Study-Guide: Netzwerkdienste

Linux wird in der Praxis hauptsächlich als Serversystem eingesetzt, also als System, das Netzwerkdienste zur Verfügung stellen soll. Grundlegende Dienste werden in den folgenden sechs Kapiteln behandelt. Auch für diesen Abschnitt gilt, dass eine weiterführende Behandlung im nächsten Prüfungslevel erfolgt. Konfiguration und Verwaltug von inetd, xinetd und verwandten Diensten Verwendung und allgemeine Konfiguration von Sendmail Verwendung und allgemeine Konfiguration von Apache Richtiges Verwalten von NFS, smb und nmb Dämonen Einrichtung und Konfiguration grundlegender DNS-Dienste Einrichten von Secure Shell (OpenSSH)

Seite:	-= LinuxLernSystem =- (http://www.lpi-test.de)
Kurs:	LPIC-1 [102]
Buch:	Study-Guide: Netzwerkdienste
Gedruckt von:	André Scholz
Datum:	Dienstag, 1 November 2005, 10:50 Uhr

Inhaltsverzeichnis

- 1.113 - Netzwerkdienste
  - 1.113.1 - Konfiguration und Verwaltung von inetd, xinetd und verwandten Diensten
  - 1.113.2 - Verwendung und allgemeine Konfiguration von Sendmail
  - 1.113.3 - Verwendung und allgemeine Konfiguration von Apache
  - 1.113.4 - Richtiges Verwalten von NFS, smb und nmb Dämonen
  - 1.113.5 - Einrichtung und Konfiguration grundlegender DNS-Dienste
  - 1.113.7 - Einrichten von Secure Shell (OpenSSH)

1.113 - Netzwerkdienste

Linux wird in der Praxis hauptsächlich als Serversystem eingesetzt, also als System, das Netzwerkdienste zur Verfügung stellen soll. Grundlegende Dienste werden in den folgenden sechs Kapiteln behandelt. Auch für diesen Abschnitt gilt, dass eine weiterführende Behandlung im nächsten Prüfungslevel erfolgt.

- Konfiguration und Verwaltug von inetd, xinetd und verwandten Diensten
- Verwendung und allgemeine Konfiguration von Sendmail
- Verwendung und allgemeine Konfiguration von Apache
- Richtiges Verwalten von NFS, smb und nmb Dämonen
- Einrichtung und Konfiguration grundlegender DNS-Dienste
- Einrichten von Secure Shell (OpenSSH)

1.113.1 - Konfiguration und Verwaltung von inetd, xinetd und verwandten Diensten

Beschreibung: Prüfungskandidaten sollten in der Lage sein, die über inetd verfügbaren Dienste zu konfigurieren, tcpwrappers für das Erlauben oder Verweigern von Diensten auf Host-Ebene zu verwenden, Internetdienste manuell zu starten, stoppen und neu zu starten und grundlegende Netzwerkdienste einschließlich **Telnet** und **FTP** zu konfigurieren. Ebenfalls enthalten ist das Ausführen von Diensten unter einer anderen als der voreingestellten Benutzerkennung in `inetd.conf`.

Die wichtigsten Dateien, Bezeichnungen und Anwendungen:

- `/etc/inetd.conf`
- `/etc/hosts.allow`
- `/etc/hosts.deny`
- `/etc/services`
- `/etc/xinetd.conf`
- `/etc/xinetd.log`

Der inetd-Superserver

Bei der großen Anzahl von möglichen Servern, die auf einem Rechner laufen müssten, damit alle Internetdienste immer bereit sind, wäre es schnell soweit, daß der Speicher mit untätigen Servern überfüllt wäre. Damit das vermieden wird, gibt es einen "Superserver", der alle möglichen Portnummern abhört und bei Anfrage einen entsprechenden Server startet. Damit kann es ermöglicht werden, daß nur die Server laufen, die gerade arbeiten, obwohl alle Dienste verfügbar sind.

Der zuständige Daemon heißt **inetd** und wird in der Datei `/etc/inetd.conf` konfiguriert. Der Aufbau dieser Datei ist verhältnismäßig einfach, meist sind alle notwendigen Einträge schon vorhanden. Sie müssen nur jeweils auskommentiert werden, wenn sie nicht erwünscht sind. Das Format der Datei ist folgendermaßen organisiert:

```
Dienstname Socket-Typ Transportprotokoll Flags User Programm Argumente
```

Die Felder haben folgende Bedeutung:

Dienstname
: Hier steht der Name eines Dienstes, so, wie er in der Datei `/etc/services` eingetragen ist. Die ganze folgende Zeile bezieht sich darauf, daß ein Client versucht, auf dem Port zuzugreifen, der in `/etc/services` unter diesem Namen genannt ist.

Socket-Typ
: Gültige Werte für den Socket-Typ sind **stream, dgram, raw, rdm** und **seqpacket**, wobei in der Praxis meist nur die ersten beiden eine Rolle spielen. Über den Daumen gepeilt kann behauptet werden, daß jedes Programm, daß als Transportprotokoll TCP benutzt, hier ein **stream** eingetragen hat, jedes, das mit UDP arbeitet hat hier ein **dgram** benutzt.

Transportprotokoll
: Das hier eingetragene Transportprotokoll muß eins der in `/etc/protocols` angegebenen Protokolle sein. In der Regel spielen hier nur **tcp** und **udp** eine Rolle.

Flags
: An dieser Stelle gibt es nur zwei gültige Einträge, **wait** und **nowait**. Die Unterscheidung spielt nur für UDP-basierte Anwendungen eine Rolle. Alle anderen sollten hier ein **nowait** eingetragen haben. Wenn ein Datagram-Server (ein Server, der mit UDP arbeitet) nach der Versendung eines Datagrams den benutzten Socket löscht, so daß **inetd** weitere Anfragen auf diesem Socket empfangen kann, dann spricht man von einem *multi-threaded* Server und sollte hier **nowait** benutzen. Datagram-Server, die alle eingehenden Datagramme über einen Socket empfangen und letztendlich mit einem TimeOut reagieren, werden *single-threaded* genannt und benötigen hier den Wert **wait**. Typische Vertreter dieses Typs sind die **biff** und **talkd**

Daemonen.

Optional kann diesem Eintrag noch ein Maximalwert mitgegeben werden, der angibt, wieviele dieser Server maximal innerhalb von 60 Sekunden gestartet werden dürfen. Dieser Wert wird durch einen Punkt getrennt an **wait** oder **nowait** angehängt. Seine Voreinstellung ist 40.

User
Der User-Eintrag legt fest, unter welcher UserID das Serverprogramm laufen soll. Nachdem **inetd** selbst unter der UserID 0 läuft, kann hier verhindert werden, daß bestimmte Dienste auch unter dieser gefährlichen UID laufen. Optional kann diesem Eintrag ein durch Punkt vom Usernamen getrennter Gruppennamen folgen, der entsprechend die GID setzt.

Programm
Hier wird das Programm angegeben, das gestartet werden soll, wenn eine auf diesen Eintrag passende Anfrage eingeht. Es wird der komplette Pfad und Programmname angegeben. Wenn **inetd** diesen Dienst selbstständig anbietet, so steht hier `internal`.

Argumente
Die Argumente (Kommandozeilenoptionen) für das zu startende Programm. Normalerweise wird der Programmname selbst (Argument 0) mit angegeben. Auch hier wird das Wort `internal` angegeben oder der Eintrag wird leer gelassen, wenn **inetd** den Dienst intern zur Verfügung stellt.

So bedeutet die folgende Zeile also:

```
ftp     stream  tcp     nowait  root    /usr/sbin/in.ftpd       in.ftpd
```

Wenn eine Nachfrage nach dem Service *ftp* ankommt, (über den Sockettyp stream, mit dem Protokoll TCP) wird unter der UserID root das Programm `/usr/sbin/in.ftpd` ohne weitere Parameter gestartet und mit der Anfrage verbunden.

Die nötigen Portnummern bezieht *inetd* aus der Datei `/etc/services` .

Der **inetd** ließt diese Konfigurationsdatei nur beim Start. Wenn also Veränderungen daran vorgenommen wurden, so muß entweder der **inetd** neu gestartet werden, oder ihm muß ein HUP-Signal geschickt werden. Dieses Signal zwingt ihn, seine Konfigurationsdatei neu einzulesen.

Der TCP-Daemon (tcpwrapper)

Neben dem oben gezeigten Beispiel gibt es noch eine spezielle Form des Aufrufs über den inetd. In der Datei inetd.conf finden sich oft auch Zeilen wie die folgenden:

```
klogin    stream  tcp   nowait  root    /usr/sbin/tcpd  rlogind -k
eklogin   stream  tcp   nowait  root    /usr/sbin/tcpd  rlogind -k -x
kshell    stream  tcp   nowait  root    /usr/sbin/tcpd  rshd -k
```

Hier scheint auf den ersten Blick immer das gleiche Programm aufgerufen zu werden, nämlich `/usr/sbin/tcpd`. Erst als dessen Parameter kommen die eigentlichen Server an die Reihe. In der Tat ist es so, daß hier der TCP-Daemon (**tcpd**) aufgerufen wird, der dann erst die Server aufruft. Das klingt auf den ersten Blick unsinnig, aber es ermöglicht eine Einstellung, wer diesen Service benutzen darf.

Der **tcpd** überprüft anhand der Dateien `/etc/hosts.allow` und `/etc/hosts.deny` ob der jeweilige Host überhaupt berechtigt ist, diesen Dienst in Anspruch zu nehmen. Eine genaue Darstellung dieser Technik:

TCP-Wrapper

Ursprünglich alleine für die Verwendung mit **inetd** gedacht, hat sich das Prinzip des TCP-Wrappers inzwischen auch für Stand-Alone-Dienste wie **ssh** durchgesetzt. Worum geht es?

Wenn ein bestimmter Dienst aufgerufen wird, so kann er - anstatt direkt aufgerufen zu werden - durch das Programm **tcpd** aufgerufen werden. Dazu wird einfach statt dem zu startenden Dienst der **tcpd** aufgerufen und ihm wird der Name des zu startenden Dienstes als Parameter mitgegeben. Das Programm **tcpd** überprüft jetzt anhand von Einträgen in den Dateien

- `/etc/hosts.allow`
- `/etc/hosts.deny`

ob der Dienst von dem entsprechenden Host in Anspruch genommen werden darf. Analog überprüfen auch Stand-Alone-Dienste diese beiden Dateien.

Die Überprüfung erfolgt auf eine etwas eigenwillige Weise:

- Existiert ein passender Eintrag in der Datei /etc/hosts.allow, so wird Zugriff gegeben. Wenn nicht, dann
- Existiert ein passender Eintrag in der Datei /etc/hosts.deny, so wird kein Zugriff gegeben. Wenn nicht, dann
- wird Zugriff gegeben.

Die klassische Form der TCP-Wrapper

Das Format beider Dateien ist in der Handbuchseite hosts_access(5) genauestens dargestellt, es handelt sich um Zeilen des Formats:

```
Serverliste : Clientliste [: Shellkommando]
```

- Serverliste ist eine Liste von Servern (Programmnamen), oder Wildcards. Server werden mit ihrem Programmnamen - nicht über ihr Protokoll - angegeben, also z.B.: **in.telnetd**
- Clientliste ist eine Liste von einem oder mehreren Hostnamen, IP-Adressen, Suchmustern oder Wildcards,
- Shellkommando ist ein Kommando, das die lokale Shell ausführt, wenn der Dienst angefordert wurde und der Eintrag ausschlaggebend für seine Ausführung oder Nichtausführung war. Damit kann etwa eine Warnmeldung an root gegeben werden, wenn jemand versucht auf einen verbotenen Service zuzugreifen.

Als Suchmuster kommen zwei einfache Verfahren in Frage:

1. Beginnt ein Suchmuster mit einem Punkt (z.B. .foo.bar), so gelten alle Hostnamen als Treffer, deren Ende mit dem Muster übereinstimmt also etwa hal.foo.bar
2. Endet ein Suchmuster mit einem Punkt (z.B. 192.168.200.), so gelten alle Namen und Adressen als Treffer, deren erster Teil mit dem Muster übereinstimmt.

Als Wildcards können unter anderem folgende Werte verwendet werden:

ALL
: Die universelle Wildcard, alles gilt...

LOCAL
: Alle Hostnamen ohne Punkt (also lokale Namen) gelten.

UNKNOWN
: Passt auf alle Usernamen, die unbekannt sind und alle Hosts, deren Namen oder Adressen nicht bekannt sind. Wird gerne in /etc/hosts.deny verwendet.

KNOWN
: Passt auf alle Hosts und User, die bekannt sind

EXEPT
: Ist ein Operator, um zwei Listen auszuschließen (Liste1 EXEPT Liste2) also etwa ALL EXEPT UNKNOWN

Das Shellkommando sollte grundsätzlich mit einem & beendet werden, weil sonst auf seine vollständige Abarbeitung gewartet wird, bevor ein Service evt. gestartet wird. Je nach Kommando kann das natürlich dauern...

Als zusätzliche Platzhalter in Shellkommandos können folgende Konstrukte verwendet werden:

%a

Die IP-Adresse des anfordernden Hosts

%A
Die IP-Adresse des aufgerufenen Servers

%c
Clientinformationen - User@Host oder User@IP-Adresse oder nur IP-Adresse des Anrufers, je nach dem, wieviel Informationen zur Verfügung stehen.

%d
Der Name des Daemon-Prozesses, der angefordert wurde.

%h
Name (oder falls nicht vorhanden IP-Adresse) des Clients

%H
Name (oder falls nicht vorhanden IP-Adresse) des Servers

%p
Die ProzessID des Daemon-Prozesses

%s
Serverinformationen - Daemon@Hostname oder Daemon@Adresse oder nur Daemon, je nach dem, wieviel Informationen zur Verfügung stehen.

%u
Der Username des Anrufers oder "unknown"

%%
Das %-Zeichen

Die modernere Form der Wrapper

Moderne Systeme arbeiten heute zwar immer noch mit dem dargestellten Prinzip, bieten aber auch die Möglichkeit, alle Einstellungen in nur einer der beiden Dateien vorzunehmen. Statt der Handbuchseite `hosts_access(5)` wird diese Methode in `hosts_options(5)` beschrieben. Der Aufbau der beiden Dateien sieht jetzt folgendermaßen aus:

```
Serverliste : Clientliste [: Option ] [: Option ...]
```

Statt einem Shellkommando können also Optionen angegeben werden. Die wichtigsten Optionen sind:

ALLOW
Erlaubt den angegebenen Dienst für die angegebenen Clients.

DENY
Verbietet den angegebenen Dienst für die angegebenen Clients.

spawn *Shellkommando*
Führt das angegebene Shellkommando aus. Wie in der klassischen Form werden die oben beschriebenen Ersetzungen vorgenommen.

twist *Shellkommando*
Führt das angegebene Shellkommando aus und schickt seine Ausgaben an den Client, anstatt den gewünschten Dienst zu starten. Wie in der klassischen Form werden die oben beschriebenen Ersetzungen vorgenommen.

user *Username*[.*Gruppe*]
Startet den angegebenen Dienst unter der angegebenen User (und optional Gruppen) ID.

Der Vorteil dieser Methode liegt darin, daß alle Einstellungen in einer Datei vorgenommen werden können. Da die Optionen ALLOW und DENY zur Verfügung stehen, können in `/etc/hosts.allow` auch Verbote ausgesprochen werden und umgekehrt.

Aber Achtung: Es gilt immer noch die oben angegebene Reihenfolge. Es nützt also nichts, einem Host etwas zu erlauben, ohne allen anderen es zu verbieten. Die Einträge werden der Reihe nach gelesen und der erste passende wird benutzt. Um also nur dem Rechner `marvin.foo.bar` zu erlauben, den FTP-Daemon zu benutzen, müssten wir schreiben:

```
in.ftpd : marvin.foo.bar : ALLOW
in.ftpd : ALL : DENY
```

Wenn der Rechner `marvin.foo.bar` jetzt den Dienst anfordert, greift die erste Zeile und der Zugriff wird gewährt. Versucht aber ein anderer Rechner den Zugriff, so greift die erste Zeile nicht, dafür aber die zweite - der Zugriff wird verwehrt.

Der xinetd-Server

Der Server **xinetd** ist ein sicherer Ersatz für **inetd**. Er bietet eine eingebaute Zugriffskontrolle und eine direkte Unterstützung der Dateien `/etc/hosts.allow` und `/etc/hosts.deny` ohne den Umweg über **tcpd**. Außerdem werden noch viele weitere Verbesserungen angeboten wie etwa

- Beschränkung der Verbindungen entweder generell, oder pro Dienst oder pro Client.
- Beschränkung von Verbindungen anhand der Tageszeit.
- Dienste können an bestimmte IP-Adressen gebunden werden, so daß z.B. interne Clients andere Server-Programme nutzen als externe Clients, obwohl sie den selben Dienst aufgerufen haben.
- Schutz vor *denial of service* Angriffen.
- Ausgiebige Log-Möglichkeiten (auch unabhängig von **syslogd**).
- Umleitung von TCP-Verbindungen zu einem anderen Rechner, der selbst nicht von außen erreichbar sein muß. Damit können Server hinter einem Masquerading-Server auch von außen benutzt werden.
- Volle Unterstützung von IPv6.
- Interaktion mit dem User des Clients (Meldungen bei erfolgtem oder gescheitertem Zugriff).

