. Die einfachste ist, im entspechenden Feld in der Passwortdatei eine nicht interaktive (nicht existierende) Shell (Standard: `/bin/false`) einzutragen; andererseits kann man in das Passwortfeld in `/etc/passwd` bzw. `/etc/shadow` einen `*` eintragen, oder einfacher ein Rufzeichen an den Anfang des verschlüsselten Passworts anfügen. Letzteres kann durch den Aufruf `passwd -l` (lock) geschehen und mittels der Option `-u` (unlock) wieder aufgehoben werden. Alle diese Varianten verhindern den Login des entsprechenden Benutzers, haben aber den Nachteil, dass dieser nicht erfährt, warum er nicht einloggen kann, was meistens eine Anfrage an den Administrator nach sich zieht.

Eine elegantere Methode ist es daher, eine Nachricht auszugeben zu lassen, wenn der Benutzer versucht, sich am System einzuloggen. Dazu verwendet man oft sog. „Tailscripts“ die als Loginshell in `/etc/passwd` eingetragen werden und eine bestimmten Text (zB „Your account has been temporarily disabled due to a security problem“) ausgeben.

Darüber hinaus gibt es Situationen, in denen man allen Benutzern den Zugang zum System verbieten will, zB wegen Wartungsarbeiten am Server, auf dem die Userdaten liegen etc. Man legt dann eine Datei `/etc/nologin` an, in der eine Erklärung zur Rechnersituation stehen sollte. Diese wird dann bei einem Loginversuche mit gültigem Passwort ausgegeben; zB wird bei laufendem Shutdown ein Nologinfile anlegt.

Accounts sperren

- nicht interaktive Loginshell (`/bin/false`)
- `*` im Passwortfeld
- `!` am Anfang des Passworts (`passwd -l`, `-u`)
- Tailscript (Benachrichtigung!)
- `/etc/nologin`-File (alle Benutzer und Nachricht)

Folie 34

Ein dauerhaftes Entfernen *(Löschen)* eines Benutzeraccounts solltenur im Einverständnis mit dem Benutzer geschehen und/oder diesem rechtzeitig vorher unter Nennung von Gründen angekündigt werden. Insbesonders ist zu klären, was mit den Daten des Benutzers passieren soll.

Beim effektiven Löschen eines Accounts sollte dieser zuerst deaktiviert werden, um sicherzustellen, dass keine Daten mehr verändert werden können (die dann möglicherweise nicht mehr mit einem eventuellen Backup übereinstimmen). Sodann entfernt man entweder mit dem oben genannten Tool `userdel` oder einem anderen Werkzeug die Einträge in den drei Benutzerverwaltungsdateien. Nachdem die offensichtlich dem User zugehörigen Dateien (sein Homeverzeichnis und etwaige Mail, Croneinträge, Druckerjobs, ...) gelöscht wurden (kann mittels geeigneter Optionen von `userdel` erledigt werden), sollte man sich daran machen, auch alle übrigen dem User gehörenden Dateien zu suchen und zu löschen (`find -user`).
Obwohl das Löschen von Benutzern in vielen Umgebungen weit seltener und wegen des Backups sensibler ist, sollte auch hier ziemlich bald ein automatisiertes Script dafür sorgen, „Karteileichen“ aus dem System zu entfernen, da unbenutzte Accounts immer ein Sicherheitsrisiko darstellt. Es fällt zB niemandem auf, wenn der Account durch unberechtigte Personen verwendet wird.

Benutzer entfernen

- Deaktivieren des Accounts
- Einverständnis/Benachrichtigung des Users
- Entfernen der relevanten Zeilen aus
  `/etc/passwd`, `/etc/group` und `/etc/shadow`
  zB mit `userdel`
- Löschen des Home-Verzeichnisses (ev. vorher Backup)
- Löschen der Mailbox (in `/var/spool/mail`), anderer Daten in `/var`
  (Cron- und Drucker-Jobs)
- Übrige Dateien des Benutzers (`find / -user name`) (Backup!?)
- Automatisierung, Karteileichen sind Sicherheitsrisiko

Folie 35

4 Software-Installation

Selbst auf dem vollständigsten System ist (natürlich) nur eine generell nützliche Auswahl an Programmpaketen installiert und so kommt jeder Administrator (oder auch Benutzer) früher oder später in die Lage, eine Anwendung selbst in das System einspielen zu müssen. In diesem Kapitel wird erklärt, wie unter Unix Software zu Paketen zusammengefasst wird, wo man die gewünschten Pakete findet und wie man sie in die Installation integriert; insbesondere besprechen wir das Softwarepackage-Management unter Linux.

Die einfachste Form, in der in der Unix-Welt Software verteilt wird, ist ein gezipptes Tarfile, auch *Tarball* genannt, das man an der Dateiendungen `tar.gz`, `tgz` (verkürzt) oder auch `tar.bz2` erkennt. (vgl. Teil 1, Abschnitt 6.4). Meist enthält es die zum Compilieren der Anwendungen benötigten (C-)Sourcecodes mit Anleitung und Dokumentation aber auch zusätzliche Files, wie etwa Vorlagen für Konfigurationsdateien.

Die meisten Linux-Distributionen stellen weitgehende Ansprüche bezüglich vereinheitlichter Konfiguration des Gesamtsystems, Abhängigkeiten von Softwarepaketen untereinander und eigener Pfadstandards. Daher verwenden heute beinahe alle Distributionen ein eigenes *Softwarepackage-Management* mit eigenem Package-Format und dazugehörigen Managementtools statt einfacher Tarballs; diese Packages enthalten statt des Sourcecodes (wie beim Tarball üblich) die gebrauchsfertigen für genau die entsprechende Distribution (und Version) compilierten Binärdateien (genauer gesagt gibt es auch Sourcecode Packages; siehe unten). Das Package-Management vereinfacht und vereinheitlicht (De)Installation und Upgrade von Softwarepaketen.
Am weitesten verbreitet ist das `rpm`-Format (RPM Package Manager, ursprünglich RedHat Package Manager), das neben RedHat auch SuSE, Mandrake und andere Distributionen verwenden. Achtung, das bedeutet *nicht*, dass ein SuSE-`rpm`-Paket auf einem RedHat-System problemlos installiert werden kann, da oft verschiedene Pfadkonventionen gelten oder zueinander nicht kompatible Versionen von Bibliotheken installiert sind.

Softwarepakete

- Tarballs (gezippte Tarfiles)
  - Sourcecode zum selber Compilieren
  - Anleitung, Dokumentation
  - Beispielkonfigurationsdateien, ...
- Package-Management
  - distributionsspezifische Standards
  - „fertige“ Programme

Folie 36

Package-Management

- Distributionen definieren eigene Standards (Pfade, Konfiguration, ...)
- Packages für Distribution
  enthalten
  - Programmdateien, Bibliotheken
  - Dokumentation, Programminformation
  - Konfigurationsfiles, ...

  ermöglicht einfaches und vereinheitlichtes
  - (De-)Installieren
  - Paketabfrage und -verifikation
  - Upgrade
  - Abhängigkeitsinformation

  Formate
  - `rpm` RedHat, SuSE, Mandrake, ...
  - `deb` Debian, Corel

Folie 37

Initiativen wie die *Linux Standard Base (LSB)* haben es sich aber zum Ziel gesetzt, die verschiedenen Linux-Distributionen auch in Bezug auf das Package-Management aneinander anzugleichen, sodass die Möglichkeit, in Zukunft ein `rpm`-Paket für alle Distributionen verwenden zu können, immer wahrscheinlicher wird.
Unter den großen Distributionen verwendet einzig Debian (und darauf aufbauende Distributionen wie Corel Linux etc.) nicht `rpm`, sondern das `deb`-Package-Format, das in der Konzeption ähnlich ist, aber seine eigenen Stärken und Schwächen hat.

Bevor wir auf die Details im Umgang mit Softwarepaketen eingehen befassen wir uns mit der Frage: Woher bekommt frau Linux Software?

4.1 Linux-Softwarequellen

Die erste Adresse für Linux-Software ist natürlich immer die Homepage der entsprechenden Distribution resp. ihre Mirrorseiten. Oft stellen die Distributoren auf ihrer Homepage weitere nicht direkt im Umfang der Basisinstallation enthaltenen oder nicht auf den Installations CDs mitgelieferte Softwarepakete (im geeigneten Package-Format) zur Verfügung. Beispiele sind hierfür etwa die RedHat PowerTools und die RedHat Contributed Packages. Darüberhinaus bietet das Internet schier unbegrenzte Möglichkeiten Linux-Software herunterzuladen.

Der größte Index für Open Source Software im Internet findet sich unter `http://sourceforge.net`. Über eine bequeme Suchmaske ist es einfach, die gewünschte Software zu finden. Die Links zu den Homepages bzw. Downloads der Softwareprojekte werden täglich aktualisiert und viele Informationen über freie Software werden angeboten. Ein ähnliches Service bietet `http://www.freshmeat.net`.
Mittels einer Suchmaschine wie `google.com` gelangt man im Allgemeinen ebenfalls recht schnell zur Homepage des jeweiligen Programms, wo man meistens den aktuellen Source-Tarball bekommt, oft aber auch schon vorcompilierte Pakete für die verschiedenen Distributionen.
Eine übrigens besonders in Österreich sehr attraktive Möglichkeit, Programme zu finden, ist der Goodie Domain Ser-

ver der TU Wien (`gd.tuwien.ac.at`). Unzählige von großen FTP-Servern mit freier Software werden hier täglich gespiegelt. Über ein Webinterface lässt sich gezielt nach dem gewünschten Programm suchen.

Software speziell im `rpm`-Format sucht man am besten unter `http://rpmfind.net` bzw. dem Mirrorserver `http://at.rpmfind.net`. Auch hier ist eine Suchmaske verfügbar. Es gibt aber ebenso nach mehreren Kriterien geordnete Indices. Außerdem ermöglicht das Paket `rpmfind` ein automatisches Suchen im Internet nach `rpm`-Paketen.

Softwarequellen

- CDs, Homepage der installierten Distribution
  (zB Contributed RedHat-Packages und RedHat-PowerTools)
- `http://sourceforge.net` großer Open Source Index
- `http://rpmfind.net` `rpm`-Datenbank
  (Mirror `http://at.rpmfind.net`)
- Suchen mittels `rpmfind`-Paket
- Homepage der jeweiligen Anwendung
- `gd.tuwien.ac.at`

Folie 38

4.2 Sourcepackages

Obwohl die Installation von `rpm`- oder `deb`-Paketen einfacher und wegen der Einhaltung der Distributionsstandards in jedem Falle vorzuziehen ist, kommt man als Administrator (oder auch Benutzer) oft in Situationen, wo die gewünschte Software nicht im richtigen Format vorhanden ist und man so gezwungen ist, ein Sourcepackage zu installieren. Gründe für das Fehlen von `rpm`- oder `deb`-Paketen können sein, dass eine Software besonders neu ist und noch keine Packages erzeugt wurden oder es sich um eine sehr exotische oder noch im Entwicklungsstadium befindliche Software handelt. In solchen Fällen bieten die Package-Managementtools der Distributionen die Möglichkeit selbst `rpm`- oder `deb`-Pakete zu bauen, die Software selbst muss aber zuerst compiliert, installiert und getestet werden.
Unter Linux gehört die GNU Compiler Collection (GCC) zum Umfang jeder Distribution; damit ist das compilieren von C/C++ Code und auch einer Reihe weiterer wichtiger Pragrammiersprachen möglich. Der Compiler selbst heißt – wie auch das ganze Paket – `gcc`, was hier aber GNU C Compiler bedeutet.
Schließlich verwenden die meisten anderen Unix-Versionen entweder kein Package-Management oder die Packages sind proprietär, sodass etwa auf AIX oder Solaris Systemen das Compilieren von Sourcepaketen eine sehr große Rolle spielt.

Wir diskutieren die Handhabung von Sourcepackages nun am Beispiel das Quellpakets von `ascii`, einem sehr einfachen Programm, das zu einer gegebenen Taste den ASCII-Code (und andere Informationen) zurück liefert. Nachdem das Paket von der via `freshmeat.net` gefundenen Seite `http://www.tuxedo.org/~esr/software.html` heruntergeladen wurde, kann man es mit `tar xzf ascii-3.0.tar.gz` entpacken (es empfiehlt sich übrigens, dies nicht direkt im Hauptpfad des Homeverzeichnisses auszuführen, sondern zB ein Verzeichnis `packages` anzulegen, in dem man mit den Paketen arbeiten kann; als root kann man auch das Directory `/usr/src/` verwenden). Wenn wir nun in das Verzeichnis wechseln und uns seinen Inhalt anschauen, sehen wir neben dem Sourcecode `ascii.c` und weiteren Dateien (zB `ascii.1` ist das Manual, das nach der Installation des Paketes bei `man ascii` angezeigt wird) auch noch die Datei `README`, die in beinahe jedem Source-

package vorhanden sein sollte; sie enthält gewöhnlich Informationen über das Programm selbst, aber auch über die Schritte, die zum Compilieren notwendig sind. Diese Datei sollte man auf jeden Fall zumindest kurz durchschauen, damit man mögliche Probleme beim Compilieren hier schon verstehen kann; eventuell gibt es auch noch eine Datei `INSTALL`, die besonders auf den Compiliervorgang eingeht. Die für uns wichtigste Datei im Verzeichnis `ascii-3.0` ist aber das `Makefile`, das alle Regeln enthält, wie ein in C (oder auch anderen Sprachen wie C++) geschriebenes Programm aus seinen einzelnen Sourcefiles zu einer ausführbaren Datei zusammengebaut werden soll. Dessen Funktionsweise muss man aber nur verstehen, wenn man das Paket selbst verändern will. Im Normalfall reicht es aus, `make` einzugeben, das Makefile wird abgearbeitet und das Programm – hoffentlich erfolgreich – compiliert. Wenn man keine Fehlermeldung erhalten hat, ist es auch gleich möglich, mit einem ähnlichen Kommando, nämlich `make install` das erzeugte Programm inklusive Manpage etc. zu installieren. Achtung: Dazu muss man `root` sein, da man sonst nicht in Verzeichnissen wie `/usr/bin` schreiben darf, wo ja die systemweit zu verwendenden Binaries abgespeichert werden müssen.
Ein normaler Benutzer kann die mit `make` erstellten Binärfiles an einen geeigneten Platz in seinem Homedirectory (zB `$HOME/bin`) kopieren und von dort aus ausführen.

Sourcepackages installieren

- Paket herunterladen
- Mit `tar` entpacken
- In das erzeugte Verzeichnis wechseln
- `README` (und `INSTALL`) lesen
- Gibt es ein `configure`?
  Wenn ja `./configure` ausführen
- `make`
- `make install` als `root`
  sonst Binaries in eigenen Suchpfad kopieren

Folie 39

Natürlich können beim Compilieren Fehler aufgetreten sein, weil zB auf dem eigenen System Dateien nicht installiert sind, die zum Bauen unbedingt erforderlich sind (meistens C-Headerfiles) oder der Programmautor den Quellcode nicht richtig getestet hat etc. Um solche Probleme möglichst schon am Anfang des (oft sehr lange dauernden) Compiliervorgangs zu erkennen, wurde ein Tool namens `autoconf` entwickelt, das insbesondere bei moderneren und größeren Paketen zum Einsatz kommt. Dieses untersucht die aktuelle Installation auf Verträglichkeit mit dem Sourcecode (also zB unterstützt der verwendete Compiler bestimmte Optionen, sind alle Include-Dateien vorhanden, ...). Als Benutzer kann man ganz einfach erkennen, ob ein Sourcepackage `autoconf` verwendet, indem man nachsieht, ob es im Hauptverzeichnis des entpackten Tarballs eine Datei namens `configure` gibt. Wenn diese vorhanden ist, muss man sie *vor* dem Kommando `make` ausführen (also `./configure` aufrufen). Von den nun relativ schnell vorbei fliegenden Meldungen lässt man sich besser nicht irritieren, `configure` gibt bei einem wirklichen Fehler eine (aussagekräftige) Fehlermeldung, alle anderen Checks kann man gewöhnlich ignorieren. Wenn `configure` fertig ist, geht man wie oben weiter vor: nach einem `make` und `make install` ist das Paket installiert und meistens auch schon einsatzbereit.

Wenn `configure` wirklich einmal eine Fehlermeldung über eine fehlende Datei ausgibt, ist die Chance relativ groß, diese Datei in einem Paket der jeweiligen Distribution zu finden (wenn nicht in den Dateien `README` oder `INSTALL` explizit auf eine andere Möglichkeit hingewiesen wird): Die Namen von Paketen, die Headerfiles zu einer Bibliothek enthalten, enden gewöhnlich auf `-dev` (SuSE, Debian) oder `-devel` (RedHat, Mandrake); zB befindet sich die Datei `/usr/include/png.h` unter RedHat im Paket `libpng-devel`.

4.3 Softwarepackage-Management

Der *RPM Package Manager*, kurz `rpm` ist das Tool zur Verwaltung von Softwarepaketen auf `rpm`-basierten Systemen. Ursprünglich von RedHat entwickelt und GPL lizenziert, ist es heute weit verbreitet. Die Homepage befindet sich auf `http://www.rpm.org`. Das `rpm`-Format sieht vor, dass Pakete neben den (binären) Programmdateien und Dokumentation auch Pre- und Post-Installationsscripts sowie Deinstallationsskripts und vor allem Paketabhängigkeits Informationen enthalten. Eine Berücksichtigung letzterer garantiert, dass ein installiertes Paket auch richtig funktioniert, weil alle vom Paket benötigten Funktionen bereits von anderen Paketen zur Verfügung gestellt werden. Beim Upgraden von Programmpaketen kann es manchmal Probleme mit eben diesen Abhängigkeiten geben, da Zirkel entstehen können; die Abhängigkeitsprüfung kann dann mit der Option `--nodeps` abgeschaltet werden (siehe Syntax auf den Folien). Weiters überprüft `rpm` bei der Installation eines Pakets, ob Konflikte vorliegen, d.h. ob das Paket eine Datei enthält, die im System schon vorhanden ist und zu einem anderen Paket gehört. Diese Überprüfung kann mit der Option `--replacefiles` überschrieben werden. Alle Informationen über installierte Pakete werden in der `rpm`-Datenbank in `/var/lib/rpm/` am System gespeichert.

Der Dateiname von `rpm`-Paketen setzt sich aus folgenden Teilen zusammen: Paketname, Versionsnr., Releasenr., Architektur und der Endung rpm. D.h. der Name `ftp-0.17-7.i386.rpm` bedeutet, dass es sich um das FTP-Paket (File Transfer Protocol) Version 0.17, Releasenummer 7 für den Intel 80386 Prozessor (und höhere) handelt, das den `ftp`-Client enthält. Der Quellcode ist jeweils in einem eigenen `rpm`-Paket erhältlich; dass es sich dabei um ein Quellcode- und nicht um ein Programmpaket handelt, wird durch ein zusätzliches „`src`“ im Dateinamen ausgedrückt, zB `ftp-0.17-7.i386.src.rpm`; `rpm`-Pakete werden auch als RPMS bezeichnet, Quellcodepakete als SRPMS.

Der RPM Package Manager

- Homepage `http://www.rpm.org`
- entwickelt von RedHat; weit verbreitet; GPL (SuSE, Mandrake, ...)
- (De)Installieren von Programmpaketen
- Upgrade, Refresh von Programmpaketen
- Abfragen und Verifizieren von Programmpaketen
- Kommandozeile und GUI (GnoRPM, `kpackage`)
- Programmpakete als .rpm-Dateien
- `rpm`-Datenbank am System (`/var/lib/rpm/`)
- Dokumentation: Maximum RPM `http://www.rpm.org/max-rpm`

Folie 40

`rpm`-Pakete

- Paketdateiname=
  Paketname+Versionsnr+Releasenr+Architektur.rpm
  zB `diald-0.16.4-1.i386.rpm`
- enthalten
  - Name, Version, Beschreibung, Info
  - Paketabhängigkeits- (Dependency-) Information
  - Das Programmpaket selbst (gezippt)
  - Pre- und Postinstallationsskripts
- herausgegeben von RedHat oder anderen
  (mit rpm kann man selbst Pakete erzeugen)

Folie 41

`rpm` verwenden

- 5 Grundverwendungsarten:
  - Installieren
  - Upgraden und Freshen
  - Deinstallieren
  - Abfragen (query)
  - Verifizieren (verify)
- für Fortgeschrittene: Pakete erzeugen

Folie 42

Drei der grundlegenden Funktionen des `rpm` (*Installation, Upgrade, Freshen*) wurden schon in Teil 1, Abschnitt 9.3 behandelt; die Syntax ist auf Folie 43 angeben. Beim Installieren wird geprüft, ob ein Paket gleichen Namens schon installiert ist und (standardmäßig) die nochmalige Installation verweigert. Will man ein Paket durch eine neuere(n) Version/Release ersetzten, muss Upgrade oder Freshen verwendet werden. Der Unterschied zwischen diesen beiden Modi ist, dass bei letzterem nur eine Installation erfolgt, falls das alte Paket tatsächlich bereits installiert ist. Bei Upgrade erfolgt immer eine Installation des entsprechenden Pakets; falls vorhanden, wird vorher die alte Version entfernt. Bei Upgrade und Freshen werden Konfigurationsfiles des alten Pakets mit der Endung `.rpmsave` gespeichert bzw. neue Konfigurationsfiles mit der Endung `.rpmnew` erzeugt und eine entsprechende Warnung ausgegeben.

Weitere grundlegende Funktionen des `rpm` sind *Deinstallieren, Abfragen und Verifizieren* von Paketen. Die Syntax ist auf den Folien 44 – 46 erläutert.
Beim Deinstallieren eines Pakets wird natürlich geprüft, ob etwaige Abhängigkeiten verletzt und so die Funktion von anderen Programmen beeinträchtigt wird.
Das Abfragen von Paketen dient dazu, Informationen über das Paket und seine Installation zu erhalten; diese sind zB der Vertreiber des Pakets und Adresse seiner Homepage, allgemeine Informationen zum Paket, eine Liste aller im Paket befindlichen Dateien, das Installationsdatum etc. Es können sowohl installierte Pakete, als auch nicht installierte Paketdateien abgefragt werden.
Das Verifizieren eines Pakets erlaubt festzustellen, ob die Dateien des Pakets seit der Installation verändert wurden. Bei Konfigurationsfiles ist es selbstverständlich, dass diese verändert werden. Ist allerdings eine binäre Datei verändert worden, so ist das ein Alarmsignal und oft ein Zeichen für einen Hackereinbruch. Ähnliches gilt für den Fall, dass die `rpm`-Datenbank am System oder `rpm` selbst korrumpiert ist. Im Fall, dass die `rpm` Datenbank nicht mehr verlässlich funktioniert, kann die Verifikation eines installierten Packages auch gegenüber einem (Original-) `rpm`-File erfolgen.

RPMS installieren/upgraden/freshen

Syntax: `rpm -i|U|F[hv] Paketdatei [Optionen]`

- `-i|U|F` install, upgrade, freshen
- Upgrade: vor Installation altes Paket deinstalliert
  Konfigurationsdateien mit Extension:
  `.rpmsave` gesichert – `.rpmnew` erzeugt
- Freshen: detto, aber nur falls altes Paket wirklich installiert war
- Paketdatei: zB `joe-2.8-40.i386.rpm`
- Optionen: `-v(v)` (sehr) verbose
  - `-h` zeigt Verlauf mittels hash marks (#)
  - `--test`, `--nodeps`, `--replacefiles`, `--replacepkgs`, `--oldpackage`, `--force`, `--noscripts`

Folie 43

RPMS deinstallieren

- Syntax: `rpm -e Paketname [Optionen]`
- Paketname: zB `joe`
  ACHTUNG: nicht Paketdatei zB `joe-2.8-40.i386.rpm`
- Optionen:
  - `--allmatches` alle Versionen
  - `--test`
  - `--noscripts`
  - `--nodeps` (gefährlich!)

Folie 44

RPMS abfragen

- Syntax: `rpm -q Paketname [Optionen]`
- Output: Versions- und Releasenummer
- Optionen:
  - `-a` frage alle installierten Pakete ab
  - `-i` zeige Paketinformation an
  - `-l` liste alle Dateien im Paket
  - `-d` liste alle Dokumentationsfiles im Paket
  - `-c` liste alle Konfigurationsfiles im Paket
  - `-f file` zu welchem Paket gehört `file`
  - `-p Paketdatei` für nicht installierte Pakete

Folie 45

Um die Verbreitung von manipulierten `rpm`-Paketen (Trojanischen Pferden) zu verhindern, können Pakete vom Vertreiber mit einem PGP/GPG (Pretty Good Privacy/Gnu Privacy Guard, einem Public-Key-Verschlüsselungsprogramm) signiert werden. Die Authentizität dieser Pakete kann dann mittels `rpm --checksig rpm-file` getestet werden.

Schließlich kann der RPM Package Manager auch dazu verwendet werden, um `rpm`-Pakete zu erzeugen.

Der RPM Package Manager besitzt auch graphische Interfaces namens GnoRPM (für den Gnome Desktop) und `kpackage` (für KDE). Diese können alle vorher besprochenen Funktionen erfüllen und bieten ein (mehr oder weniger) übersichtliches GUI.

Eine Erweiterung des `rpm` das ein einfaches und automatisches Upgraden von Paketen (allerdings nur auf RedHat Systemen) ermöglicht und es so insbesondere erlaubt, alle Packages am System auf dem neuesten Stand zu halten ist der *Red Hat Update Agent* `up2date`. Dieser verfügt sowohl über ein Kommandozeileninterface als auch ein grafisches Interface, erlaubt aber nur das Upgraden von Packages die von RedHat selbst herausgegeben wurden. Für alle Details verweisen wir auf die RedHat Homepage.
Ein anderes Tool, das in etwa die selben Aufgaben übernehmen kann und auf allen `rpm`-basierten Systemen funktioniert ist *autorpm*; für Details siehe `http://www.autorpm.org`.

Die wahrscheinlich größte Schwäche `rpm`-basierter Systeme ist, dass es unter Umständen sehr mühsam ist, händisch ein bestimmtes Paket zu installieren; nämlich wenn eine Vielzahl von Abhängigkeiten nicht erfüllt ist und diese Pakete auch noch weitere unerfüllte Abhängigkeiten nach sich ziehen. Tools, die solche Situationen meistern sind `rpmfind` (`http://www.rpmfind.net`) und `autoget` (aus dem Autoupdate-Paket von Gerald Teschl, `http://www.mat.univie.ac.at/~gerald/ftp/autoupdate`).
Im Debian Package-Management kann die Situation der „Abhängigkeitsschleife“ beim *Installieren* von Paketen vom Updatetool `apt-get` gehandelt werden (siehe unten); es gibt auch eine Version, die das `rpm`-Format unterstützt.

RPMS verifizieren

- Syntax: `rpm -V Paketname [Optionen]`
- Output: 5STUGM
  veränderte MD5-Prüfsumme, Größe, Mtime, Besitzer, Gruppe, Mode (Berechtigung, Typ)
- Optionen:
  - `-f file` verifiziere `file`
  - `-a` verifiziere alle installierten Pakete
  - `-p package-filename.rpm` verifiziere gegenüber `package-filename.rpm` statt Datenbank

Folie 46

`rpm`-Erweiterungen

- Graphische Interfaces
  - GnoRPM (Gnome Desktop)
  - `kpackage` (KDE)
- Automatische Updates
  - `up2date` (RedHat)
  - `autorpm`
  - `autoupdate`
  - `rpmfind`
  - `apt-get` auch für `rpm`-Format

Folie 47

Das *Debian Package-Management* funktioniert ähnlich dem `rpm`; das entsprechende Tool heißt `dpkg`. Wesentliche Unterschiede zu `rpm` sind eine Staffelung der Paketabhängigkeiten auf drei Ebenen (mandatory, recommended, optional), die Endung `deb` der Paketdateien an Stelle von `rpm` und die fehlende Unterstützung für PGP/GPG-Signaturen der Paketdateien. Das `up2date` entsprechende Tool für Debian Linux heißt `apt-get` und zeichnet sich durch seine besonders problemlose und einfache Anwendbarkeit aus.

Schließlich gibt es das Tool *Alien*, das es ermöglicht, Pakete von einem Format in ein anderes und auch zwischen Distributionen zu konvertieren (siehe `http://kitenet.net/programs/alien`).

Das Debian Package-Management

- Homepage `http://www.debian.org`
- `dep`-Format: Package-Managment der Debian GNU/Linux Distribution
- `dpkg`: Äquivalent zu `rpm` Kommandozeilenprogramm für `deb`s
- Standardoperationen wie `rpm` (De/Installieren, Update, ...)
- Konfiguration in `/etc/apt`
- `deb`-Datenbank am System in `/var/lib/apt/`
- Programmpakete als `.deb`-Dateien
- `apt-get`: Automatisches Install/Update von Paketen über Internet
- Umfangreiche Dokumentation und Tools fürs Selbstbauen von Paketen

Folie 48

5 Speichermanagement

In diesem kurzen Kapitel erklären wir die Grundzüge der Speicherverwaltung unter Linux und stellen einige nützliche Tools zum Speicher- und System-Monitoring vor.