Die Konfiguration des **xinetd** erfolgt analog zu seinem Vorgänger in der Datei `/etc/xinetd.conf`. Allerdings hat diese Datei einen vollkommen anderen Aufbau als `/etc/inetd.conf`. Für UmsteigerInnen gibt es ein spezielles Shellscript, das eine bestehende `inetd.conf` Datei in das neue Format konvertiert. Dieses Programm ist ein Perl-Script (`xconv.pl`) und wird mit **xinetd** mitgeliefert.

Die Konfigurationsdatei von **xinetd** beginnt mit einem Abschnitt für die Voreinstellungen, der *default section*. Die hier angegebenen Attribute werden von jedem Dienst genutzt, den **xinetd** verwaltet. Nach diesem Abschnitt folgen für jeden einzelnen zu startenden Dienst ein eigener Abschnitt. Wenn in einem solchen dienstspezifischen Abschnitt Angaben gemacht werden, die denen der *default section* widersprechen, so gelten die des dienstspezifischen Abschnitts.

Die typische Form einer *default section* ist folgendermaßen aufgebaut:

```
defaults
{
    Attribut Operator Wert(e)
    ...
}
```

Alle weiteren Abschnitte beginnen mit dem Schlüsselwort `service`, dem der Name des Dienstes (wie er in `/etc/services` eingetragen ist) folgt.

```
service Dienstname
{
    Attribut Operator Wert(e)
    ...
}
```

Es gibt drei unterschiedliche Operatoren, mit denen Attribute mit Werten versehen werden:

=
: Setzt einen Wert für ein Attribut.

+=

Fügt einem Attribut einen Wert zu den bereits bestehenden Werten hinzu.

-=

Entfernt einem Attribut den angegebenen Wert.

Es existieren sehr viele möglichen Attribute, deren Beschreibung den Rahmen hier sprengen würde. Ein Beispiel für eine solche Datei sagt hier mehr als eine endlose Beschreibung:

```
defaults
{
  instances       = 15
  log_type        = FILE /var/log/xinetd.log
  log_on_success  = HOST PID USERID DURATION EXIT
  log_on_failure  = HOST USERID RECORD
  only_from       = 192.0.0.0/8

  disabled = shell login exec comsat
  disabled += telnet ftp
  disabled += name uucp tftp
  disabled += finger systat netstat
}

service ftp
{
  socket_type  = stream
  wait         = no
  user         = root
  server       = /usr/sbin/in.ftpd
  server_args  = -l
  instances    = 4
  access_times = 7:00-12:30 13:30-21:00
  nice         = 10
  only_from    = 192.168.1.0/24
}
```

In der `defaults`-Section definieren wir hier das Attribut `instances` mit dem Wert 15. Das bedeutet, daß grundsätzlich nicht mehr als 15 Dienste gleichzeitig gestartet werden können. Später in der Definition des FTP-Dienstes setzen wir den Wert 4 für dieses Attribut. In diesem Fall bezieht es sich auf die Menge der gleichzeitigen FTP-Verbindungen.

Die Angabe `log_type = FILE /var/log/xinetd.log` bestimmen wir, daß nicht der **syslogd** benutzt werden soll, sondern **xinetd** direkt in die angegebene Datei schreibt. Anstatt diesem Eintrag wäre auch folgender möglich gewesen:

```
log_type = SYSLOG daemon info
```

Dann wären alle Meldungen über den Syslog-Daemon geschrieben worden und zwar mit der Herkunft `daemon` und der Priorität `info`.

Die beiden Angaben `log_on_success` und `log_on_failure` legen fest, was alles in das Logbuch aufgenommen werden soll, wenn ein Zugriff erfolgreich war oder abgelehnt wurde.

Die Angabe `only_from = 192.0.0.0/8` bestimmt, daß grundsätzlich (wenn beim entsprechenden Dienst nichts anderes angegeben wurde) nur die Rechner Zugriff haben, die auf die angegebene Adresse passen. Das `/8` bestimmt, daß nur die ersten 8 Bit der Adresse gewertet werden. In unserem Beispiel werden also alle Rechner Zugriff bekommen, deren erstes Adressenbyte 192 ist.

Die folgenden Zeilen der *default section* schalten die angegebenen Dienste mit dem Attribut **disabled** komplett ab.

Im Abschnitt FTP werden jetzt die einzelnen Attribute für den Dienst gesetzt. Die ersten vier Einträge entsprechen dem, was auch in der alten **inetd** Konfiguration eingetragen war. Interessant sind hier noch die Angaben über die erlaubten Tageszeiten, in denen der Dienst zur Verfügung steht und der gesetzte NICE-Wert, unter dem der Daemon laufen soll. Zugreifen dürfen nur Rechner, deren Adresse mit 192.168.1 beginnt.

Eine Liste aller Attribute und ihrer Bedeutungen sind auf der Handbuchseite von xinetd.conf(5) zu finden. Eine genaue Beschreibung der Formate der Logdateien sind in der Handbuchseite xinetd.log(5) nachlesbar.

1.113.2 - Verwendung und allgemeine Konfiguration von Sendmail

Beschreibung: Prüfungskandidaten sollten in der Lage sein, einfache Einstellungen in den Sendmail Konfigurationsdateien vorzunehmen (einschließlich des "Smart Host" Parameters, wenn notwendig), Mail-Aliases anzulegen, die Mail-Queue zu verwalten, Sendmail zu starten und zu stoppen, Mail-Weiterleitung zu konfigurieren und allgemeine Sendmail-Probleme zu lösen.

Die wichtigsten Dateien, Bezeichnungen und Anwendungen:

- `/etc/sendmail.cf`
- `/etc/aliases` oder `/etc/mail/aliases`
- `/etc/mail/*`
- `~/.forward`

Die Einrichtung eines Mailservers mit sendmail gilt als eines der kompliziertesten Kapitel der Unix-Serverinstallation. Es existiert ein tausendseitiges Buch alleine über die Sendmail-Konfiguration, es gibt auch die Weisheit:

> *Ein richtiger Unix-Systemadministrator ist man erst, wenn man einmal eine Sendmail Konfigurationsdatei geschrieben hat.*
>
> *Ein richtiger Unix-Systemadministrator wird sich aber hüten, es ein zweites Mal zu tun.*

Aber so schlimm ist es heute nicht mehr, keine Angst. Die modernen Sendmail Versionen bringen heute ausgesprochen leicht zu handhabende Konfigurationsdateien mit, die die Arbeit mit Sendmail erleichtern. Wenn also kein zu komplizierter Mailserver aufgesetzt werden soll, dann können wir uns das Leben sehr einfach machen.

Die LPI102 Prüfung verlangt kein tiefergehendes Wissen über die Geheimnisse der **sendmail** Konfiguration, es genügen die notwendigen Informationen, um einen einfachen Mailserver einzurichten.

Die Architektur von E-Mail

Das SMTP Prinzip verteilt seine Aufgaben auf verschiedene Programme. Die zwei grundlegenden Programmtypen sind dabei

Der Mail User Agent (MUA)
: Das Programm, mit dem der User seine Mails liesst und erstellt. Es existieren hunderte verschiedener MUAs, für jedes Betriebssystem. Beispiele hierfür sind Eudora, OutlookExpress, Outlook auf der Windows-Seite und elm, pine, xfmail, kmail, mutt auf der Linux-Seite.

Der Mail Transfer Agent (MTA)
: Das Programm, das die abgehende Mail von einem beliebigen MUA erhält und sich darum kümmert, daß diese Mail weitergeleitet wird. Andererseits das Programm, das - bei permanenter Internet-Anbindung - die ankommenden Mails von anderen MTAs entgegennimmt und lokal abspeichert, damit die MUAs darauf zugreifen können. Der verbreitetste und mächtigste MTA ist das Programm sendmail.

Hier geht es prinzipiell um die Konfiguration von MTAs - konkret um die Konfiguration von sendmail. Die genannte Aufteilung der Aufgaben in Userprogramm und Versendeprogramm machen eine Unterscheidung hinsichtlich der Netzanbindung nötig. Es ist ein erheblicher Unterschied, ob unser Mailserver eine permanente Netzanbindung hat, oder nicht.

In einem lokalen Netz kann auf jeder Unix-Workstation ein sendmail-Daemon laufen, jedoch hat nur einer der Rechner die tatsächliche Anbindung ans Internet. Das bewirkt, daß die anderen Rechner ihre Mails an den einen

mit Netzanbindung schicken, erst der versendet sie dann weiter ins Internet. Diese Architektur hat in vielen Fällen Vorteile, zum Beispiel

- Bei temporärer Netzanbindung bauen nicht alle User dauernd Verbindungen ins Internet auf, sondern schicken ihre Mails an einen Rechner, der die Mails regelmäßig weiterleitet.
- Bei einer Firewalleinrichtung kann eingestellt werden, daß nur SMTP-Pakete von und zu diesem einen Rechner durchgelassen werden.

Im Regelfall läuft sendmail als stand-alone Dienst. Das heißt, auch bei temporärer Anbindung ist der Daemon ständig im Speicher. Das ist notwendig, damit ein Rechner Mails von anderen MTAs empfangen kann. Ob sendmail die ausgehenden Mails gleich losschickt, oder wohin es sie weiterleitet, das muss konfiguriert werden.

Die zentrale sendmail-Konfigurationsdatei ist /etc/sendmail.cf. Diese Datei selbst zu bearbeiten oder gar zu erstellen ist ausgesprochen kompliziert - das Anfangs angeführte Zitat bezieht sich eben darauf. Wir werden hier nicht die komplette Konfiguration von **sendmail** behandeln, sondern ausschließlich kleine Veränderungen an dieser Datei vornehmen.

Grundlagen der Mailverzeichnisse im Linux-System

Jedes Unix/Linux-System hat zwei verschiedene Mail-Verzeichnisse. Zum einen existiert eine Warteschlange für ausgehende Mails (meist `/var/spool/mqueue` oder `/var/mqueue`) und zum anderen ein Verzeichnis `/var/spool/mail`, in dem eine Datei für jeden User existiert, die seine angekommene Mail bereithält. Die MUAs und MTAs arbeiten beide mit diesen Verzeichnissen.

Ein MUA, mit dessen Hilfe eine Mail versendet wird, legt diese abgehende Mail mitsamt aller dazu nötigen Headerinformation (von wem kommt die Mail, an wen geht die Mail usw.) in das Spoolverzeichnis für die ausgehende Mail. Der MTA arbeitet diese Warteschlange in regelmäßigen Abständen ab und verschickt die Mails ins Internet - entweder direkt an die empangenden Rechner oder an einen Mailserver des Providers, der dann die Weiterversendung übernimmt.

Umgekehrt bekommt der MTA eingehende Mails aus dem Netz und sortiert sie in die Eingangspostfächer (z.B. `/var/spool/mail/hans`). Beim nächsten Aufruf eines MUA findet dieser im jeweiligen Verzeichnis dann die neue Mail vor und kann sie darstellen.

Die Programme, die unter Unix ständig dafür sorgen, daß einem User mitgeteilt wird, daß er neue Mail hat (biff/xbiff), nützen ebenso diese Verzeichnisse. Der Vorteil an diesem Prinzip liegt in der Tatsache, daß das Mailsystem so integraler Bestandteil des Systems selbst ist, und nicht ein aufgepfropftes Anwenderprogramm. Nur deshalb können z.B. Programme wie **cron** oder **at** ihre Ausgaben als Mail an den User schicken, der den Auftrag gegeben hatte.

Um den Inhalt der Warteschlange anzusehen, existiert der Befehl **mailq**, der in Wahrheit nur ein symbolischer Link auf **sendmail** ist.

Starten von sendmail

Die zentrale Aufgabe von **sendmail** ist es, neben dem Empfang von Mails über SMTP, die Warteschlange für die ausgehende Mail (mqueue) zu bearbeiten. Dazu existieren zwei völlig unterschiedliche Architekturen, entweder ist der Rechner, auf dem **sendmail** läuft tatsächlich mit einer festen IP-Adresse ans Internet angebunden und von überall her unter einem bestimmten Namen zu erreichen, oder er ist nur temporär ans Internet angebunden und somit nur für das Versenden von Mails gedacht.

Starten des Mailservers mit permanenter Netzanbindung

Auf einem Rechner mit permanenter Internet-Anbindung hat **sendmail** in der Regel zwei Aufgaben:

- Empfang von Mails aus dem Internet über SMTP
- Versand von ausgehenden Mails über SMTP

Zumindestens die erste Aufgabe bedingt es, daß **sendmail** permanent als Daemon im Speicher liegt und arbeitet. Nur so ist es gewährleistet, daß eingehende Mails tatsächlich jederzeit angenommen werden können. Der Start von **sendmail** benötigt hier zwei wesentliche Parameter, der erste gibt an, daß das Programm als Daemon gestartet werden soll und der zweite stellt die Frequenz ein, in der **sendmail** die Warteschlange bearbeiten soll (in diesem Beispiel 30 Minuten):

```
sendmail -bd -q30m
```

Auf diese Weise wird **sendmail** alle 30 Minuten die Mailqueue lesen und anstehende Aufträge (ausgehende Mails) versenden. Die eingehenden Mails werden jederzeit empfangen.

Starten des Mailservers mit temporärer Netzanbindung

In diesem Fall wird **sendmail** nur für die Versendung von Mail herangezogen. Das Empfangen wird wahrscheinlich über POP3 oder IMAP geregelt. In diesem Fall muß **sendmail** nur regelmäßig aufgerufen werden, um die ausgehenden Mails zu versenden. Dazu wird der Parameter `-bd` weggelassen, um zu verhindern, daß das Programm als Daemon gestartet wird. Der Parameter `-q` wird ohne Angabe eines Zeitintervalls angegeben. Das führt dazu, daß **sendmail** die Mailqueue abarbeitet und anschließend beendet wird.

```
sendmail -q
```

Dieser Befehl kann jetzt z.B. in regelmäßigen Abständen von cron aufgerufen werden.

Um die interne Weitergabe von Mails zu realisieren, wird meist jedoch auch auf Rechnern ohne permanente Anbindung ein Sendmail-Daemon gestartet, der allerdings nicht die Warteschlange abarbeitet. Die Versendung geschieht mit dem oben genannten Befehl über cron gesteuert, der Aufruf von **sendmail** heißt dann

```
sendmail -bd
```

So ist die interne Funktionalität weiter gewährleistet, ohne daß in regelmäßigen Abständen die Queue abgearbeitet wird.

Mailaliase

sendmail hat die Möglichkeit, Alias-Adressen zu bilden. Das sind E-Mail Adressen, die in Wahrheit keinem normalen User zugeordnet sind, sondern auf andere User verweisen. Häufig benutzte solche Adressen sind z.B.

- webmaster@...
- abuse@...
- ftpadmin@...

In den seltensten Fällen sind das wirkliche User des Systems. Normalerweise werden einfach Aliase angelegt, die diese Adressen dem entsprechenden User zuweisen, der den Job übernommen hat. Dazu kennt **sendmail** die Datei `/etc/aliases` oder manchmal auch `/etc/mail/aliases`.

Das Format dieser Datei ist sehr einfach, pro Zeile wird ein Alias definiert:

Aliasname: Username

Um die drei obigen Beispiele bestimmten Usern zuzuordnen könnten in einer solchen Datei also folgende Einträge stehen:

```
webmaster: root
abuse:     root
ftpadmin:  hans
```

Alle Mails an `webmaster@`*`mydomain.de`* und `abuse@`*`mydomain.de`* würden also an den User `root@`*`mydomain.de`* weitergeleitet, während Mails an `ftpadmin@`*`mydomain.de`* an `hans@`*`mydomain.de`*

weitergegeben würden. *mydomain.de* würde natürlich durch die entsprechende lokale Domain ersetzt.

Es ist auch möglich, bestimmte Adressen nicht an andere Adressen zu versenden, sondern an bestimmte Programme weiterzuleiten. In diesem Fall werden Mails, die an diese Adresse gingen an ein angegebenes Programm weitergepiped. So können z.B. Mailinglists organisiert werden. Statt einem Usernamen wird dann ein Programmname angegeben, dem ein Pipe-Symbol vorangestellt wurde. Programmname und Pipesymbol müssen in doppelte Anführungszeichen gesetzt werden:

```
Aliasname: "|Pfad/zu/Programm Optionen"
```

Nach jeder Veränderung der Alias-Datei muß das Programm **newaliases** aufgerufen werden, um diese Aliase für **sendmail** bekannt zu machen. Wie **mailq** ist auch **newaliases** nur ein symbolischer Link auf **sendmail**.

Mails an andere Adressen weiterleiten

Einen ähnlichen Mechanismus wie das Setzen von Aliasen bietet die Datei `~/.forward` im Homeverzeichnis eines Users. Diese Datei enthält entweder E-Mail-Adressen, Dateinamen oder Programmnamen, an die eingehende Mails des Users, in dessen Homeverzeichnis die Datei liegt, weitergeleitet werden.

Die Syntax der drei möglichen Einträge ist einfach:

E-Mail-Adressen
Enthält eine `.forward` Datei eine Zeile mit einer einfachen E-Mail-Adresse, so wird jede eingehende Mail des Users an diese Mailadresse weitergeleitet. Die E-Mail-Adresse wird ohne Anführungszeichen angegeben. Enthält sie keine Domain-Angabe, so wird eine lokale Adresse angenommen. Wenn z.B. auf einem Rechner der Useraccount `hans` der Normallogin des Systemverwalters ist, so kann der Systemverwalter in sein Homeverzeichnis eine `.forward` Datei anlegen, mit dem Eintrag

```
hans
```

Jede Mail an `root` wird jetzt an Hans weitergeleitet. So empfängt der Systemverwalter seine Mails auch, wenn er sich als `hans` anmeldet.

Dateinamen
Wenn eingehende Mails in eine bestimmte Datei kopiert werden sollen, so wird der Name der Datei in Anführungszeichen in `.forward` angegeben. Die angegebene Datei entspricht vom Aufbau her einer normalen Mailbox-Datei, wie sie normalerweise in `/var/spool/mail` liegt. Alle Mails werden an diese Datei angehängt.

```
"/var/spool/mail2/hans"
"./Mail/meinemail"
```

Die Angaben einer Datei werden immer entweder als kompletter Pfad zu der Datei oder als Pfad mit führendem Punkt angegeben. Im zweiten Fall bezieht sich der Pfad auf das Homeverzeichnis des jeweiligen Users.

Bei der Angabe von normalen Pfaden muß darauf geachtet werden, daß der User Schreibrecht auf die entsprechende Datei besitzt.

Programmnamen
Soll eine Mail an ein bestimmtes Programm weitergegeben werden, so wird der Programmname in die Datei `.forward` in Anführungszeichen mit führendem Pipe-Symbol angegeben.

```
"| /Pfad/zu/Programm"
```

Die Mail wird dann an das angegebene Programm gepiped, das heißt, das Programm muß Eingaben von der Standard-Eingabe verarbeiten können. Die Angabe eines Programms erfolgt mit vollem Pfad.

Mit Hilfe dieses Mechanismuses ist es möglich, daß auch Normaluser ihre Mails weiterleiten oder Mailinglisten

aufbauen, auch wenn sie keinen Zugriff auf `/etc/aliases` besitzen.

Ausgehende Mailserver

Wenn Mails von **sendmail** versendet werden sollen, dann gibt es zwei Möglichkeiten, wie das organisiert werden kann. Entweder verschickt **sendmail** die Mails direkt an die angegebenen Adressen, oder alle Mail wird an einen sogenannten *Smarthost* weitergeleitet, der sich dann um das weitere Versenden kümmert. Die meisten Internet-Provider stellen einen solchen SMTP-Server für ihre Kunden zur Verfügung.

Die Versendung ohne Smarthost bedingt eine funktionierende DNS-Installation, da **sendmail** sowohl die Adressen der ausgehenden Mails auflösen muß, als auch von den Empfänger-Rechnern wieder angesprochen werden muß.

Wird stattdessen aber ein Smarthost angegeben, so sendet **sendmail** alle ausgehende Mail an diesen Rechner, der dann natürlich die selben Ansprüche erfüllen muß, wie ein lokaler Rechner, der keinen Smarthost verwendet, er muß eine funktionierende DNS-Anbindung haben.

Der Eintrag für den Smarthost findet in der zentralen Konfigurationsdatei von **sendmail** statt, die Datei `/etc/sendmail.cf`. Hier muß die Einstellung

```
DSRechnername
```

eingetragen sein. Soll stattdessen kein solcher Rechner verwendet werden, so wird der Parameter *Rechnername* einfach weggelassen.

```
DS
```

Die Dateien in /etc/mail

Das Verzeichnis `/etc/mail` enthält verschiedene Dateien, die Einstellungen für **sendmail** ermöglichen. Die meisten davon existieren in zwei Formaten. Zum einen existiert eine einfache Textdatei, in der wir die Einstellungen vornehmen können, zum anderen eine Datei, die den selben Namen, aber die Endung `.db` trägt. Sie enthält die Informationen, die **sendmail** selbst verarbeitet. Nach jeder Änderung einer der Textdateien ist es notwendig, diese Datei in das Datenbankformat (`.db`) zu konvertieren. Dazu wird folgender Befehl eingegeben:

```
makemap hash -f /etc/mail/datei.db < /etc/mail/datei
```

Statt *datei* wird natürlich der Name der zu konvertierenden Datei angegeben.

/etc/mail/access

Diese Datei regelt den Zugriff auf den Mailserver. Also die Frage, welcher Rechner darf ausgehende Mail an diesen Server schicken, damit er sie weiterleitet. Ein paar Beispiele:

```
127                     RELAY
192.168.100.123         OK
mydomain.de             OK
```

Alle Rechner, deren Adresse mit 127 anfangen (das sind nur wir selbst) senden ihre Mail grundsätzlich (RELAY) über diesen Rechner. Der Rechner 192.168.100.123 darf jederzeit Aufträge an diesen Rechner weitergeben und alle Mitglieder der Domain mydomain.de ebenfalls. Sonst werden alle Aufträge zur Weiterleitung abgelehnt.

Wie oben erwähnt muß diese Datei nach jeder Veränderung in das Datenbankformat konvertiert werden.

/etc/mail/genericstable

Diese Datei dient der Übersetzung ausgehender E-Mail Adressen in weltweit gültige. Bin ich z.B. auf dem Server

der Systemverwalter, so lautet meine E-Mail Adresse intern z.B. `root@server.local`. Würde meine ausgehende Mail nun diese Adresse als Absender tragen, bekäme ich wohl nie eine Antwort. Denn mit `root@server.local` kann im Internet niemand etwas anfangen. Dort gilt meine Adresse `echter.name@echter.provider.org`. In der Datei genericstable stehen jetzt Umrechnungstabellen für solche Fälle:

```
root                    echter.name@echter.provider.org
root@server.local       echter.name@echter.provider.org
hans                    hMueller@gmx.de
hans@server.local       hMueller@gmx.de
```

Auch diese Datei muß nach einer Veränderung in das Datenbankformat gebracht werden.

/etc/mail/mailertable

Hier werden bekannte Rechner genannt, samt den Methoden, mit denen Mail an sie verschickt wird. Dabei können auch Tricks angewandt werden, wie etwa der Domainname, der an einen bestimmten Rechner weitergeleitet wird:

```
marvin.local.de        smtp:marvin.local.de
hal.local.de           smtp:hal.local.de
golem.local.de         smtp:golem.local.de
local.de               smtp:marvin.local.de
```

Alle Aufträge an bestimmte Rechner (marvin, hal und golem) gehen an die Rechner selbst via smtp. Alle Adressen, die keinen Rechner sondern nur eine Domain nennen (z.B. `hans@local.de`) werden an marvin weitergeschickt.

Auch diese Datei muß nach einer Veränderung in das Datenbankformat gebracht werden.

/etc/mail/virtusertable

Hier werden - ähnlich wie bei aliases - zwei verschiedene Adressen miteinander verknüpft. Dabei darf allerdings über Domaingrenzen gegangen werden, so daß - sollte ein Rechner mehrere Domains verwalten - hier Umleitungen zur jeweils passenden Adresse vorgenommen werden können.

```
efka@local.de root@golem.local.de
```

Die Adresse `efka@local.de` wird in Wahrheit an `root@golem.local.de` weitergegeben.

Auch diese Datei muß nach einer Veränderung in das Datenbankformat gebracht werden.

1.113.3 - Verwendung und allgemeine Konfiguration von Apache

Beschreibung: Prüfungskandidaten sollten in der Lage sein, einfache Einstellungen in den Apache Konfigurationsdateien vorzunehmen, **httpd** zu starten, anzuhalten und neu zu starten und das automatische starten von **httpd** bei Systemstart einzurichten. Dies beinhaltet nicht fortgeschrittene Konfiguration von Apache.

Die wichtigsten Dateien, Bezeichnungen und Anwendungen:

- **apachectl**
- **httpd**
- `httpd.conf`

Die Konfiguration des Webservers **apache** ist ein weites Feld, das problemlos ganze Bücher füllen kann. An dieser Stelle wird aber nur ein Grundwissen über den Webserver und seine Konfiguration gefordert, das relativ schnell erlernbar ist.

Konfigurationsdateien und Verzeichnisse des Webservers

Der Webserver **apache** benutzte ursprünglich drei Konfigurationsdateien, die in der folgenden Reihenfolge abgearbeitet wurden:

httpd.conf
: Die zentrale Konfigurationsdatei des Webservers. Hier werden alle Einstellungen vorgenommen, die den Server selbst, den Hauptserver und eventuell existierende virtuelle Server angehen.

srm.conf
: In dieser Datei wurden die `ResourceConfig` Anweisungen des Webservers plaziert. Seit einiger Zeit wird diese Datei leer ausgeliefert und alle Einträge werden in `httpd.conf` vorgenommen.

access.conf
: In dieser Datei wurden die Anweisungen des Webservers plaziert, die mit Zugriffsrechten zu tun hatten. Seit einiger Zeit wird auch diese Datei leer ausgeliefert und alle Einträge werden in `httpd.conf` vorgenommen.

Die beiden Dateien `srm.conf` und `access.conf` sind zwar noch vorhanden, werden jedoch nicht mehr benötigt, da alle Einstellungen heute in der Datei `httpd.conf` vorgenommen werden.

Standardmäßig erwartete der Webserver seine Konfigurationsdateien in `/usr/local/etc/httpd/conf`. Dieses Verzeichnis entspricht aber nicht dem Dateisystem-Standard von Linux. Aus diesem Grund wird heute der Webserver in der Regel über eine Kommandozeilenoption `-f` *`Dateiname`* gestartet, die ihm ein anderes Verzeichnis für die Konfigurationsdatei angibt. In der Regel ist das Verzeichnis entweder `/etc/httpd` oder `/etc/apache`. Dort liegen dann alle Konfigurationsdateien des Webservers.

Neben diesem Verzeichnis gibt es noch zwei weitere, die für den Webserver eine große Rolle spielen.

ServerRoot
: Das Verzeichnis, in dem die Dateien liegen, die für den Gebrauch des Webservers wichtig sind. Das sind z. B. die Icons, die er für Dateien und Verzeichnisse benutzt oder die CGI-Programme.

DocumentRoot
: Das Verzeichnis, das die HTML-Dateien enthält, die der Webserver der Allgemeinheit anbieten soll.

Die Festlegung, welche Verzeichnisse für diese beiden Aufgaben benutzt werden sollen, wird in der Konfigurationsdatei `httpd.conf` gemacht.

Die zentrale Konfigurationsdatei des Webservers

Auch heute noch versucht der Server alle drei oben genannten Dateien abzuarbeiten, aber die beiden Dateien `access.conf` und `srm.conf` sind in der Regel einfach leer. Alles, was früher dort untergebracht wurde, finden wir heute in der zentralen Konfigurationsdatei `httpd.conf`.

Diese Datei ermöglicht es, den Webserver komplett zu konfigurieren, von der Einstellung, auf welchen Port er hören soll, bis hin zur Angabe welcher User welche Verzeichnisse sehen darf. Die standardmäßige Konfigurationsdatei des Webservers enthält bereits eine wohldurchdachte und sehr gut kommentierte Konfigurationsdatei, die entsprechend angepasst werden kann.

Grunsätzlich ließt der Webserver diese Datei beim Start. Das heißt, wenn in dieser Datei Veränderungen vorgenommen werden sollen, dann muß der Server danach neu gestartet werden. **Bitte nochmal zur Definition: Server meint hier das Programm und nicht den Rechner!**

Die Konfigurationsdatei des Webservers kennt hunderte verschiedener Anweisung, die hier nicht alle dargestellt werden können. Wir werden uns hier also auf die wichtigsten Anweisungen beschränken. Die Konfigurationsdatei ist in drei Bereiche aufgeteilt: Die globalen Einstellungen, die Einstellungen des Hauptservers und die Einstellungen für die virtuellen Server.

Globale Einstellungen

Diese Einstellungen beziehen sich auf die grundsätzliche Funktionalität des Servers selbst. Folgende Einstellungen sind wichtig:

`ServerType standalone|inetd`
: Diese Anweisung legt fest, ob der Webserver standalone oder durch inetd gestartet werden soll. Wie schon gesagt ist es nicht empfehlenswert, den Apache durch inetd zu starten, also solte hier der Begriff `standalone` stehen.

`ServerRoot "/usr/local/httpd"`
: Die Angabe des Verzeichnisses, in dem der Apache seine Dateien vorfindet. Hier liegen in der Regel alle wichtigen Dateien und Verzeichnisse, die der Server zum Betrieb benötigt. Nicht verwechseln mit DocumentRoot.

`LockFile /var/lock/subsys/httpd/httpd.accept.lock`
: Das Lock-File des Servers. Diese Datei wird vom Server beim Start angelegt und beim Herunterfahren gelöscht. Damit kann der Server selbst feststellen, ob schon ein Apache-Server im Speicher gestartet ist.

`PidFile /var/run/httpd.pid`
: Auch diese Datei wird vom Server beim Start angelegt. Er schreibt seine ProzessID hier hinein, so daß ein späterer Zugriff - etwa zum Versenden eines Kill-Signals - auf den Server möglich ist, ohne erst lang mit dem ps-Kommando die PID herausfinden zu müssen.

`Timeout 300`
: Die Anzahl von Sekunden, die der Server beim Senden und Empfangen von Daten wartet, bis er aufgibt und einen Timeout-Fehler ausgibt.

`KeepAlive On`
: KeepAlive ermöglicht es, mehrere Transfers pro Verbindung zuzulassen (On) oder zu verbieten (Off). Damit wird verhindert, daß wegen jeder einzelnen Nachfrage erneut der komplette TCP-Handshake erfolgen muß.

`MaxKeepAliveRequests 100`
: Dieser Wert bestimmt die maximale Menge der zulässigen aufeinanderfolgenden Verbindungen, ohne erneuten TCP-Handshake. Steht der Wert auf 0, so bedeutet das, daß es keine Einschränkungen gibt (unendlich).

`KeepAliveTimeout 15`
: Die Anzahl von Sekunden nach der Übertragung einer Datei bis zum Beginn der nächsten Nachfrage, die ohne zusätzlichen Handshake stattfinden darf.

`StartServers 1`
: Die Anzahl der Serverprozesse, die beim Start geladen werden sollen.

`MinSpareServers 1`
: Die minimale Anzahl der unbenutzten Serverprozesse (Childs). Wird dieser Wert unterschritten, so werden neue Server gestartet.

`MaxSpareServers 1`

Die maximale Anzahl unbenutzter Serverprozesse (Childs). Wird dieser Wert überschritten, so werden unbenutzte Serverprozesse heruntergefahren.

MaxClients 150
Die maximale Anzahl gleichzeitig bedienbarer TCP-Verbindungen. Kommen gleichzeitig mehr als angegeben herein, so bekommen die übrigen eine Fehlermeldung und werden nicht mehr bedient.

MaxRequestsPerChild 0
Nach der angegebenen Zahl von abgearbeiteten Aufträgen wird ein Child-Prozess heruntergefahren und durch einen neuen ersetzt. Steht der Wert auf 0, so findet keine Ersetzung statt (0 = unendlich). Damit kann z.B. im Dauerbetrieb festgelegt werden, daß kein Server-Child mehr als 30 Anfragen beantwortet. So wird verhindert, daß ein eventuell hängender Prozess über Tage hinweg den Server stört.

Nach diesen Grundeinstellungen werden die Module geladen. An diesen Einstellungen ist eigentlich keinerlei Veränderung nötig, es sei denn, Sie haben eigene Module entwickelt und wollen sie einbinden.

Konfiguration des Hauptservers

Die zweite Sektion der Konfigurationsdatei bezieht sich auf die Angaben zum "Hauptserver". Das ist der eigentliche HTTP-Server, der ohne weitere Gimmicks läuft. Im Regelfall ist das der einzige Server, nur in Spezialfällen werden sogenannte virtuelle Server dazugenommen.

Port 80
Die Portnummer, die verwendet werden soll, wenn der Server als standalone-Server läuft. Der Standard-Port ist 80. Ports unter 1023 müssen mit root-Rechten ausgestattet sein, es handelt sich ja um die sogenannten *privilegierten Ports*. Um Sicherheitslücken zu vermeiden, werden weitere Server unter anderen UserIDs gestartet.

User wwwrun
Die UserID, unter der die Child-Prozesse des Servers laufen. Alle Zugriffe des Servers aufs Dateisystem werden unter dieser ID ausgeführt.

Group nogroup
Die Gruppenmitgliedschaft der Child-Prozesse.

Listen 80
Diese Anweisung entspricht der Port-Anweisung, nur sind damit auch die Angabe mehrerer Ports möglich. Eine typische Form, um z.B auch das HTTPS-Protokoll (Secure Socket Layer) zu ermöglichen, wäre:

```
<IfDefine SSL>
  Listen 80
  Listen 443
</IfDefine>
```

ServerAdmin root@localhost
Die E-Mail Adresse des Administrators dieses Webservers

DocumentRoot "/usr/local/httpd/htdocs"
Die Wurzel des Dokumentenbaums. Alle Pfade in URLs (außer CGI) beziehen sich auf die diesen Pfad.

DirectoryIndex index.html
Der Name (oder die Namen) der Datei, die in einem Verzeichnis aufgerufen werden soll, wenn nur der Verzeichnisname angegeben wurde. Wenn mehrere Namen angegeben werden, müssen sie durch Leerzeichen voneinander getrennt werden.