5.1 Speicherverwaltung

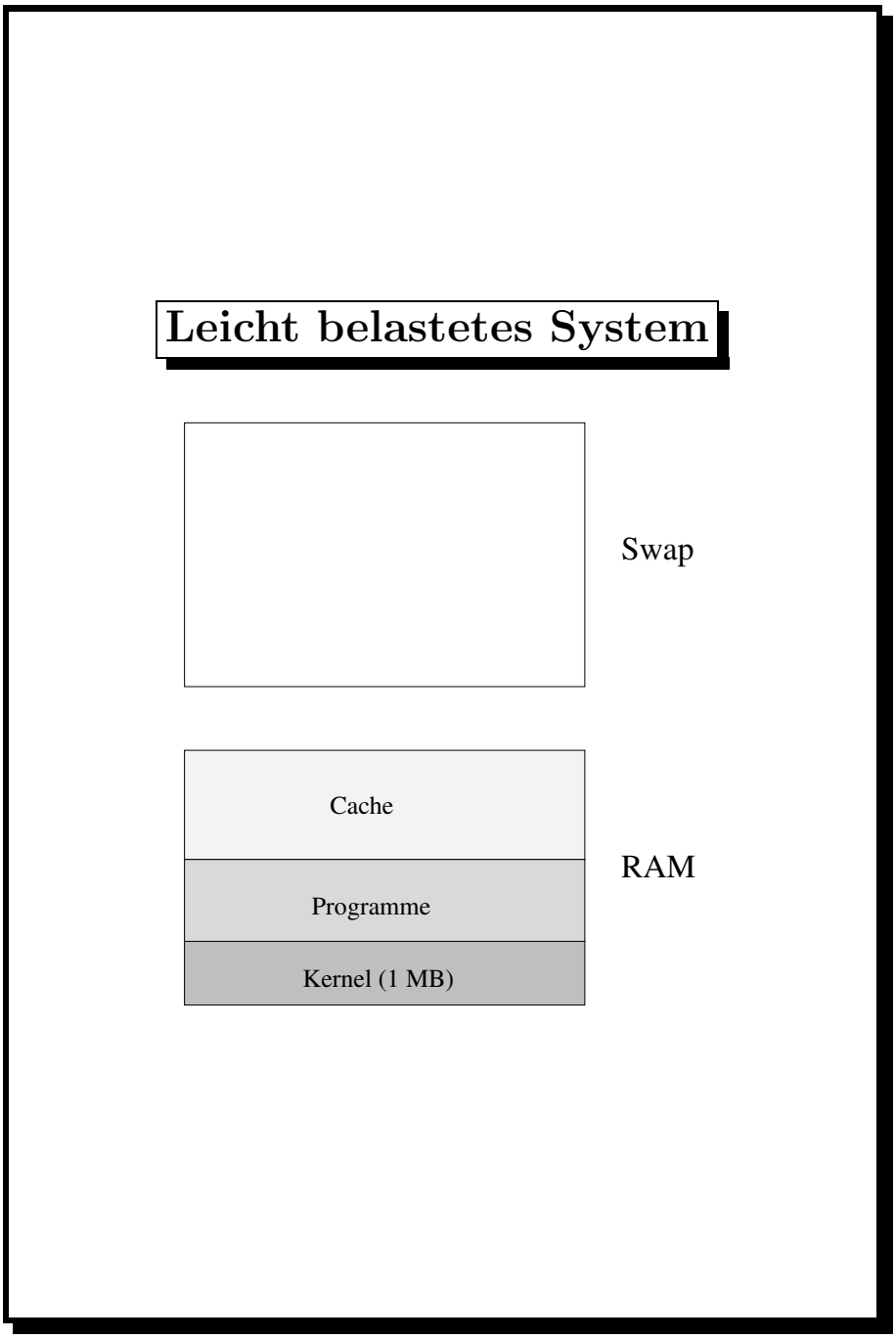
Unter Unix wird neben dem physikalisch vorhandenen Hauptspeicher (RAM) Auslagerungsspeicher (Swap oder Paging Space, meist in Form einer Swap-Partition auf einer der Festplatten des Systems) verwendet. Der so vorhandene gesamte Speicher wird in gleich große Einheiten sogenannte *(Memory) Pages* (1–8 KB, meist 4 KB) unterteilt, die vom Kernel verwaltet werden. Jedem Prozess wird bei seinem Start ein gewisser Adressbereich bestehend aus einer Anzahl von Pages zugewiesen und bleibt für diesen reserviert.
Etwas genauer wird der Hauptspeicher unter Linux wie folgt aufgeteilt: Das erste MB ist für den Kernel reserviert, der Rest des RAM wird auf die laufenden Prozesse und den Disk Cache aufgeteilt. Letzterer dient zum Zwischenspeichern von Daten, die von den Massenspeichergeräten des Systems eingelesen bzw. dorthin geschrieben werden.

Ist das RAM voll belegt (d.h. es gibt keine freien Pages mehr im Hauptspeicher) – wobei immer ein, wenn auch eventuell kleiner Teil, für den Disk Cache reserviert bleibt – beginnt der Kernel Speicherseiten in den Swap-Bereich auszulagern. Dabei wird versucht, zuletzt benutzte Pages im RAM zu behalten und weniger benutzte in den Swapspeicher auszulagern, der nur um Größenordnungen langsamer angesprochen werden kann (sog. LRU System; die least recently used Pages werden ausgelagert (paged out) und VM (Virtual memory) System; für Details und auch den (hier verschwiegenen) Unterschied zwischen Swapping und Paging siehe [1], Kap. 25).

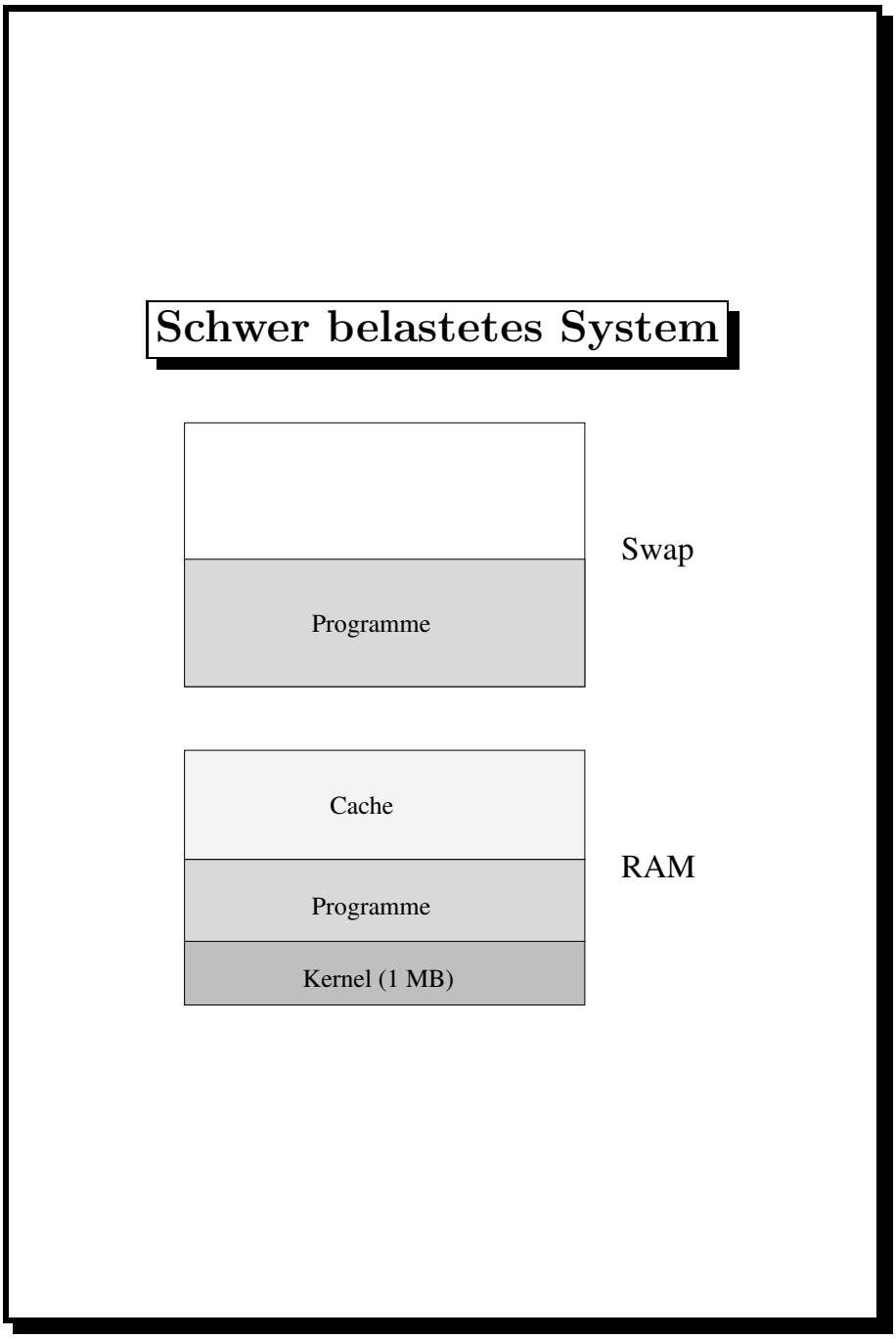
Linux Speichermanagement

- Gesamtspeicher = RAM + Swapspeicher
- unterteilt in (Memory)Pages (4 KB)
- für Prozesse verfügbarer Speicher = RAM + Swap Space − 1 MB
  - 1 MB reserviert für Kernel
  - Rest für Prozesse
- Pages im RAM werden – wenn nötig – auf HD ausgelagert
- Unbenutztes RAM wird als Disk-Cache verwendet
- effektiver Algorithmus, nicht konfigurierbar

Folie 49


Leicht belastetes System
Swap
Cache
Programme
Kernel (1 MB)
RAM

Folie 50


Schwer belastetes System
Swap
Programme
Cache
RAM
Programme
Kernel (1 MB)

Folie 51

Neben dem unangenehmen „Dauerrattern“ der Festplatte(n) verlangsamt das Swapping den Betrieb des Systems wesentlich, besonders dann, wenn sogar Daten von aktiven Prozessen ausgelagert werden müssen. Ist auch der Swapspeicher voll belegt, so beginnt der Kernel Prozesse (nach Zufallsprinzip) zu stoppen, eine Situation, die vermieden werden sollte!

Unter Linux kann *Swap—ßpeicher* in Form von Swap-Partitionen aber auch Swap-Files verwendet werden. Die Methode der Wahl ist die Verwendung von Swap-Partitionen, die wesentlich schneller und effektiver sind. Dazu muss eine eigens dafür vorgesehene Festplattenpartition angelegt werden (meist schon bei der Installation; siehe Teil 1, Kapitel 8). Swap-Partitionen können mit jedem der Tools zu Festplattenpartitionierung, also insbesondere zB mit `fdisk` erstellt werden. Der Partitionstyp muss dabei auf den Hex Code 82 (Linux Swap) gestellt werden. Zum Anlegen des Swapspeichers (analog zum Formatieren einer „gewöhnlichen“ Festplattenpartition) verwendet man das Kommando `mkswap partition`. Aktiviert werden kann der Swapspeicher mit dem Kommandozeilenaufruf `swapon partition`; bei Systemstart wird dies vom Script `/etc/rc.sysinit` erledigt (vgl. Teil 1, Kapitel 9). Im Unterschied zu den meisten Unix-Varianten kann unter Linux Swapspeicher im laufenden Betrieb mittels `swapoff partition` deaktiviert werden; das ist allerdings nur dann möglich, wenn im Hauptspeicher (resp. anderen Swap-Bereichen) genug Platz ist und die Daten dorthin geschrieben werden können.

Die Größe des Swapspeichers richtet sich sinvoller Weise nach der Größe des RAM; der Swapspeicher sollte größenordnungsmäßig dem RAM entsprechen. Sinnvoll ist es daher, im Hinblick auf zukünftiges Hardwareupgrade die Swap-Partition ca. zweimal so groß wie das RAM zu wählen; es ist oft wesentlich einfacher, zusätzliches RAM einzubauen als die Formatierung der Festplatte(n) zu ändern.
Von der Verwendung von Swap-Files ist im Allgemeinen aus Performancegründen abzuraten. Im Notfall (etwa kurzfristig bedingter Speicherengpass) stellt sie aber eine gangbare Alternative dar. Wir erklären die Vorgangsweise zum Anlegen eines Swap-Files auf Folie 53.

Swap-Partition

- Größe
  - minimal 10 Pages (40KB); maximal 2GB
  - maximal 8 Partitionen
- Anlegen
  - Partition erzeugen mit `fdisk` (Typ 82)
  - Swap-Partition mit `mkswap` anlegen (formatieren)
- Management
  - aktivieren mit `swapon /dev/hdxy`
  - deaktivieren mit `swapoff /dev/hdxy`
    - ohne Reboot möglich
    - nur wenn genug Speicher vorhanden

Folie 52

Swap-File

- weniger effizient als Partition (Notfall!)
- Anlegen
  - finde Platz für Swap-File (`df`)
  - erzeuge großes File
    zB `dd if=/dev/zero of=swapfile bs=1024 count=64000`
  - verwandle es in ein swapfile `mkswap swapfile`
  - aktivieren `swapon swapfile`

Folie 53

Gegenwärtig (Februar 2002) kann der Linux Kernel bis zu 64 GB Hauptspeicher verwalten. Dazu muss er aber mit High-Memory-Support compiliert werden (erlaubte Werte der entsprechenden Option sind: OFF, 4 GB und 64 GB, vgl. Kapitel 10). Die Standardkernels der meisten Distributionen (High-Memory-Support OFF) können bis zu 960 MB RAM adressieren.
Darüber hinaus sind bis zu 8 verschiedene Swap-Bereiche (Partitionen und Dateien) möglich. Ihre Größe muss mindestens 10 Pages betragen und ist nach oben mit 2 GB limitiert. Für den aktuellen Stand dieser sich stetig ändernden Werte empfiehlt es sich, immer in der aktuellsten Dokumentation nachzusehen.

5.2 Monitoring

Zum Schluss dieses Kapitels stellen wir einige einfache Tools zur Überwachung der Speicherverwendung bzw. der Systemperformance vor.

Das einfachste Kommando, das eine Statistik über die laufenden Prozesse ausgibt ist der `ps`-Befehl. Etwa gibt `ps aux` eine Statistik aller laufenden Prozesse geordnet nach PID (Prozess Identity; siehe Teil 1, Abschnitt 6.3) aus. Angezeigt werden neben dem Besitzer eines Prozesses vor allem Speicher- und CPU-Nutzung. Die Prozesshierarchie kann mittels `pstree` angezeigt werden. Beide Kommandos haben den Nachteil, dass sie nur einen Snapshot zu einem bestimmten Zeitpunkt wiedergeben. Abhilfe schafft das `top`-Kommando, das eine periodisch aktualisierte Statistik der laufenden Prozesse anzeigt. Außerdem können interaktiv Signale an die Prozesse gesendet werden.

Die Speicherstatistik kann mittels `free` abgefragt werden; Belegung von Haupt- und Swapspeicher, sowie Größe des Festplattencaches werden angezeigt. Um detaillierte Information über die Statistik des Virtual Memory System auszugeben, kann der Befehl `vmstat` verwendet werden.

Das `uptime`-Kommando gibt die *Uptime* des Systems aus, also die Zeit, die seit dem letzten Systemstart vergangen ist. `uptime` zeigt aber auch die *Systemlast* an. Die Systemlast ist die Zahl der „lauffähigen“ Prozesse, also der Prozesse, die auf CPU Zeit oder I/O Operationen warten. `uptime` gibt 3 Werte gemittelt über die letzten 1, 5 und 15 Minuten aus. Steigt die Systemlast signifikant über 1 an, so ist das System schwer belastet und für den Benutzer entsteht der Eindruck, das System arbeite langsam.

Schließlich geben die graphischen Tools `xosview, xload` und `xsysinfo` einen bunten Überblick über die Systemnutzung; weitere Tools sind in die Desktopsysteme KDE und GNOME integriert.

Monitoring-Tools

- `top` zeigt Speicher-, CPU- und Prozessstatistik an
- `ps, pstree` zeigt Prozesse/Prozesshierarchie an
- `free` zeigt Speicherstatistik an
- `vmstat` detaillierte VM Statistik
- `uptime` zeigt Uptime und Systemlast

fürs GUI

- `xosview` zeigt Systemüberblick an
- `xload` zeigt Systemlast
- `xsysinfo` zeigt Systeminformationen an
- ...

Folie 54

6 Scheduling

Eine wichtige Aufgabe des Systemadministrators ist es Routineaufgaben verlässlich durchzuführen; Automatisierung ist hier die Methode der Wahl. In diesem Kapitel beschreiben wir wie sich (periodische) Routineaufgaben mittels Scheduling automatisieren lassen und wie Programme ausgeführt werden können, ohne dass der Benutzer interaktiv am System angemeldet ist.

Durch den konsequenten Einsatz von Automatisierungen lassen sich viele Aufgaben am System – unter der Voraussetzung der richtigen Implementierung – sehr effizient und bequem durchführen. Eine mögliche Form ist das Schreiben von Scripts (zB zum Hinzufügen von Benutzern, vgl. Kapitel 3), eine andere, der wir uns hier zuwenden wollen, ist das sogenannte *Scheduling*. Es kann im Wesentlichen in zwei Gruppen eingeteilt werden:

- regelmäßig wiederholtes Starten bzw.
- einmaliges Ausführen von Programmen (Scripts) (möglicherweise unter speziellen Randbedingungen).

Für den ersten Punkt steht unter Unix das Programm `cron` zur Verfügung, für den zweiten allgemein `at`, im speziellen `batch`, das Jobs ausführt, sobald die Systemlast (siehe Abschnitt 5.2 unter einen gewissen Wert (default 0.8) gefallen ist.
Alle drei Varianten werden durch Daemonen gesteuert (`crond` bzw. `atd` für `at` und `batch`), die jede Minute überprüfen, ob ein „Auftrag“ erledigt werden muss. Das Einsenden solcher Jobs an die Warteschlange (ähnliches Konzept wie beim Drucken, siehe Kapitel 7) geschieht bei `at` und `batch` direkt über die Kommandozeile. `cron` bezieht die Informationen aus einer Crontab genannten Datei. Es gibt einerseits die globale Crontab des Systems `/etc/crontab`, andererseits kann jeder Benutzer eine persönliche Crontab in `/var/spool/cron/$USER` anlegen.

Scheduling

- Automatisiertes Ausführen von Programmen
  (ohne eingeloggt zu sein)
- Standardoutput in Mail umgeleitet
  - periodische Starten von Prozessen, Routinetasks zB
    (Backup, `/tpm`-löschen, `logrotate`, ...)
  - Prozesse ein Mal zu einem spezifischen Zeitpunkt in der Zukunft starten
  - Prozesse starten, wenn Systemlast niedrig (genug)

Folie 55

Scheduling aber wie?

- periodische Aufgaben
  - in Crontab-Dateien festgelegt
    `/etc/crontab` fürs System,
    `/var/spool/cron/$USER` für Benutzer
  - `crond` liest diese und startet Prozesse
  - `anacron`: auch wenn Rechner nicht durchgehend in Betrieb
- einmalige Aufgaben
  - zu einem bestimmten Zeitpunkt: `at`-Kommandozeileninterface
  - wenn Systemlast gering: `batch`-Kommandozeileninterface
  - beide vom `atd` gemanagt

Folie 56

Verschiedene Distributionen (darunter RedHat) verwenden außerdem folgenden Standard: Der Administrator kann Programme und Scripts, die vom System stündlich, täglich, wöchentlich oder monatlich ausgeführt werden sollen in die Verzeichnisse `/etc/cron.{hourly,daily,weekly,monthly}` legen. Programme in diesen Verzeichnissen werden nach dem in `/etc/crontab` festgelegten Schema einmal pro Stunde, Tag, Woche bzw. Monat ausgeführt.

Weiters gibt es unter Linux noch das Service `anacron`, das ähnlich wie `cron` funktioniert, aber dafür sorgt, dass Jobs auch dann ausgeführt werden, wenn der Rechner nicht durchgehend in Betrieb ist. `anacron` überprüft nach seiner Aktivierung (meist beim Booten), ob gemäß seiner Konfiguration (in `/etc/anacrontab`) unerledigte Jobs vorhanden sind und startet diese (jeweils mit einer kleinen Zeitverschiebung, sodass nicht alle Jobs aufeinmal ausgeführt werden und die Systemlast plötzlich aus für den Benutzer nicht ersichtlichen Gründen ansteigt). Ist der letzte Job erledigt wird `anacron` beendet. Außerdem wird `anacron` (meist) einmal täglich von `cron` selbst aufgerufen, um die Checks vorzunehmen.

Wir wollen uns nun anhand eines einfachen Beispiels aus der täglichen Praxis die grundlegende Funktionsweise von `cron` ansehen.

Systemcrontab (1)

- in `/etc/crontab` festgelegt
- führt standardmäßig Scripts in den folgenden Verzeichnissen aus
  - `/etc/cron.hourly`
  - `/etc/cron.daily`
  - `/etc/cron.weekly`
  - `/etc/cron.monthly`

Folie 57

6.1 Ein „Real World“ Beispiel

In einem großen (meistens auch räumlich getrennten) Netzwerksystem kann man nicht jedes Problem eines Benutzers vor Ort lösen. In einer solchen Situation nützt man die Stärke eines textbasierten Systems, wie es nun einmal Unix/Linux ist, aus, um mittels eines geeigneten Mechanismus Rechner aus der Ferne zu administrieren. War hier bis vor einigen Jahren hauptsächlich `telnet` in Verwendung, setzt man heute wegen ihrer höheren Sicherheit auf die *Secure Shell* (`ssh`). Um sich so zu einem anderen Rechner zu verbinden, muss dort der `sshd` Daemon laufen, der einkommende Verbindungsanfragen übernimmt (siehe Kapitel 15). Sollte dieser Daemon aber einmal aus irgendeinem Grund (zB Absturz) nicht laufen, bleiben einem im Wesentlichen nur zwei Alternativen: Selbst zum betroffenen Rechner gehen und manuell den `sshd` neu starten bzw. hoffen, dass jemand vor dem Rechner sitzt und ihn bitten, die Maschine zu rebooten (den `sshd` darf nur root starten). Beide Möglichkeiten kommen nicht immer in Frage (zB Rechner ist weit entfernt bzw. ein Server, den man nicht einfach so rebooten darf). Es wäre in der Tat schön, gäbe es einen Mechanismus, der zweimal täglich nachsieht, ob der `sshd` ordnungsgemäß läuft, ihn gegebenenfalls neu startet und dann eine Mail über den Status des Daemonen an den Administrator schickt.
Ein Script, das diese Aufgabe erledigt (und das man unbedingt zuerst einmal manuell genügend testen sollte), lautet (siehe auch Kapitel 11):

```
#!/bin/sh
#/sbin/check-sshd (restarts sshd if down)
if [ -z "‘pidof sshd‘" ]; then
 /etc/init.d/sshd restart >/dev/null 2>&1
 echo "sshd restarted" | mail -s "‘hostname --fqdn‘"\
                         root@logserver
fi
```

Kurze Erklärung: `pidof sshd` ermittelt die `PID`, unter der der `sshd` läuft; wenn aber kein `sshd` läuft, muss das Ergebnis leer (`-z`) sein, was hier ausgenutzt wird. Sodann wird der Daemon neu gestartet und letztendlich noch eine Mail an root auf einem zentralen Logserver gesendet.

Nun wollen wir erreichen, dass dieses Script zweimal täglich, nämlich um 11.30 und 23.30 Uhr, ausgeführt wird. Dazu rufen wir die globale `cron` Konfigurationsdatei `/etc/crontab` in einem Editor auf oder (besser!) editieren sie mit dem Kommandoaufruf `crontab -e` (siehe auch Abschnitt 6.2) und fügen am Ende die Zeilen

```
30 11 * * * /sbin/check-sshd
30 23 * * * /sbin/check-sshd
```

ein. Das (etwas sperrige) Format der `crontab` Dateien verdient nähere Aufmerksamkeit; die Bedeutung der sechs leerzeichengetrennten Felder (siehe auch Folie 58) ist der Reihe nach: Minute (0-59), Stunde (0-23), Tag des Monats (1-31 oder Name), Monat (1-12), Tag der Woche (0-7 oder Name), auszuführendes Programm.
Mit einem `*` – wie in unserem Fall – zeigt frau an, dass das jeweilige Feld alle Werte annehmen kann. Weitere Möglichkeiten des Feldformats sind

- Explizite Auflistung (zB `5,9,13` für die Stunde)
- Intervalle (`5-9`)
- Kombinationen daraus (`4,7-19,23`)
- eigene Schrittweiten mittels `wert/schritt` (`5-10/2` in Spalte 2 bedeutet von 5 bis 10 Uhr in 2 Stunden-Abständen)
- Kombination von `*` und Schrittweiten (`*/2` jede 2. Stunde)
- Kombinationen aus allen obigen

Mit diesem Wissen können wir unser Beispiel auf eine einzige Zeile reduzieren:
`30 11,23 * * * /sbin/check-sshd`
oder auch
`30 */12 * * * /sbin/check-sshd`.

Crontab-Syntax

Min	Std	T/M	Mo	T/W	
1	*	*	*	*	jede Stunde
/5	*	*	*	*	alle 5 Minuten
0,30	4	*	*	*	zweimal am Tag
22	17	1	*	*	einmal im Monat
17	17	*	*	1	einmal pro Woche
15,45	9-18	*	*	*	24 mal am Tag
11	11	11	11	*	einmal im Jahr
59	23	1	1	6	jeden Samstag und jeden 1.1.

Folie 58

Systemcrontab (2)

```
# less /etc/crontab
  SHELL=/bin/bash
  PATH=/sbin:/bin:/usr/sbin:/usr/bin
  MAILTO=root
  HOME=/
  # run-parts
  01 * * * * root run-parts /etc/cron.hourly
  02 4 * * * root run-parts /etc/cron.daily
  22 4 * * 0 root run-parts /etc/cron.weekly
  42 4 1 * * root run-parts /etc/cron.monthly
```

Folie 59

6.2 Benutzung von `cron`, `at` und `batch`

Wie oben erwähnt, kann jeder User ein eigenes Crontab File anlegen und so Cronjobs ausführen lassen. Zum Bearbeiten von Crontab Dateien verwendet man am besten das Werkzeug `crontab`, das dem direkten Editieren der Datei in `/var/spool/cron` vorzuziehen ist. `crontab` kennt mehrere Optionen, darunter `-e` zum Editieren der `cron` Tabelle des Benutzers, die – sobald gespeichert – auch aktiviert wird. Mit dem Parameter `-l` kann man sich die `cron` Einträge ansehen (listen) und mit `-r` die gesamte Datei löschen (remove). KDE bietet auch einen grafischen „Taskmanager“ names `kcron` zur Verwaltung von Cronjobs an.

`at` und `batch` werden über ein Spool System verwaltet. Dadurch ergeben sich ähnliche Bedienungsmuster wie beim Drucken (vgl. Kapitel 7). Man kann zB durch `at -f script zeit` ein Script zu einer bestimmten Zeit ausführen. Es sind mehrere Zeitformat erlaubt zB HH:MM (2-stellige Stundenangabe: 2-stellige Minutenangabe), DD.MM.YY (2-stellig Tag.Monat.Jahr) und das handliche `now + n min` oder `4pm + 3 days`. Auf jeden Fall lohnt sich ein Blick in die Manpage.
Das `batch` Kommando funktioniert völlig analog. Die `at` Queue kann man mittels `atq` bzw. `at -l` ansehen. Aufträge löscht man mit `atrm`.

Den Output eines `cron-, at-` oder `batch-` Jobs erhält der entsprechende Auftraggeber, sofern (etwa wie im Script oben) nichts anderes angegeben ist als Mail an den lokalen Account; d.h. der Standardoutput eines solchen Jobs wird in eine Mail umgeleitet.
Oft ist es wünschenswert, bestimmten Usern zu verbieten, eigene `cron` oder `at` Einträge bzw. Jobs anzulegen. Dazu existieren die Dateien `/etc/{cron|at}.{allow|deny}`, wobei jeweils die `allow` Datei stärker ist, d.h. was in den `allow` Dateien steht, hat höhere Priorität. Ein Eintrag in diesen Dateien besteht einfach aus einem Usernamen. Standardmäßig ist es den privilegierten Pseudobenutzern (zB `bin`, `daemon`, ...) aus vernünftigen Gründen verboten, `at` Jobs auszuführen, allen anderen Benutzern erlaubt.

`crontab` bearbeiten

- crontab listen

  ```
  crontab -l
  ```

- crontab editieren

  ```
  crontab -e
  ```

  default=`vi`, mittels Umgebungsvariable EDITOR konfigurierbar

- crontab löschen

  ```
  crontab -r
  ```

- GUI: `kcron` (Taskmanager des KDE)

Folie 60

at-Jobs

- Jobs in Queue eintragen:

```
# at 4am (now+xx min)
  who
  CTRL d
# at -f myatjob 16:00 + 2 days
```

  Output als Mail an Auftraggeber

- Queue listen: `atq, at -l`
- Job löschen: `atrm jobnr, at -d jobnr`

Folie 61

batch-Jobs

- Kommando ausgeführt wenn Systemlast gering (default < 0.8)
- Jobs in Queue eintagen: analog `at`

```
$ batch
at> echo System wenig ausgelastet
<CTRL d>
```

- Output als Mail an Auftraggeber
- Jobs listen/löschen: `atq, atrm`
- Zugangsbeschränkung zum `atd`
  - `/etc/at.allow` (stärker)
  - `/etc/at.deny`

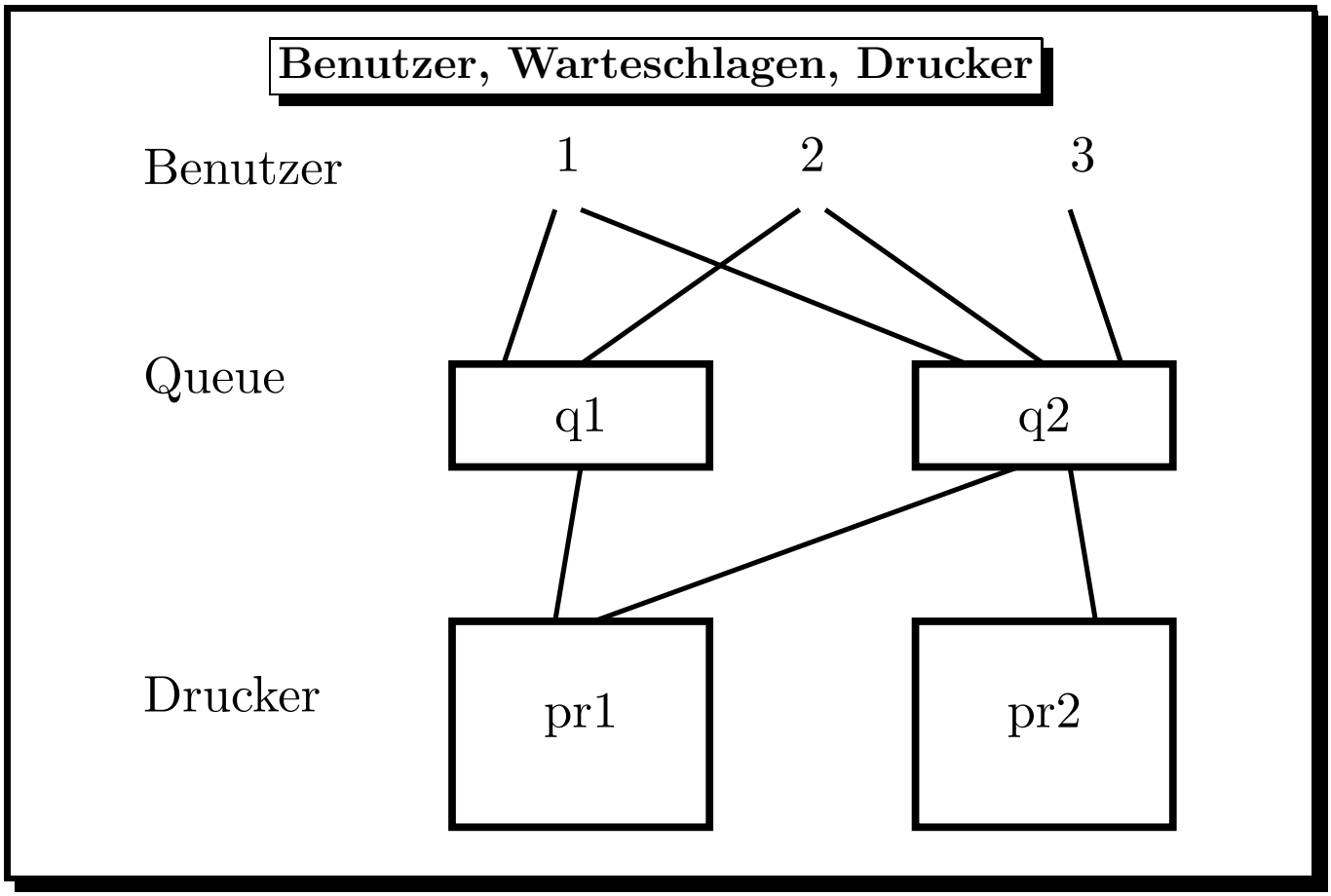
Folie 62

7 Drucker

In diesem Kapitel erklären wir die Grundlagen des Druckerwarteschlangenkonzepts (Queuings) unter Unix/Linux und besprechen den unter Linux häufig verwendeten BSD Druckerdaemon `lpd` sowie seine Erweiterungen LPRng und Cups. Außerdem erklären wir das praktische Vorgehen beim Installieren sowohl eines lokalen wie auch eines Netzwerkdruckers und das Verwalten von Druckerwarteschlangen.