ScriptAlias /cgi-bin/ "/usr/local/httpd/cgi-bin/"
Mit dieser Anweisung wird festgelegt, welches Verzeichnis CGI-Scripts enthalten darf, die vom Server ausgeführt werden sollen. Der Alias `/cgi-bin` wird also auf das Verzeichnis `/usr/local/httpd/cgi-bin/` gelegt. Damit werden URLs wie

```
http://www.myhost.mydomain.de/cgi-bin/irgendwas.pl
```

auf das genannte Verzeichnis umgeleitet. Es ist zwar zu empfehlen, daß das Script-Verzeichnis außerhalb des Dokumentenbaums liegt, jedoch nicht notwendig. Falls mehrere solcher Aliase angelegt werden sollen, müssen sie jeweils andere Aliasnamen tragen:

```
ScriptAlias /cgi-bin/ "/usr/local/httpd/cgi-bin/"
```

```
ScriptAlias /test/cgi-bin/ "/usr/cgi-bin/"
ScriptAlias /users/hans/cgi-bin/ "/home/hans/cgi-bin/"
```

Die entsprechenden URLs für die drei verschiedenen Verzeichnisse wären jetzt:

```
http://www.myhost.mydomain.de/cgi-bin/irgendwas.pl
http://www.myhost.mydomain.de/test/cgi-bin/irgendwas.pl
http://www.myhost.mydomain.de/users/hans/cgi-bin/irgendwas.pl
```

Die meisten weiteren Einstellungen benützen jetzt ein HTML-ähnliches Format, um die Gültgkeit auf bestimmte Bereiche zu reduzieren. Dazu werden die gewünschten Bereiche (Verzeichnisse, Dateien und URLs) in spitze Klammern geschrieben und mit einem entsprechenden Abschlußtag beendet. Ein paar Beispiele:

```
<Directory "/usr/local/httpd/htdocs">
  Options FollowSymLinks
  DirectoryIndex default.htm
</Directory>
```

Diese Angabe legt fest, daß die beiden Befehle in den Klammern nur für das Verzeichnis "/usr/local/httpd/htdocs" und alle darin enthaltenen Unterverzeichnisse gelten.

Statt <Directory> kann auch <Location> stehen. Dann wird statt einer absoluten Pfadangabe eine URL angegeben. Das erspart einem Systemverwalter das Wissen über die tatsächliche Positionierung des DocumentRoot. Das muß dann selbstverständlich mit </Location> abgeschlossen werden.

Um einzelne Dateien direkt anzusprechen kann auch die Klammerung mit <Files> erfolgen. In der Klammer stehen dann Dateinamen, auf die sich die Angaben beziehen.

Konfiguration von virtuellen Servern

Die dritte Sektion der zentralen Konfigurationsdatei von Apache dient der Definition von virtuellen Servern. Virtuelle Server sind sozusagen Nebenserver, die vom selben Webserver verwaltet werden, aber eine andere Dokumentenwurzel benutzen. Dazu kommt, daß diese Server auch noch entweder über andere IP-Adressen oder andere DNS-Namen ansprechbar sind.

Die Definition eines virtuellen Servers ermöglicht es also dem Webserver zu entscheiden, an welchen DNS-Namen bzw. welche IP-Adresse die Anfrage ging. Diese Entscheidung führt dann zu unterschiedlichen DocumentRoots also werden unterschiedliche Seiten angezeigt.

Linux bietet zu diesem Zweck die Möglichkeit, daß eine einzige Ethernetkarte mehrere IP-Adressen bekommen kann. Dazu wird der Bezeichnung der Ethernetkarte (z.B. `eth0`) einfach ein Doppelpunkt und eine Nummer zugefügt (`eth0:1, eth0:2, ...`). Jeder dieser "virtuellen Netzwerkkarten" kann jetzt eine eigene IP-Adresse vergeben werden. Entweder mit dem entsprechenden Konfigurationsprogramm (yast, linuxconf,...) oder mit dem Befehl:

```
ifconfig eth0:1 Adresse netmask Maske
```

Wenn diese zweite (dritte, vierte, ...) Adresse jetzt im Nameserver einen eigenen Eintrag erhält, so kann tatsächlich der Webserver darauf reagieren. Dazu müssen in der Konfigurationsdatei ein paar zusätzliche Einträge vorhanden sein:

```
NameVirtualHost  IP-Adresse
```

Dieser Eintrag ist nur nötig, um mehrere namensbasierte Virtuelle Server aufzubauen. Die IP-Adresse ist die der "virtuellen Netzwerkkarte" wie oben beschrieben.

Jetzt können wir die einzelnen virtuellen Server definieren. Dazu werden wiederum Angaben in spitzen Klammern gemacht, entweder mit IP-Adressen (von virtuellen Karten) oder mit Hostnamen (Alias-Einträge im Nameserver,

die auf die virtuelle Karte verweisen).

Jeder dieser Einträge bekommt jetzt einen eigenen DocumentRoot und kann alle bisher besprochenen Direktiven des normalen Servers enthalten. Ein Beispiel:

```
NameVirtualHost 10.230.1.101

<VirtualHost 10.230.1.101>
  ServerAdmin root@marvin.mydomain.de
  DocumentRoot /www2
  ServerName virtual1.mydomain.de
</VirtualHost>

<VirtualHost 10.230.1.101>
  ServerAdmin hans@marvin.mydomain.de
  DocumentRoot /www3
  ServerName virtual2.mydomain.de
  <Directory /www3/specialdir>
     AllowOverride All
  </Directory>
</VirtualHost>
```

Hier haben wir also zwei virtuelle Hosts definiert, beide wurden über ein und dieselbe IP-Adresse (10.230.1.101) definiert, aber beide unterscheiden sich anhand ihres Namens. Natürlich muß der zweite Name entsprechend im Nameserver vorhanden sein und als Alias auf den ersten Rechner gesetzt sein.

Die zweite Möglichkeit virtueller Hosts ist die Adressenbasierte. Hier ist die Angabe des NameVirtualHost-Befehls nicht nötig. Dafür muß jeder virtuelle Server eine eigene IP-Adresse besitzen. Mit der oben gezeigten Methode ist das bis zu einem bestimmten Punkt möglich. Ab einer gewissen Anzahl (etwa ab 4 virtuellen Hosts) bietet es sich aber an, namensbasierte Hosts zu generieren, statt mit x virtuellen Netzwerkkarten zu arbeiten...

Innerhalb der <VirtualHost> Klammerung können alle Direktiven stehen, die auch schon für die Konfiguration des Hauptservers zur Anwendung kamen. Es sind also vollständig autarke Server, denen sogar eigene CGI-Verzeichnisse gegeben werden können. Das folgende Beispiel zeigt zwei virtuelle Hosts, die Adressenbasiert aufgebaut sind und je ein eigenes cgi-bin Verzeichnis besitzen:

```
<VirtualHost 10.230.1.105>
  ServerAdmin root@marvin.mydomain.de
  DocumentRoot /www1/htdocs
  ScriptAlias /cgi-bin/ "www1/cgi-bin/"
  ServerName virtual1.mydomain.de
</VirtualHost>

<VirtualHost 10.230.1.106>
  ServerAdmin hans@marvin.mydomain.de
  DocumentRoot /www2/htdocs
  ScriptAlias /cgi-bin/ "www2/cgi-bin/"
  ServerName virtual2.mydomain.de
</VirtualHost>
```

Starten und Anhalten des Webservers

Normalerweise wird der Webserver beim Start des Systems durch ein Init-Script gestartet. Dieses Script wird beim Wechsel in den Standard-Runlevel ausgeführt und heißt meistens `/etc/init.d/apache` oder `/etc/init.d/httpd`.

In diesem Script wird - wie oben schon erwähnt - als Parameter für den Webserver die Konfigurationsdatei mit angegeben. Typische Einträge zum Starten wären also z.B.

```
/usr/bin/httpd -f /etc/httpd/httpd.conf
```

Auch das Herunterfahren des Webservers kann in der Regel mit diesem Script ausgeführt werden, indem ihm der Parameter `stop` mitgegeben wird.

Nach jeder Änderung an der Konfigurationsdatei muß der Server neu gestartet werden. Damit das nicht nur über das Init-Script erfolgen kann, gehört zum Lieferumfang von **apache** ein Serverkontrollprogramm, mit dem diese Aufgaben erledigt werden können. Dieses Programm ist ein Shellscript, das eventuell an abweichende Pfade angepasst werden muß.

Die Aufrufform des Programms ist

```
apachectl Kommando
```

Es stehen verschiedene Kommandos zur Verfügung. Die wichtigsten sind:

start
: Startet den Apache-Daemon (Webserver). Gibt eine Fehlermeldung aus, wenn er bereits läuft.

stop
: Hält den Daemon an.

restart
: Stopt den laufenden Daemon und startet ihn neu. Zu diesem Zweck wird dem Daemon ein HUP-Signal geschickt. Wenn der Daemon noch nicht gestartet war, wird er neu gestartet.

graceful
: Dieses Kommando startet den Server mit dem Signal USR1 neu. Der Unterschied zu **restart** ist, daß bestehende Verbindungen nicht unterbrochen werden.

configtest
: Die Konfigurationsdatei wird einem Syntax-Test unterworfen. Rückgabe ist entweder **Syntax Ok** oder eine detailierte Information über die gefundenen Fehler in der Konfigurationsdatei.

1.113.4 - Richtiges Verwalten von NFS, smb und nmb Dämonen

Beschreibung: Prüfungskandidaten sollten wissen, wie man im Netzwerk freigegebene Dateisysteme mittels NFS einbindet, wie man NFS für den Export lokaler Dateisysteme konfiguriert und wie man den NFS-Server startet, anhält und neustartet. Ebenfalls enthalten ist die Installation und Konfiguration von Samba unter Verwendung des mitgelieferten GUI-Tools oder durch direkte Bearbeitung von `/etc/smb.conf` (Achtung: Bewußt ausgeschlossen sind fortgeschrittene Themen der NT-Domänen; einfaches Freigeben von Verzeichnissen und Druckern sowie das korrekte Einrichten von nmbd als WINS-Client sind jedoch enthalten).

Die wichtigsten Dateien, Bezeichnungen und Anwendungen:

- `/etc/exports`
- `/etc/fstab`
- `/etc/smb.conf`
- **mount**
- **umount**

Einhängen freigegebener Verzeichnisse

Unter Linux werden freigegebene Verzeichnisse ähnlich wie bei Windows als Netzlaufwerke betrachtet. Allerdings werden sie nicht mit einem Laufwerksbuchstaben versehen, sondern - wie alle anderen Laufwerke auch - in den Dateibaum gemountet.

Es gibt prinzipiell zwei grundlegende Techniken der Freigabe. Entweder ist ein Verzeichnis über das *Network File System* (NFS) freigegeben, oder über das *Server Message Block* Protokoll (SMB). NFS ist in der Regel von Unix/ Linux Maschinen angeboten, SMB ist die Technik der Windows-Freigaben.

Einhängen von NFS-Freigaben

Eine NFS-Freigabe wird durch einen einfachen **mount** Befehl eingebunden. Dazu muß aber statt einer Gerätedatei ein NFS-Pfad eingegeben werden. Dieser Pfad hat die Form

Hostname:/absoluter/Pfad

Um beispielsweise das freigegebene Verzeichnis `/usr/public` des Rechners `hal` in unser lokales Verzeichnis `/mnt` einzuhängen, wird der Befehl

```
mount -t nfs hal:/usr/public /mnt
```

eingegeben. Normalerweise kann die Angabe des Dateisystemtyps (`-t nfs`) auch weggelassen werden, da der **mount**-Befehl aus der Struktur der Pfadangabe erkennt, daß es sich um NFS handeln muß.

Dieser Prozess kann natürlich auch automatisiert werden, damit das Verzeichnis bei jedem Start des Rechners wieder gemountet wird. Dazu muß wie bei lokalen Partitionen ein Eintrag in die Datei `/etc/fstab` gemacht werden. Unsere Beispielfreigabe würde folgenden Eintrag benötigen:

```
hal:/usr/public /mnt    nfs   defaults  0  0
```

Dabei ist allerdings darauf zu achten, daß der Fileserver (in diesem Fall `hal`) schon läuft, bevor der Computer gestartet wird, dessen `/etc/fstab` diese Zeile enthält. Sonst versucht der Rechner das Verzeichnis von `hal` zu mounten und es kann eine ganze Weile dauern, bis der Timeout dafür sorgt, daß der Bootvorgang weitergeht.

Einhängen von SMB-Freigaben

Freigaben von Windows-Rechnern (oder von Samba-Servern) werden über das SMB-Protokoll gesteuert. Diese Freigaben müssen im Gegensatz zu NFS-Freigaben nicht mit ihrem vollen Pfad angegeben werden, sondern besitzen einen sogenannten *Freigabenamen*. Dieser Name wird normalerweise unter Windows in der folgenden Form zusammen mit dem Rechnernamen zu einem Netzwerkpfad:

*Hostname**Freigabename*

Da der Backslash aber eine Sonderbedeutung für die Shell hat, gibt es unter Linux verschiedene Möglichkeiten, wie diese Angabe gemacht werden darf:

*Hostname**Freigabename*
//Hostname/Freigabename
"*Hostname**Freigabename*"

Entweder wird also für jeden Backslash ein doppelter Backslash angegeben, oder statt Backslashs werden normale Slashs benutzt. Die dritte Alternative ist die Ausblendung der Sonderbedeutung des Backslashes durch Anführungszeichen.

Um eine SMB-Freigabe einzuhängen, wird entweder der spezielle Befehl **smbmount** benutzt, oder der normale **mount**-Befehl, zusammen mit dem Dateisystemtyp `smbfs`. Um also die Windows-Freigabe mit dem Freigabenamen `public` des Rechners `winserver1` ins lokale Verzeichnis `/mnt` zu mounten, kann einer der beiden folgenden Befehle verwendet werden:

```
mount -t smbfs //winserver1/public /mnt
smbmount //winserver1/public /mnt
```

Allerdings wird auf diese Weise immer nach einem Passwort gefragt, selbst wenn das Verzeichnis auf dem Windows-Rechner ohne Passwort freigegeben wurde. Mit den folgenden Optionen kann mit diesen Passwörtern umgegangen werden:

username=*Name*
: Der Usernamen des Windows-Users wird angegeben

password=*Passwort*
: Das Passwort für die Freigabe wird angegeben

guest
: Es existiert kein Passwort, es muß also auch nicht nach einem gefragt werden.

Diese Optionen werden dem **mount**-Befehl zusammen mit dem Schalter **-o** angegeben, also etwa

```
mount -t smbfs -o guest //winserver1/public /mnt
```

Der **smbmount** Befehl hängt diese Optionen - auch durch **-o** eingeleitet an den Befehl hinten an:

```
smbmount //winserver1/public /mnt -o guest
```

Um den Prozeß zu automatisieren, wird wiederum ein Eintrag in die Datei `/etc/fstab` gemacht, der unbedingt als Option entweder **guest** oder Username und Passwort benötigt.

```
//winserver1/public  /mnt   smbfs  defaults,guest 0  0
```

Wenn die Windows-Freigabe die Angabe eines Usernamens und Passwortes erwartet, so könnten hier natürlich auch die Optionen `username=...` und `password=...` angegeben werden. Das ist aber keine gute Idee, denn die Datei `/etc/fstab` ist von allen Usern lesbar. Aus diesem Grund gibt es die Möglichkeit, hier einen Dateinamen anzugeben, mit der Option **credentials=***Dateiname*. Die angegebene Datei kann dann Einträge der Form

```
username = Wert
password = Wert
```

enthalten. Diese Datei wird nicht von aller Welt lesbar sein, also sind die Passwörter sicher.

Verzeichnisse mit NFS freigeben

Um selbst Verzeichnisse mit NFS freizugeben, müssen ein paar Bedingungen erfüllt sein, die hier näher erläutert werden sollen.

Technisch gesehen baut das Network-File-System auf einer Entwicklung auf, die *Remote Procedure Call* (RPC) genannt wird. Dieser "entfernte Prozeduraufruf" nutzt die Portnummern oberhalb 10000 um es zu ermöglichen, Client-Server Software zu steuern. Der Server stellt bestimmte Prozeduren zur Verfügung, die jeweils einer bestimmten Portnummer zugeordnet sind. Ein Client im Netz kann nun diese Prozeduren aufrufen, es entsteht eine Art gemischtes Programm, ein Teil läuft auf dem Client ab, ein anderer Teil auf dem Server.

Die Zuweisung der Ports für das RPC-System übernimmt der **portmapper**, ein Programm, das immer laufen muß, wenn ein Rechner RPC-Service anbieten will.

Die oben genannte Aufteilung macht sich NFS zu Nutzen, indem dieses System Prozeduren zur Verfügung stellt, die sich auf den Zugriff auf Dateisysteme beziehen. Diese Prozeduren werden jetzt vom NFS-Client aufgerufen, um Zugriff auf ein Dateisystem zu bekommen.

Die folgenden Voraussetzungen müssen erfüllt sein, um Verzeichnisse mit NFS freizugeben:

- Der **portmapper** muß laufen.
- Die Daemonen **rpc.nfsd** und **rpc.mountd** müssen laufen.
- Beide erhalten ihre Konfigurationsinformationen über die Datei `/etc/exports` - Diese Datei enthält die Informationen, wer was mounten darf.

Probleme bei den Zugriffsrechten

Linux erweckt zwar den Anschein, als ob es die Zugriffsrechte nach Usernamen verwalten würde, hinter den Kulissen arbeitet es aber ausschließlich mit den numerischen UserIDs. Das führt beim Mounten von Dateisystemen übers Netz zu Problemen mit der Sicherheit, was Dateizugriffe angeht.