Oft kann es zweckmäßig sein, die eigenen Kreationen, Dokumentation oder Konfigurationsdateien nicht nur am Bildschirm zu betrachten, sondern als „Hardcopy“ in Händen zu halten, also auszudrucken. In Multiusersystem wie Unix sind folgende Überlegungen relevant: Was geschieht, wenn zwei User gleichzeitig eine Datei an den Drucker senden. Welcher der beiden sollte im System die höhere Priorität haben? Wie kann man sicherstellen, dass die Ausdrucke der beiden nicht durcheinander geraten?
Überlegungen wie diese führen zum Konzept des *Druckservers*. Man schickt die zu druckende Datei nicht direkt an den Drucker, sondern an einen Druckerdaemonen (*Spooler*), der in einer FIFO- (First In First Out) Warteschlange (engl.: Queue) die Druckaufträge verwaltet und (je nach Priorität) alle Aufträge abarbeitet, bis die Queue leer ist. Unter Linux kommt hier meistens ein Daemon namens *lpd* (Line Printer Daemon) bzw. ein Paket namens `lpr` (Line Printer) zum Einsatz, der ursprünglich für *BSD* Unix entwickelt wurde. Meist wird der vom Server `lpd` definierte Standard als LPD Protokoll bezeichnet.

Trotz des Namens unterstützt LPD beinahe alle Arten von Druckern, die sich einerseits durch die Art, wie sie drucken (Tintenstrahler, Laserdrucker, ...), andererseits durch die Art des Anschlusses an den Rechner (parallele Schnittstelle, Netzwerkdrucker) charakterisieren lassen. Einzig von sogenannten *GDI*-Druckern (Graphical Device Interface) sollte man unter Unix/Linux die Finger lassen. Verschiedene Hersteller wollten diese Drucker möglichst kostengünstig produzieren und haben daher beinahe die gesamte Steuerung des Druckvorganges dem PC überlassen; daher benötigt man unbedingt die Hersteller-Treiber, die aber – bis auf wenige Ausnahmen – nur für Windows-kompatible Betriebssysteme existieren.


Benutzer, Warteschlagen, Drucker
Benutzer
1
2
3
Queue
q1
q2
Drucker
pr1
pr2

Folie 63

Auf neueren Linux-Systemen (zB RedHat ab 7.0) wird anstatt des BSD-Systems der LPRng (lpr Next Generation) verwendet, der eine Erweiterung von LPD darstellt und (fast) 100%ig rückwärtskompatibel ist.

Ein anderes modernes Drucksystem ist *Cups* (Common Unix Printing System), das das Internet Printing Protocol (IPP) benutzt und (eingeschränkte) Kompatibilität mit sowohl lpd als auch SMB und AppSocket (a.k.a. JetDirect) besitzt. Es besteht eine hohe Wahrscheinlichkeit, dass IPP respektive Cups in Zukunft die Standards für Drucken in Netzwerksystemen resp. auf Unix-Systemen setzen werden.

Drucksysteme unter Linux

- BSD Drucker-Spooling-Mechanismus (`lpr`)
  - gut dokumentiert und verstanden
  - wenig flexibel
- LPRng (lpr Next Generation)
  - Erweiterung des BSD-Systems
  - rückwärtskompatibel
  - `http://www.lprng.com`
- Cups (Common Unix Printing System)
  - verwendet Internet Printing Protocol (IPP)
  - einfach konfigurierbar/administrierbar
  - `http://www.cups.org`

Folie 64

7.1 Das Drucksystem

Das *BSD* Drucksystem ist – wie viele andere Unix-Dienste – ein Client/Server System. Wenn der Server (beim Bootvorgang) gestartet wird, liest er seine Konfigurationsdatei `/etc/printcap` ein, in der alle dem System zur Verfügung stehenden Drucker mit ihren jeweiligen Optionen (lokal, remote, ...) aufgelistet sind.

Wie sieht nun ein typischer Linux-Druckvorgang aus? Als erstes teilt man dem Client-Prozess *lpr* die zu druckende Datei mit; das geschieht entweder direkt an der Kommandozeile oder aus einer beliebigen Anwendung heraus. `lpr` überprüft nun, in welche Queue er den Auftrag stellen soll. Dazu sieht er zuerst nach, ob diese explizit dem *lpr* Kommando übergeben wurde (Syntax: `lpr -P Queue file`). Wenn dem nicht so ist, wertet er die Umgebungsvariable `PRINTER` aus. Falls diese nicht gesetzt sein sollte, nimmt er den Drucker aus `/etc/printcap`, der `lp` heißt bzw. wenn dieser auch nicht vorhanden sein sollte, den ersten Druckereintrag in dieser Datei.
Hat der Client auf diesem Weg nun die Queue bestimmt, sieht er in der `/etc/printcap` nach, welches *Spool*-Verzeichnis der Queue zugeordnet ist. Dieses Directory befindet ist gewöhnlich `/var/spool/lpd/`. Er legt dort jeweils eine Kontrolldatei mit Informationen über den Druckjob und die eigentlich zu druckende Datei ab und benachrichtigt den Druckerdaemon davon. Dieser entscheidet durch den Druckereintrag in der Konfigurationsdatei die weitere Vorgehensweise. Wenn der Drucker remote (über Netzwerk) angesteuert wird, dann leitet er den Auftrag an den entfernten Druckserver weiter. Wenn lokal (also zB über die parallele Schnittstelle) gedruckt wird, dann übergibt er den Auftrag, wenn er in der Queue an der Reihe ist, direkt oder über einen Druckfilter, der die Daten möglicherweise in ein anderes Dateiformat (zB Postscript) konvertieren muss, an die Druckerhardware. Die in `/var/spool/lpd/` temporär angelegten Dateien werden nach dem erfolgtem Ausdruck wieder gelöscht.

Drucksystem: Grundlagen

- Client/Server System `lpr/lpd(cupsd)`
- Benutzer erzeugt Druckauftrag mittels `lpr`
- `lpr`-Syntax: `lpr [-P Queue] file`
  Kommandozeile oder aus Anwendung
- `lpr` sendet Druckauftrag an Warteschlange `/var/spool/lpd/`
- `lpd` wartet auf Benachrichtigung von `lpr`
- `lpd` arbeitet Druckaufträge in der Reihenfolge des Eintreffens gemäß seiner Konfiguration über die Drucker ab

Folie 65

Drucksystem: Weitere Funktionen

- mehrere Warteschlangen für einen Drucker
- mehrere Drucker von einer Warteschlange bedient
  (nur LPRng, Cups)
- Verschieben von Aufträgen zwischen Warteschlangen
- Prioritätensetzung für Druckaufträge
- Authentifizierung
- Accounting

Folie 66

Der *LPRng Druckserver* – der ebenfalls schlicht und einfach `lpd` heißt – berücksichtigt neben `/etc/printcap` auch die Konfigurationsdatei `/etc/lpd.conf`, die eine wesentlich umfangreichere und feinere Konfigurationsabstimmung erlaubt; siehe dazu die entsprechende Manpage. Eine wesentliche Neuerung von LPRng ist, dass der Druckserver in der Lage ist, Warteschlangen zu verwalten, die mehr als einen Drucker bedienen und dass das Verschieben von Druckaufträgen zwischen verschiedenen Queues wesentlich erleichtert wurde. Die Clientkommandos des LPRng sind gegenüber den BSD-Kommandos nur minimal verändert.

Der *Cups Druckserver* heißt `cupsd` und wird grundlegend anders konfiguriert als der BSD-Server und seine Abkömmlinge. Die Konfigurationsfiles befinden sich im Verzeichnis `/etc/cups/` und die primäre Konfigurationsdatei heißt `/etc/cups/cupsd.conf`; ihre Syntax ist der Syntax der Konfigurationsdateien des Webservers Apache (siehe Kapitel 17) verwandt. Cups speichert alle verfügbaren Queues in `/etc/cups/printers.conf` und in `/etc/cups/ppd/` wird die Konfiguration für jede Queue in einem eigenen File namens `queue.ppd` verwaltet. Cups verfügt über ein Webinterface zur Druckerverwaltung, das auf Port 631 (also unter `http://myhost:631`) erreichbar ist, per Default aber nur vom lokalen Rechner aus angesprochen werden kann. Cups emuliert die BSD Clientkommandos; sie können also ganz wie unter BSD verwendet werden (und nicht nur mit dem Cupsserver zusammenarbeiten, sondern auch mit einem LPR(ng) Server, der auf einem entfernten Rechner in einem Netzwerk läuft; siehe unten). Darüber hinaus bietet Cups auch weitere unter BSD nicht verfügbare Clientkommandos; nähere Information dazu findet sich unter `http://myhost:631/documentation.html`.

Schließlich bietet KDE ein eigenes Drucker Clientkommando namens `kprinter`, das sich im Wesentlichen genau wie `lpr` verhält.

Benutzerkommandos (BSD, LPRng, Cups)

- Drucken
  - `lpr [-P queue] filename`
  - `pr filename | lpr [-P queue]` (formatiert Textdateien)
- Warteschlange listen `lpq [-P queue]`
- Druckaufträge löschen `lprm [-P queue] jobnr`
- Standarddrucker festlegen `export PRINTER=queue`

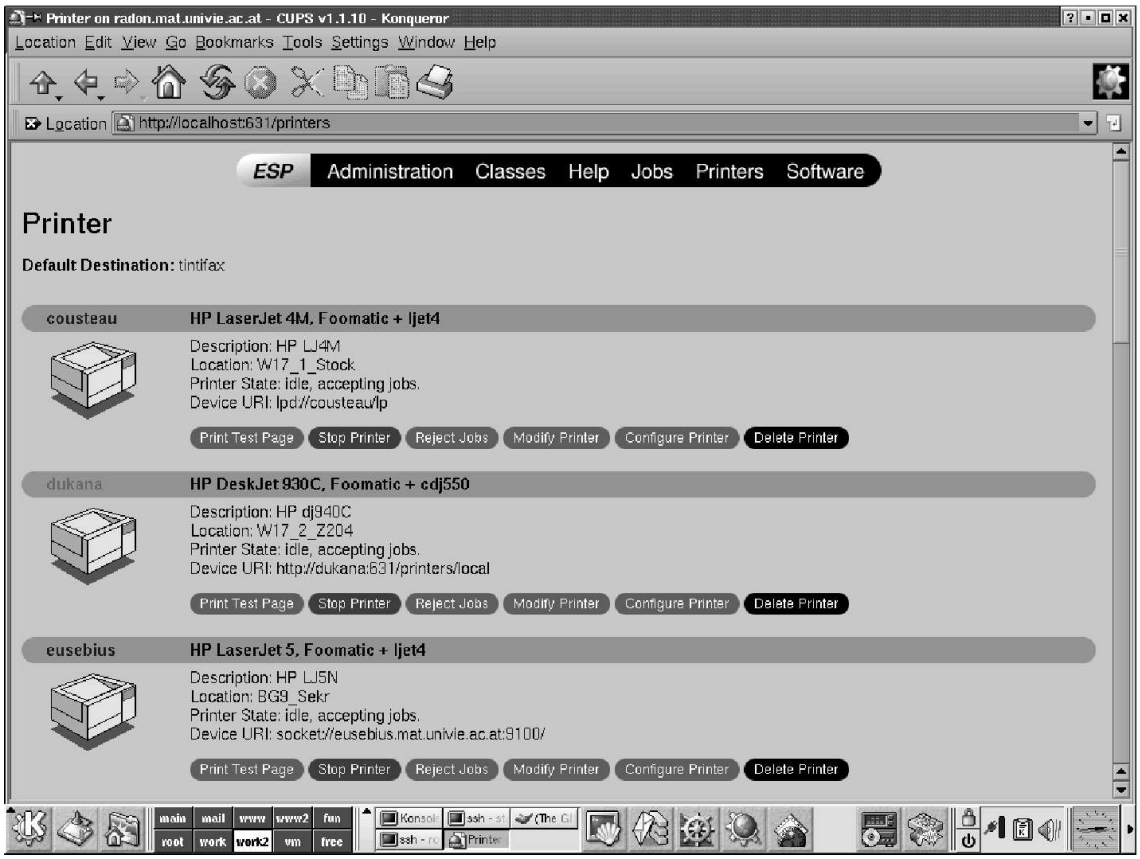
Folie 67

Die Printcap-Datei (BSD, LPRng)

```
# /etc/printcap
#
# Please don't edit this file directly unless you
# know what you are doing!
#
lp|local|Local Printer:\
       :sd=/var/spool/lpd/local:\
        :mx\#0:\
        :sh:\
        :lp=/dev/lp0$:\
        :if=/var/spool/lpd/local/filter:
#
lp|zra|W17 2.Stock|HP 2000 TN:\
        :sd=/var/spool/lpd/zra:\
        :mx\#0:\
        :sh:\
        :rm=zra.mat.univie.ac.at:\
        :rp=lp:\
        :if=/var/spool/lpd/zra/filter:
```

Folie 68

Cups Web-Interface



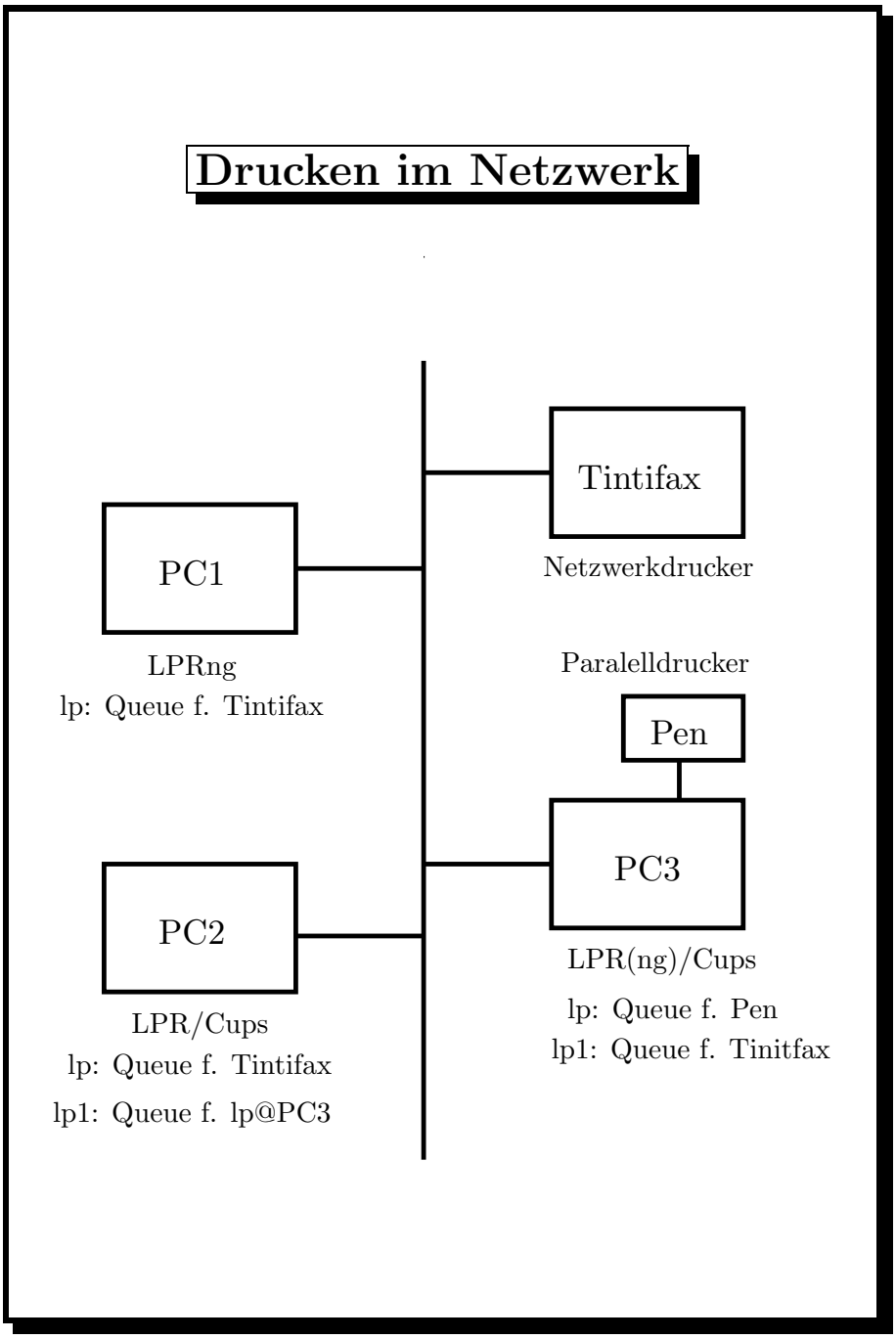
Der BSD Druckserver `lpd` kann eine lokale Druckerqueue nicht nur lokalen Benutzern zur Verfügung stellen, sondern auch den Zugang über ein *Netzwerk* ermöglichen. Dieser wird über die Dateien `/etc/hosts.lpd` oder `/etc/hosts.equiv` (regelt auch den Zugang zu den „alten“ Unix-„r-Diensten“, wie `rlogin` und `rcp`; siehe Kapitel 15) und einen Eintrag in `/etc/printcap` geregelt. In einem der beiden ersteren Files werden die Hostnamen der berechtigten Druckerclients angegeben. Wird zusätzlich in der Printcapdatei eine Queue mit der Option `rs` versehen, so wird die Verwendung ebendieser nur Benutzern erlaubt, die auch auf dem lokalen System einen Account besitzen. Unter LPR muss aber immer am lokalen Rechner eine Queue eingerichtet werden, die den entfernten Druckserver bedient (siehe auch Folie 70).
Die Netzwerk-(und auch die lokalen Berechtigungen) für den LPRng `lpd` können sehr fein abgestuft in der Datei `/etc/hosts.perms` vergeben werden; siehe auch dazu die entsprechende Manpage. Wesentliche Neuerung des LPRng `lpr`-Clientkommandos ist, dass damit auch Remotequeues bedient werden können, d.h. mit der Syntax `lpr -P Queue1@serverhost file` kann ein Druckauftrag an die Queue1 an den `lpd` auf dem Host mit Namen serverhost (das Vorhandensein der entsprechenden Berechtigung vorausgesetzt) gesendet werden. Das hat zur Folge, dass der LPRng Druckerdaemon nur auf Systemen laufen muss, auf denen das tatsächliche Queuing oder Drucken stattfindet und die unter BSD nötige Einrichtung einer lokalen Queue entfällt.
Der Netzwerkzugang zum Cups Druckerserver wird in `/etc/cups/cupsd.conf` (über eine Allow-Directive) geregelt.

Netzwerkdrucker – also Drucker die nicht an der parallele oder sonstige Schnittstelle eines PCs angeschlossen sind, sondern über eine Netzwerkkarte (etwa: JetDirect) oder einen eigenen (Hardware)Printserver (etwa Print Mate) verfügen – unterstützen in (fast) allen Fällen LPR und können vom einem Clientrechner wie ein LPD Druckserver angesprochen werden. Es sollte aber auf jeden Fall eine lokale Queue für den Netzwerkdrucker eingerichtet werden.

Netzwerkdruckserver

- BSD `lpd`
  Clients müssen am Server explizit angegeben werden
  `/etc/hosts.lpd` (nur Druckerclients)
  `/etc/hosts.equiv` (lpd, rlogind, ect...)
  `rs` Option im Printcapfile
- LPRng `lpd` feinere Abstimmung
  `/etc/lpd.perms`
  `/etc/lpd.conf`
- LPRng `lpr` Client kann Remotequeues bedienen
  `lpr -P Queue1@host file`
  LPRng `lpd` daher nur nötig auf Hosts wo Drucker oder Queue
- `cupsd` Netzwerkzugang in `/etc/cups/cupsd.conf` geregelt

Folie 69


Drucken im Netzwerk
Tintifax
Netzwerkdrucker
PC1
LPRng
lp: Queue f. Tintifax
Paralelldrucker
Pen
PC3
LPR(ng)/Cups
lp: Queue f. Pen
lp1: Queue f. Tinitfax
PC2
LPR/Cups
lp: Queue f. Tintifax
lp1: Queue f. lp@PC3

Folie 70

7.2 Druckeradministration

Steht ein Druckjob einmal in der Queue, die man mittels `lpq` ansieht, kann er einerseits mittels `lprm Jobnr` auch wieder daraus entfernt werden (wenn der Druckauftrag dem User gehört, der diesen Befehl ausführt; root darf durch Eingabe von `lprm -` alle Jobs löschen). Die Jobnummer erfährt man durch das `lpq`-Kommando, das den Inhalt der Druckerqueue, also die anstehenden Druckjobs ausgibt. Siehe dazu auch Folie 71. Bei jedem dieser Befehle muss die zu bearbeitende Queue (falls es nicht die Defaultwarteschlange ist) explizit mittels der Option `-P queue` angegeben werden.

Die Kommandos `lpq` und `lprm` stehen sowohl unter LPD, LPRng als auch Cups zur Verfügung. Ebenso das generelle Kommandozeilen Administrationstool `lpc` (Line Printer Control Programm), mit dessen Hilfe der Administrator Druckaufträgen eine höhere Priorität als anderen einräumen, die Reihenfolge der Aufträge in der Queue verändern oder den Status (Drucken, Queuing en/disablen) einer Warteschlange verändern und abfragen kann. Für die `lpc` Syntax siehe Folie 73; das Kommando bietet auch einen interaktiven Bedienmodus an. Das Cups `lpc` bietet nur eine sehr eingeschränkte Funktionalität; *das* Kommandozeilentool unter Cups zum Konfigurieren und Administrieren von Druckerwarteschlangen ist `lpadmin`, die Queues können aber selbstverständlich auch über das Webinterface verwaltet werden. Für Details siehe die entsprechenden Manpages und die Cups Online Dokumentation unter `http://localhost:631/documentation.html`.

Darüber hinaus gibt es grafische Programme zur Verwaltung der Warteschlangen (aller Server) wie etwa `klpq` im KDE.

Druckerwarteschlange manipulieren

```
[roli@pablo tmp]$ lpq
ken is ready and printing
Rank    Owner   Job     File(s)                         Total Size
active  roli    35      blue_angle_swirl.jpg            12288 bytes
1st     roli    36      bluegreencrisscross.jpg         19456 bytes
2nd     roland  65      yp.conf                          1024 bytes
3rd     root    66      cupsd.conf                      18432 bytes
[roli@pablo tmp]$ lprm 36
[roli@pablo tmp]$ lpq
ken is ready and printing
Rank    Owner   Job     File(s)                         Total Size
active  roli    35      blue_angle_swirl.jpg            12288 bytes
1st     roland  65      yp.conf                          1024 bytes
2nd     root    66      cupsd.conf                      18432 bytes
```

Folie 71

Druckermanagement

- mit dem `lpc` Kommando
  - Druckerstatus anzeigen
  - Drucker en/disablen
  - Queue en/disablen
  - Druckaufträge löschen
  - Druckauftrag an die Spitze einer Queue stellen
  - Signale an lpd senden
- GUI `klpq`, ...

Folie 72

lpc-Syntax

- Kommandozeile: `lpc [Optionen] [Kommando [Argument]]`
  - Optionen: `-P Queue, -a=-P all`
  - Kommandos: start, stop, status, redirect, topq, lprm
    (nur LPRng): move Queue1 [jobid] Queue2
- Interaktiver Modus

```
$ lpc
lpc$>$ start lp
lpc$>$ status lp
lpc$>$ exit
$
```

Folie 73

Das direkte *Einrichten eines neuen Druckers* über die Datei `/etc/printcap` ist eine eher komplizierte Angelegenheit. Daher bietet sich der Weg über zusätzliche (graphische) Tools wie `printtool` (RedHat) oder Administrationsschnittstellen anderer Distributionen (zB YaST) oder anderer zB auf `freshmeat.net` verfügbarer Programme an.
Wer dennoch den „steinigen“ Weg gehen will, muss über die Syntax der Datei Bescheid wissen. Das erste Feld enthält die Namen, unter denen die Queue ansprechbar sein soll, jeweils durch | separiert. Die darauf folgenden Optionen haben die Form `keyword=value` und müssen durch Doppelpunkte getrennt sein. Jeder Druckereintrag sollte mindestens die Felder (siehe auch Folie 68)

```
:sd=/pfad/zur/queue
:lf=/var/log/lpd-errors
:lp=/dev/parport0
```

enthalten, wobei `sd` für das Spool Directory, `lf` für Logfile und `lp` für die Hardwareschnittstelle zum Drucker steht. Handelt es sich um einen nichtlokalen Drucker (Netzwerkdrucker; Druckserver auf entferntem Rechner) so ist der `lp`-Eintrag durch `rm=hostname` (Remote) und `rp=queue` (Remote Printer) zu ersetzen. Weitere wichtige Felder (etwa `if` oder `sh`) können bei Bedarf in der Manpage von `printcap` nachgesehen werden. Anschließend muss man noch das Spoolverzeichnis erstellen und die Rechte dort richtig setzen.

LPRng bietet auch die Möglichkeit, mittels `checkpc` zu überprüfen, ob der Druckereintrag korrekt ist. Unter Cups schließlich kann das Installieren eines neuen Druckers einfach über das Webinterface erfolgen.

Installation, Konfiguration

- BSD/LPRng
  - `/etc/printcap`
  - RedHat Printtool
    - lokaler Drucker
    - remote Unix Queue (lpd)
    - LAN (SMB, Netware)
- Cups Webinterface

Folie 74

8 Dateisysteme

Hier wiederholen wir kurz die Festplattenpartitionierung, erklären den praktischen Umgang mit Dateisystemen, stellen das Standard-Linux-Dateisystem `ext2` vor und erwähnen andere von Linux unterstützte Dateisysteme. Wir besprechen die Grundlagen von Journaling-Filesystemen und die zwei Beispielsysteme `ext3`, und ReiserFS.

8.1 Praktischer Umgang mit Dateisystemen

Partitionierung
Auf Intel-basierten Systemen werden Festplatten in logische Bereiche unterteilt; diese *Partitionen* (vgl. Teil 1, Abschnitt 8.2) können wie eigene Festplatten angesprochen werden und unterschiedliche Dateisysteme beinhalten. Achtung beim Partitionieren gehen alle Daten auf dem Datenträger verloren (es gibt auch Tools, bei denen in eingeschränktem Ausmaß Daten erhalten bleiben; es ist aber empfehlenswert, trotzdem ein Backup zu machen). In der Regel wird die Partitionierung beim Installationsvorgang erledigt und später nur freigelassener Plattenplatz oder neue Platten partitioniert.
Unter Linux ist `fdisk` das Standardwerkzeug zur Partitionierung (vgl. Teil 1, Abschnitt 8.3); ein etwas einladenderes Look and Feel bietet `cfdisk` und der RedHat-Installer verwendet per Default den Disk Druid. Beim Partitionieren muss der Typ der Partition angegeben werden. Dieser Typ entspricht dem auf dieser Partition einzurichtenden Dateisystem – also zB Linux (für ext2) oder FAT für DOS und VFAT für Windows 9x.
Eine kurze Wiederholung der Grundlagen der Partitionierung (unter Linux) findet sich auf Folie 75.

Dateisysteme erzeugen
Auf einer neu erstellten Partition muss nun ein (zunächst leeres) *Dateisystem* erzeugt werden, auf dem die Daten gespeichert werden können; dieser Schritt wird auch *Formatieren* genannt. Klarerweise muss der Partitionstyp mit dem Filesystemtyp übereinstimmen. Details über den Aufbau von Dateisystemen und vor allem den Standardtypen, die unter Linux verwendet werden besprechen wir in eigenen Abschnitten.

Partitionierung

- maximal vier primäre Partitionen
- davon maximal eine als erweiterte Partition
- Erweiterte Partitionen sind Behälter für (max. 12) logische Partitionen
- alle Partitionen (logisch oder primär) lassen sich gleich ansprechen
- Partitionstypen bezeichnen das verwendete Dateisystem
- Namensgebung unter Linux
  - IDE: `/dev/hd[a-h]`, SCSI: `/dev/sd[a-z]`
  - primär: `/dev/?d?[1-4]`
  - logisch: `/dev/?d?[5-16]`

Folie 75

Unter Linux können nicht nur eine Vielzahl verschiedener Dateisysteme (Überblick auf Folie 81, genaueres in Abschnitt 8.2) verwendet, sondern auch erstellt werden. Das Programm `mkfs` (Make File System) ist ein Front-End für die verschiedenen Programme, die dann tatsächlich die Dateisysteme verschiedenen Typs erzeugen: Die Syntax ist:

```
mkfs [-t FsType] [FsOptions] Device
```

Das `Gerät` (Device) ist üblicherweise eine spezielle *Gerätedatei*, über die die Hardware (zB Festplattenpartition `/dev/hda1` oder Floppy `/dev/fd0`) angesprochen wird. Der Filesystemtyp `FsType` wird hier ein für alle mal festgelegt und kann später im laufenden Betrieb nicht verändert werden (ohne die am Filesystem gespeicherten Daten zu verlieren).
Festplattenpartitionen unter Linux werden meist mit dem 2nd extended Filesystem (Option `ext2`; wird in Abschnitt 8.3 detailliert behandelt) formatiert, Disketten mit der Option `msdos` für DOS-formatierte Floppies, oder (wenn Dateiberechtigungen verwendet werden sollen) mit der Option `minix`.
Die `FsOptions` hängen freilich von `FsType` ab; unterstützt werden meistens: `-v` (verbose) für ausführliche Meldungen und `-c` (check) zum Prüfen auf defekte Blöcke.