Ein Beispiel:

Auf dem Rechner hal existieren die User:

```
Username     UID
root          0
peter         501
hans          502
```

Der Rechner hal exportiert sein `/usr` Verzeichnis jetzt an alle anderen Rechner im Netz. Dabei bleiben natürlich die Zugriffsrechte erhalten. Doch die exportierten Dateien

```
-rw-r-----  root   root  geheim.txt
-rw-------  peter  user  noch_geheimer.txt
```

werden eben nicht nach Usernamen (root, peter) verwaltet sondern nach ihren User IDs also 0 und 501.

Der Rechner marvin will das freigegebene Verzeichnis von hal jetzt mounten. marvin hat jedoch andere User:

```
Username     UID
root          0
otto          501
```

```
hugo          502
```

Die oben genannten UserIDs sind also auch auf marvin vergeben, was dazu führt, daß marvin jetzt mit den neuen Namen arbeitet, nicht nur mit den Namen sondern eben auch mit den Rechten dieser User. Die gemounteten Dateien sähen jetzt also so aus:

```
-rw-r-----  root  root  geheim.txt
-rw-------  otto  user  noch_geheimer.txt
```

Die ganze Sache hätte zur Folge, daß otto plötzlich Dateien lesen und beschreiben kann, die eigentlich peter gehören und für alle anderen unlesbar sein sollten.

In den Fällen, in denen diese Erscheinung unvermeidbar ist (weil es z.B. nicht möglich ist, alle User überall gleich zu nummerieren) kann ein Mechanismus angewandt werden, der etwas genauere Betrachtung verdient, das Squashing.

Diese Technik wird vorwiegend bei root-Zugängen angewandt; root hat immer die UserID 0 egal ob es sich um die selbe Person handelt oder nicht. Also kann dafür gesorgt werden, daß sich der root-Zugriff auf ein gemountetes Dateisystem in eine andere UserID verwandelt. NFS versucht es zunächst mit dem Usernamen nobody, wenn es den nicht gibt, so weist es dem zugreifenden root die UID und GID -2 zu. Man spricht hier von der anonymen UserID.

Manchmal ist es auch erwünscht, anderen Usern ebenfalls die anonyme UserID zuzuweisen, NFS bietet die Möglichkeit diese Zuweisung zu übernehmen.

Der Aufbau der Datei /etc/exports

Die Datei `/etc/exports` steuert die Zugriffsrechte anderer Rechner auf die Verzeichnisse des Servers. Hier werden Verzeichnisse freigegeben und die entsprechenden Optionen (etwa ReadOnly) gesetzt.

Die Datei hat eine einfache Form, zeilenweise werden die einzelnen Verzeichnisse angegeben und mit Zugriffsbestimmungen versehen. Dabei sind Wildcards möglich um möglichst flexibel in der Unterscheidung zu sein, wer was mounten darf.

In Klammern können noch Optionen gesetzt werden, die die Zugriffsrechte betreffen (Read only ...) oder die das Squashing (siehe oben) steuern. Am Besten sehen wir uns das einmal an einem Beispiel (der Handbuchseite entnomme) an:

```
# Beispielhafte /etc/exports Datei
/               master(rw) trusty(rw,no_root_squash)
/projects       proj*.mydomain.de(rw)
/usr            *.mydomain.de(ro)
/home/joey      pc007(rw,all_squash,anonuid=501,anongid=100)
/pub            (ro,all_squash)
```

Der erste Eintrag bestimmt zwei Rechner (`master` und `trusty`) die das ganze Wurzeldateisystem mounten dürfen. Das `rw` in Klammern besagt, daß sie auch schreibenden Zugriff haben. Standardmäßig ist vorgegeben, daß der root-account über Squashing auf die Anonyme UID gelegt wird. Mit der Option `no_root_squash` wird dieses Squashing ausgeschaltet. Der Systemverwalter von `trusty` hat also tatsächlich auf dem gemounteten Dateisystem Rootrechte.

Der nächste Eintrag beschreibt die Möglichkeit aller Rechner, deren Namen mit `proj` beginnen und auf `.mydomain.de` enden, das Verzeichnis `/projects` zu mounten. Schreibender Zugriff ist möglich.

Das Verzeichnis `/usr` wird in der dritten Zeile für alle Rechner der Domain `mydomain.de` freigegeben. Es kann aber nur Read-Only gemountet werden, Veränderungen sind also nicht möglich.

Die nächste Zeile beschreibt den Rechner `pc007`, der das Verzeichis `/home/joey` mounten darf. Es handelt sich

um einen typischen Eintrag für PCs, alle User werden gesquasht, als Anonyme UID wird 501 angenommen, als GID 100.

Das letzte Beispiel zeigt einen Eintrag, der von der ganzen Welt (sofern sie Zugriff auf unser Netz hat) benutzt werden kann und das Verzeichnis `/pub` als Read-Only freigibt. Alle UserIDs werden auf die Anonyme UID (nobody) gesetzt.

Eine genaue Möglichkeit zu bestimmen, welche UserIDs auf die anonyme abgebildet werden sollen, gibt es auch noch. Die Optionen `squash_uids` und `squash_gids` ermöglichen eine genaue Angabe, welche UIDs und GIDs verwandelt werden sollen. Eine gültige Angabe kann so aussehen:

```
...   (squash_uids=0-15,20,25-50)
```

Jedesmal, wenn Einträge in der Datei `/etc/exports` verändert werden, muß den beiden Daemonen **rpc.mountd** und **rpc.nfsd** ein HUP-Signal geschickt werden, um sie zu zwingen, diese Datei neu einzulesen.

Statt einem HUP-Signal kann auch der Befehl **exportfs -a** eingegeben werden, der die Daemonen zwingt, die Datei neu einzulesen.

Einfache Freigaben mit Samba

In den meisten Fällen arbeiten in Computernetzen heute die Arbeitsstationen mit Windows-Betriebssystemen, entweder Windows 95/98/ME oder WindowsNT/2000/XP. Es wäre natürlich genauso möglich, alle Arbeitsstationen mit Linux auszurüsten, das würde aber eine sehr große Umstellung bedeuten, sowohl für die einzelnen Anwender, als auch für die Verwaltung. Oft kommen auch Programme zur Anwendung, die für Linux nicht zur Verfügung stehen und die Akzeptanz von Linux ist sicher auch nicht besonders hoch, bei Menschen, die eben gerade mal gelernt haben, mit Windows umzugehen...

Damit Linux als Server für Windows-Systeme eingesetzt werden kann, ist es nötig, daß ein Serverprogramm installiert wird, das die Netzwerktechnik von Windows zur Verfügung stellt. Die Protokollfamilie, mit der Windows-Netze ihre Datei- und Druckerdienste verwalten, heißt SMB (Server Message Block) und wird von allen Windows-Systemen (von Win 3.11 bis Win 2000/XP) verwendet. Die Implementierung dieser Protokollfamilie unter Linux (und anderen Unixen) heist Samba.

Samba bietet von der einfachen Fähigkeit, Datei- und Druckerdienste in Arbeitsgruppen zur Verfügung zu stellen, über die Integration in bestehende NT-Domains bis hin zur kompletten Emulation eines NT-PDC alle Funktionen, die gebraucht werden, um Linux-Rechner in ein bestehendes oder neu zu erstellendes Windows-Netz zu integrieren. Die Windows-Arbeitsstationen merken gar nicht, daß es sich bei dem Server um einen Linux-Rechner handelt, aus ihrer Sicht arbeitet hier ein Windows Server.

Das grundlegende Prinzip ist zunächst einmal sehr einfach. Um Samba auf einem Linux-Rechner zu installieren müssen zwei Daemonen gestartet werden:

- **smbd**
  Der SMB-Daemon, der die Datei- und Druckerdienste zur Verfügung stellt.
- **nmbd**
  Der NetBIOS Name Server, der das Prinzip der Rechnernamen in einem Windows-Netz unter Linux zur Verfügung stellt.

Beide dieser Daemonen erhalten ihre Konfigurationsinformationen aus einer einzigen Datei, `/etc/smb.conf`. Diese Datei enthält alle Einstellungen, die nötig sind, um SMB-Dienste zu aktivieren.

Samba ist üblicherweise ein Stand-alone Dienst, kann aber auch via **inetd** gestartet werden. Nachdem aber in den meisten Fällen ein Rechner mit Samba-Dienst hauptsächlich als solcher arbeitet, wird der stand-alone Variante meist der Vorzug gegeben. In diesem Fall erledigt das entsprechende Init-Script (z.B.: `/etc/init.d/smb/`) die Aufgabe, sowohl den SMB-Server **smbd**, als auch den NetBIOS-Nameserver **nmbd** jeweils mit dem Parameter `-D` (für Daemon) zu starten (oder zu stoppen).

Eine erste Samba Konfiguration

Samba wird grundsätzlich über eine einzige Konfigurationsdatei verwaltet, `/etc/smb.conf`. Diese Datei muß existieren, bevor der Server selbst gestartet wird, damit er überhaupt weiß, was er tun und lassen soll.

Wir werden jetzt eine sehr einfache `smb.conf` Datei durchsprechen, die die grundlegenden Prinzipien dieser Konfigurationsdatei zeigt. Die einzelnen Bereiche werden jeweils kommentiert, erklärt und denkbare Alternativen werden aufgezeigt.

Der grundsätzliche Aufbau dieser Konfigurationsdatei entspricht dem von Windows-INI-Dateien. Das heißt, die Datei besteht aus einzelnen Abschnitten, die immer durch Überschriften in eckigen Klammern voneinander getrennt sind. Kommentare innerhalb der Konfigurationsdatei werden durch Strichpunkte eingeleitet, nicht durch Doppelkreuze (Doppelkreuze funktionieren jedoch auch).

Alle Freigaben, also sowohl Drucker, als auch Dateifreigaben werden als sogenannte *shares* bezeichnet. Jedes Verzeichnis und jeder Drucker, der freigegeben wird ist also ein *share* und hat einen eigenen Abschnitt innerhalb der Konfigurationsdatei. Ein spezieller solcher Abschnitt ist `[global]`, der globale Einstellungen ermöglicht. Aber genug der Theorie, fangen wir an. Das folgende steht in unserer `/etc/smb.conf`:

```
[global]
  workgroup = ARBEITSGRUPPE
  netbios name = MARVIN
  security = SHARE
  guest account = nobody

[public]
  comment = Oeffentliches Verzeichnis
  path = /usr/public
  guest ok = Yes
  read only = Yes
```

Wir haben nur zwei einfache Abschnitte in dieser Datei, `[global]` und `[public]`. Der erste Abschnitt beschreibt die globalen Einstellungen und muß diesen Namen tragen. Der zweite beschreibt eine Freigabe, deren Namen willkürlich auf "public" gesetzt wurde. Der Name könnte auch anders heißen, es ist einfach nur ein Name.

Die Sektion [global]

Sehen wir uns die `[global]`-Sektion einmal genauer an: Die erste Anweisung ist klar, hier wird die Windows-Workgroup definiert. Das Prinzip der Workgroups wurde von Windows 3.11 (Windows for Workgroups) eingeführt und definiert eine Arbeitsgruppe, um ein Netz überschaubarer zu halten. Windows NT (und Win 2000) arbeiten stattdessen mit dem Domain-Prinzip, aber Samba benutzt den Workgroup Eintrag später auch, um die Domain-Einstellungen vorzunehmen. Wir definieren hier also die Zugehörigkeit des Samba Servers zur Workgroup "Arbeitsgruppe".

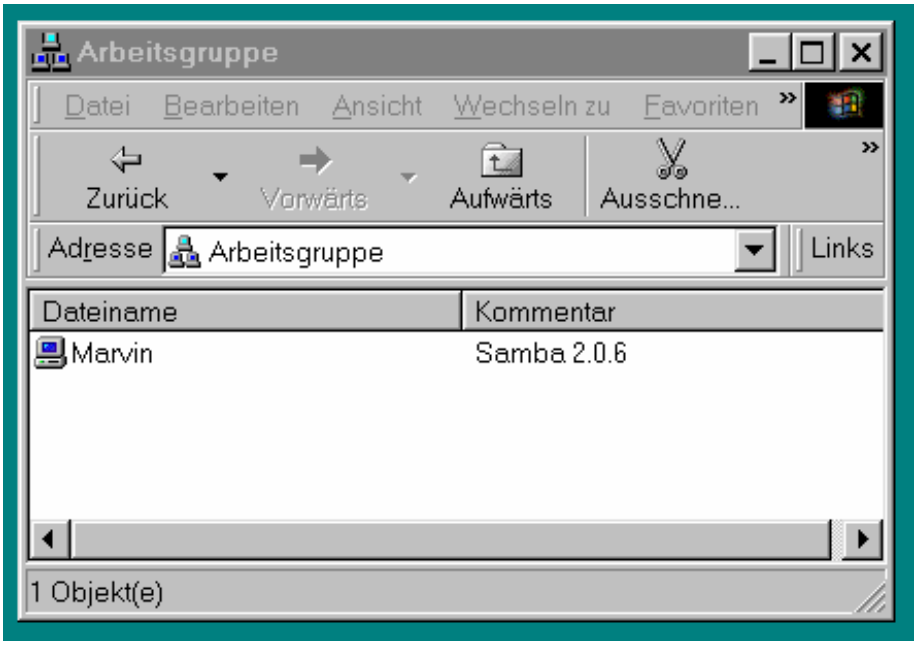
Der zweite Eintrag (`netbios name = marvin`) definiert den Namen, der aus der Sicht von Windows Rechnern für den Samba Server gültig ist. Wir dieser Eintrag weggelassen, dann wird der Standard-Unixname (DNS Name) des Servers gewählt.

Der Eintrag `security = share` legt fest, daß die Zugangssteuerung auf Freigabeebene erfolgt, das ist die simpelste Möglichkeit, die einzige, um keine Passwörter zu brauchen. Wir werden uns später mit dieser Einstellung noch intensiv auseinandersetzen.

Der nächste (und letzte) Eintrag der Sektion `[global]` beschreibt, unter welcher UserID der Dateizugriff stattfinden soll, wenn ein nicht authorisierter User (also ein Gast ohne Passwort) auf einen Dienst zugreift. Die Tatsache, daß Windows selbst keine vergleichbaren Usereinstellungen kennt, zwingt uns hier, einen User anzugeben, damit Linux weiß, welche Rechte hier zur Verfügung stehen. In diesem Fall ist es also der User nobody, ein User ohne besondere Rechte.

Wenn wir jetzt also von einem Windows-Rechner aus die Netzwerkumgebung ansehen und die Workgroup

Arbeitsgruppe aufrufen, werden wir folgendes Bild zu sehen bekommen:



Der Kommentar Samba 2.0.6 ist der voreingestellte Kommentar, den wir in der Sektion `[global]` auch ändern könnten, indem wir die Anweisung

```
server string = ...
```

eingefügen. Diese Einstellung bietet noch die Möglichkeit, den Rechnernamen mit einem %h darzustellen, die Version von Samba bekommen wir mit %v. Hätten wir also in der Sektion [global] geschrieben:

```
server string = Samba Versuchsserver auf %h (Samba %v)
```

dann stünde im Kommentarfeld der Netzwerkumgebung unter Windows

Samba Versuchsserver auf marvin (Samba 2.0.6)

Das %h und %v sind also sogenannte Zeichenkettensubstitutionen, die vom Server durch eine bestimmte Zeichenkette ausgetauscht (substituiert) werden. Alle denkbaren Zeichenkettensubstitutionen, die Samba unterstützt, finden Sie hier:

Zusammenfassung Zeichenkettensubstitutionen

Samba kennt eine Menge von Zeichenkettensubstitutionen, die jeweils mit einem Prozentzeichen beginnen und vom Server durch eine bestimmte Zeichenfolge ausgetauscht (substituiert) werden. So ist es z.B. möglich, userbezogene Pfade zu setzen, wie in

```
path = /home/%u
```

Dieser Pfad würde - da %u für den aktuellen Usernamen ersetzt wird - zu /home/hans, wenn sich der User mit dem Namen hans eingeloggt hätte, sie würde aber zu /home/otto. wenn otto angemeldet wäre...

Samba 2.0 unterstützt die folgenden Substitutionen:

- %S
  Der Name des aktuellen shares - sofern es einen gibt.
- %P
  Das Wurzelverzeichnis des aktuellen shares - sofern es einen gibt.
- %u
  Der Name des Users, der die Anfrage macht.

- %g
  Name der primären Gruppe des Users %u.
- %U
  Session User Name - Der Name, den der Client angab, nicht notwendigerweise der, den er dann bekommen hat.
- %G
  Name der primären Gruppe von %U
- %H
  Das Homeverzeichnis des Users %u
- %v
  Die Versionsnummer von Samba
- %h
  Der Internet Hostname des Rechners, auf dem Samba läuft.
- %m
  Der NetBIOS Name des Windows-Clients - Sehr nützlich
- %L
  Der NetBIOS Name des Servers. Das erlaubt es, unterschiedliche Konfigurationen anzubieten, je nachdem, wie der Server vom Client angesprochen wurde.
- %M
  Der Internet Name der Client-Maschine
- %N
  Der Name des NIS Home Directory Servers.
- %p
  Der NIS Pfad des Homedirectories des aktuellen shares.
- %R
  Der SMB-Protokoll-Level nach dem Handshake. Eins von CORE, COREPLUS, LANMAN1, LANMAN2 oder NT1.
- %d
  Die Prozess ID des aktuellen Server-Prozesses.
- %a
  Die Architektur der Client-Maschine. Nicht hundertprozent verlässlich. Eins von Samba, WfWg, WinNT, Win95 und UNKNOWN.
- %I
  Die IP-Adresse der Client-Maschine.
- %T
  Aktuelles Datum und Uhrzeit

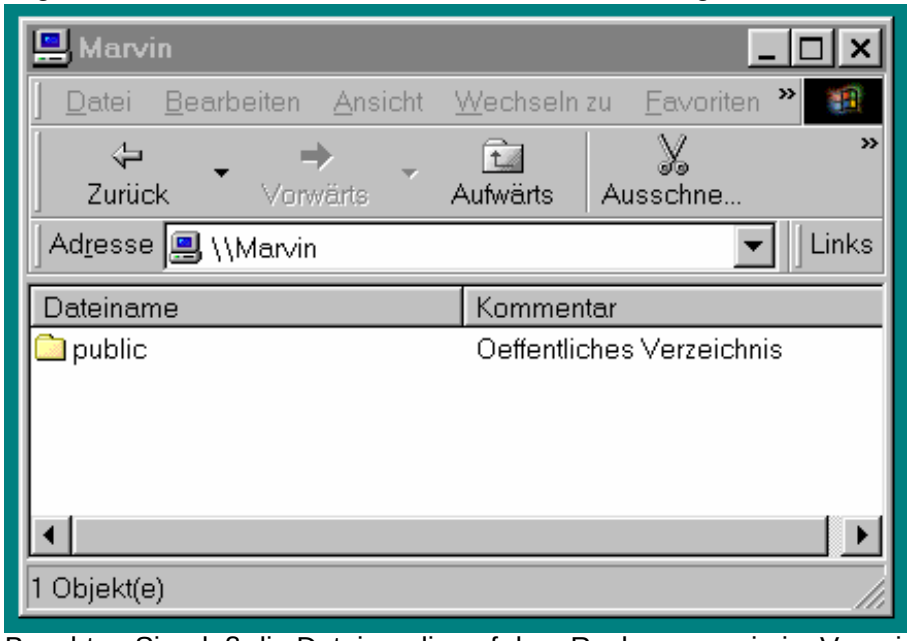
Die Sektion [public]

Unserer zweite Sektion in der Datei `/etc/smb.conf` heißt `[public]`. Das ist - wie oben schon erwähnt - nur ein beliebiger Name und hat keinerlei syntaktische Bedeutung. Der Name ist allerdings der Freigabename, unter dem das freigegebene share (Verzeichnis) unter Windows dann zu sehen sein wird. Die Zeilen in diesem Abschnitt haben folgende Bedeutung:
Die erste Anweisung (`comment = Oeffentliches Verzeichnis`) definieren einen Kommentar, den wir dann in der Netzwerkumgebung unter Windows wiederfinden werden. Beachten Sie, daß Unix und Windows völlig anders mit Umlauten umgehen. Hätten wir Öffentlich statt Oeffentlich geschrieben, so hätte der Windows-User statt dem Ö ein | gesehen, was sicherlich weniger informativ gewesen wäre...
Die zweite Anweisung definiert den absoluten Pfad unter Linux, auf den sich diese Freigabe bezieht. Die Angabe `path = /usr/public` bedeutet also, daß unter dem Freigabenamen public das Verzeichnis /usr/public zu erreichen ist.
Die Angabe `guest ok = yes` bedeutet, daß dieses share ohne Passwort erreichbar ist. Stattdessen hätten wir auch schreiben können:

```
public = Yes
```

Es ist also ein öffentlich zugängliches Verzeichnis für alle Windows-User. Kein Passwort wird verlangt. Allerdings haben alle User in diesem Verzeichnis dann auch nur das Recht des Gast-Users.
Die letzte Zeile erklärt sich von selbst, `read only = Yes` meint natürlich, daß die Freigabe nur ReadOnly

erfolgt, daß die Windows-User also dort nur lesen dürfen. Aus der Sicht von Windows bekommen wir also jetzt folgendes Bild, wenn wir oben den Rechner marvin angeklickt hätten:



Beachten Sie, daß die Dateien, die auf dem Rechner marvin im Verzeichnis `/usr/public` liegen nur dann lesbar sind, wenn sie - aus der Sicht von Linux - vom User nobody lesbar sind. Wir haben ja oben angegeben, daß ein Gastzugriff unter dieser UserID verwaltet wird. Jeder passwortlose Zugriff auf dieses Verzeichnis wird jetzt unter der UserID von nobody vorgenommen.
Es ist auch möglich, für jedes angegebene share einen eigenen Gast-User festzulegen. Wenn die Anweisung `guest account = ...` in einem Abschnitt vorkommt, der nicht die Sektion `[global]` ist, so gilt diese spezielle Abmachung nur für diesen share.
Zwei weitere interessante Anweisungen für jeden share sind

- **`browseable = Yes|No`**
  Wenn die Option browseable = No gesetzt wurde, dann erscheint der share nicht in der Liste der freigegebenen Verzeichnisse in der Windows Netzwerkumgebung, steht aber trotzdem zur Verfügung, wenn man den entsprechenden Pfad (\Rechnername\Freigabename) angibt.
  Standardmäßig steht browseable auf Yes.
- **`available = Yes|No`**
  Mit dieser Option kann - wenn sie auf No gesetzt ist, ein share für den Augenblick abgeschaltet werden, ohne ihn aus der Datei /etc/smb.conf zu streichen oder langwierig auszukommentieren. Standardmäßig ist available auf Yes gestellt.

Selbstverständlich können beliebig viele shares angelegt werden, um verschiedene Verzeichnisse auf unterschiedliche Arten freizugeben. Jedes share bekommt einen eigenen Abschnitt in der `smb.conf`, beginnend mit dem Freigabenamen in eckigen Klammern, gefolgt von den entsprechenden Angaben...

Userverzeichnisse einbeziehen

Die primäre Aufgabe eines Fileservers ist es nicht nur öffentliche Dateisysteme anzubieten, sondern auch userbezogene Verzeichnisse anzubieten. Linux hat ja für jeden User der dem System bekannt ist, ein Verzeichnis, in dem dieser User alle Rechte bekommt. Er kann dort Dateien und Verzeichnisse anlegen und ist alleiniger Eigentümer dieser Daten.
Samba bietet einen sehr einfachen Mechanismus, der einem Windows User erlaubt, sein Home-Verzeichnis auf dem Linux-Rechner zu nutzen, wenn sein Username unter Linux der selbe Name wie der unter Windows ist.
Erweitern wir unsere /etc/smb.conf Datei doch einmal um diesen Mechanismus zum laufen zu bringen:

```
[global]
  workgroup = ARBEITSGRUPPE
  netbios name = MARVIN
  security = SHARE
  guest account = nobody
```

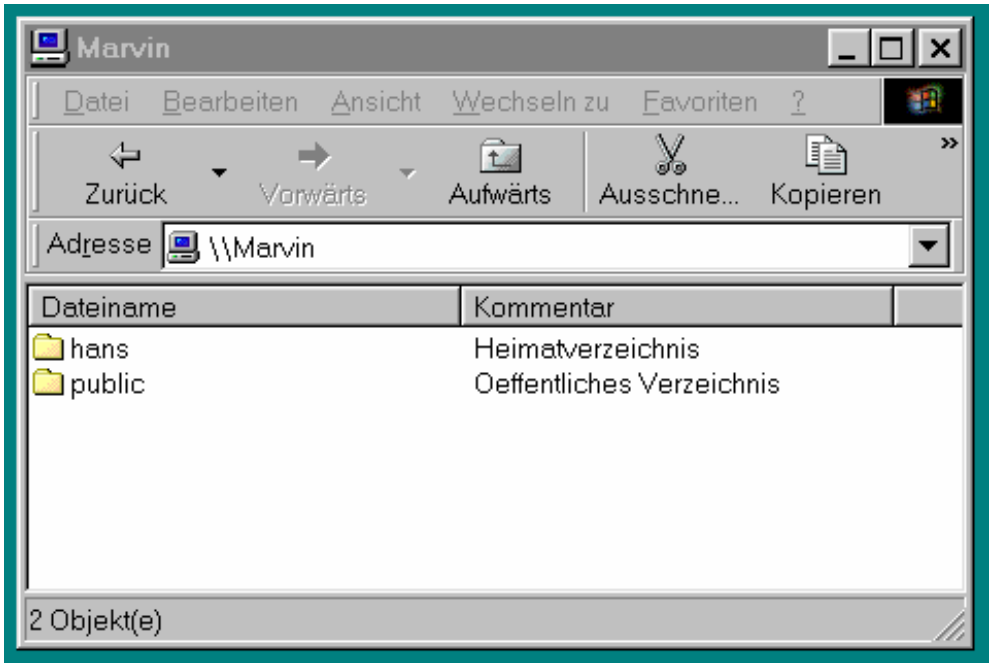
```
[homes]
  comment = Heimatverzeichnis
  read only = No
  create mask = 0750
  browseable = No

[public]
   comment = Oeffentliches Verzeichnis
   path = /usr/public
   guest ok = Yes
   read only = Yes
```

Um diese Erweiterung auch aktiv zu machen müssen wir den Samba Server neu starten, am einfachsten mit

```
/etc/init.d/smb restart
```

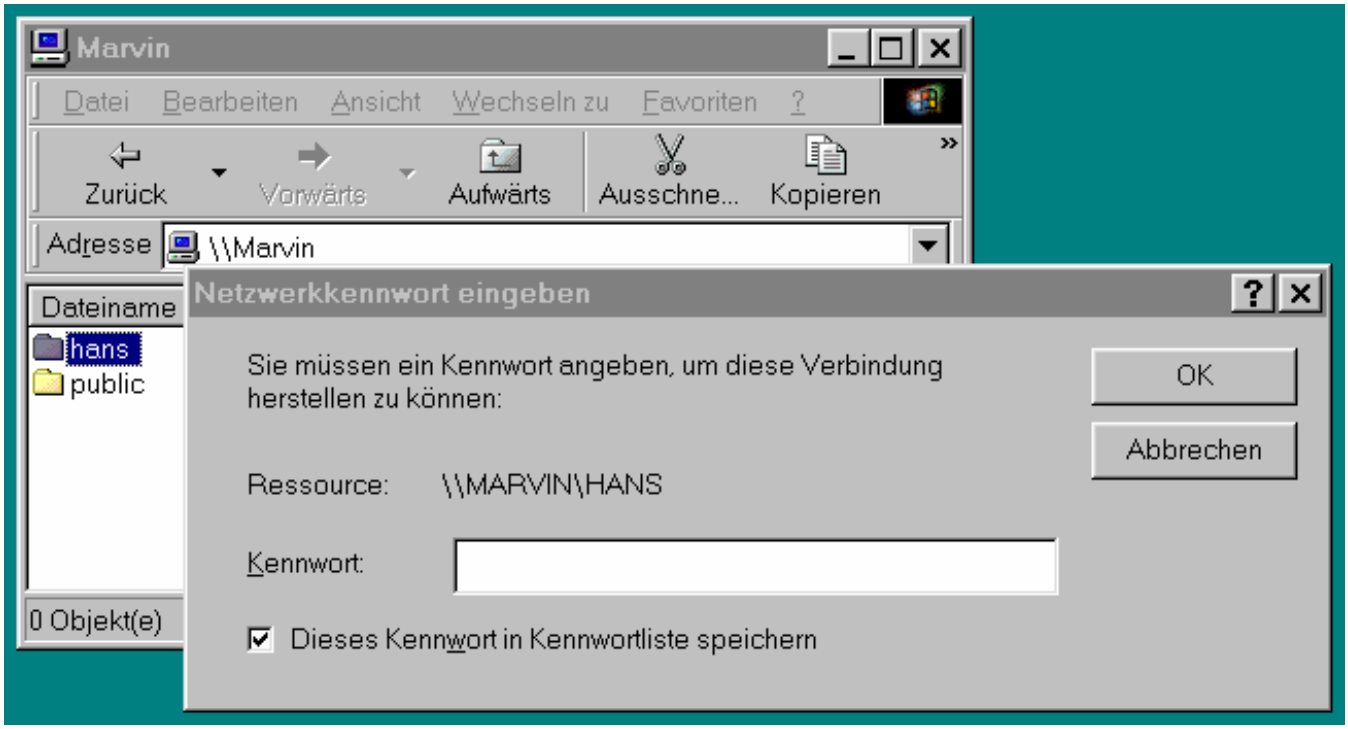
Wenn wir uns jetzt auf unserem Windows-Rechner als - sagen wir mal - User hans anmelden - und ein User hans auf dem Linux-Rechner existiert, dann sieht die Liste der freigegebenen Shares in der Netzwerkumgebung jetzt folgendermaßen aus:



Hätten wir uns aber als otto eingeloggt (und gäbe es einen User otto unter Linux) so hätten wir statt der Angabe der Freigabe von hans jetzt eben die von otto gesehen.
Die konsequente nächste Fragestellung ist jetzt natürlich die des Passworts, das wir für diesen Zugriff benötigen. Grundsätzlich gilt:

- Wenn das Passwort unter Windows das selbe Passwort wie unter Linux ist, so gewährt Samba Zugriff.
- Wenn das Windows-Passwort nicht das selbe wie das unter Linux ist, so muß ein Passwort angegeben werden.

Windows frägt im zweiten Fall dann einfach nach dem Passwort nach und gibt auch die Möglichkeit frei, das angegebene Passwort in einer Passwortliste zu speichern. Wird das Passwort gespeichert, so muß es beim nächsten Mal nicht erneut angegeben werden.



Probleme ab Windows 98

Diese Passwort-Weitergabe funktioniert so bis Windows 95 ohne Probleme, ab Windows 98 kommt es aber nur zu der befremdlichen Fehlermeldung, das eingegebene Passwort sei falsch. Das liegt nicht daran, daß Sie es falsch eingegeben haben, sondern an einer Eigenschaft von Windows 98/NT/2000, die festlegt, daß Passwörter nicht unverschlüsselt über das Netz gegeben werden. Das ist ja eine eigentlich erfreuliche Eigenschaft, weil es die Sicherheit im Netz vergrößert, aber andererseits funktioniert jetzt der normale Mechanismus der Passwortüberprüfung des Unix-Passworts natürlich nicht mehr.
Die Lösung besteht darin, eine eigene Passwortverwaltung für den Sambadienst zu installieren. Das geschieht durch zwei einfache Schritte:

1. Die Zeile

   ```
   encrypt passwords = Yes
   ```

   wird in die `[global]` Sektion der `/etc/smb.conf` aufgenommen

2. Eine spezielle Datei `/etc/smbpasswd` enthält die Userinformationen über die Samba-User samt ihrer verschlüsselten Passwörter.

Um jetzt also diese Passwort-Datei anzulegen und verschlüsselte Passwörter zu generieren benötigen wir wiederum ein kleines Programm mit Namen **smbpasswd**. Dieses Programm erlaubt es, sofern es der Superuser root anwendet, neue Samba-User anzulegen und ihnen Passwörter zu vergeben. Ein Normaluser kann nur sein Passwort damit ändern. Um einen neuen User anzulegen schreibt root jetzt

```
smbpasswd -a Username
```

Er wird dann gleich nach dem Passwort des neuen Users gefragt, das Passwort wird verschlüsselt und zusammen mit der UserID des Users in der Datei `/etc/smbpasswd` abgelegt.
Damit also alles jetzt wirklich funktioniert muß die `[global]`-Sektion unserer smb.conf Datei jetzt folgendermaßen aussehen:

```
[global]
  workgroup = ARBEITSGRUPPE
  netbios name = MARVIN
  security = SHARE
  guest account = nobody
```

```
encrypt passwords = Yes
```

Und schon funktioniert die Freigabe der Userverzeichnisse.
Die Sektion `[homes]` ist also - wie die Sektion `[global]` eine spezielle Möglichkeit. Sie muß grundsätzlich diesen Namen tragen denn Samba übernimmt ja sehr spezielle Ersetzungsmechanismen, wenn dieser share angesprochen wird.