Eine Liste der Blöcke eines Mediums, die sich nicht lesen lassen, erhält man auch durch `badblocks Gerät`; Schreibtests werden nur dann durchgeführt, wenn man die Option `-w` einsetzt. `mkfs -c` oder `badblocks` werden üblicherweise nur bei Disketten eingesetzt oder wenn man Probleme erwartet, da das ein sehr, sehr langes Lesen des gesamten Mediums erfordert (das DOS-Tool `format` macht dies immer). Bei vielen Dateisystemen ist das überflüssig, da defekte Blöcke beim Schreiben von Daten im laufenden Betrieb erkannt werden und dann nicht benutzt werden.

Man kann statt des Aufrufs des Front-Ends `mkfs` auch direkt die Programme verwenden, die die Formatierung durchführen zB `mkfs.ext2` = `mke2fs`, `mkfs.msdos`, etc. Die jeweiligen Optionen sind ausführlich in den Manpages dokumentiert.

Erstellen von Dateisystemen/Formatieren

- erstellt Strukturen des FS auf Partition
- bereitet die Partition auf den Einsatz vor
- Filesystemtyp muss mit Partitionstyp übereinstimmen
- kann später nicht mehr geändert werden
- Front-End für alle Typen:
  `mkfs [-t type] [-c] Device`
- Einzelne Tools:
  `mke2fs = mkfs.ext2 [Optionen] device` für Feintuning

Folie 76

Mounten

Einer der großen Vorzüge des Konzepts des Unix/Linux Verzeichnisbaums ist, dass er vollständig unabhängig von der zu Grunde liegenden Partitionierung der Festplatten des Systems immer die gleichen Verzeichnisse aufweist (vgl. das Laufwerk-Konzept unter MS-Windows). Verschiedene Festplattenpartitionen (sogar mit verschiedene Dateisystemtypen) können an beliebigen Punkten des Verzeichnisbaums „eingehängt“ oder „montiert“ werden; etwas technischer spricht man vom *Mounten* von Dateisystemen.
Dem Zugrunde liegt das sog. *virtuelle Dateisystem* von Unix/Linux, das den Zugriff auf die verschiedensten Medien vereinheitlicht, die für alle Programme völlig transparent als Verzeichnisse dargestellt werden. Ein Programm muss nicht wissen (und weiß auch nicht!), auf welchem Medium die Datei liegt, die es gerade schreibt oder liest. Ein weiterer Vorteil dieses Konzepts liegt darin, dass sich der Zugriff auf jedes Medium einfach über die Dateiberechtigungen steuern lässt.

Das Mounten von Dateisystemen ist prinzipiell root vorbehalten (Ausnahmen siehe unten) und wird mittels `mount`-Kommando bewerkstelligt; Syntax:

```
mount [-t FsTyp] Device Mountpoint
```

Hier Bezeichnet `Device` wieder die Hardwaregerätedatei für das entsprechende Medium. `Mountpoint` bezeichnet den Platz im Verzeichnisbaum, wo das entsprechende Gerät eingefügt werden soll. Dabei sollte es sich um ein *leeres* Directory handeln; ist es nicht leer, sieht man die darin befindlichen Dateien solange nicht, wie ein Medium auf das Verzeichnis gemountet ist. Üblicherweise reicht ein `mount Gerät Mountpoint`, um ein formatiertes Medium auf `Gerät` unter dem Namen `Mountpoint` ansprechen zu können, da `mount` den Dateisystemtyp selber festzustellen versucht. Andernfalls muss man mit `-t Typ` nachhelfen. Bei Medien, die man nicht schreiben kann (CDROMs) oder will (Backups), setzt man die Option `-r` für „read only“.

Braucht man ein Medium nicht mehr, hängt man es per `umount Gerät` oder `umount Mountpoint` (kein *n* obwohl „unmount“) wieder aus dem Verzeichnisbaum aus. Verwendet ein Programm noch Dateien auf diesem Medium oder ist sein Arbeitsverzeichnis (pwd) noch in einem Unterverzeichnis des Mediums, verweigert der Kernel das Aushängen. Da es *keine* `--force`-Option gibt, die ein Unmounten erzwingt (würde die Konsistenz des Filesystems gefährden), muss man zuerst die das Medium benutzende Programme schließen (oder töten). Die Tools `fuser` und `lsof` (List Open Files) zeigen, welche Programme das sind.

Die standardmäßig vom System verwendeten Festplattenpartitionen müssen natürlich nicht (mit oben beschriebener Syntax) händisch gemountet werden, sondern werden automatisch beim Systemstart gemountet. Genauer wird das Wurzelverzeichnis `/` in einem sehr frühen Stadium vom Kernel gemountet (vgl. Teil 1, Abschnitt 9.1, der Bootloader übergibt die Informationen über die Rootpartition an den Kernel, der einen Fallback-Default fix eincompiliert hat, der aber mit `rdev` geändert werden kann). Die weiteren Dateisysteme werden von `/etc/rc.sysinit` nach eventuell durchzuführenden Filesystemchecks gemäß dem in der Datei `/etc/fstab` vorgegebenen Schema an die entsprechenden Stellen im Verzeichnisbaum gemountet (mit einem Aufruf von `mount -a`). Eine Beispiel-Fstab befindet sich auf Folie 78. Jede Zeile beschreibt die Verwendung eines Gerätes; als Trennzeichen zwischen den Einträgen können Leerzeichen und Tabs verwendet werden. Die Reihenfolge ist: Gerät, Mountpoint, Dateisystem, Optionen, dump-Eintrag (veraltet), fsck-Eintrag. der Geräteeintrag kann die Gerätedatei (zB `/dev/hda7`), `none` (für virtuelle Dateisysteme wie `proc` oder ein *Label* sein. (Jede `ext2`-Partition kann durch ein sog. Label identifiziert werden; siehe `man e2label`). Der fsck-Eintrag bestimmt die Reihenfolge eventuell durchzuführender Filesystemchecks; 0 bedeutet keinen Prüfung. Einträge mit der Option `noauto` werden nicht automatisch (von `mount -a`) gemountet, sondern dienen zum definieren von Standardmountpoints zB für Floppy und CDROM, d.h. ruft man `mount` auf und lässt `Gerät` oder `Mountpoint` weg, wird das fehlende Argument mit Hilfe der `/etc/fstab` ergänzt.

Üblicherweise erlaubt der Kernel nur `root` Medien zu mounten, das SUID-Programm `mount` erlaubt aber auch normalen Benutzern `/etc/fstab`-Einträge zu nutzen, falls die Option `user` eingetragen ist. Ist die Option `owner` gesetzt, so kann außer root auch (und nur) der Besitzer der Gerätedatei den Mount ausführen; dieser Mechanismus wird von den Desktopsystemen genutzt: Loggt sich ein Benutzer auf der grafischen Oberfläche ein, wird er Eigentümer einiger Gerätedateien aus `/dev`, so zB von `/dev/cdrom` und `/dev/fd0`. `mount` erlaubt es dann diesem Benutzer die CDROM und die Floppy zu mounten, allerdings nur *genau* mit den Optionen in `/etc/fstab`; ist für `/dev/cdrom` als Dateisystem `iso9660` eingetragen ist, kann der Benutzer nur dann den Mount ausführen, wenn es sich wirklich um ein solches Dateisystem handelt. Für die Floppy ist meist `auto` für Dateisystemtyp eingetragen, was die Verwendung beliebiger Dateisysteme auf der Diskette ermöglicht. Für weitere Details siehe `man fstab`.

Das `mount`-Kommando schreibt bei jedem erfolgreichen (U)Mount eines Mediums eine Zeile nach dem Muster der `/etc/fstab` in die Datei `/etc/mtab`; diese enthält daher immer den aktuellen Stand über alle verwendeten Medien; ein `mount`-Aufruf ohne Optionen und Argumente wertet genau diese Informationen aus. Komfortabler zeigt `df` (Display Filesystem) die gemounteten Medien und eine Statistik über ihrer Benutzung.

Mounten/Unmounten

- mounten: `mount [-t FsType] Gerät Mountpoint`
- gemountete Dateisysteme listen
  - `mount`
  - `df`
  - `less /etc/mtab`
- unmounten: `umount Gerät` *oder* `Mountpoint`
  - FS darf nicht geöffnet sein (Files, Programme, Directories)
  - `fuser`, `lsof` zeigt Prozesse, die FS verwenden
  - kann **nicht** erzwungen werden!!

Folie 77

/etc/fstab

```
/dev/hda2     /              ext3    defaults          1 1
/dev/hda1     /boot          ext3    defaults          1 2
none          /dev/pts       devpts  gid=5,mode=620    0 0
none          /proc          proc    defaults          0 0
none          /dev/shm       tmpfs   defaults          0 0
/dev/hda3     swap           swap    defaults          0 0
/dev/cdrom    /mnt/cdrom     iso9660 noauto,owner,ro   0 0
/dev/fd0      /mnt/floppy    auto    noauto,owner      0 0
```

Folie 78

8.2 Dateien und Dateisysteme

Einschub: Gerätedateien
Jeder (unterstütze) Datenträger – wie prinzipiell jedes Hardwaredevice – wird unter Unix durch eine sogenannte Gerätedatei (Devicefile, eine spezielle Datei im Verzeichnis `/dev/`) dargestellt, über die der Zugriff auf das entsprechende Gerät erfolgt. Auf diese Weise sind keine speziellen Programme nötig, um (low-level) auf die Geräte zuzugreifen. Zum Beispiel kann man mittels

```
$ cat foo > /dev/lp0
```

den Inhalt der Datei `foo` auf dem Drucker (genauer der ersten parallelen Schnittstelle) ausgeben. Die Daten müssen klarerweise in einem Format sein, das der Drucker auch versteht. Da es, wie schon erwähnt, keine gute Idee ist, dass mehrere Benutzer direkt ihre Daten an den Drucker schicken, schaltet man eine Druckerwarteschlange (Queue) dazwischen (vgl. Kapitel 7). Das gilt für auch für die meisten anderen Geräte, weshalb man nur in den seltensten Fällen direkt auf die Gerätedateien zugreifen muss bzw. warum normalen Benutzern dafür auch die Rechte fehlen. Wir diskutieren im Folgenden kurz die Grundlagen im Umgang mit Gerätedateien.

Linux unterscheidet zwischen zwei Gerätearten: *Random-Access Block Devices* (zB Festplatte, CD-ROM) und *Character Devices* (zB Bandlaufwerk, Tastatur, Maus oder serielle Schnittstellen). Ein langes Listing zeigt bei jeder speziellen Datei, ob es sich um eine Block Device (b) oder ein Character Device (c) handelt; zB:

```
stein@dukana:stein;-) ls -l /dev/hda1 /dev/cua1
crw-rw----    1 root     uucp     5,  65 Aug 30  2001 /dev/cua1
brw-rw----    1 root     disk     3,   1 Aug 30  2001 /dev/hda1
```

Bei den meisten Systemen sind alle Gerätedateien vorhanden, unabhängig davon, ob das Gerät tatsächlich vorhanden ist. Nur weil man eine Datei `/dev/sda` am System hat, bedeutet das also nicht, das tatsächlich eine SCSI-Festplatte eingebaut ist. Die Dateien werden vom Installationsprogramm automatisch erstellt, sodass man sie beim Einbau eines neuen Geräts nicht „von Hand“ erzeugen muss (was auch eher kompliziert ist; das entsprechenden Kommando heißt `mknod`).

Gerätedateien

- jedes Gerät wird über eine Gerätedatei angesprochen
- es gibt zwei Arten von Geräten:
  - Random-Access Block Devices
  - Character Devices
- die Gerätedatei existiert auch, wenn es das Gerät nicht gibt
- Gerätedateien besitzen ganz normale Berechtigungen

Folie 79

Konsequenterweise haben auch die Gerätedateien Berechtigungen und können gelinkt werden. Dadurch kann man bestimmten Benutzern die Verwendung zB des Modems sehr leicht verbieten. Besitzer ist meistens root, bei Wechselmedien allerdings der lokal (unter X) angemeldete Benutzer. Symbolische Links bieten bequem eine Möglichkeit ein Standardgerät festzulegen; zB ist `/dev/cdrom` oft ein Link auf das entsprechende IDE-Gerät also etwa `dev/hdc`.

Was genau ist eine Datei?
Ein gewöhnliches File enthält ein Dokument (Text, Graphik, Datenbankfile) oder ein Programm. Die Files besitzen keine Satzstruktur, sondern werden als sequentielle Ströme von einzelnen Zeichen behandelt. Eine Blockstruktur, wie sie auf Disketten oder Festplatten vorhanden ist, ist für die Anwendung/den Benutzer nicht erkennbar. Jede Anwendung kann so ihre eigene Struktur definieren.

Was genau tut ein Dateisystem?
Die Aufgabe eines Dateisystems besteht darin, die einzelnen Datenblöcke einer Datei auf die Festplatte (oder einen anderen Datenträger) abzulegen. Es legt fest, an welcher Stelle des Datenträgers sie gespeichert werden sollen und archiviert diese Informationen – die sog. *Metadaten* – gleichzeitig in einer Tabelle.
Die Zugriffs- und Lokalisierungsstrategie eines Dateisystems hat direkten Einfluss auf die Durchsatzrate der Schreib- und Lesezugriffe auf das Medium und auf die Zuverlässigkeit des Dateisystems an sich.

Files und Filesysteme

- Eine Datei ist eine fortlaufende Anzahl von Bytes
  - unabhängig vom tatsächlichen Ort auf dem Speichermedium
  - kein interner Aufbau
- Ein Dateisystem verwaltet die Dateien auf einem Speichermedium
  - unabhängig von der Art der Speicherung
  - Metadaten beinhalten Verwaltungsüberbau

Folie 80

Dateisysteme unter Linux
Obwohl `ext2` das Standard Dateisystem unter Linux ist, kommt das Betriebssystem mit einer ganzen Reihe von Dateisystemen zurecht. Hier eine etwas ausführlichere – aber beileibe nicht vollständige – Übersicht (vgl. Folie 81):

- `ext2`: 2nd extenden Filesystem; der Standard unter Linux (Details siehe Abschnitt 8.3)
- `ext3`: 3rd extended Filesystem; Weiterentwicklung von `ext2`, Journaling (siehe Abschnitt 8.4)
- ReiserFS: Reiser-Dateisystem, nach seinem Schöpfer Hans Reiser benannt; Journaling (siehe auch Abschnitt 8.4
- JFS: Von IBM entwickeltes Journaling-Filesystem, ursprünglich für AIX (IBMs UNIX) entwickelt; seit Februar 2000 ist eine Open Source Version für Linux verfügbar. Die Entwicklung geht dank finanziellem Hintergrund von Big Blue sehr rasch von statten.
- XFS: Journaling-Filesystem unter UNIX. Ursprünglich von SGI für IRIX, aber mittlerweile unter der GPL in einer Beta-Version als Patch für Linux verfügbar. Ist neben JFS und ReiserFS ein weiterer aussichtsreicher Kandidat für den nächsten Journaling-Filesystem Standard.

- iso9660: Standardformat für CD-ROMs und DVDs. Es gibt hier verschiedene Erweiterungen (zB Rock Ridge Interchange Protocol), um etwa die 8.3 Beschränkungen aufzuheben.
- udf: Universal Disk Format (CDRWs und DVDs) ist die Weiterentwicklung von ISO9660 und heißt eigentlich ISO13346
- MS-DOS: Dateisystem für MSDOS-Partitionen (FAT12, FAT16, FAT32) und -Disketten (kurze Dateinamen). Es gibt, trotz gleichen Namens (FAT), einen Unterschied zwischen FAT12 und FAT16. Ersteres ist nur für Dateisysteme mit weniger als 16 MB, also für Disketten brauchbar.
- VFAT: DOS/Windows 9x-Dateisystem (lange Dateinamen)

- NTFS: Windows NT/2000 Dateisystem (nur Lesezugriff empfohlen)
- UMSDOS: Ein unixartiges FS das auf einer FAT Partition angelegt werden kann.
- minix: Dateisystem von Minix, wird oft für Linux-Disketten verwendet; war das erste Dateisystem für Linux, kann aber nur Dateisysteme mit max. 64MB und Dateinamen mit 30 Zeichen anlegen.
- NCPFS: Novell FS
- HPFS, HFS, AFFS: FS von OS/2 (wie schon OS/2 selbst, (fast) ausgestorben), MAC und Amiga
- Weitere in der Unix-Welt verbreitete FS wie `ext`, System V, Xenix, ...

- proc: virtuelles Dateisystem zur Prozessverwaltung (`/proc`).
- swap: Swap-Partitionen oder -Dateien; siehe Kapitel 5.
- autofs: für das automatische Mounten eines Dateisystems bzw. Datenträgers ins Gesamtsystem – also ebenso kein echtes Dateisystem
- usbdevfs: Einbinden und Verwalten von USB-Geräten
- devpts: für Pseudoterminals (nach UNIX-98-Spezifikation)

- NFS: Netzwerkdateisystem für UNIX, um Daten von anderen Rechnern zu lesen (siehe Kapitel 16)
- SMBFS: Samba (Netzwerkdateisystem unter Windows); siehe Kapitel 19

Wir besprechen in den folgenden Abschnitten das Standarddateisystem unter Linux, das 2nd extenden Filesystem (`ext2`), sowie die Journaling-Dateisysteme `ext3` (3rd extenden Filesystem) und ReiserFS.

Unterstützte Filesysteme

- `ext2`... 2nd extended FS, Linux-Standard
- ReiserFs, `ext3`... Journaling-FS für Unix/Linux
- JFS, XFS... Journaling-FS von IBM, SGI
- iso9660, udf ... (CDROM)
- FAT-12, -16, -32, VFAT... (WIN 3x, 9x)
- NTFS... Win NT (read only !?)
- minix, ext, xiafs, System V, Xenix, Coherent... Unix
- HPFS, HFS, AFFS... OS/2, Mac, Amiga (read only)
- UMSDOS... Unix like on top of MSDOS
- `/proc`... Linux Kernel und Prozessinfo (virtuelles FS)
- NFS... Network File System
- SMBFS... SMB
- NCPFS... Novell

Folie 81

8.3 Das `ext2`-Dateisystem

In diesem Abschnitt besprechen wir die Interna des 2nd extended Filesystems, d.h. die Art und Weise wie unter `ext2` die Dateien verwaltet werden und stellen einige Tools vor, mit denen ins Dateisystem eingegriffen werden kann.

Das `ext2`-Filesystem bewältigt seine Verwaltungsaufgabe, indem es den vorhandenen Speicherplatz in *Blöcke* gleicher Größe (Defaultwert 1024 Byte) einteilt und diese Blöcke durchnummeriert. Die Menge der Blöcke wird anschließend in verschiedene *Gruppen* aufgeteilt, die jeweils zur Speicherung unterschiedlicher Datentypen genutzt werden.
Wichtigste dieser Gruppen sind neben den Datenblöcken (wo die eigentlichen Daten gespeichert werden) die *Inodes* (Index Nodes, oder auch Informationsknoten), die mit Ausnahme des Dateinamens alle relevanten Daten über die gespeicherten Dateien enthalten, nämlich: Benutzer- und Gruppen-ID; Zugriffsrechte; Größe der Datei; Anzahl der (harten) Links, die auf diese Datei verweisen; Daten der Erstellung, der letzten Änderung oder des Löschens dieser Datei und Verweise auf die (ersten zwölf) Datenblöcke wo die Datei gespeichert ist. Die *Dateinamen* schließlich sind in den Verzeichnissen gespeichert; diese sind im Prinzip wie in Zeilen gegliederte Textdateien: jede Zeile beinhaltet den Name einer Datei (oder eines Unterverzeichnisses) und die zugehörige Inode-Nummer (siehe Folie 83). Zusätzlich findet sich in jedem Verzeichnis ein Verweis auf das Verzeichnis selbst (`.`) und das übergeordnete Verzeichnis (`..`).

Eine Datei besteht also aus ihrem entsprechenden Inode und aus mehreren Datenblöcken, die die eigentlichen Daten enthalten. Die Verwaltungsinformationen im Inode und der eigentliche Dateninhalt der Datei sind vollkommen getrennt und können auch unabhängig voneinander bearbeitet werden. Wird zB eine Datei in ein neues Verzeichnis verschoben, wird der Inode und die entsprechenden Verzeichniseinträge verändert, aber niemals werden Datenblöcke direkt verändert.

Das ext2-Filesystem

- aufgeteilt in Blöcke gleicher Größe (Defaultwert 1024 Bytes)
- 6 Gruppen von Blöcken: Bootblock, Superblock, Inode Bitmap, Datablock Bitmap, Inodes, Datablocks (Redundanz!)
- Wichtigste Bestandteile:
  - Superblock
  - Inode (Index Node, Informationsknoten)
  - (double, triple) indirekter Block
  - Datenblock

Folie 82

Mit diesem Hintergrundwissen sehen wir uns ein langes Listing eines Verzeichnisses an und Diskutieren die Dateiverwaltung an diesem Beispiel.

```
bash-2.04$ ls -ail
223244 drwxr-xr-x 4 jj users 1024 Jan 21 17:00 .
180247 drwxr-xr-x 3 jj users 1024 Jan 21 17:03 ..
 84002 drwxr-xr-x 2 jj users 1024 Jan 21 17:00 dir1
223245 drwxr-xr-x 2 jj users 1024 Jan 21 17:01 dir2
```

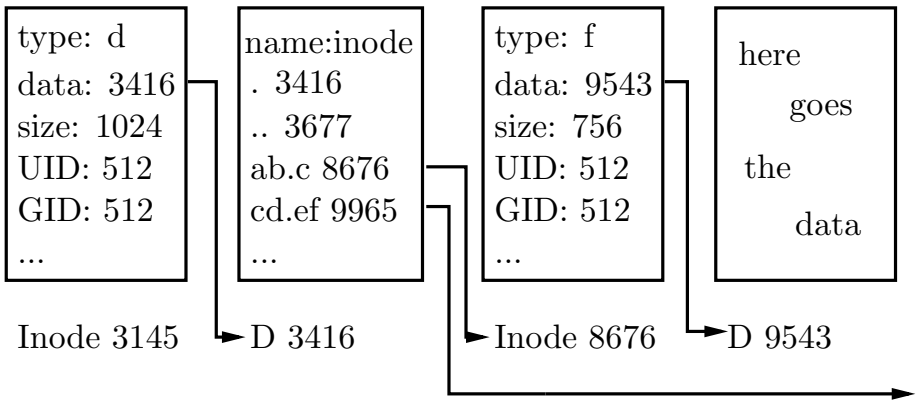
Von links nach rechts erklärt: Die linke Spalte gibt die Inode-Nummern an, unter der die Zeiger, Unterverzeichnisse oder Daten abgelegt sind. In der nächsten Spalte werden die Zugriffsrechte aufgelistet. Darauf folgt die Spalte mit der Anzahl harter Links, die auf diese Dateien oder Verzeichnisse zeigen (Linkcount). Eine Datei hat zwar nur einen Inode, aber in diesem können mehrere (harte) Links auf diese Datei gespeichert werden. Beispielsweise ist für das Unterverzeichnis `dir1` die Zahl von zwei Hardlinks angegeben. Konkret besagt dies: auf `dir1` verweist einmal `dir1` in `jj` und einmal `.` in `dir1`. Explizit:

```
bash-2.04$ cd dir1; ls -ail
 84002 drwxr-xr-x 2 jj users 1024 Jan 21 17:00 .
223244 drwxr-xr-x 4 jj users 1024 Jan 21 17:00 ..
```

Ein Inode kann mittels *Pointer* maximal 12 Datenblöcke direkt adressieren. Umfasst eine Datei mehr als 12 KByte (12 mal 1024 Byte oder zwölf Datenblöcke), kommt ein indirektes Adressierungsschema zum Einsatz. Jeder der 12 Pointer kann statt auf einen Datenblock auf einen *indirekten Block* zeigen, der weitere Pointer enthält. Nachdem ein Pointer 4 Bytes Speicherplatz verbraucht, kann jeder der indirekten Blöcke maximal 256 Pointer beinhalten, die entweder auf Datenblöcke oder auf weitere indirekte Blöcke verweisen. Letztere enthalten wiederum maximal 256 Pointer, die entweder auf Datenblöcke oder indirekte Blöcke zeigen. Nach dem dritten Level der indirekten Referenzierung ist aber Schluss: Man nennt Blöcke deren Pointer nur auf Datenblöcke zeigen *(einfach) indirekte Blöcke*, Blöcke deren Pointer auf (einfach) indirekte Blöcke zeigen *doppelt indirekte Blöcke* und schließlich Blöcke, deren Pointer auf doppelt indirekte Blöcke zeigen *dreifach indirekte Blöcke*; vierfach indirekte Blöcke sind nicht erlaubt.

Dateien und Verzeichnisse

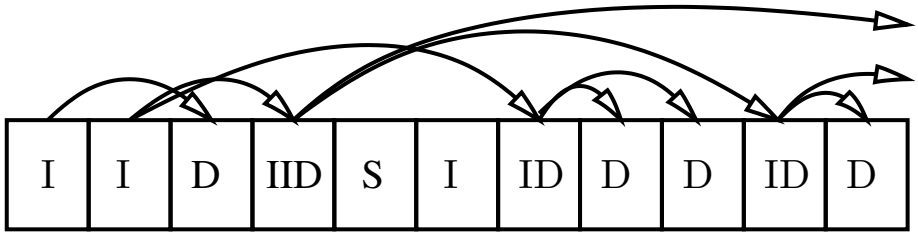
Verzeichnisse enthalten Tabellen: Dateiname – Inode



Folie 83

Indirekte Adressierung

- Inode zeigt auf ersten 12 Datenblöcke
- Indirekte Blöcke (ID): 256 Zeiger auf Datenblöcke
- Doppelt indirekte Blöcke (IID): 256 Zeiger auf ID
- 3-fach indirekte Blöcke (IIID): 256 Zeiger auf IID
- theoretische Maximalgröße für Dateien 16 GB; praktisch 2 GB



Folie 84

Rechnet man nun alles richtig zusammen, so ergibt sich eine theoretische Maximalgröße für Dateien von 16 GB. Aufgrund verschiedenen anderer Beschränkungen kann eine Datei auf einem `ext2` System allerdings nur höchstens 2 GB groß sein; (das ist natürlich für Datenbankanwendungen etwas klein und man wird hier zu einem anderen Dateisystem greifen müssen).

Neben den Inodes und Datenblöcken kennt das `ext2`-Layout noch weitere (Gruppen von Blöcken). Der wichtigste ist der *Superblock*. Er enthält wichtige Informationen über das Layout des Dateisystems, wie zB die Größe der Blöcke und der folgenden Gruppen, Informationen über den letzten Mount und Filesystemcheck, etc. Beim Mounten wird auf Informationen im Superblock zurückgegriffen; ist der Superblock korrumpiert, so kann das Dateisystem nichtmehr gemountet werden. Daher wird der Superblock repliziert und mehrere Kopien davon an verschiedenen Stellen im Dateisystem aufbewahrt (vgl. Folie 85).
Der *Bootblock* enthält eine Menge von Minimaldaten, die benutzt werden, falls von der entsprecheden Partition gebootet wird.
Schließlich werden die Nummern von freien Datenblöcken und freien Inodes in sog. Bitmaps gespeichert; diese *Block Bitmaps* und *Inode Bitmaps* stellen die 4. und 5. Gruppe von Blöcken im `ext2`-Dateisystem dar (vgl. Folie 86).

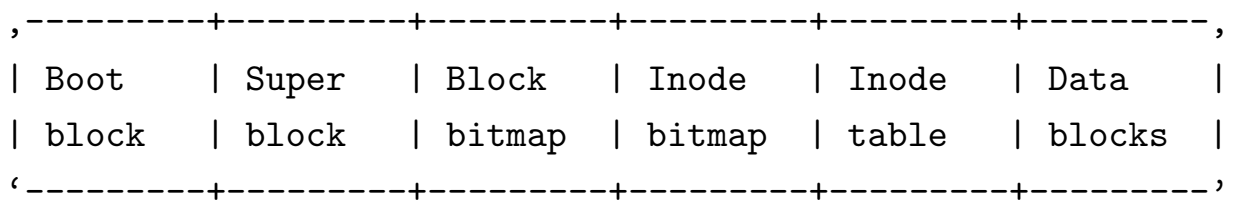
.

Superblocks, Inodes und Datenblöcke

- Superblock
  - 1. Block des FS, Kopien (8193, 16385)
  - Info über FS (letzter Mount, Blockgröße, Zeiger auf freie B/I)
- Inode
  - 256 Bytes (4/Block)
  - Benutzer, Gruppe, Berechtigungen, s/c/a/m-time
  - Zeiger auf Datenblöcke (max 12)
  - Zeiger auf indirekte Blöcke (3 Levels)
- Datenblöcke
  - eigentliche Daten der Dateien
  - Verzeichnisse (Inode-Tabellen)

Folie 85

Gruppen von Blöcken in ext2



Folie 86

Finetuning des Dateisystems

Man kann – allerdings nur beim Erstellen des Dateisystems – einige Finetuning vornehmen. Die *Inode-Dichte* gibt an, welche Dateigröße für ein Dateisystem erwartet wird. Die Anzahl der Inodes bestimmt, wie viele Dateien angelegt werden können. Ist die Zahl zu klein, kann es vorkommen, dass die Partition als voll gilt, obwohl durchaus noch Datenblöcke frei sind.
Wird zum Beispiel ein Wert von 4096 Bytes pro Inode (Standardwert) gewählt, bedeutet dies, dass im Schnitt alle Dateien 4 kB groß sein werden. Werden auf diesem Dateisystem ausschließlich Dateien mit einer Größe von nur einem Kilobyte angelegt, so kann man nur ein Viertel der Platte verwenden, da das Dateisystem dann als voll gilt.

Die Blockgröße ist die kleinste adressierbare Einheit in einem Dateisystem. D.h. jede Datei verbraucht ein Vielfaches dieser Blockgröße, unabhängig von ihrer eigentlichen Größe. Die Blockgröße beeinflusst die Dauer des Dateisystemchecks; sie steigt exponentiell mit der reziproken Blockgröße. Für Partitionen mit mehr als 4 GB wird oft eine Blockgröße von 4096 Bytes gewählt.

Nützliche Kommandos

- `fsck` überprüft das Dateisystem auf kleinere Fehler und kann diese beheben. Meistens wird fsck automatisch gestartet, zum Beispiel beim Booten von `/etc/rc.sysinit` nach einem Systemabsturz. Das funktioniert so: Im Superblock (Gruppe 2) wird beim Mounten des Dateisystems ein sogenanntes Valid Bit gelöscht, das nach einem sauberen Unmount wieder gesetzt wird. Fehlt dieses Bit beim nächsten Start des Systems, bedeutet dies, dass das System nicht ordnungsgemäß heruntergefahren wurde. Jetzt wird ein Programm gestartet, e2fsck, das die Daten auf Konsistenz überprüft und wenn nötig zu reparieren versucht. Dies ist nicht immer erfolgreich, Dateien (oder Datenblöcke), die nicht ordnungsgemäß wiederhergestellt werden konnten oder nicht richtig referenziert werden können, landen in einem extra dafür vorgesehenen `/lost+found` Verzeichnis der Partition.