Drucker freigeben

Um Drucker für Windows-Rechner freizugeben, die an Linux Rechnern hängen, sind ein paar kleine Erweiterungen an unserer smb.conf Datei nötig. Zunächst einmal muß geklärt werden, wie Linux druckt. In der Regel wird Linux das BSD-Drucksystem benutzen, manchmal - seltener - aber auch das PLP System. Samba muß in der `[global]`-Sektion einen entsprechenden Eintrag besitzen:

```
[global]
  workgroup = ARBEITSGRUPPE
  netbios name = MARVIN
  security = SHARE
  guest account = nobody
  encrypt passwords = Yes
  printing = BSD
```

Um jetzt Drucker selbst freizugeben haben wir zwei Möglichkeiten:

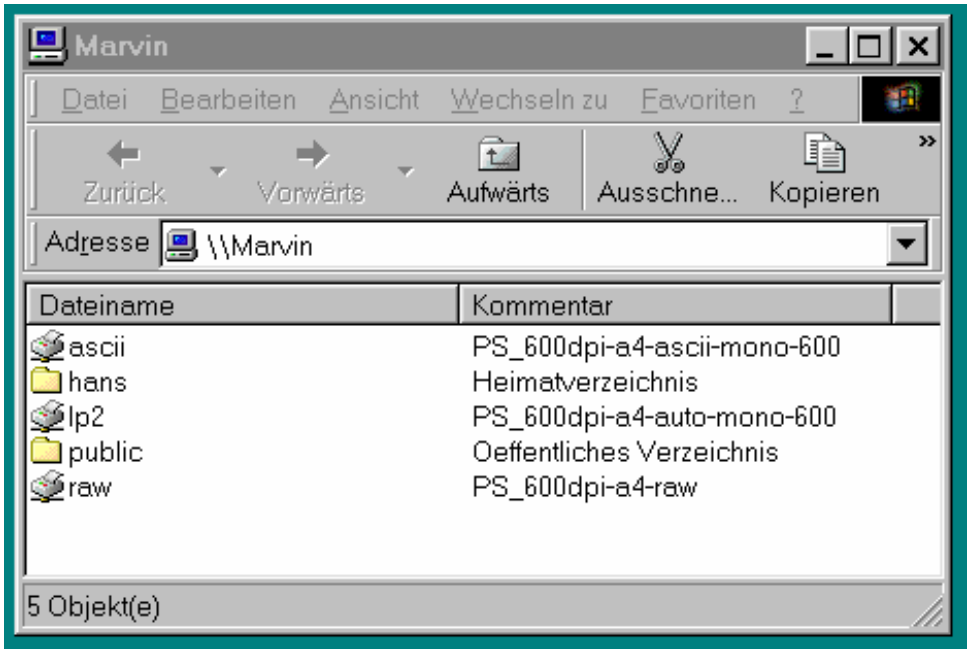
- Alle Drucker werden freigegeben
- Bestimmte Drucker werden freigegeben

Wenn alle Drucker freigegeben werden sollen wird ein Mechanismus benutzt, der dem der Userverzeichnisse ähnelt. Wir legen ein spezielles share an, das zwingend [printers] heissen muß. Dieses share enthält folgende Anweisungen:

```
[printers]
  comment = All Printers
  path = /tmp
  create mask = 0700
  printable = Yes
  browseable = No
  guest ok = Yes
```

Der angegebene Pfad bezieht sich auf ein Verzeichnis, in dem alle User Schreibrecht haben (und dem also das Sticky-Bit gesetzt sein sollte). Die Create Mask legt fest, welche Permissions Druckaufträge haben sollen, die in diesem Verzeichnis liegen. Die Angabe printable = Yes bedeutet, daß - selbst wenn das Verzeichnis selbst Read only = Yes als Parameter hat, Druckaufträge dort abgelegt werden dürfen. Die Angabe browseable = No bedeutet, daß der share `[printers]` selbst nicht erscheint.

Auf diese Weise werden alle Drucker, die unter Linux in der `/etc/printcap` Datei auftauchen, dem Windows-Rechner angezeigt und freigegeben. Das heißt, daß auch wenn - wie bei vielen Distributionen üblich - mehrere logische Drucker angelegt werden, all diese logischen Drucker auch angezeigt werden:

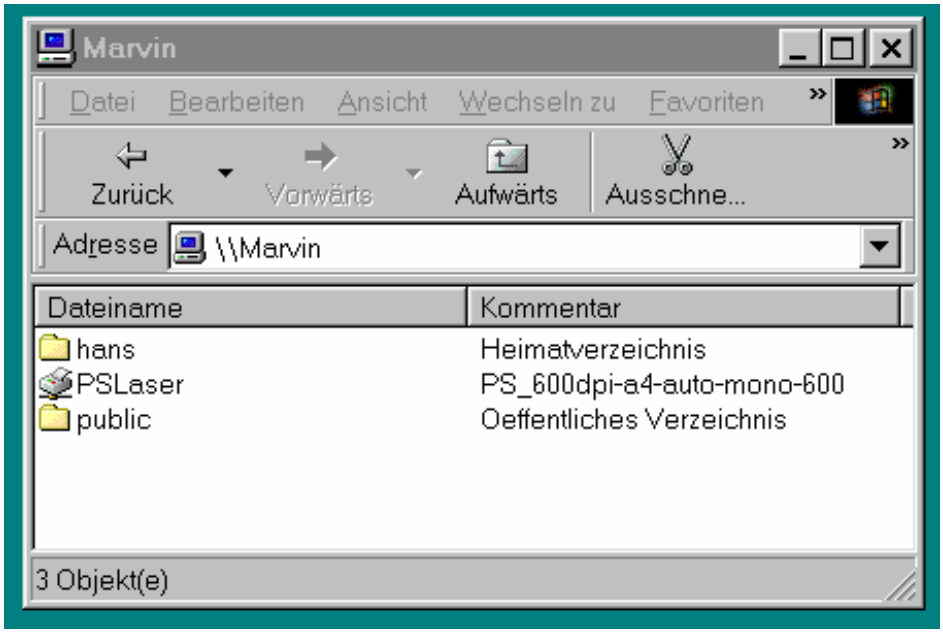


Das kann einen unbedarften Windows-User natürlich verwirren. Um aber nur einen einzigen Drucker anzuzeigen, wird statt des `[printers]` Abschnitts ein Abschnitt für den freizugebenden Drucker angelegt, dessen Name jetzt wieder frei wählbar ist:

```
[PSLaser]
  comment = PS_600dpi-a4-auto-mono-600
  path = /tmp
  create mask = 0700
  guest ok = Yes
  printable = Yes
  printer name = lp2
```

Der entscheidende Unterschied zu einem share für Verzeichnisse sind eigentlich nur die Anweisungen `printable = yes` und `printer name = lp2`

Daran merkt Samba, daß es sich hierbei um einen Drucker-Dienst und nicht um einen Dateidienst handelt. Der Windows-User bekommt jetzt folgendes Bild zu sehen:



Die Angabe printer `name = lp2` bezieht sich auf den Namen des Druckers, der unter Linux gültig ist. In diesem Fall darf natürlich auch nicht `browseable = No` eingetragen werden, sonst sieht der Windows-User den Drucker nicht in seiner Liste.

Die Angabe `guest ok = yes` ermöglicht das Drucken ohne Passworteingabe.

Sicherheitsebenen

Samba kennt verschiedene Sicherheitsebenen, die alle durch die Angabe

```
security = ...
```

in der `[global]`-Sektion eingestellt werden. Das heißt, sie gelten immer für den gesammten Server und sind nicht für jedes einzelne share einstellbar. Denkbare Werte für security sind `share, user, server` und `domain`. Diese vier Sicherheitsebenen werden hier einzeln erläutert.

security = share

Bisher haben wir in unsere Samba Konfiguration immer die Zeile

```
security = share
```

in die `[global]`-Sektion eingetragen. Dieser Eintrag entspricht der "Zugriffssteuerung auf Freigabeebene" unter Windows. Es ist die geringste Sicherheitsstufe, die Zugriff auf bestimmte shares nur über die Frage klärt, ob ein share öffentlich ist oder nicht. Allerdings kann auch unter dieser Einstellung mit Samba eine userbezogene Freigabe realisiert werden, indem dem share die Anweisung

```
guest ok = No
```

gesetzt wird. In diesem Fall wird Samba ein Passwort anfordern, das zu dem Usernamen passen muß, der unter Windows angegeben wurde. Auch hier gilt, daß ab Win98 die Passwörter verschlüsselt übermittelt werden, also die Zeile

```
encrypt passwords = Yes
```

in der `[global]`-Sektion stehen muß. Zusätzlich muß der entsprechende User sowohl Linux bekannt sein, als auch ein verschlüsseltes Passwort in der Datei `/etc/smbpasswd` besitzen. (Siehe Userverzeichnisse)

Die Frage, wer dann nun was in dem Verzeichnis anstellen darf wird wiederum von Linux selbst beantwortet. Es gelten die normalen Unix-Dateizugriffsrechte des jeweiligen Users. Das ist auch der Grund, warum jeder Samba-User auch eine gültige Unix UserID besitzen muß.

Zusammenfassend ist also zu sagen, daß Samba mit dieser Methode noch wesentlich sicherere Zugriffe zulässt, als es Windows95/98 zulassen würde. Andererseits ist diese Methode erheblich eingeschränkt, was die Möglichkeiten der userbasierten Zugriffskontrolle angeht.

`security = share` wird hauptsächlich in Systemen eingesetzt, die alle Dienste frei - ohne Passwörter - anbieten wollen, oder in Systemen, deren User nicht identisch mit den Windows-Usern sind.

security = user

Eine andere Möglichkeit wird durch die Einstellung

```
security = user
```

in der `[global]`-Sektion ermöglicht. Das ist - seit Samba 2.0 - die voreingestellte Methode. In dieser Einstellung muß sich ein Windows-User zuerst "einloggen", bevor er irgendwelche Dienste des Samba-Servers in Anspruch nehmen darf. Das heißt, er muß einen Usernamen und ein Passwort an den Samba-Server schicken, der wiederum überprüft, ob die beiden Angaben stimmen und erst nach erfolgreicher Prüfung Zugriff gewährt. Der Zugriff ist dann wiederum abhängig von den Linux-Rechten, die dieser User auf dem Samba-Server hat. Das heißt, daß diese Methode nur dann funktioniert, wenn auf dem Windows-Rechner die selben Usernamen

existieren, wie auf dem Samba-Server.

Wie schon bei früheren Passwortübermittlungen wird auch hier wieder zwischen unverschlüsselter Passwortübermittlung (bis einschließlich Win95) und verschlüsselter Passwortübergabe (ab Win98) unterschieden. Das heißt, wenn die Windows-Rechner mindestens Win98 fahren, muß die Zeile

```
encrypt passwords = Yes
```

in der `[global]`-Sektion stehen und die User müssen mittels **smbpasswd** angelegt werden.

Beachten Sie, daß der Name der angeforderten Resource nicht an den Server weitergeschickt wird, bevor sich der Windows-User nicht korrekt angemeldet hat. Das ist der Grund, warum Gast-shares (guest ok) in diesem Modus nicht funktionieren, ohne daß User automatisch in einen Gastaccount umgewandelt werden. Aber auch dazu müssen sie sich zuerst korrekt anmelden, also einen Useraccount besitzen.

security = server

In diesem Modus versucht Samba die Passwortüberprüfung nicht selbst vorzunehmen, sondern die angegebenen Username/Passwort Kombinationen an einen anderen SMB-Server zur Überprüfung weiterzuleiten (z.B. einen NT-Rechner). Wenn das fehlschlägt, wird automatisch auf die Einstellung `security = user` zurückgeschaltet.

Damit Samba weiss, an welchen Server es die Passwortüberprüfung weiterleiten soll, muß der entsprechende Server mit der Angabe `password server = ...` in der `[global]`-Sektion angegeben werden. Der angegebene Name muß ein NetBIOS Name sein, also der Name, unter dem der Rechner auch unter Windows ansprechbar ist. Dieser angegebene Server muß selbst im `security = user` Modus laufen, um die entsprechende Überprüfung vorzunehmen.

Achtung: Versuchen Sie niemals hier den Samba Server selbst einzutragen! Es würde zu einer Endlosschleife kommen, die den gesammten Samba-Server blockieren würde.

Aus der Sicht des Windows-Clients ist der Server-Modus absolut identisch mit dem User-Modus. Der Unterschied bezieht sich nur auf die Frage, wie der Samba-Server die Überprüfung der Zugriffsrechte vornimmt. Im User-Modus macht er es selbst, im Server-Modus leitet er es an einen anderen Server weiter. Natürlich kann dieser andere Server auch wieder ein Samba Rechner sein, der dann aber eben im User-Modus laufen muß...

Beachten Sie, daß der Name der angeforderten Resource nicht an den Server weitergeschickt wird, bevor sich der Windows-User nicht korrekt angemeldet hat. Das ist der Grund, warum Gast-shares (guest ok) in diesem Modus nicht funktionieren, ohne daß User automatisch in einen Gastaccount umgewandelt werden. Aber auch dazu müssen sie sich zuerst korrekt anmelden, also einen Useraccount besitzen.

security = domain

Dieser Modus funktioniert nur dann, wenn der Samba-Server mittels des Programms **smbpasswd** an eine bestehende Windows NT Domain angeschlossen wurde. Der Parameter `encrypted passwords` muß aktiviert sein. In diesem Modus versucht der Samba-Server alle Authentifizierungen an einen Primären- oder Backup Domain Controller einer Windows-NT Domain weiterzuleiten, genau so, wie es eine NT-Maschine selbst tun würde.

Trotzdem müssen User dem Linux-System bekannt sein, damit eine Bestimmung ihrer Zugriffsrechte auf einzelne Verzeichnisse möglich ist. Jeder User braucht also einerseits einen Account auf dem PDC der Windows-Domain und andererseits einen gültigen Acccount auf dem Linux-Server.

Wie schon bei dem Server-Modus, so ist dieser Modus aus der Sicht des Windows-Clients einfach der User-Modus. Der Unterschied liegt nur darin, auf welche Weise der Samba-Server die Passwörter authentifiziert.

Wie schon im Server-Modus, so muß auch hier der Rechner angegeben werden, der die Passwortüberprüfung vornimmt. Wieder benutzen wir die Angabe

```
password server = ...
```

in der `[global]`-Sektion. Nur jetzt geben wir nicht einen Server an, sondern eine Liste, durch Kommata getrennt. Hier können wir den PDC und alle existierenden BDCs angeben, also etwa

```
password server = NT-PDC, NT-BDC1, NT-BDC2
```

Wird statt einer Liste einfach nur ein Pluszeichen (+) angegeben, so versucht Samba den PDC und die BDCs selbst herauszufinden.

Beachten Sie auch, daß der Name der Domain, der wir uns anschließen, wiederum mit dem Parameter `workgroup = ...` angegeben wird.

Damit unser Samba-Server aber überhaupt Mitglied einer Domain werden kann, müssen wir erst dafür sorgen, daß er davon auch weiss. Dazu kommt wiederum das Programm smbpasswd zur Anwendung, das wir ja schon zum Anlegen der User und Wechseln der Passwörter benutzt hatten.

Mit dem Aufruf von

```
smbpasswd -j Domain
```

wird der Samba-Server in die genannte Domain aufgenommen. Damit das funktioniert, muß der Administrator der NT-Domain mit dem Programm *Server Manager for Domains* den primären NetBIOS Namen des Samba-Servers in die Liste der Domain-Mitglieder aufgenommen haben.

Erst nachdem dieser Befehl ausgeführt wurde, sollte die smb.conf auf `security = domain` gesetzt werden. Von nun an sendet der Samba-Server alle Anfragen an den PDC zur Authentifizierung weiter.

NMBD Einstellungen

Alle bisherigen Einstellungen, die wir in den vorangehenden Beispielkonfigurationen getroffen hatten, bezogen sich auf den SMB-Daemon smbd. Es ging um die Fragen der Freigaben und Sicherheitskonzepte. Damit ein Windows-Netz aber richtig funktioniert, muß zusätzlich noch ein Dienst laufen, der die NetBIOS-Namen in IP-Adressen umsetzt und die Liste der Windows-Netzwerkumgebung verwaltet, der NMBD. Dieser Daemon wird grundsätzlich zusammen mit dem SMBD gestartet, entweder über inetd oder als Stand-Alone-Server.

Die primäre Aufgabe dieses Daemons ist es, die Namensauflösung im SMB-Netz zu ermöglichen. Dabei ist es unerheblich, ob die entsprechenden NetBIOS-Namen den DNS-Namen entsprechen, es gibt aber viele Fälle, wo sich Probleme vermeiden lassen, wenn zumindestens der Samba-Server hier immer den selben Namen trägt.

Die Einstellungen des NMBD werden auch in der Datei /etc/smb.conf vorgenommen, dort aber nur in der `[global]`-Sektion. Dieses Kapitel beschreibt die notwendigen Einstellungen für die Namensauflösung, im nächsten Kapitel betrachten wir die Verwaltung der Browsing-List (Windows-Netzwerkumgebung), die auch vom NMBD übernommen wird.