Da `fsck` über die gesamte Partition (oder Platte) laufen muss, kann diese Prozedur unter Umständen je nach Größe des Dateisystems einige Minuten kostbarer Uptime in Anspruch nehmen. Ist man auf eine effizientere (schnellere) Art der Dateiverwaltung angewiesen, sollte man eine andere Art der Dateiverwaltung in Betracht ziehen (Journaling, Abschnitt 8.4).

- `debugfs:` Dieses Programm setzt dort an, wo fsck aufgibt. Bevor man allerdings Zeit und Mühe investiert um verloren gegangene Daten vielleicht zu retten, wäre ein regelmäßiges Backup eine Überlegung wert.

- `tune2fs:` kann einige Parameter für fsck setzen:

  - Die maximale Anzahl der Mounts, nachdem eine gründliche Überprüfung erzwungen wird (meistens beim Booten)
  - Die maximale Zeit zwischen zwei Checks
  - Anzahl der Blocks, die für root reserviert sind. Dadurch kann man auch auf einer an sich vollen Platte noch Administrationsarbeiten durchführen, ohne dass man etwas löschen muss.

- `dumpe2fs:` zeigt Informationen über das Dateisystem, ausgehend vom Superblock, an.

Inode-Dichte, Blockgröße

- wichtigste Komponenten: Inodes und Datenblöcke
- FS voll, wenn
  - keine freien Datenblöcke, ODER
  - keine freien Inodes
- Finetuning des Filesystems (nach Bytes/File)
  - Blockgröße (1024, 2048, 4096)
  - Bytes/Inode (4096)

Folie 87

Nützliche Kommandos

- `fsck` Konsistenzüberprüfung von FS
- `debugfs` Debugging
- `badblocks`, überprüft auf defekte Blöcke
- `tune2fs` Finetuning (Blockreservierung für root)
- `df` zeigt freien Platz auf dem Dateisystem an
- `du` zeigt Größe von Files/Directories

Folie 88

8.4 Journaling: `ext3` und ReiserFS

Wie werden Daten korrumpiert?
Stellen wir uns vor, wir editieren gerade eine Textdatei. Angenommen, das System stürzt ab, bevor wir eine Änderung abgespeichert haben, so ist diese verloren, aber die alte Version ist noch vorhanden; so haben wir zwar Daten verloren, aber das Malheur hält sich in Grenzen. Stürzt allerdings das System ab, gerade während die Datei geschrieben wird, ist der Schaden möglicherweise viel größer. Teile der neuen Datei sind schon überschrieben, Teile noch nicht und eventuell geht die gesamte Datei verloren.
Noch schlimmer ist es, wenn das System beim Schreiben von Verwaltungsinformationen, (der Metadaten) abstürzt, im Falle von `ext2` etwa der Inodes. Geschieht dies zB beim Verschieben eines Directories, so kann der ganze Verzeichnisbaum darunter verloren gehen. Daher bauen alle modernen Dateisysteme Redundanzen bei der Speicherung von Metadaten ein. Nach einem Systemcrash wird beim nächsten Booten das gesamte Filesystem einem Check unterzogen (zB `fsck`), wobei die Metadaten aus der redundanten Information rekonstruiert werden; trotzdem können unter Umständen gewisse Blöcke nicht mehr richtig referenziert werden. Dann gehen einzelne Dateien verloren und die respektiven Blöcke landen (unter `ext2`) in `lost+found/` im obersten Verzeichnis des Filesystems. Bei Dateisystemen mit einigen GB Größe kann dieser Vorgang mehr als 10 Minuten in Anspruch nehmen.

Journaling

- Herkömmliches FS
  - Metadaten redundant
  - Langer Dateisystemcheck nach Fehlern
- Journaling
  - Journal/Log für Metadatenänderungen/alle Schreibzugriffe
  - Dateisystemprüfung durch Auswerten des Journals
  - mehr Schreibzugriffe
- Journaling-Filesysteme
  - `ext3` (3rd extended FS; Nachfolger von `ext2`)
  - ReiserFS
  - JFS, XFS

Folie 89

Journaling
ist nun ein Verfahren, das die Filesystemchecks ersetzt. Es ist zwar im Betrieb manchmal langsamer als herkömmliche Dateisysteme, aber die Datenintegrität ist wesentlich verbessert und Filesystemchecks sind wesentlich verkürzt bzw. unnötig.
Im Gegensatz zu den unproblematischen Lesezugriffen werden Schreibzugriffe besonders behandelt. Bevor die eigentlichen Änderungen am Dateisystem vorgenommen werden, wird ein „Backup“ der zu überschreibenden Daten in eigens reservierten Bereichen, dem sog. *Journal* oder *Log*, gemacht. Erst dann werden die tatsächlichen Änderungen durchgeführt und nach Erfolg der eine Journal-Eintrag gelöscht. Stürzt der Rechner während einer solchen Prozedur ab, sind entweder noch die alten Daten vorhanden oder sie können aus dem Log rekonstruiert werden.

Dieser Zugang erfordert also keinen vollen Scan des Filesystems beim Checken und ermöglicht auch bei großen Partitionen Überprüfungszeiten im Sekundenbereich. Außerdem können die `lost+found`-Verzeichnisse entfallen. Der Nachteil des Journaling ist wegen der zusätzlichen Schreibarbeit eine eventuell geringere Geschwindigkeit. Die `fsck`-Programme könnten zwar theoretisch weggelassen werden; sie sind aber zur Sicherheit vorhanden um Hardwarefehler und etwaige Programmierfehler im Kernel zu mildern.
Prinzipiell wird für Metadaten Journaling gemacht, um immer ein konsistentes Dateisystem zu haben. Weiters unterscheidet man Journaling-FS wo auch ein Daten-Journal existiert.

ReiserFS
ist ein Filesystem, das Journaling nur für die Metadaten macht: Das bringt einen Geschwindigkeitsvorteil aber auch schlechtere Datenintegrität. Da es zur Verwaltung der Dateien datenbankähnliche Strukturen wie Hash-Werte und balancierte B^*-Bäume nutzt, kann auf kleine Dateien schnell zugegriffen werden. Bislang fehlt ein ausgereiftes `fsck.reiserfs`.

Das Dateisystem `ext3`
macht Journaling für Daten und Metadaten. Sehr nützlich ist die Abwärtskompatibilität zu `ext2`: Die Daten werden genauso wie unter `ext2` gespeichert, weshalb man sofort alle Werkzeuge wie zB `fsck.ext2` = `e2fsck` einsetzen kann und – falls notwen-

dig – eine ext3-Partition auch als ext2 mounten kann. Da ext3 durch das Journaling die Möglichkeit hat, die Bewegung der Festplattenköpfe zu optimieren, erzielt es einen höheren Durchsatz als ext2, was sich für große Dateien positiv auswirkt; für kleinere Dateien kommt es durch das Journaling selbst aber zu einer geringeren Geschwindigkeit.

Das Umsteigen von ext2 auf ext3 ist sehr einfach: Egal ob gemountet oder nicht lässt sich ein ext2-Dateisystem durch

```
tune2fs -j /dev/Gerät
```

umstellen. Ein neues ext3-Dateisystem lässt sich mit

```
mke2fs -j /dev/Gerät
```

einrichten; soll das Journal nicht mit dem Dateisystem sondern auf einem anderen Medium gespeichert werden, macht man das mit

```
mke2fs -O journal_dev /dev/myLog
mke2fs -J device=/dev/myLog /dev/myDisk  .
```

Allerdings muss bei mount stets -t ext3 bzw. in der fstab stets ext3 als Typ eingegeben werden. Um auf einem nicht gemounteten ext3-System nicht nur das Dateisystem zu überprüfen sondern auch das Journal durchzugehen, verwendet man

```
e2fsck -fy /dev/Gerät  .
```

Möchte man die Annehmlichkeit nutzen, dass fsck niemals (vor dem Mounten) aufgerufen werden muss, dreht man das mit

tune2fs -i 0 -c 0 /dev/hdxx

ab. Durch Optionen in fstab bzw. für mount kann beeinflusst werden, welcher Grad des Journaling verwendet werden soll. Die Voreinstellung ist data=ordered, womit Daten und Metadaten ins Journal kommen. Um dies mit data=writeback und data=journal teilweise auszuschalten, sehe man in der Online-Dokumentation nach.
Will man auch die Partition die auf / gemountet wird mit ext3 betreiben, so muss entweder der ext3-Treiber fix in den Kernel compiliert oder eine Initramdisk verwendet werden (siehe man mkinitrd).

9 Backup

In diesem kurzen Kapitel erklären wir warum das Backup von Daten wichtig ist, stellen grundlegende Backupstrategien und einfache Backupwerkzeuge vor.

Die Daten auf einem System sind oftmals wesentlich wertvoller und schwieriger (wieder-) zu beschaffen, als die Hardware. Eine der wichtigsten Aufgaben des Systemadministrators ist daher die Sicherung von (Benutzer)daten und Konfigurationsdateien, kurz auch *Backup* genannt. „Irren ist menschlich“, sagt man; daher kann es schon einmal passieren, dass ein Benutzer zB seine Diplomarbeit löscht. Andererseits ist ein Backup auch ein guter Schutz vor Datenverlust durch Hardwaredefekte (zB Ausfall einer Serverfestplatte), Softwarefehler oder Hackereinbrüche. Weiters sind Backups auch hervorragend geeignet, Daten von einem System auf ein anderes zu migrieren. Schließlich dienen Backups zum Archivieren von Daten über längere Zeiträume hinweg; zB Wirtschaftsbetriebe, Versicherungen, Banken, etc. müssen ihre Buchhaltungen einige Jahre lang aufbewahren.

9.1 Backupstrategien

Bei der Planung von Backups sind einige grundlegende Dinge zu bedenken und Entscheidungen zu treffen, die voneinander nicht unabhängig sind und die von der Art und dem Einsatzbereich des Systems bestimmt werden (*Backupstrategie*):

- Zu allererst muss man sorgfältig die *Frequenz und Art* des Backups für die verschiedenen Daten und Datenbereiche des Systems überlegen. Hierbei unterscheidet man zweckmäßigerweise zwischen den *Benutzerdaten* und den *Systemdaten*, wie etwa Konfigurationsdateien, wichtigen Scripts etc. Die Benutzerdaten werden in der Regel häufiger zu sichern sein als die Systemdaten; zB die Benutzerdaten täglich, die Systemdaten zB nur vor und nach Eingriffen in das System, mindestens aber monatlich.

Warum Backup?

- Daten teurer/schwerer (wieder) zu beschaffen als Hardware
- Disasterrecovery
  - Hard-/Softwareschaden
  - Schaden aufgrund von Installation/Upgrade
  - unabsichtliches Löschen
  - böse Benutzer, Hacker
- Langzeitarchiv
- Systemadministration
  - Datentransfer zwischen Systemen
  - Reorganisation des Filesystems
  - Checkpoint vor und nach Upgrade

Folie 90

Weiters gibt es verschiedene Arten des Backups: Bei *vollen Backups* wird der gesamte Datenbestand gesichert, bei *inkrementellen Backups* nur die Änderungen seit der letzten Sicherung; dazu muss natürlich eine entsprechende Software verwendet werden. Vorteil des inkrementellen Backups ist der geringere Zeitaufwand beim Sichern, das Zurückspielen der Daten wird aber prinzipiell komplizierter und daher langsamer.

- Als nächstes sollte man sich für *Backupmedien bzw. die Hardware* entscheiden. Prinzipiell stehen hier viele Möglichkeiten offen. Bandlaufwerke sind aufgrund ihrer hohen Speicherkapazität weit verbreitet. Backups auf (Wechsel-) Festplatten sind sehr schnell und effizient aber teuer. Zip- und Jaz-Drives haben unter Umständen den Nachteil, dass sie nicht wirklich „Standard“ sind. Schließlich kann man auch Backups auf CD machen, was sich positiv auf die Transportierbarkeit auswirkt; im Gegensatz zu Bandlaufwerken verfügt fast jeder PC über ein CD-Laufwerk. Backups auf Disketten sind aufgrund der geringen Speicherkapazität nur für sehr kleine Datenmengen bzw. einzelne Dateien möglich. In großen Installationen gibt es Möglichkeiten Backups über das Netzwerk zu machen, was meist eine spezielle (kommerzielle) Software erfordert.

- Schließlich muss man natürlich ein geeignetes *Backupprogramm* finden. Meistens ist ein einfaches `tar` nicht ausreichend sondern man muss zu fortgeschritteneren und intelligenteren (möglicherweise) kommerziellen Anwendungen greifen, was fast immer auch mit größerer Benutzerfreundlichkeit einhergeht. Um die Belastung des Administrators gering zu halten, ist es zB wünschenswert, dass die Benutzer die Wiederherstellung ihrer (und zwar nur ihrer) Daten selbst vornehmen können.

- In kritischen Umgebungen darf man Backups nicht am Aufstellungsort des Rechners lagern. Im Falle eines Feuers, Wasserschadens oder anderer „mittlerer Katastrophen“ sollten wenigstens die Backupmedien „überleben“.

Backupstrategie

- Backupschema
  voll/inkrementell, System/Daten
- Backupprogramm/tools
- Medium
  Band, CDR, HD, Floppy(?), Zip, Jaz, Netzwerk
- Lagerung?
- Dokumentation!!!

Folie 91

Hat man sich nun einen Backupplan zurechtgelegt, ist auf sorgfältigste Dokumentation desselben zu achten, um auch anderen (eventuell Administratoren) während eigener Abwesenheit (Urlaub, Wochenende, ...) die Möglichkeit zu geben, in Krisenfällen sofort und richtig zu reagieren.
Schließlich ist ein wichtiger aber oft unterschätzter Punkt das Erproben und Testen des Restorevorgangs (Zurückspielen der Daten vom Backup auf das System), sowie die geordnete Lagerung (inkl. Dokumentation) der Backupmedien. Schließlich sollten Backupmedien auch regelmäßig erneuert (Bänder!), bzw. geprüft werden.

Ein Beispiel für eine ausgearbeitete Backupstrategie inklusive einiger weiterer Tipps ist auf den Folien 92 und 93 zu finden.

Ein Beispielbackupschema

- volles Gesamtbackup: jeden Monat
- Systemdaten: volles Backup
  - vor und
  - nach Systemumstellungen
- Userdaten:
  - volles Backup jedes Wochenende
  - inkrementell jede Nacht

Folie 92

Wichtige Tipps

- nicht nur Sysadmin soll Backup machen können, DOKUMENTATION
- immer mit dem schlimmsten Fall rechnen
- Hardcopies aller verwendeten Scripts aufheben
- Installationsmedien und Backuptools mit Backup aufheben
- Backupmedien beschriften
- regelmäßig neue Medien benutzen
- Recovery testen !!!
- alte Backups regelmäßig prüfen

Folie 93

9.2 Werkzeuge

An einfachen Unix-Werkzeugen zum Backup stehen `tar`, `cpio` und `dump` zur Verfügung, jedes mit seinen spezifischen Vor- und Nachteilen.

Das Kommando `tar` (vgl. auch Teil 1, Abschnitt 6.4) dient nicht nur zum Backup sondern auch dem einfachen Archivieren und Transportieren größerer Datenmengen und von ganzen Verzeichnisbäumen.
Allerdings ist es schwierig, mit `tar` komplette Festplattenpartitionen sichern, was mit `dump` dagegen einfach ist. `cpio` wiederum hat Probleme mit symbolischen Links. Außerdem unterstützt `dump` im Gegensatz zu den beiden anderen Tools inkrementelle Backups. Die Syntax der Befehle und einige Beispiele dazu können entweder den jeweiligen Manpages bzw. den folgenden Folien entnommen werden.
Dort findet man auch Hinweise auf einige kommerzielle Produkte, die, wie bereits erwähnt, meistens wesentlich leistungsfähiger und dabei einfacher zu verwenden sind. Für welches Werkzeug man sich im Endeffekt entscheidet, hängt wesentlich von der benötigten Sicherheit und einer Abwägung der Kosten und Nutzen ab.

Linux-Backuptools

- `tar`
  - Backup individueller Files
  - weit verbreitet (Unix-Standard)
  - Datentransfer zwischen Plattformen
- `cpio`
  - Backup individueller Files
  - weit verbreitet
  - Probleme mit Symlinks
- `dump`
  - Backup ganzer Filesysteme
  - inkrementelle Backups

Folie 94

tar

- traditionelles UNIX Tape-Archiv Kommando
- Basis-Syntax
  - backup: `tar cvf tarfile.tar directory_or_file_to_tar`
  - restore: `tar xvf tarfile.tar`
  - listen: `tar tvf tarfile.tar`
  - zusätzlich komprimieren: `tar xvzf tarfile.tar`
  - komprimieren mit `bzip2`: `tar xvjf tarfile.tar`
  - Multivolume: `tar cvMf /dev/fd0 files_to_tar`
    keine Komprimierung, alternativ: `split`

Folie 95

cpio

- Unix-Standard
- backup: `cpio -ov files > device`
  `find /home | cpio -ov > /dev/fd0`
- restore: `cpio -iv[-dum] [files] < device`
  `cpio -ivdum "/home/j*≪ /dev/fd0`
- listen: `cpio -itv < device`

Folie 96

dump

- Backup kompletter Filesysteme
- auch Symlinks und spezielle Files (`/dev`)
- inkrementelles Backup bis zu 9 Stufen
- backup: `dump -0 -a -u -f /dev/fd0 /home`
- inkrementell: `dump -5 -a -u -f /dev/fd0 /home`
- restore: `cd /home; restore -r -f /dev/fd0`

Folie 97

Weitere Tools und Tipps

- Taper: menügesteuertes Band-Backup
- BRU2000: `http://www.bru.com`
- Lone-Tar: `http://www.cactus.com`
- ADSL/TSL: Netzwerkbackup, IBM
- `http://www.linux-backup.net`
- Linux Backup HOWTO
  `http://www.biochemistry.unimelb.edu.au/pscotney/backup/Backup-HOWTO.html`

Folie 98

10 Kernel compilieren

Hier erklären wir warum es manchmal sinnvoll ist, selbst einen Kernel zu compilieren. Wir gehen auf die Verwendung von Kernel-Modulen ein und erklären kurz die wichtigsten Schritte beim Konfigurieren und Compilieren eines Linux-Kernels.

Der Linux-Kernel ist nichts anderes als ein ca. 1 MB großer Code, der vom Bootmanager (zB `lilo`, `grub`) in den Speicher geladen und ausgeführt wird und die Grundfunktionen des Betriebssystems implementiert (vgl. auch Teil 1, Kapitel 2).
Nach wie vor koordiniert *Linus Torvalds* das riesige Projekt der Weiterentwicklung des Linux-Kernels: Neben Bugfixes für die aktuelle Version wird gleichzeitig auch an sog. Entwicklerkernels getüftelt, in denen neue Konzepte entwickelt und getestet werden und die daher nicht (so) stabil sind.

Da ein Kernel einen möglichst großen Leistungsumfang bieten und sich schnell an neue Hardware und an neue Problemstellungen anpassen lassen soll, kann man ihn nicht monolithisch (als einen großen Programmklotz) programmieren, sondern muss die verschiedenen Aufgaben klar in weitgehend unabhängige Teile trennen. Durch diese *Modularisierung* des Quellcodes können viele Leute gleichzeitig an der Kernelentwicklung mitarbeiten. zB ist ein Festplattentreiber (zB IDE oder SCSI) völlig unabhängig von einem Dateisystemtreiber (zB `ext3` oder ReiserFS), solange ein vereinbartes Programmierinterface benutzt wird; daher kann man schnell und einfach zB Treiber für neue Dateisystemformate programmieren, ohne Änderungen in anderen Quellcodeteilen vornehmen zu müssen. Werden gewisse Treiber, Protokolle, etc. überhaupt nicht benötigt oder sollen sie nicht benutzt werden (Performance- und Sicherheitsaspekt), lässt man sie entweder beim Compilieren des Kernels weg oder lädt und entlädt die entsprechenden *Module* bei Bedarf (im laufenden Betrieb): dies lässt sich auch automatisieren; der Kernel lädt zB die Module für die Floppyhardware und das DOS-Dateisystem erst, wenn eine DOS-Diskette gemountet wird, und entlädt sie selbständig kurz nach dem Unmounten (Details in Abschnitt 10.1).

Ein weiterer Vorteil der Modularisierung ist, dass sich Linux relativ rasch auf andere Hardware-Architekturen übertragen lässt: Die wenigen Prozeduren, die stark von CPU, Motherboard, Bus-System, etc. abhängen, sind in der Assemblersprache des jeweiligen Systems maßgeschneidert programmiert, sind klein und übernehmen nur ganz elementare Aufgaben (in Protected Mode umschalten, Festplattensektor lesen, ...). Der andere Teil des Quellcodes ist hardwareunabhängig, in C geschrieben und implementiert die ganze Funktionalität des Kernels (Netzwerkprotokolle, Dateisysteme, ...) und schließlich die Funktionen, die die Anwendungsprogramme aufrufen dürfen (sog. System Calls; zB Lese/schreibe Datei, ...). Jener Teil kann also ohne wesentliche Änderungen auf ein neues System übernommen werden (Recycling), allerdings braucht man dazu zuerst einen C-Compiler (den man aber ohnehin benötigt, um Anwendungsprogramme zu compilieren).

Die *Nachteile* dieses Konzepts der Linux-Kernelprogrammierung liegen auf der Hand: Eine perfekt funktionierende C-Compilermaschinerie, die optimierten Code für eine neue Hardware-Architektur erzeugen soll, ist ein aufwändigeres Projekt als der Kernel selbst.
Bei der Programmierung des Kernels muss man sich strengstens an gewisse Konzeptionen und Spielregeln halten, da sonst seine Stabilität leidet und der Quellcode (noch) unübersichtlicher. Schließlich bringt die Vielfalt der Konfigurationsmöglichkeiten des Kernels und der Treiber einen „Overhead“ in Form eines nicht unwesentlichen Verwaltungsaufwands einerseits *im* Quellcode andererseits des Quellcodes *selbst* mit sich.

Kernel-Versionsnummern

`uname -r` und `dmesg | head -1` zeigen die Versionsnummer des laufenden Kernels an. `2.4.9-12` bedeutet zB, dass wir mit der *Version* 2, *Patchlevel* `4` und *Sublevel* `9` arbeiten. Die gerade Patchlevel-Nummer (`4`) zeigt an, dass dies ein stabiler Kernel (auch: *Produktionskernel*) ist; eine ungerade steht für Entwicklerkernel, die instabil sind. Manche Distributoren ändern einen Kernel leicht ab und hängen eine Zahl oder Zeichenkette an, (hier die RedHat-Releasenummer `12`). Innerhalb der stabilen Versionen wird nur dann ein neuer Kernel veröffentlicht und das Sublevel weitergezählt, wenn Fehler im Code ausgebessert („Patches“ herausgegeben) werden (beachte die nicht ganz schlüssige Benennung des Patchlevels).

Grundsätzlich Neues (wie Restrukturierung des Kernelaufbaus; Hardwareunterstützung, Protokolle, ...) wird in Entwicklerkernels ausprobiert und getestet und erst beim nächsten Versionssprung „freigegeben“, d.h. in einem Kernel mit geradem Patchlevel veröffentlicht.

Heutiger Stand (2002-03-13) ist die Version `2.4`, die aus der Entwicklerversion `2.3` entstanden ist. Das aktuelle Sublevel ist `18` und somit hat der neueste Produktionskernel die Versionsnummer `2.4.18`; der neueste RedHat-7.2-Kernel hat die Versionsnummer `2.4.9-31`.

Derzeit läuft dazu parallel die Entwicklung der Version 2.5, aus der später die stabile Version 2.6 werden soll. Der neueste Entwicklerkernel ist `2.5.7`.

Kernel-Versionen

```
[oli@sqirrel ~] uname -r 2.4.10-4GB
[oli@sqirrel ~] dmesg | head -1
  Linux version 2.4.10-4GB (gcc version 2.95.3)
  #1 Tue Sep 25 12:33:54 GMT 2001
```

- Version: 2
- Patchlevel: 4; (un)gerade bedeutet (in)stabil
- Sublevel: 10; (Bugfixes innerhalb 2.4)
- Release: SuSE-Kernel mit Unterstützung für 4 GB RAM

Folie 99

10.1 Kernel-Module

Der Linux-Kernel bietet die Möglichkeit, viele Treiber als sogenannte Module zu verwenden. Diese Treiber sind dann nicht fix in den Kernel hineincompiliert, sondern können im laufenden Betrieb geladen werden. Prominente Beispiele hierfür sind Netzwerkkartentreiber und Soundkartentreiber (vgl. auch oben). Die Module befinden sich in `/lib/modules/kernel-versionsnr/` und enden auf `.o` (Objektdatei, i.e., eine Sammlung compilierter C-Funktionen).
Die Vorteile beim Verwenden von Modulen sind, dass der Kernel weniger Speicher benötigt und schneller bootet. Der Nachteil dabei ist allerdings, dass das Laden der Module selbst eine gewisse Zeit in Anspruch nimmt. Außerdem sollten Treiber nicht als Module verwendet werden, die beim Booten benötigt werden. Es gibt zwar die Möglichkeit, sogenannte *Initial Ramdisks* (Initramdisks) zu verwenden und etwa den SCSI-Treiber modular zu verwenden, obwohl man von einer SCSI-Platte bootet. Wir gehen hier darauf aber nicht ein; siehe zB `http://sdb.suse.de/en/sdb/html/initrd.html` und `man mkinitrd`.
Die Module eines Kernels können nicht mit einem anderen Kernel verwendet werden, selbst bei gleicher Versionsnummer und selbem Quellcode. Wenn man einen Kernel installiert/compiliert, der die gleiche Versionsnummer wie der laufende hat, muss man selbst ein Backup der Module machen und selbst dafür sorgen (zB Symlink auf Backup), dass der Kernel die richtigen Module lädt.

Im Folgenden stellen wir die Werkzeuge vor, mit denen Module gehandelt werden; im Wesentlichen sind das die die Programme `lsmod`, `modprobe`, `insmod`, `rmmod`, `depmod` in `/sbin/` und die Datei `/etc/modules.conf` (siehe auch Folie 101).
Mit `lsmod` kann man die gerade in Verwendung befindlichen Module auflisten. `depmod` bestimmt die Modulabhängigkeiten und schreibt diese in die Datei `/lib/modules/kernel-versionsnr/modules.dep`. Üblicherweise wird dieser Befehl nach dem Booten von `/etc/rc.sysinit` ausgeführt.

Kernel-Module

- Teile (Treiber) des Kernels als Module
- können im laufenden Betrieb geladen werden
- Vorteile: kleinerer Kernel, schnelleres Booten
- Nachteil: Laden braucht Zeit
- Nur für Hardware, die nicht beim Booten benötigt wird

Folie 100

Umgehen mit Modulen

- Listen: `lsmod`
- Laden: `insmod` (einfach), `modprobe` (clever)
- Entfernen: `rmmod` (nur wenn unbenutzt)
- Weiters: `depmod`, `modinfo`
- Automatisch: `/etc/modules.conf`

Folie 101

Um Module in den Kernel zu laden hat man zwei Alternativen: `insmod` ist einfach und brutal, `modprobe` intelligenter; es prüft Modulabhängigkeiten und lädt vorher alle benötigten Module wobei die Information aus `/etc/modules.conf` bezogen wird. `insmod` reagiert in so einem Fall mit einer Fehlermeldung; man muss hier also wissen, in welcher Reihenfolge die Module zu laden sind. Beiden Befehlen kann man Parameter übergeben, z.B. IRQ oder IO-Port. Üblicherweise finden die Module die notwendigen Einstellungen allerdings durch Probieren (*Autoprobe*) selbst heraus; Hat man den Kernel-Quellcode unter `/usr/src/linux` installiert, findet man dort in `Documentation/` detaillierte Informationen zu Modulparametern. Eine störrische Sound-Blaster-Karte kann vielleicht mit

```
modprobe sb io=0x220 irq=5 dma=1 dma16=5
```

zum Arbeiten überredet werden; stimmen die Optionen nicht, kann sich das System aber auch aufhängen.
Module aus dem Kernel entfernen kann man nur, wenn sie nicht benutzt werden und zwar mit `rmmod`; ggf. muss man zuerst davon abhängige Module entfernen, Medien unmounten oder Programme schließen.

Wirklich bequem wird das Verwenden von Modulen durch die Automatisierung des Ladevorgangs (und des Entfernens). Dazu muss die Datei `/etc/modules.conf` (oder je nach Distribution auch `/etc/conf.modules`) angelegt werden, in der angegeben wird, welches Modul (eventuell mit welchen Parametern) für welches Hardwaregerät zu verwenden ist. Üblicherweise wird diese Datei bereits vom Installer richtig geschrieben, und braucht nur im Falle, dass man eine neue Hardwarekomponente einbaut verändert zu werden. Eine typische `/etc/modules.conf`-Datei befindet sich auf Folie 102.

/etc/modules.conf

```
alias eth0 rtl8139
alias eth1 3c59x

alias scsi_hostadapter aic7xxx
alias parport_lowlevel parport_pc

alias sound-slot-0 sb
options sb io=0x240 irq=7 dma=0

alias midi opl3
options opl3 io=0x388
```

Folie 102

10.2 Kernel compilieren – Warum?

Auf jedem funktionierenden Linux-System ist natürlich (mindestens) ein Kernel vorhanden. Warum sollte man dann einen neuen Kernel compilieren (außer aus Faszination an der Sache selbst)?
Nun, dafür gibt es mindestens drei gute Gründe: der verwendete Kernel hat entweder zu wenig oder zu viel Hardwareunterstützung, oder es muss auf einen neuen Kernel-Release upgegraded werden (und es steht noch kein neues, bzw. adäquates Kernel-Paket der entsprechenden Distribution zur Verfügung).
Der erste Grund tritt ein, wenn etwa am System ein Hardwaregerät vorhanden ist, das vom gegenwärtig verwendeten Kernel nicht oder nicht optimal unterstützt ist.
Andererseits haben die Standardkernel der einzelnen Distributionen viele Treiber fix eincompiliert, die man üblicherweise nicht benötigt. Dann kann durch Herstellen eines auf die spezifische Hardwarekonfiguration des Systems getrimmten Kernels die Performance verbessert werden. Insbesondere belegt ein kleinerer Kernel, der keine unnötigen Treiber beinhaltet, weniger Speicher und bootet schneller. Manchmal kann es auch wichtig sein, wegen verbesserter Features (neue Treiber, neu unterstützte Hardwarekomponenten) oder ausgebesserter Bugs (vor allem, wenn sicherheitsrelevant) auf einen neuen Kernel-Release upzugraden. Das kann auch – falls vorhanden – mittels `rpm`-Pakets geschehen, was einem das Compilieren erspart; Kernel RPMS sollten aber niemals mit den Upgradeoptionen `-U` oder `-F` installiert werden, da sonst der alte Kernel und seine Module überschrieben werden. Das kann dazu führen dass der laufende Kernel nicht mehr richtig funktioniert, weil er keine brauchbaren Module mehr findet. Außerdem wird der Bootloader nicht entsprechend upgedatet. Das muss man zwar auch dann händisch machen, wenn man `rpm -i` verwendet, allerdings bleibt der alte Kernel funktionstüchtig und kann auch später noch gebootet werden („beliebte"' Fallgrube bei automatischen Updates). Für weitere Detials siehe zB `http://www.redhat.com/support/resources/howto/kernel-upgrade`.

Kernel compilieren – Warum?

- Standard-Kernel nicht adäquat, weil
  - vorhandene Hardware nicht unterstützt
  - zu viel nicht benötigte Hardwareunterstützung
    - Speicherverbrauch
    - Booten dauert länger
- Upgrade auf neuere Version
  - neue Hardwareunterstützung
  - neue Protokolle, Dateisysteme, ...
  - Bugs, Sicherheistlücken
- Rechner-Spezialeinsatz zB: Nur-Firewall; alter Rechner mit wenig RAM
- Experimentellen Kernel verwenden ?!? Spaß !?!

Folie 103

10.3 Kernel-Quellcode

Für den Fall, dass nur neue Hardwareunterstützung ins System integriert werden soll, können die Kernel-Quellcode Pakete der Distribution verwendet werden (z.B. `kernel-source-2.2.17-14.i386.rpm`). Achtung, diese dürfen weder mit den Kernel RPMS selbst (z.B. `kernel-2.2.17-14.i586.rpm`), noch mit den Kernel SRPMS (z.B. `kernel-2.2.17-14.src.rpm`) verwechselt werden. Letztere enthalten die Quellen um das Kernel RPM und das Kernel-Source RPM (mittels `rpm`) neu zu bauen. Die Kernel-Quellcode Pakete sind allerdings meist nur für den Standardkernel und eventuell ein bis zwei Upgrades für den entsprechenden Release der Distribution verfügbar.

Will man eine dadurch nicht abgedeckte Kernel-Version verwenden, so muss man die Kernel-Source selber aus dem Internet herunterladen; diese findet man auf `http://www.kernel.org` resp. auf dem Mirror `http://www.kernel.at` oder auf der Originalsite `ftp://ftp.funet.fi/pub/Linux/PEOPLE/Linus`. Im Directory für jedes Patchlevel befindet sich jeweils eine Datei namens `LATEST-IS-x.y.z`, die auf die neueste Kernel-Version verweist. Man kann die gesamte Kernel-Source im `gz`- oder `bz2`-Format, wobei letzteres stärker komprimiert ist, herunterladen. Die Größe von z.B. `linux-2.4.3.tar.bz2` ist 19.9 MB. Um nicht den gesamten Quellcode herunterladen zu müssen, wenn man bereits einen älteren hat, kann man Patches verwenden; diese enthalten nur die Änderungen seit der letzten Version und können zB mittels des Kommandos `zcat patch-2.4.3.gz | patch -p0` eingespielt werden; dadurch wird der 2.4.2er Sourcetree in einen 2.4.3er verwandelt. `patch-2.4.3.bz2` hat z.B: nur 1 MB. Will man allerdings von einem 2.4.1er Kernel auf einen 2.4.3er upgraden, braucht man beide Patches, d.h. `patch-2.4.2.bz2` und `patch-2.4.3.bz2`.

Kernel-Quellen

- Kernel RPMS
  nur installieren, nicht compilieren
  z.B. `kernel-2.2.17-14.i686.rpm`
  NUR: `rpm -i`, NIEMALS: `-F|U`
- Kernel-Source RPMS
  installieren dann compilieren
  z.B. `kernel-source-2.2.17-14.i386.rpm`
- NICHT: Kernel SRPMS
  Quelle zum Neupaken der RPMS mit `rpm`
  z.b. `kernel-2.2.17-14.src.rpm`
- Kernel-Quellcode Tarballs
  als `tar.gz` oder `tar.bz`
  - `http://www.kernel.org`
  - `http://www.kernel.at`
  - `ftp://ftp.kernel.at`
  - `ftp://ftp.funet.fi/pub/Linux/PEOPLE/Linus`

Folie 104

Schließlich kann man noch mittels der ebenfalls im Download Directory befindlichen Signatur (z.B. `linux-2.4.2.tar.bz2.sign`) die Authentizität des Codes prüfen. Dazu muss man von `http://www.kernel.org/signature.html` den Kernel PGP Public Key herunterladen und mittels `gpg --import` in den Keyring aufnehmen. Sodann verifiziert man das `tar.gz`-file etwa mit `gpg --verify linux-2.3.9.tar.gz.sign linux-2.3.9.tar.gz`.

Achtung jedes `tar.gz` oder `tar.bz2` Kernel-Quellcode File entpackt sich nach `/usr/src/linux`, sodass man aufpassen muss, einen eventuellen vorhandenen älteren Sourcetree nicht zu überschreiben. Man kann nach dem Entpacken den Sourcetree auf `/usr/src/linux-versionsnr` umbenennen; dann sollte man aber immer in `/usr/src/` einen Link `linux` anlegen, der auf das aktuellen Quellcodeverzeichnis zeigt.

10.4 Kernel compilieren – Eine Kurzanleitung

Die folgende Anleitung ist als Kochrezept zu verstehen, wo man die eine oder andere Zutat hinzugeben oder weglassen muss, da sich von Version zu Version Dinge ändern können. Folgende Pakete müssen installiert (und upgedatet) sein um den Kernel erfolgreich compilieren zu können; meist sind diese bei der Systeminstallation in der Gruppe Development und/oder Kernel-Development zu finden.

- Gnu Compiler Collection (`gcc`)
- `binutils`, `modutils`, `make`

Führt man die Konfiguration unter der Konsole durch:

- `ncurses`, `ncurses-devel`[opement]

Für X-Window-Konfiguration:

- `Tcl/Tk`

Viele der folgenden Schritte (Ausnahme: Konfiguration und Compilierung) brauchen `root`-Rechte. Wenn man nicht paranoid ist, führt man der Einfachheit halber alles als `root` aus.
Bevor es richtig losgeht, macht man Backups von `/lib/modules/` und ggf. `/usr/src/linux/`; weiters informiert man sich möglichst genau über die Hardware des Computers. Anhaltspunkte sind `/proc/cpuinfo`, `/proc` allgemein, das Bios-Menü, etc.

Sodann wechselt man ins Kernel-Sourcedirectory `/usr/src/linux/` und führt der Reihe nach folgende Schritte durch.

1. Die *Kernelkonfiguration* erfolgt über einen der drei Befehle `make config` (Frage Antwort-Spiel), `make menuconfig` (menügesteuert) oder `make xconfig` (GUI). Der längeren Beschreibung der Kernel-Optionen und -auswahlmöglichkeiten ist der ganze Abschnitt 10.5 gewidmet. Hat man die Konfiguration abgeschlossen, sichert man sicherheitshalber die Konfigurationsdatei `.config` per Menü oder per `cp` zB nach `/root/kernel.config`

2. Zum *Compilieren des Kernels und der Module* müssen der Reihe nach folgende Befehle ausgeführt werden: `make dep, (make clean), make bzImage, make modules, make modules_install`. Dadurch wird der Kernel compiliert, die neuen Module compiliert und installiert. Das neue Kernel Image liegt dann in `usr/src/linux/arch/i386/boot/bzImage`. Mittels

   ```
   $ make clean dep bzImage modules modules_install \
     2>&1 | tee mylogfile
   ```

   erfolgt alles in einem Aufwaschen und wrnungen und fehlermeldungen werden in `mylogfile` gespeichert. Der gesamte Compiliervorgang kann je nach Hardware und Konfiguration einige Minuten oder (mehr als) eine Stunde dauern.

3. Um den Kernel zu *installieren* muss er nach `/boot/` kopiert werden; zB mit

   ```
   $ cp /usr/src/linux/arch/i386/boot/bzImage \
        /boot/meinvmlinuz
   ```

   und der Bootloader `lilo` für das neue Image eingerichtet werden; als Vorlage kann man die bestehende Konfiguration nehmen, wobei man unbedingt die alte Konfiguration beibehalten und die neue (als alternatives Image) hinzufügen sollte – für den Fall, dass der neue Kernel nicht richtig funktioniert kann man so immer noch den alten verwenden, um einen weiteren Compilierversuch durchzuführen. Um den Bootsektor neu einzurichten, ruft man schließlich `lilo` auf.

Folgender Schritt ist noch empfehlenswert, um harmlose Fehlermeldungen des neuen Kernels zu eliminieren, aber nicht unbedingt notwendig:

```
$ cp /usr/src/linux/System.map \
     /boot/meineSystem.map
$ mv /boot/System.map /boot/System.map.old
$ ln -s /boot/meineSystem.map System.map
```

4. Rebooten und die Bootmessages sehr genau beobachten (und die Daumen drücken).

Troubleshooting: Treten beim Compilieren Probleme auf, sollte man seine Konfiguration auf Widersprüchlichkeiten überprüfen; manchmal ist es auch sinnvoll zu testen, ob sich die benutzte Version des `gcc` bzw. die der `binutils` mit der Kernel-Version verträgt (siehe Internet).

Die Fehlersuche, wenn sich der neue Kernel anders verhält als erwartet, beginnt mit einer Inspektion der `/var/log/messages`. Hier sollte man vergleichen, welche Meldungen der vorher installierte Kernel ausgegeben hat. Die Dateien und Unterverzeichnisse von `/proc/` geben sehr detailliert Auskunft darüber, welche Teile des Kernels funktionieren. Auf Informationen über Bugfixes, etc. sei wegen der Aktualität generell auf das Internet verwiesen.

Kernel compilieren

```
cd /usr/src/linux

make xconfig
   (auch: make menuconfig, make config)
make dep
(make clean)
make bzImage
   (auch: make zImage)
make modules
make modules_install

vi /etc/lilo.conf
lilo
reboot
```

Folie 105

10.5 Kernel-Konfiguration – Einige Details

Da sich die Optionen bei der Kernelkonfiguration ständig ändern, ist eine vollständige Aufzählung aller Konfigurationsmöglichkeiten hier nicht sinnvoll. Für Detailfragen sei auf `/usr/src/linux/Documentation/`, die Online-Dokumentation während der Kernelkonfiguration und das Kernel-HOWTO verwiesen.
Generell steht „y/∗“ für „fix in den Kernel einbinden“, „m“ für „als Modul“, „n“ für „Unterstützung weglassen“. Zu beachten ist, dass sich einige Punkte erst dann „aufklappen“, wenn man frühere Fragen mit „y“ oder „m“ beantwortet hat.

Auf folgende Fragen sollte man **yes** sagen, wenn man kein sehr außergewöhnliches System hat und wenn man (noch) nicht so genau weiß, was man tut.
Code maturity level options: *Enable loadable module support*, *Kernel module loader*; **nein** bei *Prompt for development and/or incomplete code/drivers*;
Für **Processor type and features** befragt man das BIOS oder `/proc/cpuinfo`. Nur Besitzer von Multiprozessor-Systemen nehmen *Symmetric multi-processing support*.
Bei **General Setup** muss man sich Zeit nehmen und genau durchlesen. *Networking Support*, *PCI support*, *System V lpc*, *BSD Process Accounting*, Sysctl support, *Kernel support for Elf binaries*. Notebook-Besitzer wählen üblicherweise PCMCIA-Unterstützung.
Parallel port support: *PC-style hardware*.
Plug and Play support: *ISA Plug and Play support* (zumindest schon wegen einer Soundkarte).
Block devices: *Normal PC floppy disk support*, *Loopback device support*.
Networking options: Vorsicht: Auch auf Rechnern ohne Netzwerkkarte verwenden X Window und andere Programme intern TCP/IP!!! Unbedingt: *Packet socket*, *Network Packet Filtering*, *Unix domain sockets*, *TCP/IP networking*, *Advanced Router*, *Verbose route monitoring*, *TCP/IP syncookie support*, *Ip Multicasting*;
bei *Netfilter Configuration* (Firewalling) unbedingt: *Connection tracking*, Unterstützung für *FTP protocol*, *Iptables*, *MAC address match*, *Connection state match*, *Owner match*, *Packet*

filtering, *Reject target*, *Full Nat*, Masquerade target, Redirect target, *Log target* und weitere Optionen bei Bedarf. Unbedingt Fast Switching abschalten. Vorsicht: Auch auf Rechnern ohne Netzwerkkarte verwenden X Window und andere Programme intern TCP/IP!
Auf **Ata/Ide/Mfm/Rll support** dürfen bloß die Nur-SCSI-Besitzer verzichten. IDE-Besitzer aktivieren hingegen den Support/die Option für *Enhanced IDE/MFM/RLL disk/...*, IDE/ATA-2 Disk, IDE/ATAPI CDROM, *Generic PCI IDE chipset*, *Sharing PCI IDE interrupts*, *Generic PCI bus-master DMA*, *Use PCI DMA by default*.
Ggf. **Scsi**-Unterstützung. Achtung: Wer einen IDE-Brenner hat, muss hier „y“ sagen (siehe unten).
Ggf. **Network device support** für seine Netzwerkkarte. Für Modem-Verbindungen aktiviert man PPP.
In **Character Devices** zu *Virtual terminal*, *Support for console on virtual terminal*, *Unix98 PTY support* sowie ggf. *Parallel printer support*. Auch eine nicht-serielle Maus lässt sich hier konfigurieren. Der *Direct Rendering Manager* unter XFree86 $\geq$4.1 für eine gehobenere Grafikkarte lässt sich hier einstellen.
In **File systems** hat man unbedingt das Dateisystem der `/`-Partition fix einzubinden (und nicht als Modul); für DOS-Disketten, Win-Partitionen bzw. CDROMs sind VFAT bzw. *iso9660* auszuwählen.
Bei **Sound** nehmen Sound-Blaster-64-Besitzer *Creative Ensoniq AudioPCI 97*; weiters sollte man neben den Alsa-Treibern auch die (proprietären) OSS-Treiber enablen.
Bei **Usb** ist i.a. (nur) der UHCI-Treiber zu aktivieren; in Zukunft sind hier sicher Änderungen zu erwarten.

Hier erwähnen wir einige **speziellere Details**, die weniger oft relevant sind.

Kerneloptionen: Ähnlich wie man jedem Programm auf der Kommandozeile Optionen übergeben kann, kann man dies auch für den Kernel in Form eines Eintrags in `/etc/lilo.conf` per `append="`*Option*`"`. Diese „Kernelkommandozeile“ sieht man in `/proc/cmdline`. Da man diese Optionen üblicherweise nur benutzt, wenn spezielle Hardware beim Booten konfiguriert werden muss, sei dazu auf die Kernel-Dokumentation verwiesen.

CD-Brenner: SCSI-Brenner werden wie ganz normale SCSI-Geräte unterstützt. Da es bis jetzt noch keine echten Treiber für IDE-Brenner gibt, muss man dafür die SCSI-Emulation einschalten:
Man wähle *SCSI support*, *SCSI CDROM support*, *SCSI generic support* und füge in `/etc/lilo.conf` bei dem Boot-Eintrag für Linux hinzu: `append="hdx=scsi"`, wobei *hdx* für den IDE-Namen des Brenners steht, zB `hdc`. Sollte es dennoch Probleme geben, streicht man den *IDE CDROM support*. In `/etc/fstab` sollte man berücksichtigen, dass der Brenner jetzt mit `/dev/sr0` o.ä., angesprochen wird. `cdrecord -scanbus` zeigt, ob man erfolgreich war. Für weitere Details siehe das augezeichnete CD-Writing-HOWTO.

11 Shell Scripts (Bash)

In diesem Kapitel geben wir eine kleine Einführung in die (Bash) Shell Programmierung. Unterwegs wiederholen und erweitern wir viele Konzepte aus Teil 1, Kapitel 6.

11.1 Wozu Shell Scripts?

Shell Scripts bieten die Möglichkeit schnell und einfach Aufgaben unter Unix zu automatisieren; eine Shell kann als Programmierinterface verwendet werden. Jedoch unterscheiden sich Shell Scripts von Programmen in Sprachen wie C beträchtlich. Im Shell Script sind alle Systemprogramme direkt verwendbar. Es gibt nur einen Typ von Variablen; diese müssen nicht initialisiert werden, etc. Statt einer langen Liste weiterer Unterschiede sei hier nur Folgendes gesagt: Shell Scripts eignen sich besonders gut um Aufgabe „quick and dirty“ zu lösen. Ein Systemadministrator, der vor einer größeren Aufgabe steht – etwa der Erzeugung von 100 durchnummerierten Accounts – wird in der Regel versuchen, dies mit einem Script zu lösen. Wird die Aufgabe größer, so gehen die Überlegungen in Richtung Perl und letztlich zu einer Compilersprache. Shell Scripts sollte man nicht verwenden, wenn

- die Geschwindigkeit des Programms eine große Rolle spielt.
- komplizierte Aufgaben gelöst werden sollen, die ein stark strukturiertes Programmieren erfordern.
- Sicherheitsmechanismen verwendet werden sollen.
- viele Dateien gelesen und geschrieben werden sollen.
- man ein GUI haben will – GUI's are for wimps anyway.

Wozu Shell Scripts?

- Automatisierung von Aufgaben
- „quick and dirty“
- **nicht** wenn:
  - Geschwindigkeit wichtig
  - Sicherheitsmechanismen wichtig
  - hohe Komplexität
  - GUI
- Alternativen:
  - Perl
  - Compilersprachen

Was ist ein Shell Script?

Ein Shell Script ist eine Textdatei bestehend aus

- Programm- und Funktionsaufrufen
- Shell internen Kontrollstrukturen
- Variablenzuweisungen

Das Schreiben von Shell Scripts verlangt Wissen über

- shell-interne Kontrollstrukturen
- möglichst viele Unix Programme und
- deren Optionen

Folie 107

Verschiedene Shells

sh	Bourne shell, die erste Unix-Shell von S. R. Bourne
bash	Bourne Again Shell, Standard auf Linux, von der FSF
csh	Berkeley Unix C Shell, C-ähnliche Syntax, Standard auf BSD
tcsh	TENEX C Shell, erweiterte csh
ksh	Korn Shell, proprietäre Shell von David Korn
pdksh	Public Domain Korn Shell
rc	Shell für Plan 9-OS
es	Erweiterbare Shell auf Basis von rc
zsh	Z Shell von Paul Falstad
ash	kleine, POSIX konforme Shell, /bin/sh auf NetBSD
esh	Easy Shell, kleine, leicht bedienbare Shell
kiss	Karel´s Interactive Simple Shell
lsh	Shell mit DOS Kommandos für Umsteiger
osh	Operator´s Shell, mit erweiterten Sicherheitsmechanismen
sash	Standalone Shell, braucht keine Libraries
psh	Perl Shell, Perl Syntax

Die Shell unserer Wahl

Shells gibt es sehr viele. Die Aufstellung auf Folie 108 erhebt keinerlei Anspruch auf Vollständigkeit. Die Bourne Shell (*sh*), benannt nach ihrem Erfinder, ist die Mutter aller Shells auf UnixSystemen. Die Bourne Shell bot schon zu Anfang fast alle Möglichkeiten der Programmierung, die die *bash* heute bietet; die interaktive Bedienung war allerding nicht sehr bequem. Dieser Umstand führte zur Geburt der *csh*. *csh*-Scripts folgen einer anderen Syntax, die der Programmiersprache C ähnlich ist. Ein weiterer Meilenstein in der Geschichte der Shells war die Korn Shell *ksh*, ebenfalls nach ihrem Erfinder benannt. Diese war ein sehr erfolgreiches proprietäres Produkt. Die *bash* (Bourne Again Shell) ist der Versuch einer Zusammenmischung der Vorzüge verschiedener Shells, wobei die Programmierung in einer erweiterten *sh*-Syntax erfolgt.
Im weiteren werden wir unser Hauptaugenmerk auf die *bash* legen, die heute wohl die meistverwendete Shell unter Linux und vielleicht auch Unix ist. Die meisten unserer Konstruktionen werden allerdings ohne weiteres auch mit einer *sh* funktionieren.

Ein Shell Script schreiben

Ein Shell Script ist eine Textdatei, die Kommandos enthält. Welchen Texteditor man zur Erzeugung dieser Datei verwendet ist egal, aber... `http://www.thinkgeek.com/images/products/zoom/vi-emacs.jpg`.

Beginnt diese Textdatei mit der Zeichenfolge
`#!/bin/bash`
und wird die Datei ausführbar gemacht
`$ chmod 755 scriptname`
so kann das Script direkt mit seinem Namen aufgerufen werden.
`$ ./scriptname`
Das Script wird dann Zeile für Zeile, vom Interpretor, in unserem Fall der Bash, abgearbeitet. Im Unterschied dazu müssen Programme, die in Compilersprachen geschrieben sind, erst compiliert werden, bevor sie ausgeführt werden können.
Wie in allen Programmiersprachen ist das Kommentieren eines Scripts ein oft ignoriertes Merkmal guten Programmierstils. Zeilen die mit `#` beginnen gelten als Kommentare und werden ignoriert. Unser erstes Shellscript befindet sich auf Folie 110.

Die Login Shell

Nach dem Login landet man auf Unix-Systemen in einer Shell. Welche Shells auf einem System als mögliche Login Shell installiert sind steht in der Datei `/etc/shells`.

```
$ cat /etc/shells
```

Die Login Shell jedes Users steht in der Datei `/etc/passwd`.

```
$ grep $USER /etc/passwd | cut -d : -f 7
```

ändern kann man seine Login Shell mit dem Befehl `chsh` (aka change shell). Die Standard Login Shell auf Linux-Systemen ist die **bash**.

Folie 109

Hello World

Das obligate Hello World Programm ist als Shell Script denkbar einfach.
Folgender Text soll in der Datei `helloworld` gespeichert sein.

```
#!/bin/bash
echo "hello world"
```

Ist es ein Shell Script?

```
$ file helloworld
helloworld: Bourne-Again shell
  script text executable
```

Ausführbar machen mit
`$ chmod 755 helloworld`
und ausführen mit

```
$ ./helloworld
hello world
```

Folie 110

Kommentare & Interpreter

- Kommentare: Zeilen, die mit `#` beginnen werden beim Ausführen ignoriert.

```
# Kommentare koennen helfen die
# Lesbarkeit von Shell Scripts
# wesentlich zu erhoehen.
```

- Interpreter: Angabe des Interpreters am Anfang des Scripts durch `#!`
  macht Scripts direkt ausführbar und gilt für beliebige Interpretersprachen.
  - `#!/bin/bash`
  - `#!/bin/csh`
  - `#!/bin/sh`
  - `#!/usr/bin/perl`

Folie 111

11.2 Variablen

Variablenname bestehen aus Buchstaben, dem Underscore (_) und Zahlen, wobei oft auf Kleinbuchstaben verzichtet wird. Die Zuweisung eines Wertes funktioniert folgendermaßen:

```
$ VARIABLENNAME=wert
```

Auf den Wert einer Variable greift man mit `$VARIABLENNAME` zu:

```
$ echo $VARIABLENNAME
```

`unset` nimmt einer Variable ihren Wert:

```
$ unset VARIABLENNAME
```

In Shell Scripts gibt es nur einen Variablentyp. Man unterscheidet also nicht wie in anderen Programmiersprachen Integer-, Gleitkomma- und Stringvariablen. Wenn ein Script davon abhängt, dass in einer Variable ein Integerwert steht, so ist der Autor dafür verantwortlich, dass dem auch so ist. Variablen müssen nicht initialisiert werden. Dies bereitet oft Probleme bei Tippfehlern. Folgendes Beispiel führt zu keiner Fehlermeldung:

```
$ TEST=Legasthenie
$ echo $TSET

$
```

Der Befehl `set` gibt eine Liste der von der Bash gesetzten Variablen. Neben anderen befindet sich darunter die Variable `PS1` die das Aussehen des Prompts bestimmt. MS-DOS Nostalgiker könnten sich über Folgendes freuen:

```
$ PS1="C:\> "
C:\>
```

In Scripts sind vor allem die Variablen `$1-$9` von großer Bedeutung. Diese enthalten die an das Script übergebenen Parameter (auch *Positionsparameter*). Dazu folgendes Beispiel, das in der ausführbaren Datei *parameters* gespeichert sein soll:

```
#!/bin/bash/


# Ausgabe des ersten Parameters
echo "Erster Parameter:  $1"
# Ausgabe des zweiten Parameters
echo "Zweiter Parameter: $2"
```

Und das kommt dabei heraus:

```
$ ./parameters foo bar
Erster Parameter:  foo
Zweiter Parameter: bar
```

Subshells, Variablen exportieren und einlesen
Ein grundsätzliches Konzept des Unix Prozessmanagments ist, dass jeder Prozess von einem Parentprozess abstammt. Dieser Text entsteht in einer Instanz des Editors `vim`. Ein Auszug aus der Ausgabe des Kommandos `pstree` soll dieses Konzept verdeutlichen:

```
$ pstree
init-+-arpwatch
   [...]
     |-xdm-+-XF86_SVGA
     |     `-xdm---fvwm2-+-FvwmCommandS
     |                   |-xterm---bash---vim
   [...]               [...]
```

Der Prozess *init* ist „Ahne“ aller Prozesse. Der Loginmanager *xdm* wartet auf Benutzerauthentifizierung und startet einen Windowmanager *fvwm2*, aus dem ein *xterm* gestartet wurde in dem eine *bash* läuft, aus der ein *vim* gestartet wurde.
Eine Subshell ist eine Shell, die aus einer anderen gestartet wurde. Ruft man ein Shell Script via

`$ ./script` oder
`$ bash script`

auf, so wird es in einer Subshell gestartet. Ruft man es via

`$ . script` oder
`$ source script`

auf, so werden die darin enthaltenen Befehle in der aktuellen Shell ausgeführt. Besonders beachten sollte man, dass die Verwendung von Pipelines auch innerhalb eines Shell Scripts zu Subshells führt. Ein Überblick über verschiedene Möglichkeiten ein Shellscript aufzurufen befindet sich auf Folie 112.

Die Unterscheidung ob Befehle in einer Subshell oder der aktuellen Shell asugeführt werden ist wichtig, weil Variablenzuweisungen auf eine `bash` Instanz beschränkt sind. Die Zuweisung kann in alle Subshells mit dem Befehl `export` exportiert werden. Folgende Kommandoabfolgen sollen das verdeutlichen, wobei man sich durch `pstree -h` einen Überblick über die aktuelle Situation machen kann.

```
$ # Test wird nicht exportieren
$ TEST=rose
$ echo $TEST
rose
$ bash
$ echo $TEST

$ TEST=eros
$ echo $TEST
eros
$ exit
exit
$ echo $TEST
rose
$ unset TEST
$ exit
$ # Test wird exportiert
$ TEST=rose
$ echo $TEST
rose
$ export TEST
$ bash
$ echo $TEST
rose
$ TEST=eros
$ echo $TEST
eros
$ exit
exit
$ echo $TEST
rose
$ unset TEST
$ exit
```

Setzt man also in der Shell eine Variable und startet dann ein Shell Script, so ist diese Variable – so sie nicht exportiert wurde – im Script nicht gesetzt.

Interaktiv kann man Variablem mit dem `read` Kommando definieren. Folgendes Beispielscript `what` liest interaktiv eine Variable ein und gibt sie dann wieder aus.

```
$ cat what
#!/bin/bash

# lese variable ein
echo "What do you want to do next?"
read todo

# ausgabe
echo "You said you will $todo"

$ ./what
What do you want to do next?
fall alseep
You said you will fall alseep
$
```

Shellscripts aufrufen

- Interpreteraufruf:
  - `bash helloworld`
    oder `sh helloworld`
  - in Subshell ausgeführt
  - nur r-Bit nötig
  - überschreibt Interpreterangabe im Script
- direkter Aufruf
  - `./helloworld`
  - in Subshell ausgeführt
  - r- und x-Bits nötig
  - Interpreterangabe im Script verwendet, wenn keine angegeben, dann in verwendeter Shell
- Sourcen
  - `. helloworld`
    oder `source helloworld`
  - in aktueller Shell ausgeführt
  - nur r-Bit nötig

Folie 112

Variablen

- Variablennamen bestehen aus Buchstaben, Underscore (_) und Ziffern.
- Zuweisung über

  ```
  $ VARIABLENNAME=wert
  ```

- Auf den Wert zugreifen mit `$`

  ```
  $ echo $VARIABLENNAME
  ```

- Unterschiedliche Variablenarten mit Beispielen
  - selbstdefinierte Variablen:
    `$FOO, $BAR, ...`
  - systemweite Variablen:
    `$HOSTNAME, $HOSTTYPE, ...`
  - built-in Variablen:
    `$HOME, $PS1, $PATH, ...`
- Variablen exportieren

  ```
  $ export VARIABLENNAME
  ```

- interaktiv einlesen:

  ```
  $ read  VARIABLENNAME
  ```

Folie 113

Einige Built-in Variablen

- $$... PID
- $0... Name des Shellscripts
- $n... n.ter übergebener Parameter
- $#... Anzahl der Positionsparameter
- $* ... alle Positionsparameter
- $?... letzter Exit Code (siehe unten)
- $!... letzte Background PID

Folie 114

Subshells

Aus einer Shell kann man eine weitere Shell starten, welche dann als Subshell der ursprünglichen Shell bezeichnet wird.

```
$ bash
```

Die im Script enthaltenen Kommandos werden in

- einer Subshell ausgeführt, wenn das Script mit
  `$ ./script [&]` oder
  `$ bash script [&]`
- in der aktuellen Shell augeführt, wenn das Script mit
  `$ . script` oder
  `$ source script`

gestartet wird, wobei das optionale `&` das Script im Hintergrund startet.

Folie 115

11.3 Spezielle Zeichen und Quoting

Leerzeichen `<space>` und Tabulatoren `<tab>` werden als Trennzeichen verwendet. So werden im folgenden Beispiel das Kommando, die Optionen und die Parameter eines Befehls durch Leerzeichen getrennt:

```
$ ls -l foo bar
```

Manchmal ist es erwünscht, einem Zeichen mit spezieller Bedeutung (einem sogenannten *Shell-Metacharacter*) diese zu nehmen. So kann ein Dateiname auch ein Leerzeichen enthalten. Seit Microsofts Betriebssysteme über die 8.3 Namenskonvention hinausgekommen sind, tritt dieses Phänomen leider gehäuft auf. Als Beispiel nehmen wir eine Datei mit dem Namen *Mein Lied.mp3* an. Versucht man diese in der Shell zu löschen und schreibt:

```
$ rm Mein Lied.mp3
```

so versucht die Shell zwei Dateien mit Namen *Mein* beziehungsweise *Lied.mp3* zu löschen. Um der Shell mitzuteilen, dass das Leerzeichen in diesem Fall keine spezielle Bedeutung hat verwendet man eine der drei Quoting Methoden. Diese seien im Folgenden anhand unseres Beispiels angeführt:

```
$ rm Mein\ Lied.mp3
$ rm 'Mein Lied.mp3'
$ rm "Mein Lied.mp3"
```

Der Backslash `\` (escape character) hebt dabei die spezielle Bedeutung des unmittelbar folgenden Zeichens auf. Auch wenn dieses Zeichen ein „newline“, also das Resultat des Betätigens der Return Taste, ist. Sowohl einfache `''` (Single Quotes) als auch doppelte Anführungszeichen `""` (Double Quotes) heben die spezielle Bedeutung der Zeichen dazwischen auf, wobei dies bei den einfachen Anführungszeichen für alle Zeichen gilt. Bei doppelten Anführungszeichen bleibt die spezielle Bedeutung der Zeichen `$`, `'` und `\` erhalten. Vergleiche die Ausgaben folgender Kommandos:

```
$ TEST=rose
$ echo "In \$TEST steht \"$TEST\""
In $TEST steht "rose"
$ echo 'In \$TEST steht \"$TEST\"'
In \$TEST steht \"$TEST\"
```

Im weiteren werden wir noch des öfteren auf spezielle Zeichen stoßen.

Befehle zu Gruppen zusammenfassen
Befehle werden durch einen Zeilenumbruch `<newline>` oder ein Semikolon `;` getrennt. Im Folgenden zwei Schreibweisen, die das gleiche liefern:

```
$ cd /usr/bin
$ ls
```
bzw.
```
$ cd /usr/bin; ls
```

Es gibt zwei Möglichkeiten Kommandos zu Gruppieren. Runde Klammern `()` (Parantheses) beziehungsweise geschwungene Klammern `{}` (Curly Braces). Befehle, die in runden Klammern gruppiert sind, werden in einer Subshell ausgeführt, während in geschwungenen Klammern gruppierte Befehle in der aktuellen Shell ausgeführt werden. Dazu folgende Beispiele:

```
$ { FOO=bar; echo "\$FOO ist $FOO"; }; echo "\$FOO ist $FOO"
$FOO ist bar
$FOO ist bar
$ ( FOO=bar; echo "\$FOO ist $FOO" ); echo "\$FOO ist $FOO"
$FOO ist bar
$FOO ist
```

Die runden und die geschwungenen Klammern zählen zu den speziellen Zeichen. Diese müssen also, so sie in Dateiname auftreten, mit Quotes versehen werden.

Quoting

Quoting wird dazu benutzt, um bestimmten Zeichen ihre spezielle Bedeutung zu nehmen. Es gibt drei Möglichkeiten zu quoten.

\	escape character
''	single quotes
""	double quotes

Hat ein geistreicher Benutzer eine Datei unter „Mein Lied * Napster.mp3“ abgespeichert, so kann man diese mit einer der folgenden Zeilen löschen.

```
$ rm Mein\ Lied\ \*\ Napster.mp3
$ rm "Mein Lied * Napster.mp3"
```

Der escape character nimmt dem folgenden Zeichen seine spezielle Bedeutung. Alle Zeichen zwischen single quotes sind von ihrer speziellen Bedeutung befreit. Double quotes lassen den Zeichen `$`, `'` und `\` ihre spezielle Bedeutung. Double quotes sind also schwächer als single quotes.

Folie 116

Kommandos Gruppieren

; Trennt verschiedene Kommandos, die in der gleichen Zeile geschrieben werden. Die Kommandos werden hintereinander ausgeführt

() Ein oder mehrere Kommandos innerhalb runder Klammern werden in einer Subshell ausgeführt.

{} Ein oder mehrere Kommandos innerhalb geschwungener Klammern werden als Block in der aktuellen Shell ausgeführt.

Folie 117

Wildcards und Pattern Matching

Gehen wir im Folgenden davon aus, dass im aktuellen Verzeichnis folgende Dateien existieren:

```
$ ls
glut gut hut mut mutter
```

`*` steht für beliebige Zeichenketten; auch leere.

```
$ ls mut*
mut mutter
```

`?` steht für genau ein beliebiges Zeichen.

```
$ ls ?ut
gut hut mut
```

Alle in eckigen Klammern angegebenen Zeichen dürfen an dieser stelle alternativ vorkommen.

```
$ ls [gh]ut
gut hut
```

Ein `!` oder `^` verneint diese Auswahl.

```
$ ls [!gh]ut
mut
```

Alle in geschwungenen Klammern angegebenen Zeichenketten, die durch Beistriche getrennt sind dürfen vorkommen,

```
$ ls {gl,m}ut
glut mut
```

Diese Konstrukte sind auch kombinierbar.

```
$ ls {gl,m}ut*
glut mut mutter
```

Wildcards und Pattern Matching

Folgende Zeichen haben in der `bash` spezielle Bedeutung

`*`	beliebige Zeichenkette
`?`	genau ein beliebiges Zeichen
`[ ]`	genau eines der genannten Zeichen
`[^ ]`	genau ein nicht genanntes Zeichen
oder `[! ]`	
`{ , }`	genau eine der Zeichenketten

Folie 118

11.4 Exit Status

Jedes Kommando liefert nach seiner Beendigung seinem Parentprozess einen *Exit Status* zurück. Der Exit Status ist ein Integerwert zwischen 0 und 255. 0 bezeichnet den Erfolg des Kommandos, alle anderen Werte bezeichnen verschiedene Ausmaße des Scheiterns. Der Exit Status des letzten Kommandos wird in der Variable `$?` gespeichert.

```
$ grep root /etc/passwd
root:x:0:0:root:/root:/bin/bash
$ echo $?
0
$ grep Administrator /etc/passwd
$ echo $?
1
$ true
$ echo $?
0
$ false
$ echo $?
1
```

Die Programme `true` beziehungsweise `false` dienen einzig und allein dazu Exit Status 0 beziehungsweise 1 zu liefern. Das scheint auf den ersten Blick sinnlos, aber das tut auch `/dev/null`.
Um den Exit Status eines Shell Scripts zu beeinflussen gibt es den built-in Befehl `exit`. Dieser beendet ein Script sofort und setzt einen anzugebenden Wert als Exit Status des Scripts. Das folgende Script namens *parameterexists* liefert Exit Status 0, wenn mindestens ein Parameter übergeben wurde. Andernfalls liefert es Exit Status 1.

```
#!/bin/bash
if [ $1 ]; then
   exit 0
else
   exit 1
fi
```

Exit Status

Jedes Programm liefert nach Beendigung seinem Parent einen Exit Status zurück.

0 steht für eine erfolgreiche Beendigung des Programms

1-255 bezeichnet verschiedene Ausmaße des Scheiterns.

Der Exit Status eines Shell Scripts kann über den `bash` built-in Befehl `exit` gesteuert werden.

Der Exit Status des letzten ausgeführten Kommandos steht in der Variable `$?`.

Folie 119

Man sollte dabei unbedingt die Konvention beachten, dass 0 für einen Erfolg steht, während andere Zahlen als Misserfolg interpretiert werden.

11.5 Flow Control

Unter Flow Control versteht man *Kontrollstrukturen* wie bedingtes Ausführen von Kommandos und Schleifen. Im Wesentlichen sind dies `if`, `case`, `for` und `while`. Wir werden diese im Weiteren nacheinander abhandeln, und nebenbei allerlei Nützliches „mitnehmen“.

Bedingungen (`if`)
Zuerst wenden wir uns dem `if` zu, von dem wir schon ein Beispiel gesehen haben; die Syntax der `if`-Anweisung ist auf Folie 120 angegeben: Bei zutreffen von *Bedingung1* die wird die *Anweisung1* ausgeführt. Trifft *Bedingung1* nicht, *Bedingung2* aber schon zu, so wird *Anweisung2* ausgeführt. Trifft keine der beiden Bedingungen zu, so wird *Anweisung3* ausgeführt.

Um ein `if` formulieren zu können, müssen wir also wissen, was eine Bedingung ist. Eine Bedingung ist nichts anderes als ein Kommando. Ist dessen Exit Status 0, so gilt die Bedingung als erfüllt. Andernfalls gilt sie als nicht erfüllt. Das erklärt auch, warum man den Konventionen des Exit Status in eigenen Scripts folgen sollte.
Ein in Bezug auf Bedingungen besonders wertvoller Befehl ist der built-in Befehl `test`. Will man zum Beispiel feststellen, ob der Inhalt der Variable `TEST` gleich „rose“ ist, so kann man das so überprüfen:

```
$ TEST=rose
$ test $TEST = rose
$ echo $?
0
$ TEST=eros
$ test $TEST = rose
$ echo $?
1
```

Als Synonym für `test` können auch eckige Klammern `[]` verwendet werden, wobei darauf zu achten ist, dass nach der

öffnenden und vor der schließenden Klammer Leerzeichen stehen.

```
$ TEST=rose
$ [ $TEST = rose ]
$ echo $?
0
```

Arithmetik
Neben der Möglichkeit ganze Zahlen gegeneinander zu testen, kann man in der Shell mit ganzen Zahlen auch Rechnen. Ausdrücke die mit `$[]` oder `$(())` umschlossen sind werden als ganzzahlige Rechenoperationen ausgelegt und auszuwerten versucht.

```
$ echo "1+1=$[1+1]"
1+1=2
$ echo "3*3=$((3*3))"
3*3=9
$ echo "11/4=$[11/4] mit Rest $[11%4]"
11/4=2 mit Rest 3
```

In der Programmierung ist es häufig gefragt, den Inhalt einer Variable zu inkrementieren, sprich um 1 zu erhöhen. Das sieht in C definitiv schöner aus, als in der `bash`:

```
$ N=1
$ echo $[$N+1]
2
```

Um in der Shell oder innerhalb von Scripts Gleitkommarechnungen oder allgemein kompliziertere Rechnungen durchzuführen sei dem Leser `bc` und `dc` and Herz gelegt.

Flow Control, `if`

- erlaubt bedingte Ausführung von Kommandos
- dem `if` in Programmiersprachen sehr ähnlich
- Syntax:

```
if Bedingung1
then
  Anweisung1
elif Bedingung2
then
  Anweisung2
else
  Anweisung3
fi
```

Der `elif` und der `else`-Block sind optional.

Die *Bedingung* ist ein Kommando, das 0 (true) oder $\neq 0$ (false) als Exit Status zurückgibt.

Folie 120

Bedingungen (Strings, Dateien)

Das ist ein unvollständige Liste von Tests mit Strings (Zeichenketten) und Dateien.

Im folgender Liste bezeichnen *S1* beziehungsweise *S2* Strings und *D1* beziehungsweise *D2* Dateinamen.

Test	**Wahr wenn**
`[` *S1* `=` *S2* `]`	Strings ident
`[` *S1* `!=` *S2* `]`	Strings nicht ident
...	
`[ -e` *D1* `]`	Datei *D1* existiert
`[ -d` *D1* `]`	*D1* ist ein Verzeichnis
`[ -x` *D1* `]`	*D1* ist ausführbar
`[` *D1* `-nt` *D2* `]`	*D1* neuer als *D2*
...	

Bedingungen (Integers)

Die folgenden Tests gehen davon aus, dass in den Variablen `A` und `B` ganze Zahlen stehen. Ist dies nicht so, gibt es eine Fehlermeldung.

Test	Wahr wenn
`[ $A -lt $B ]`	$A kleiner als $B
`[ $A -gt $B ]`	$A größer als $B
`[ $A -le $B ]`	$A kleiner gleich $B
`[ $A -ge $B ]`	$A größer gleich $B
`[ $A -eq $B ]`	$A gleich $B
`[ $A -ne $B ]`	$A ungleich $B

Folie 122

Arithmetik

Integerarithmetik wird innerhalb eckiger oder doppelter runder Klammern ausgewertet.

```
$ echo $[1+1]
$ echo $((1-1))
```

Eine unvollständige Liste von Operatoren:

Operator	Bedeutung
+	Plus
-	Minus
*	Multiplikation
/	Division
%	Modulo
**	Exponent

Das Inkrementieren einer Variable funktioniert folglich so:

```
$ N=1
$ N=$[$N+1]
$ echo $N
```

Folie 123

Bedingungen Verknüpfen

Neben dem `if` gibt es noch eine einfachere, eingeschränkte Möglichkeit des bedingten Ausführens von Kommandos. So wird im Folgenden der Befehl `ls` nur ausgeführt, wenn die Programmdatei `/bin/ls` existiert und ausführbar ist:

```
$ [ -x /bin/ls ] && ls
helloworld  parameterexists  parameters
```

Der Exit Status solch einer Kette von Kommandos ist der Exit Status des letzten ausgeführten Kommandos. Ein weiteres Beispiel legt eine Datei *foo* nur an, wenn diese noch nicht existiert:

```
$ touch foo
$ ls
foo
$ [ -e foo ] || touch foo
$ echo $?
0
$ [ ! -e foo ] && touch foo
$ echo $?
1
```

Bedingtes Ausführen

```
$ Kommando1 && Kommando2
```

Kommando2 wird ausgeführt, wenn Kommando1 Exit Status 0 hat.

```
$ Kommando1 || Kommando2
```

Kommando2 wird ausgeführt, wenn Kommando1 Exit Status $\neq 0$ hat.

- Der Exit Status dieser Zeilen ist der Exit Status des letzten ausgeführten Kommandos.
- Ein `!` vor einem Kommando verneint den Exit Status.

Operator	Bedeutung
`&&`	Und
`||`	Oder
`!`	Nicht

- Kann benutzt werden, um verknüpfte Bedingungen in einer `if`-Anweisung zu erstellen.

Flow Control, `case`

Eine spezielle und für manche Anwendungen sehr angenehme Variante der `if`-Anweisung ist die `case`-Anweisung, die nach folgender Syntax verlangt.

```
case Ausdruck in
  Pattern1)
    Anweisungen ;;
  Pattern2)
    Anweisungen ;;
  ...
esac
```

Das bietet sich zum Beispiel dazu an, um Switches zu realisieren. Wie bei Initscripts üblich erlaubt folgendes Script eine „start“ und „stop“ Option:

```
case $1 in
  start) ...;;
  stop) ...;;
  *) Usage: ...;;
esac
```

Folie 125

case-Anweisung
Die `case`-Anweisung wird oft in Initscripts verwendet. Diese Scripts dienen zum Starten und Stoppen von Systemdiensten. Üblicherweise liegt für jeden Systemdienst im Verzeichnis `/etc/init.d/` (manchmal auch `/etc/rc.d/init.d` oder `/etc/rc.d/`) ein Script, das die Parameter `start`, `stop` oder `restart` versteht (vgl. Teil 1, Abschnitt 9.2).

```
$ cat /etc/init.d/inetd
#!/bin/sh
#
# start/stop inetd super server.
[...]
case "$1" in
    start)
        echo -n "Starting internet superserver:"
        [...]
        ;;
    stop)
        echo -n "Stopping internet superserver:"
        [...]
        ;;
    restart)
        echo -n "Restarting internet superserver:"
        [...]
        ;;
    *)
        echo "Usage: /etc/init.d/inetd {start|stop|restart}"
        exit 1
        ;;
esac
```

Gibt man einen Parameter an, der nicht angeführt ist, so wird der Punkt unter `*)` ausgeführt. Dieser klärt über die richtige Benutzung des Scripts auf und beendet das Script mit Exit Status 1.

for-Anweisung
Die `for`-Schleife unterscheidet sich stark von `for`-Schleifen anderer Programmiersprachen. In ihr durchläuft ein Variable alle Werte einer Liste. Die Einträge dieser Liste sind durch Leerzeichen, Tabulatoren oder Zeilenumbruch getrennt. Alle Frauen sind herzlichst dazu eingeladen das Script *polygam* nach ihren Bedürfnissen zu modifizieren.

```
#!/bin/bash

LISTE="Liese Ingrid Stefanie Frauke"
for FRAU in $LISTE; do
   echo "$FRAU ist meine Frau."
done
```

Flow Control, `for`

Syntax der `bash` `for`-Schleife wesentlich anders als in vielen Programmiersprachen

`for` *Name* `in` *Liste*
`do`
  Anweisungen
`done`

- Inhalt der Variable Name durchläuft alle Elemente der Liste
- folgender Code spiel alle mp3-Dateien im aktuellen Verzeichnis ab

```
for LIED in *.mp3
do
   mpg123 "$LIED"
done
```

`mpg123` ist ein mp3-Player.
`*.mp3` wird von der `bash` zu einer Liste aller Dateien mit der Endung `.mp3` expandiert.

Folie 126

while- und until-Anweisung
Oft will man an einer bestimmten Stelle in einem Script die Ausgabe eines Kommandos platzieren. Das nennt man Kommando-Substitution und lässt sich auf zwei Arten realisieren. Das Kommando wird entweder mit Backticks (Backquotes) ‘‘ oder mit `$()` umschlossen. Das Script *summerton* ist der Telefonzeitansage nachempfunden:

```
#!/bin/bash

# Zeitausgabe ala Zeitansage per Telefon

# Warten bis zu den naechsten vollen 10 Sekunden
while [ $[‘date +%S‘%10] -ne 0 ]; do
   sleep 1
done

# Alle 10 Sekunden Zeitansage
while true; do
   echo "Es wird mit dem Summerton ‘date +%k‘ Uhr, \
‘date +%M‘ Minuten und ‘date +%S‘ Sekunden."
   sleep 10
done
```

Die Frage ob man ‘‘ oder `$()` verwendet ist nicht ganz irrelevant. Die Verwendung von `$()` hat den Vorteil, dass man diese Konstruktion ineinander verschachteln kann. Der Nachteil ist, dass diese Konstruktion in der `bash`, nicht jedoch in der `sh` existiert. Versucht man ein Script, das `$()` verwendet auf einem System ohne einer `bash` zu verwenden, so wird dies scheitern.

Die `until`-Anweisung funktioniert völlig analog zu `while` natürlich mit dem Unterschied, dass der Aweisungsblock ausgeführt wird, wenn die Bedingung nicht erfüllt ist.

Das übliche For I

Um das in anderen Programmiersprachen übliche Verhalten einer `for`-Schleife zu erhalten kann folgender Code in *usualfor1* dienen:

```
#!/bin/bash

# Immitiert ein for wie in C, wobei dies einem
# for(int i=$START, i<=$STOP, i=i+$STEP) entspricht

START=1; STOP=10; STEP=3

for N in ‘seq $START $STEP $STOP‘
do
   echo $N
done
```

`$N` durchläuft alle Werte von `$START` bis `$STOP` in Abständen von `$STEP`.
Zur Kommando-Substitution werden hier Backticks (‘‘) verwendet, was die Portabilität des Codes erhöht. In den meisten Fällen kann auf eine einfachere Syntax von `seq` zurückgegriffen werden. Siehe dazu

```
$ man seq
```

Flow Control, `while, until`

`while`- bzw. `until`-Syntax entspricht der in anderen Programmiersprachen.
Anweisungsblock wird, solange eine Bedingung wahr bzw. nicht wahr ist, ausgeführt.

```
while Bedingung
do
  Anweisungen
done
```

Als Beispiel ein Codesegment, das alle 5 Sekunden prüft, ob eine Modemverbindung existiert und falls diese zusammengebrochen ist, eine Funktion namens `reconnect()` ausführt.

```
while ifconfig | grep ppp0
do
  sleep 5
done
reconnect()
```

Folie 127

Das übliche For II

Eine weitere Methode, das in anderen Programmiersprachen übliche `for` zu erhalten, ist die folgende Umschreibung in *usualfor2*, die natürlich nicht Shell spezifisch ist:

```
#!/bin/bash

# umschreibt ein for wie in C
# durch ein while

START=1; STOP=10; STEP=3

N=$START
while [ $N -le $STOP ]; do
   echo $N
   N=$[$N+$STEP]
done
```

Auch hier durchläuft `$N` alle Werte von `$START` bis `$STOP` in Abständen von `$STEP`.

Zum Abschluß geben wir noch einige Quellen für (weiterführende) Informationen zur Bashprogrammierung an.

Quellen

1. Manpage
   `$ man bash`
2. Bücher
   C. Newham, B. Rosenblatt, „Learning the Bash Shell"(2nd Edition, 'O' Reilly, 1998)
3. Webpages
   `www.gnu.org/manual/bash-2.02/`
   `/html_manual/bashref.html`
   `www.ldp.at/HOWTO/Adv-Bash-Scr-HOWTO/`
   `unixhelp.ed.ac.uk/scrpt/scrpt2.html`
   `www.beforever.com/bashtut.htm`
   `www.linuxgazette.com/issue25/dearman.html`
   `pegasus.rutgers.edu/~elford/`
   `/unix/bash-tute.html`
4. auf jedem System vorhandene Scripts
   `/etc/init.d/*`
   `~/.bashrc`

Folie 128

12 Grundlagen der Security und Logging

Dieses Kapitel dient der Besprechung grundlegender Sicherheitskonzepte und Sicherheitsmechanismen. Wir erklären SUID, SGID und Sticky-Bit Berechtigungen und die Grundlagen des Pluggable Authentication Module (PAM). Wir geben einen Überblick über das Logging auf Linuxsystemen – das nicht nur im Bereich der Sicherheit, sondern auch bei jeglicher Fehlersuche eine große Rolle spielt – und erklären die Konfiguration des Systemlogdaemons. Schließlich besprechen wir Varianten von Angriffen auf Computersysteme und diskutieren grundlegende Schutzmechanismen.

Bei der Entwicklung von Unix und Linux war und ist Sicherheit nicht das primäre Ziel; im Vordergrund stehen vielmehr Machbarkeit und Benutzerfreundlichkeit. Daher werden und wurden Unix/Linux-Systeme immer wieder Ziel von Attacken und Einbrüchen. Es ist eine der wesentlichen Aufgaben des Systemadministrators, die Sicherheit des Systems und seiner Daten zu überwachen und sicherzustellen.
In kaum einem Bereich der Systemadministration spielen Planung, Methode und „Politik“ eine so große Rolle, wie im Bereich der Sicherheit. Bei größeren Systemen ist es unumgänglich, ein umfassendes Sicherheitskonzept zu erarbeiten, in dem genau festgelegt wird, was wie geschützt werden soll. (Man denke zB an den juristischen Aspekt.) Die „politischen Aspekte“ (vgl. Kapitel 1) der Sicherheit von Computersystemen sind vor allem deshalb besonders zu beachten, weil die Sicherheit indirekt proportional zur Bequemlichkeit der User ist (siehe auch Folie 129).
Vor der allgemeinen Diskussion warum und wovor Computersysteme geschützt werden müssen erklären wir in jeweils eigenen Abschnitten spezielle Dateiberechtigungen, das Pluggable Authentication Module (PAM) und das Logging auf Linuxsystemen.

Security

- Unix optimiert Benutzerfreundlichkeit, nicht Sicherheit
- $$\text{security} = \frac{1}{\text{convenience}}$$
- „politische Aspekte“
- rechtliche Aspekte
- umfassendes Sicherheitskonzept

Folie 129

12.1 Spezielle Dateiberechtigungen

Ein wichtiger Punkt der Unix-Sicherheitskonzeption ist das *SUID (Set UID)-Bit*. Dabei handelt es sich um eine spezielle Dateiberechtigung, die es erlaubt, ein Kommando nicht mit den Rechten des Benutzers, der es aufruft, sondern mit den Rechten des Besitzers der Programmdatei auszuführen. Dieses Konzept ermöglicht es, dass zB ein normaler Benutzer sein Passwort ändern kann. Dazu muss ja der Passworteintrag in `/etc/passwd` bzw. `/etc/shadow` geändert werden, also eine Datei geschrieben werden, für die nur root Schreibrecht besitzt. Daher ist für das Programm zur Passwortänderung (`/usr/bin/passwd`) das SUID-Bit gesetzt. SUID-Programme stellen immer ein potentielles Sicherheitsrisiko dar und sind oft Ziel von Hackern. Daher sollte die Zahl an SUID-Programmen am System möglichst gering gehalten werden und regelmäßig überprüft werden, ob nicht neue SUID-Programme aufgetaucht sind (Zeichen für Hackereinbruch!).
Analog dazu (aber in der Praxis weniger relevant) gibt es das Konzept des *SGID-Bits*, das es ermöglicht ein Programm mit den Gruppenrechten des Gruppenbesitzers des Programmfiles auszuführen. Ist auf einem Directory das SGID-Bit gesetzt, so erhält ein, in diesem Verzeichnis erstelltes File den Gruppenbesitz des Gruppenbesitzers des Directories.

Eine weitere spezielle Berechtigung ist das sogenannte *Sticky-Bit* für Verzeichnisse. Ähnlich wie das SGID-Bit für Verzeichnisse erfüllt es eine wichtige Funktion vor allem bei Directories, die von einer aus mehreren Benutzern bestehenden Gruppe verwendet wird. Ist für ein Verzeichnis das Sticky-Bit gesetzt, so kann ein Benutzer, selbst wenn er Schreibrechte in diesem Directory besitzt nur Dateien schreiben, in deren Besitz er sich befindet. (Sonst reicht ja die Schreibberechtigung im Directory aus, um *alle* Files darin (auch die, für die keine Schreibberechtigung besteht) zu löschen; das Sticky-Bit entspricht somit der Option „Ändern, Ersteller/Besitzer“ in der NT ACL. Typischerweise ist für das `/tmp`-Directory das Sticky-Bit gesetzt. Achtung es vererbt sich *nicht* automatisch auf Subdirectories!

Die speziellen Berechtigungen SUID, SGID und Sticky-Bit können (ganz genau wie die Basisberechtigungen „rwx“) mittels `ls -l` angezeigt und mittels `chmod` verändert werden. Im symbolischen Modus werden SUID, SGID und Sticky-Bit mit „s“, bzw. „t“ bezeichnet: `chmod u=s` bzw. `g=s` setzt also das SUID resp. das SGID Bit, das Sticky-Bit wird mit `chmod o=t` gesetzt. Die oktale Syntax hierfür wird auf den Folien 130 und 131 erklärt, die auch das Setzen der Basisberechtigungen sowie des Besitzers von Files und Directories aus Teil 1, Abschnitt 3.6 wiederholen.

Fileberechtigungen

Perm.	File	Directory
r=4	lesen	listen
w=2	schreiben	Files schreiben
x=1	ausführen	cd, PATH
SUID	runs with UID=owner	
SGID	runs with GID=owner	neue Files GID=Directory
Sticky-Bit		nur Besitzer von File oder Dir. kann schreiben

Folie 130

Besitz/Berechtigungen ändern

- Jedes File hat Besitzer und Gruppenbesitzer;
  ändern mit
  - `chown karli vielgeld.tex`
  - `chgrp finanz vielgeld.tex`
  - `chown karli.finanz vielgeld.tex`
- Berechtigungen rwx(s,t) oder numerisch;
  ändern mit
  - `chmod 1755 (o+t)...rwxr-xr-t`
  - `chmod 2755 (g+s)...rwxr-sr-x`
  - `chmod 4755 (u+s)...rwsr-xr-x`

Folie 131

12.2 PAM

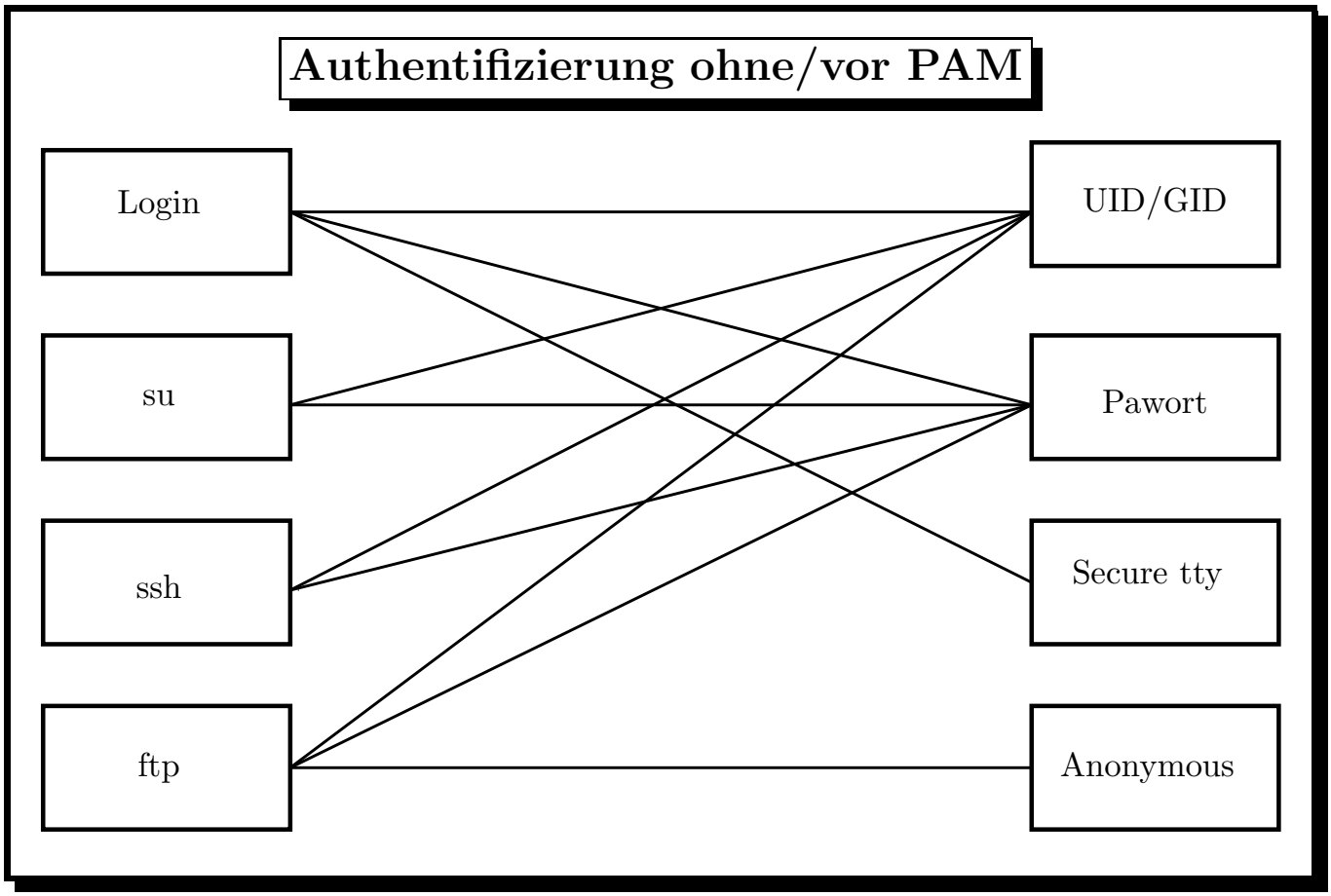
In diesem kurzen Abschnitt besprechen wir die Grundlagen des Pluggable Authentication Module- System (PAM). Dabei handelt es sich um eine Sammlung gesharter Bibliotheken, die systemweite Authentifizierungsaufgaben übernimmt und dadurch vereinheitlicht. Es gibt eine Vielzahl von Prozessen, die eine Authentifizierung vornehmen müssen (zB `login, su, sshd, ftp, telnetd`, etc.).
Müssten diese Programme jeweils direkt auf die verschiedenen Authentifizierungsmechanismen zugreifen, so würde sich ein fast unüberschaubares Konfigurationschaos ergeben (siehe Folie 133). An diesem Punkt setzt das PAM-System ein. Muss irgendein Prozess eine Authentifizierung vornehmen (zB Login) so nimmt er dafür PAM in Anspruch. Das PAM-System erledigt gemäß seiner Konfiguration in `/etc/pam.d/` die Authentifizierung und meldet dem anfragenden Prozess, ob die entsprechende Authentifizierung positiv erledigt werden konnte.

Die Konfiguration des PAM wird auf den Folien 135 und 136 erläutert. Darüber hinaus findet man Informationen über PAM auf der Manpage bzw. unter `http://www.kernel.org/pub/linux/libs/pam`.

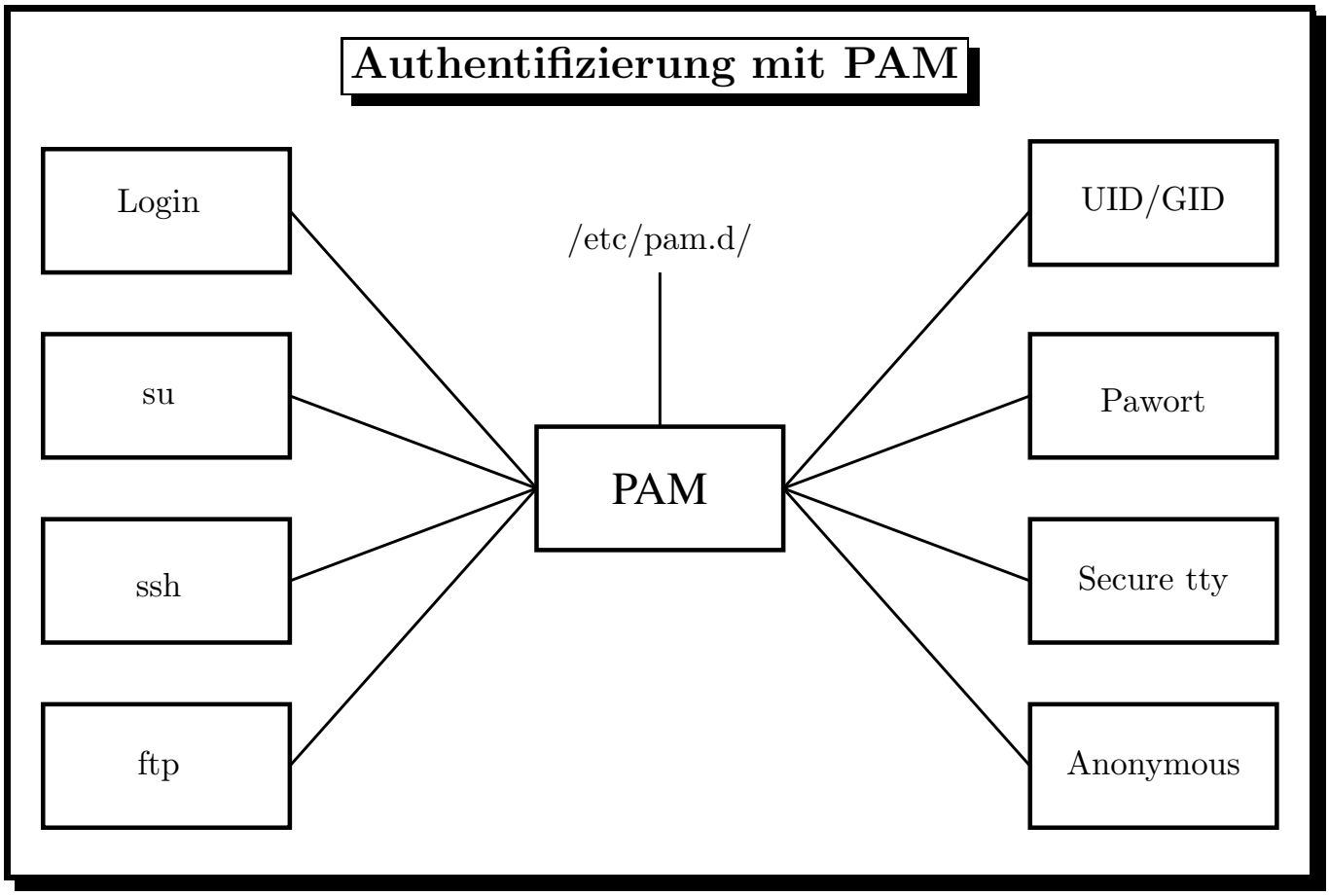
Pluggable Authentication Modules (PAM)

- Sammlung gesharter Libraries
- übersichtliche Konfiguration, wie Anwendungen Benutzer authentifizieren
  `/etc/pam.d/`
- entwickelt von Sun Microsystems
- adaptiert für Linux

Folie 132


Authentifizierung ohne/vor PAM
Login
su
ssh
ftp
UID/GID
Pawort
Secure tty
Anonymous

Folie 133


Authentifizierung mit PAM
Login
su
ssh
ftp
/etc/pam.d/
PAM
UID/GID
Pawort
Secure tty
Anonymous

Folie 134

PAM Konfiguration (1)

```
[root@pablo /tmp]# cat /etc/pam.d/login
#%PAM-1.0
auth       required     /lib/security/pam_securetty.so
auth       required     /lib/security/pam_pwdb.so
                          shadow nullok
auth       required     /lib/security/pam_nologin.so
account    required     /lib/security/pam_pwdb.so
password   required     /lib/security/pam_cracklib.so
password   required     /lib/security/pam_pwdb.so
                          nullok use_authtok shadow
session    required     /lib/security/pam_pwdb.so
session    optional     /lib/security/pam_console.so
```

Folie 135

PAM Konfiguration (2)

```
[root@pablo /tmp]# cat /etc/pam.d/ssh
#%PAM-1.0
auth       required     /lib/security/pam_pwdb.so shadow
auth       required     /lib/security/pam_nologin.so
account    required     /lib/security/pam_pwdb.so
password   required     /lib/security/pam_cracklib.so
password   required     /lib/security/pam_pwdb.so shadow
                          nullok use_authtok
session    required     /lib/security/pam_pwdb.so
```

Folie 136

12.3 Logging

Wir besprechen zunächst die wichtigsten *Logfiles* auf einem Unix/Linux-System, in denen (je nach Konfiguration) alle relevanten Ereignisse am System protokolliert werden. Diese Dateien sind neben ihrer Sicherheitsrelevanz natürlich auch für viele weitere Aufgaben des Systemadministrators wichtig (Monitoring, Fehlersuche und Behebung,...).
Auf fast allen Linux-Systemen befinden sich diese Dateien in `/var/log/`, auf anderen Unix-Systemen aber auch oft in `/var/adm/` oder `/usr/adm/`.
Das wichtigste Logfile ist `/var/log/messages`, in dem die meisten relevanten Informationen gespeichert werden; es ist eine ASCII Datei, die zB mit `less` angesehen werden kann. Weitere wichtige Logdateien sind auf Folie 137 zusammengestellt. Die Dateien `lastlog, wtmp` und `utmp` sind binär und können mittels der entsprechenden Kommandos auf Folie 138 ausgewertet werden.

Das Logging in Unix-Systemen wird mittels des Syslog Daemons `syslogd` (und des Kernel-Log Daemons `klogd`) durchgeführt. Der Syslog Daemon wird in `/etc/syslog.conf` konfiguriert und nimmt von verschiedenen anderen Prozessen bzw. dem Kernel geordnet nach sog. Facilities: (auth(=security), authpriv, cron, daemon, kern, lpr, mail, mark, news, syslog, user, uucp und local0–local7) Meldungen verschiedener Priorität (debug, info, notice, warning(=warn), error(=err), crit, alert, emerg(=panic)) entgegen und schreibt sie gemäß seiner Konfiguration in verschiedene Dateien bzw. leitet sie weiter. Mehr Information über den Syslog Daemon bzw. seine Konfiguration ist den entsprechenden Manpages mittels `man syslog` bzw. `syslog.conf` zu entnehmen.

Logfiles

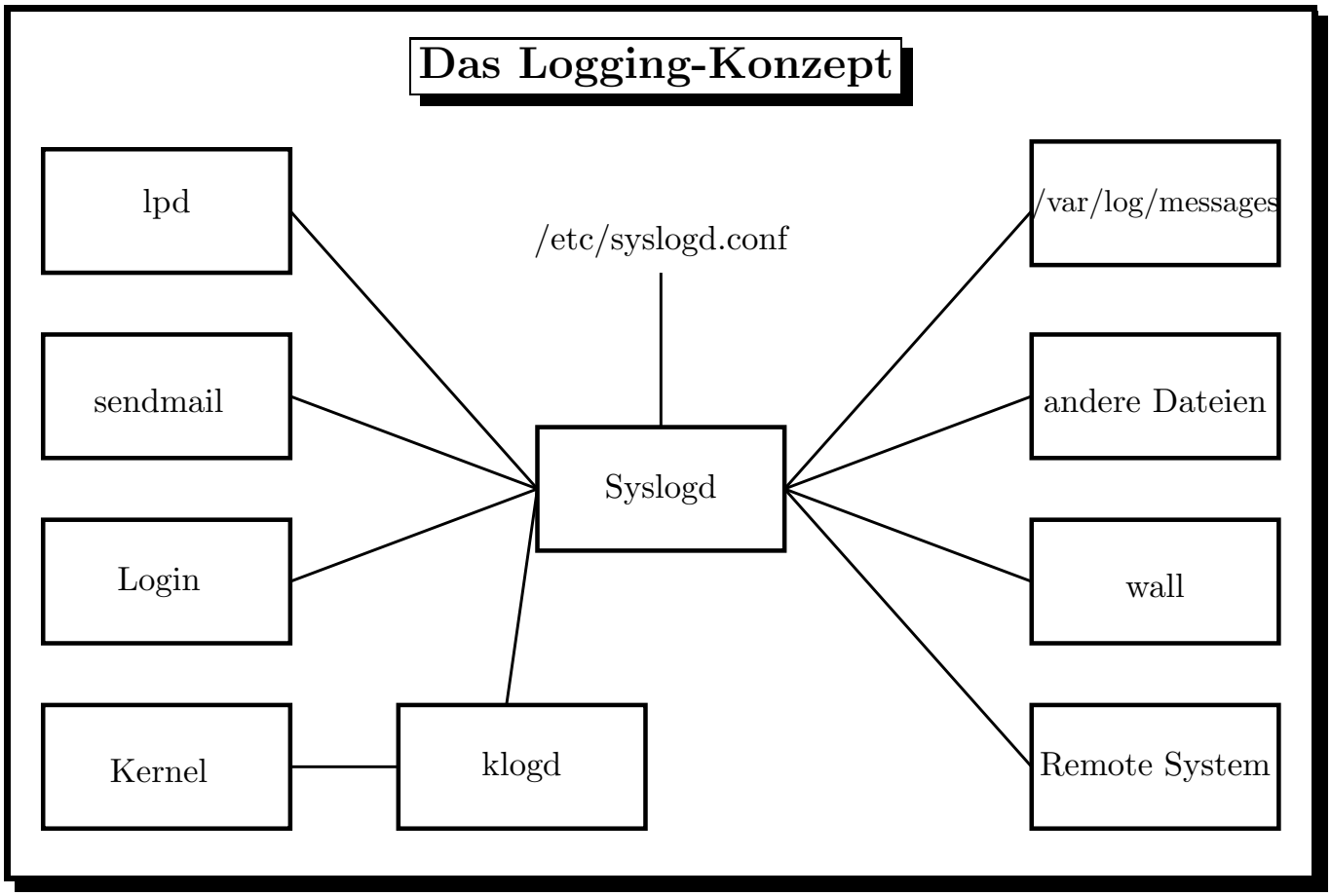
- `/var/log/messages`... Masterlogfile (ASCII)
- `/var/log/lastlog`... letzter erfolgreicher Login
- `/var/log/secure`... erfolglose Logins (ASCII)
- `/var/log/wtmp`... erfolgreiche Logins
- `/var/run/utmp`... derzeit eingeloggte User
- `/var/log/{maillog, http, cups, rpmpkgs, XFree86, samba}`, etc. ...Logfiles einzelner Programme

Folie 137

Nützliche Kommandos

- `w`... Wer ist eingeloggt und tut was?
- `who`... Wer ist eingeloggt und weitere Information aus `/var/log/utmp, wtmp`
- `last`... Wer war wann das letzte Mal eingeloggt
- `lastlog`... Letztes login aller User
- `tail -f file`... zeige die letzten 10 Zeilen des Files und neuen Input

Folie 138


Das Logging-Konzept
lpd
sendmail
Login
Kernel
klogd
/etc/syslogd.conf
Syslogd
/var/log/messages
andere Dateien
wall
Remote System

Folie 139

Der Syslog-Daemon

- Konfiguration `/etc/syslog.conf`
- Facilities
  - auth(=security)
  - cron
  - daemon
  - kern
  - mail, ...
- Priorities
  - debug
  - info
  - warning(=warn)
  - crit
  - emerg(=panic), ...

Folie 140

/etc/syslog.conf

```
# Log anything (except mail) of level info or higher.
# Don't log private authentication messages!
*.info;mail.none;authpriv.none  /var/log/messages
# The authpriv file has restricted access.
authpriv.*                      /var/log/secure
# Log all the mail messages in one place.
mail.*                          /var/log/maillog
# Everybody gets emergency messages, plus log them on another
# machine.
*.emerg                         *
# Save mail and news errors of level err and higher in a
# special file.
:uucp,news.crit                 /var/log/spooler
```

Folie 141

/var/log/messages

```
Apr 30 21:08:04 phobos rpc.mountd: authenticated mount request
from radon.mat.univie.ac.at:1000 for /users/gerald (/users)
Apr 30 21:13:43 phobos rpc.mountd: authenticated unmount request
from radon.mat.univie.ac.at:1012 for /users/gerald (/users)
Apr 30 21:23:20 phobos sshd2[1143]: connection from "131.130.145.151"
Apr 30 21:23:21 phobos sshd2[13747]: Remote host disconnected:
Connection closed.
Apr 30 21:23:21 phobos sshd2[13747]: connection lost:
'Connection closed.'
Apr 30 21:23:52 phobos sshd2[1143]: connection from "131.130.145.151"
Apr 30 21:24:06 phobos rpc.mountd: authenticated mount request
from radon.mat.univie.ac.at:602 for /users/gerald (/users)
Apr 30 21:24:13 phobos sshd2[13750]: User root's local password accepted.
Apr 30 21:24:13 phobos sshd2[13750]: Password authentication for user
root accepted.
Apr 30 21:24:13 phobos sshd2[13750]: User root, coming
from hektor.mat.univie.ac.at, authenticated.
```

Folie 142

12.4 Sicherheit – Allgemeine Diskussion

Warum sollte ein System geschützt werden?
Diese Frage umfasst zwei wesentliche Themengebiete. Einerseits muss man sensible Benutzerdaten schützen, andererseits den Rechner selbst, damit das System verfügbar bleibt und nicht mißbräuchlich verwendet wird.
Attacken gegen die *Verfügbarkeit des Systems* werden DoS (Denial of Service) Attacken genannt. Dabei bzw. bei der (unter bösartigen Hackern besonders beliebten) „erweiterten Variante" der Distributed DoS Attacken (DDoS) beanspruchen mehrere Rechner (in der Größenordnung von einigen 100, in seltenen berühmten Fällen (yahoo) noch mehr) gleichzeitig ein Service (etwa einen Webserver) des angegriffenen Systems, bis dieses unter der hohen Systemlast zusammenbricht und jeden weiteren Zugriff verbietet.
Der Schutz vor *missbräuchlicher Verwendung* ist deshalb besonders wichtig, weil der Administrator dafür verantwortlich ist, dass zB von seinem System keine Angriffe auf andere Rechner erfolgen. Viele Hacker verwenden mehrere Rechner als „Zwischenstationen", um ihre Spuren möglichst zu verwischen; es ist nur schwer möglich, über 5 und mehr Stationen hinweg Hinweise zurückzuverfolgen (besonders wenn die Rechner nicht im eigenen Netz, sondern über die ganze Welt verstreut sind).

Nicht überall wiegen die zwei genannten Bereiche – Schutz der Daten und des Systems – gleich schwer, denn zB in einem Netz mit langsamen Übertragungsraten (ev. über Modems) lässt sich ein DDoS kaum durchführen, andererseits bietet zB ein Universitätsinstitut kaum sensible Daten; allerdings ist eine Universität meistens über besonders schnelle Datenleitungen angebunden, was wiederum DDoS Angreifer anlockt. Andererseits ist es unter Umständen ein „prestigeträchtiges" Unterfangen den Webserver einer Universität lahmzulegen. Bei Banken, Technologiekonzernen und öffentlichen Einrichtungen (zB Polizei) ist dagegen der Schutz der geheimen Daten oberstes Gebot. Eine Vernachlässigung des Datenschutzes ist hier sogar strafbar. Auf jeden Fall wird das Vertrauen des Kunden in die jeweilige Organisation dauerhaft geschädigt und den Administrator kostet es vermutlich den Job.

Wer greift an?
Die Antwort ist so einfach, wie beunruhigend: Prinzipiell **jeder** ist ein potentieller Angreifer. Obwohl Paranoia im Allgemeinen ein gewisses soziales Problem darstellt, sollte sie bei einem Systemadministrator in höherem Masse ausgeprägt sein; egal ob der Kollege von nebenan, der noch eine alte „Rechnung“ zu begleichen hat, oder ein „Nachwuchshacker“ (neudeutsch „Script-Kiddie“), der gewöhnlich ein vorgefertigtes Paket verwendet, das eine bestimmte Sicherheitslücke ausnützt, bis zum Profi-Hacker, der die Bugs in den Diensten selbst entdeckt und ein Programm („Exploit“) schreibt, das diese Bugs ausnützt. Diesem Profi-Hacker kann vermutlich kaum ein Rechner standhalten, der nur irgendeinen Dienst im Internet anbietet. Das einzige Glück, wenn man es so nennen will, ist, dass sich solche Hacker wohl kaum mit kleineren Systemen abgeben, sondern gleich die „großen Kaliber“ und „lohnende Ziele“ in Angriff nehmen.

Wie wird angegriffen?
Hier unterscheidet man (hauptsächlich) zwischen drei Möglichkeiten:

1. Physischer Angriff
Dies ist der Weg, wie man am einfachsten unauthorisierten Zugriff auf einen Rechner erlangt, denn die Hardware an sich bietet kaum Schutz. Man hat hier mannigfaltige Möglichkeiten: Ausbau der Festplatte und Einbau in einen anderen Rechner; falls ein BIOS-Passwort gesetzt ist, Abklemmen bzw. Kurzschließen der BIOS-Batterie, wodurch alle BIOS-Einstellungen gelöscht werden; Starten von Linux in den Singleuser Modus oder als Init-Prozess die Shell angeben (am LILO-Prompt `linux init=/bin/sh` eintippen), danach kann man problemlos das `root` Passwort ändern bzw. die `/etc/shadow` auf eine Diskette kopieren und auf einem schnellen Rechner Benutzerpasswörter cracken (es erregt weit weniger Aufsehen, wenn man sich als normaler Benutzer einloggt, als wenn man das root-Passwort ändert).

Vorüberlegungen

- Was soll am System geschützt werden? (Daten/Verfügbarkeit)
- Vor wem schütze ich das System?
- Gibt es sensible Daten am System?
- Zugang von außen notwendig?
- (Wie viel) Internetzugang notwendig?
- Ausreichender Schutz durch Passwörter/Verschlüsselung?

Folie 143

Auch wenn der LILO-Prompt durch ein Passwort geschützt ist, könnte man immer noch von einer eigenen Bootdiskette starten (sofern die Boot-Reihenfolge im BIOS auf `A: C:` gestellt ist); wenn ein zweites Betriebssystem auf der Festplatte ist, für das man entweder einen Account hat bzw. das keine Accounts benötigt (zB Windows 98), kann man dort ein Programm installieren, welches das Dateisystem von Linux lesen kann und sich wiederum die `/etc/shadow` (oder beliebige andere Daten) auf eine Diskette kopieren.
Der größte Nachteil für den Hacker ist, dass er tatsächlich vor Ort sein muss, um den Rechner anzugreifen. Das sollte sich durch mechanische Barrieren lösen lassen (Zugang zu den Räumen kontrollieren, Server in eigene Räume sperren, Zugangskontrolle durch Fingerabdruck, Abbild der Regenbogenhaut, ...). Bei Workstations in zB einem PC Labor wird das kaum eine komfortable Möglichkeit darstellen, sodass man hier darauf angewiesen ist, einen gewissen Überblick über die Räumlichkeiten und die Hardware darin zu halten (Aufsicht, ...).

2. Lokaler Angriff
Darunter versteht man die potentiellen Attacken eines Benutzers, der einen Account am (ev. nicht im Netz befindlichen) System hat. Da man dem Benutzer im Allgemeinen vertraut, hat er vermutlich auch physischen Zugang zum Rechner, was auch das weite Feld der oben genannten Hacks offen lässt. Zusätzlich besteht hier die Gefahr, dass der User alle Dateien im System lesen kann, denen nicht explizit die Rechte entzogen wurden. Das ermöglicht auch das Ansehen von `/etc/passwd`, in der gewöhnlich alle vorhandenen Benutzer eingetragen sind. Damit hat man zumindest schon einen Überblick über die Benutzernamen und damit eventuell schon halb gewonnen, da viele User schlechte Passwörter wählen und das Durchprobieren von Namen etc. oft zum Erfolg führt.
Eine weitere große Gefahr stellen Bugs in Systemdiensten dar, die dem User mittels Exploits oft zu root Rechten verhelfen können; ein häufig gewählter Weg ist der „Buffer Overflow“. Wenn ein Programm/Dienst Benutzereingaben akzeptiert, dürfen diese meistens eine maximale Länge von 255 Zeichen aufweisen. Ist der eingegebene String länger als diese 255 Zeichen, wird dem normalerweise vom Programm mit einer

Fehlermeldung begegnet. Unterlässt der Programmierer allerdings einen Check der Stringlänge, überschreibt das, was nach dem 255. Zeichen steht, den Hauptspeicher, der sich nach dem String befindet. Dadurch kann es vorkommen, dass alles ab dem 256. Zeichen als Kommando interpretiert und ausgeführt wird; viele Daemonen/Dienste laufen aber mit dem SUID-Bit (siehe Abschnitt 12.1), d.h. mit root-Rechten. Wenn also ab dem 256. Zeichen etwa `/bin/sh` im Speicher steht, hat man Zugang zu einer root-Shell!

3. Angriffe aus dem Netz
Neben den oben erwähnten DDoS Attacken, für die der Angreifer keinerlei Rechte auf dem Zielsystem benötigt, gibt es noch zahlreiche andere Angriffe, die ebenso wie beim lokalen Angriff einen „Buffer Overflow“ ausnutzen. Häufig betroffen sind hier hochkomplexe Internet Dienste. Wir widmen dem weiten Feld der Netzwerksicherheit das gesamte Kapitel 18.

Schutz des Systems

Es sind in der Regel drei Sicherheitsformen zu gewährleisten:

- **Physische Sicherheit**: Schutz vor Angreifern, die direkten Zugriff auf die Hardware haben.
- **Lokale Sicherheit**: Schutz vor Angreifern, die einen lokalen Account besitzen.
- **Netzwerk Sicherheit**: Schutz vor Angreifern, die keinen lokalen Account besitzen, aber über ein Netzwerk Zugriff auf den Rechner haben.

Folie 144

12.5 Sicherheit – Lösungsansätze

Natürlich können hier keine vollständigen Lösungen für alle oben genannten Probleme genannt werden, dafür ist die Materie viel zu komplex. Vielmehr geben wir einen groben Überblick; jeder Administrator entwickelt mit der Zeit ein Gespür dafür, welche Dienste/Systeme er besonders schützen muss.
Von fundamentalster Wichtigkeit für jede Art von Sicherheit ist ein regelmäßiges Einspielen der vom Distributor herausgegebenen bugbereinigten Softwarepakete (Updates). Diese Aufgabe sollte möglichst automatisiert erfolgen (siehe Abschnitt 4.3).
Außerdem wird der sicheheitsbewusste Administrator relevante Mailinglisten der Distribution und/oder allgemeine Sicherheistmailinglisten (zB Bugtraq auf `http://online.securityfocus.com` oder CERT (`http://www.cert.org`)) verfolgen.

Physische Attacken
auf die Hardware frei zugänglicher Rechner kann man wie erwähnt kaum abwehren. Daher sollte man öffentlich aufgestellte Rechner (PC-Labor) nicht lange unbeaufsichtigt lassen und private Rechner immer dem Zugriff außen Stehender entziehen.
Geht man zur Software über, eröffnet sich ein weiteres Feld an Möglichkeiten: Auf jeden Fall muss man ein BIOS-Passwort setzen und die Bootreihenfolge so ändern, dass nur mehr von der Festplatte gebootet werden darf („C only“ oder „C:“). Hacken über Diskettenzugriff ist nun weitgehend beseitigt. Sodann geht man unter Linux daran, ein LILO-Passwort zu setzen (das erfolgt in der Datei `/etc/lilo.conf`, deren Rechte daher unbedingt `600` sein sollten), damit kann man nur mehr mit den voreingestellten Kernel-Parametern booten und keinen Init-Mode manuell einstellen, wenn man das Passwort nicht kennt.
Da Linux-Systeme im Normalfall wochen- bis monatelang ohne Reboot laufen, ist eine zusätzliche Option das Neustarten mittels Menü auf dem GUI oder CTRL+ALT+DEL zu unterbinden (keine Garantie gegen das Stromabschalten). Ersters erfolgt übder die Konfiguration des Desktopsystems (Loginmanager), letzteres in `/etc/inittab` durch Auskommentieren der entsprechenden Zeile, danach muss man noch `init q` ausführen, damit die Änderungen aktiviert werden.

Physische Sicherheit

Zugang zu Hardware erschweren:

- kein Zugang zur Hardware (Server)
- kein Zugang zu BIOS-Setup (kein Booten von Floppy!)
- Software Reboot disablen
  - Loginmanager des GUI
  - CTRL+ALT+DEL in `/etc/inittab`
    `#ca::ctrlaltdel:/sbin/shutdown -t3 -r now`
    mit `/sbin/init q` aktivieren

Folie 145

Lilo Passwort

Für die Übergabe von Bootparametern an den Kernel Authentifizierung verlangen.

In `/etc/lilo.conf` die Zeilen:

```
restricted
password=<password>
```

einfügen.
Danach Rechte für `/etc/lilo.conf` auf `600` setzen und

`lilo` ausführen.

Folie 146

Um **lokalen Attacken** möglichst entgegenzuwirken, sollte der Administrator sicherstellen, dass alle User ein gutes und kaum zu erratendes Passwort gesetzt haben, das auch eine höhere Verschlüsselung als normale Unix-Passwörter hat (durch Einsatz von md5-Verschlüsselung). Die Verwendung von `/etc/shadow` ist zum Glück unter beinahe allen Linux/Unix-Varianten inzwischen Standard, sodass unter Normalbedingungen kein User an ein Passwort geraten kann, das nicht für ihn bestimmt ist (vgl. Abschnitt 3.1).
Als Administrator selbst sollte man unbedingt den `su` bzw. `sudo` Befehl verwenden, statt sich als root einzuloggen (vgl. auch Abschnitt 1.3). Außerdem ist es beinahe tödlich, eine root Shell unbeabsichtigt offen zu lassen. Ein fortgeschrittener Anwender vorausgesetzt, der einige Minuten Zeit hat, kann sich im System einnisten. Einem „gehackten“ Rechner sollte man niemals wieder vertrauen und nach Möglichkeit sofort neu installieren und alle Passwörter ändern. Die Folgen der Nachlässigkeit von root lassen sich zwar durch ein Timeout der Shell, wenn eine bestimmte Zeit lang keine Tasten gedrückt werden, mindern (durch die Shell-Variable `TMOUT`, deren Wert man in Sekunden setzt), aber generell muss man natürlich trotzdem äußerst wachsam sein.
Durch die entsprechenden Dateien in `/etc/` lassen sich die Benutzer von `cron`, `at` etc. einschränken (vgl. 6.2). Speziell privilegierte Pseudobenutzer sollten niemals Cronjobs ausführen dürfen; tritt dies auf, so ist das ein relativ sicheres Zeichen, dass mit dem System etwas nicht stimmt.
Besonders gefährlich (zB bei den genannten „Buffer Overflows“) sind SUID-Programme, die von Administratoren mitunter auch als „Suizid“-Programme bezeichnet werden und das nicht ohne Grund. Nicht benötigte SUID-Dateien sind unbedingt zu deinstallieren und bei benötigten muss man das Augenmerk darauf legen, ob diese Dateien seit der Installation jemals verändert wurden (zB mittels `rpm`).

Was die **Netzwerksicherheit** betrifft, gilt hier allgemein, dass man auf jeden Fall alle nicht benötigte Dienste stoppen (zB `service httpd stop`) und per `chkconfig` aus der Liste der Dienste, die beim Hochfahren des Rechners gestartet werden, entfernen muss. Viele weitere Aspekte der Netzwerksicherheit besprechen wir im Kapitel 18.

Lokale Sicherheit (1)

- **Passwörter:**
  - sinnvolle Mindestlänge von 8 Zeichen
    mit: `# vi /etc/login.defs`
    Zeile ändern auf: `PASS_MIN_LEN 8`
  - **md5** und **shadow** verwenden
    (Default in vielen Disributionen)
    Konfig unter RedHat mit `authconfig`
- `su [-]` statt root login verwenden!
- Keine unbeaufsichtigte root Shell hinterlassen
  Umgebungsvariable `TMOUT=600` für `root` setzen

Folie 147

Lokale Sicherheit (2)

- Cronjobs nur für bestimmte Benutzer erlauben (`/etc/cron.allow`).
- Nicht benötigte SUID-Programme deinstallieren.
  Auflisten mit `find / -perm -4000`
- `su`-Befehl nur für Mitglieder der Gruppe `wheel` erlauben
  In `/etc/pam.d/su` folgende Zeile eintragen:
  `auth required /lib/security/pam_wheel.so group=wheel use_uid`
- ...

Folie 148