Etwas genaueres zu NetBIOS Namen

NetBIOS ist die grundlegende Protokollfamilie für Windows-Netze. Es arbeitet mit den folgenden drei Transport/ Vermittlungsschicht Protokollen:

- TCP/IP
- NetBEUI
- IPX/SPX

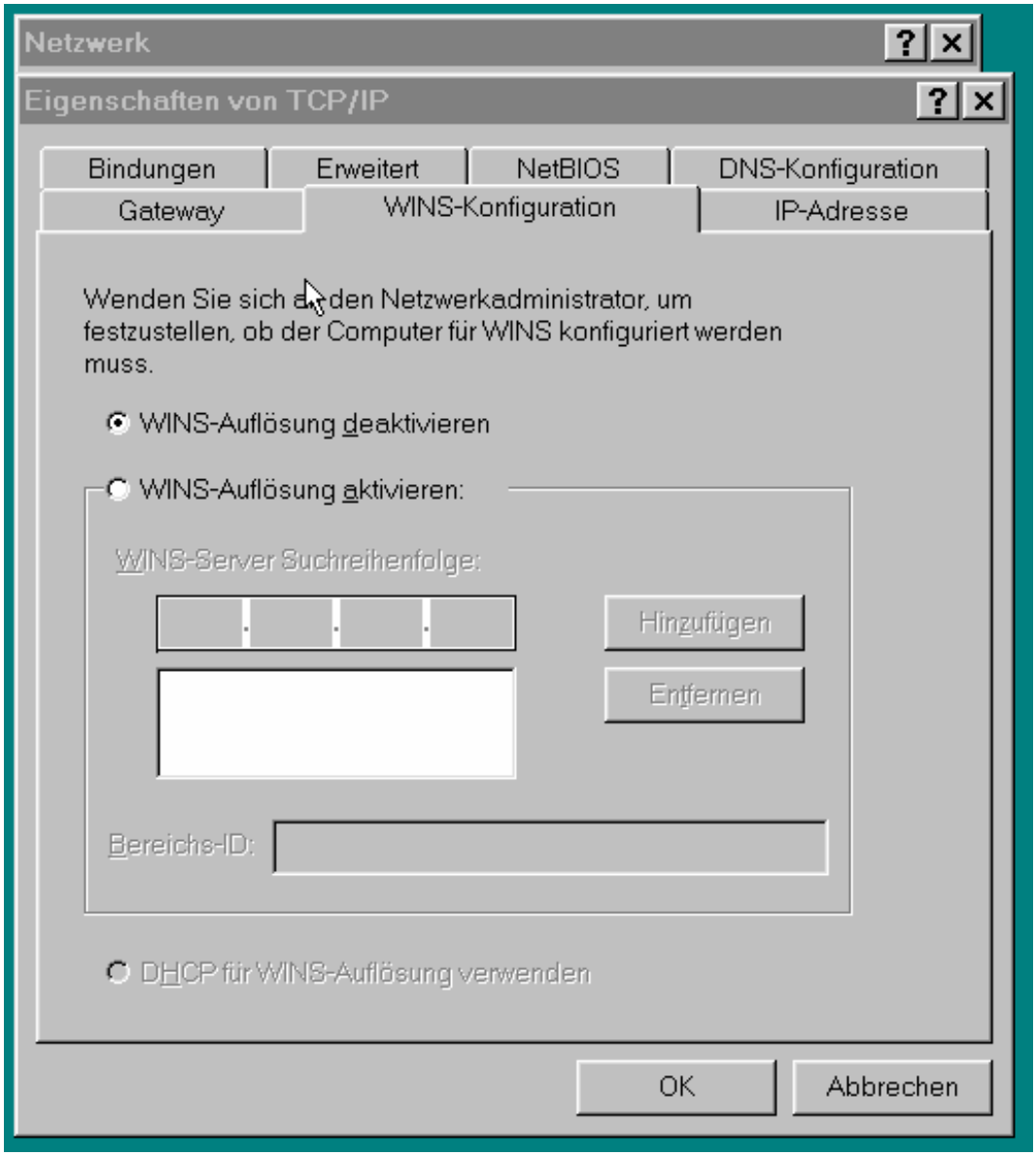
Samba benutzt grundsätzlich nur NetBIOS über TCP/IP. NetBIOS Anwendungen (wie Samba) bieten ihre Dienste

im Netz unter einem NetBIOS Namen an. Dieser Name muß vorher im Netz beansprucht worden sein. Nachdem aber diese Namen nicht zwangsläufig mit den DNS-Namen im TCP/IP Netz übereinstimmen müssen, gibt es zwei grundsätzliche Herangehensweisen, um im Netz Kommunikation zu ermöglichen. Entweder werden Broadcast-Aufrufe an alle angeschlossenen Rechner verschickt, oder es wird ein spezieller Windows Internet Name Service (WINS) angeboten, der die Auflösung von Namen in IP-Adressen vornimmt.

Größere Netze sollten grundsätzlich die zweite Möglichkeit (WINS) in Anspruch nehmen, denn sonst wird das Netz mit unnötig vielen Paketen belastet.

Um mit WINS arbeiten zu können muß ein Rechner im Netz diesen Dienst auch anbieten. Das kann entweder ein WindowsNT/2000 Rechner sein, oder eben wieder ein Samba Server. In der Regel wird der NT Rechner benutzt, wenn Samba in einer gemischten NT/Samba Umgebung arbeitet. Wenn nur Samba-Server zur Verfügung stehen, wird einer dieser Server den Dienst übernehmen.

Windows-Rechner müssen diesen Server dann entsprechend unter den Eigenschaften von TCP/IP eintragen:



Zu beachten ist, daß - um die Netzwerkumgebung richtig darstellen zu können - immer nur ein WINS-Server pro Netzsegment (bzw. Workgroup) aktiv sein sollte. Ansonsten sehen die verschiedenen Windows-Clients jeweils nur die Rechner, die den entsprechend gleichen WINS-Server benutzen.

Samba als WINS-Client

Um einen Samba Server an einen bestehenden WINS-Server anzuhängen, also ihn zu zwingen, seine Dienste und Namensauflösungen über diesen Server vorzunehmen, muß nur die folgende Zeile in der `[global]`-Sektion

eingetragen werden:

```
[global]
  ...
  wins server = 123.45.67.89
  ...
```

Der Befehl `wins server =` erfordert entweder die IP-Adresse oder den DNS-Domainnamen des entsprechenden Servers.

Viele Dienste von Samba, die die Browsing-Liste betreffen funktionieren nur, wenn ein WINS-Server im Netz aktiv ist. Ob dieser Server jetzt ein NT-Rechner oder wiederum ein Samba-Server ist, spielt keine Rolle.

Samba als WINS-Server

Wenn in einem Netz keine NT-Maschinen laufen, so kann auch Samba selbst als WINS-Server arbeiten. Allerdings darf in diesem Netz dann absolut nur ein Samba-Server als WINS-Server agieren.

Die notwendige Einstellung in der smb.conf lautet:

```
[global]
  ...
  wins support = yes
  ...
```

Beachten Sie, daß NIEMALS beide Einstellungen (Server und Client) gleichzeitig benutzt werden dürfen. Wenn Samba als Server arbeitet ist die Zeile `wins server = ...` nicht nur unnötig, sondern schlicht verboten. Das % h und %v sind also sogenannte Zeichenkettensubstitutionen, die vom Server durch eine bestimmte Zeichenkette ausgetauscht (substituiert) werden. Alle denkbaren Zeichenkettensubstitutionen, die Samba unterstützt, finden Sie in der Zusammenfassung Zeichenkettensubstitutionen.

1.113.5 - Einrichtung und Konfiguration grundlegender DNS-Dienste

Beschreibung: Prüfungskandidaten sollten in der Lage sein, Namensauflösung zu konfigurieren und Probleme mit lokalen caching-only Nameservern zu lösen. Dies benötigt ein Verständnis der Prozesse der Domainregistrierung und der DNS-Auflösung. Ebenfalls erforderlich ist ein Verständnis der wichtigsten Unterschiede der Konfigurationsdateien von BIND und BIND 8.

Die wichtigsten Dateien, Bezeichnungen und Anwendungen:

- `/etc/hosts`
- `/etc/resolv.conf`
- `/etc/nsswitch.conf`
- `/etc/named.boot` (v.4) oder `/etc/named.conf` (v.8)

Adressen in einem IP-Netzwerk sind immer nummerische Adressen. Diese Adressen sind zwar logisch aufgebaut, jedoch nicht unbedingt dafür geeignet, daß man sich davon 500 verschiedene merkt. Menschen erinnern sich einfach lieber an Namen als an Nummern.

Um dieser Tatsache Rechnung zu tragen bietet Linux ein System an, wie IP-Adressen mit Namen verknüpft werden können.

Die Bibliotheksfunktionen **gethostbyname** und **gethostbyaddr** sind für die Verwaltung der Hostnamen und IP-Adressen zuständig. Jedes Programm, das mit IP-Adressen oder Hostnamen umgeht, benützt diese Funktionen.

Es existieren mehrere Informationsquellen, die von diesen beiden Funktionen benutzt werden, um einen Namen mit einer Adresse zu verknüpfen.

- **Die Datei /etc/hosts**
  Diese Datei enthält IP-Adressen und dazu passende Namen (Siehe auch Abschnitt 1.112.3). Sie dient dazu, die Namen der Rechner im lokalen Netz zu verwalten und eventuell noch häufig benutzte Namen fremder Rechner mit Adressen zu versehen. Diese Technik ist aber sehr statisch. In einem lokalen Netz können wir uns darauf verlassen, daß sich Adressen nicht dauernd ändern, aber in einem fremden Netz gibt es dafür keine Garantie.
- **Das Domain Name System (DNS)**
  Dieses System teilt das gesamte Internet in Verwaltungseinheiten (domains) auf, die autonom administriert werden. Jede Domain betreibt sogenannte Nameserver, die die Adressen und Hostnamen ihres Verwaltungsgebietes kennen. Die Nameserver sind über eine Art Baumstruktur miteinander verbunden, so daß man die Summe aller Nameserver als globale und verteilte Datenbank betrachten kann. Jeder Nameserver kann über die Informationen über die über ihm liegenden Nameserver jede beliebige Adresse im Internet herausfinden. Welche Nameserver benutzt werden sollen, wird in der Datei `/etc/resolv.conf` eingestellt.

In welcher Reihenfolge diese Informationsquellen herangezogen werden, kann in den Dateien `/etc/host.conf` und `/etc/nsswitch.conf` festgelegt werden. All diese Zusammenhänge sind bereits im Abschnitt 1.112.3 besprochen worden.

Die Technik der Nameserver bedingt natürlich eine zentrale Verwaltung. Diese Verwaltung wird von einer Organisation mit Namen IANA (Internet Assigned Numbers Authority) abgewickelt. Die Struktur des gesamten DNS-Systems wird über eine überschaubare Menge von Toplevel-Domains gesteuert.

Es gibt zwei unterschiedliche Arten von Toplevel-Domains. Die sogenannten *generischen* und die *länderspezifischen* Toplevel-Domains. Die generischen Domains sind 14 festgelegte Domains, die alle eine bestimmte Bedeutung haben:

- **.aero**
  neu, reserviert für Mitglieder der Luftfahrtindustrie.
- **.biz**
  eine neu angelegte Toplevel-Domain nur für Unternehmen (businesses).
- **.com**
  die alte Toplevel-Domain für kommerzielle Unternehmen.
- **.coop**
  eine neue Domain für Kooperativen.
- **.edu**
  reserviert für Bildungseinrichtungen, die bei einer der sechs US-amerikanischen Agenturen akkreditiert sind.
- **.gov**
  reserviert für Einrichtungen der US-amerikanischen Regierung.
- **.info**
  eine neue Domain für informative Einrichtungen.
- **.int**
  eine neue Domain, reserviert für Organisationen, die aufgrund internationaler Verträge zwischen Regierungen zustande kamen.
- **.mil**
  reserviert für Einrichtungen des US-amerikanischen Militärs.
- **.museum**
  eine neue Domain nur für Museen.
- **.name**
  eine neue Domain, für die Verwendung im Privatbereich.
- **.net**
  ursprünglich für die Netzverwaltung gedachte Domain, die heute frei verfügbar ist.
- **.org**
  eine Domain für nicht kommerzielle Organisationen.
- **.pro**
  Eine im Aufbau befindliche Domain, reserviert für Berufsverbände.

Neben diesen generischen Toplevel-Domains existieren für jedes Land der Erde eine länderspezifische Domain. Diese Domain trägt als Namen die zwei Buchstaben ISO-Länderkennung. Eine vollständige Liste der existierenden Länderdomains erhalten Sie bei der IANA selbst unter www.iana.org/cctld/cctld-whois.htm.

Jede Toplevel-Domain wird von einer bestimmten Organisation verwaltet, die wiederum Domains unterhalb ihres Verwaltungsgebietes vergeben kann. So kann etwa die Verwaltung der deutschen Topleveldomain `.de` der Firma `foo` die Domain `foo.de` überlassen. Wichtig ist dabei, daß jede echte Domain - egal ob Toplevel oder nicht - einen oder mehrere Nameserver betreiben muß, die ihren jeweiligen Zuständigkeitsbereich abdeckt.

Auf der Wurzel des Baums des DNS arbeiten noch die sogenannten *root-server*, Nameserver, die die Namen aller Toplevel-Domains kennen und die Information über die von ihnen verwendeten Nameserver bereithalten.

Jeder Nameserver, egal ob der einer Toplevel-Domain oder einer gewöhnlichen, kennt die Liste aller *root-server*. Wenn jetzt ein Nameserver nach einer bestimmten Adresse sucht, so fragt er zunächst einmal einen *root-server* nach dem Nameserver der Toplevel-Domain dieser Adresse. Der wiederum kennt den Nameserver der Subdomain und der kennt - hoffentlich - die Adresse, die gesucht wurde.

Das gesamte Domain-System des Internets kann als eine große, weltweit verteilte Datenbank verstanden werden. Die einzelnen Knoten der Datenbank sind die Nameserver der verschiedenen Domains, die über die root-Server miteinander verbunden sind. Wie jede andere Datenbank auch, so hat auch die DNS-Datenbank eine vorgegebene Struktur und vorgegebene Feldeinträge.

Jeder Datensatz einer DNS-Datenbank besteht aus folgenden Elementen:

Domain-name
: Der Name der Domain, auf den sich der Datensatz bezieht.

Time-to-live
: Dieser Wert ist optional und bezeichnet die Stabilität eines Eintrags, also die Frage, wie lange er gültig sein

soll. Wird häufig weggelassen.

Type

Gibt an, um was für einen Typ Datensatz es sich handelt. Eine genaue Beschreibung der möglichen Typen finden Sie weiter unten.

Class

Die Informationsklasse. Für Internet Informationen steht hier immer der Begriff `IN`.

Value

Der eigentliche Wert des Datensatzes. Abhängig vom genannten Typ.

Die Datenbank besteht also aus Einträgen, die sich alle an dieses Format halten. Für den Typ der Information (Type) stehen folgende Werte zur Verfügung:

Typ	Bedeutung	Eintrag
`SOA`	Start Of Authority	Verschiedene Parameter für die Zone, die der Nameserver verwalten soll.
`A`	Address	Die Adresse eines Internet Hosts.
`MX`	Mail Exchange	Die Priorität und Name des Mailservers der Domain.
`NS`	Name Server	Name eines Nameservers der Domain.
`CNAME`	Canonical Name	Domainname eines Rechners (Aliasfunktion)
`PTR`	Pointer	Alias für eine numerische IP-Adresse
`HINFO`	Host Information	ASCII Beschreibung des Hosts (CPU, OS, ...)
`TXT`	Text	Nicht verwertbarer Text - Kommentar

Die einzelnen Einträge werden in Konfigurationsdateien geschrieben, die jeweils für eine sogenannte Zone gelten. Eine Zone entspricht entweder einem physikalischem Netz oder einer Domain selbst.

Für jede Zone existieren zwei solcher Konfigurationsdateien, eine für die Auflösung von Namen in Adressen und eine für die Auflösung von Adressen in Namen (reverse lookup).

Eine zentrale Konfigurationsdatei (natürlich im Verzeichnis `/etc`) sorgt für die Information, für welche Zonen der Nameserver verantwortlich ist und gibt die Positionen der Zoneninformationsdateien im Dateisystem an. Meist liegen diese Dateien unter `/var/named`.

Zuletzt existiert noch eine Datei `root.hint`, die nicht selbst editiert wird und die die Informationen über die root-Server im Internet enthält. Diese Information ist nötig, damit unser Nameserver in den weltweiten Verbund des DNS eingebunden ist.

Ein lokales Netz, das nicht permanent ans Internet angeschlossen ist und das vor allem nicht mit echten IP-Adressen arbeitet, muß natürlich keinen kompletten Nameserver unterhalten. In den meisten Fällen genügt es, einen sogenannten *caching only nameserver* bereitzustellen, der die Abfragen an andere Nameserver weiterleitet, aber selbst keine Informationen zur Verfügung stellt. Der Nameserver kann die Informationen über die bereits gefundenen Adressen zwischenspeichern (caching) und muß so nicht für jede Adresse erneut eine Anfrage starten.

Ein solcher Nameserver muß also keine Informationen über das lokale Netz beinhalten, sondern nur die Liste der *root-server* kennen. Diese Liste ist im Internet frei erhältlich und liegt bei der Installation eines Nameservers gewöhnlich bei. Meist heißt diese Datei `root.hint` (ab Version 8.0) oder `named.ca, named.root` (Version 4.x) und wird ins Verzeichnis `/var/named` installiert.

Es existieren heute zwei Generationen von Nameservern. Die erste benutzt den Berkley Internet Name Daemon **bind** Version 4, die zweite benutzt **bind** Version 8.0 (oder später). Dieser Generationsunterschied spielt eine große Rolle hinsichtlich der Syntax der Konfigurationsdateien. Moderne Installationen sollten heute alle mit Versionen ab 8.0 arbeiten. Die alten Versionen sind sehr unsicher!

Für den Prüfungsinhalt der LPI102 Prüfung ist noch das Wissen über beide dieser Versionen und vor allem über den Unterschied in den Konfigurationsdateien gefragt. Also werden wir beide Installationen durchspielen.

Nochmal die Aufforderung: Es sollten nur noch Versionen ab 8.0 verwendet werden!

Konfiguration eines caching-only Nameservers unter bind Version 4.0

Die zentrale Konfigurationsdatei der älteren bind-Version liegt im Verzeichnis `/etc` und heißt `named.boot`. Diese Datei enthält die grundlegenden Informationen über folgende Einstellungen:

- In welchem Verzeichnis liegen die Informationsdateien des Nameservers.
- Welche Zonen werden von diesem Server verwaltet und in welchen Dateien liegen die entsprechenden Informationen.
- Ist dieser Nameserver der primäre Server einer Domain oder ein sekundärer.
- Welche Nameserver sollen gefragt werden, wenn dieser Nameserver nicht in der Lage ist, eine Adresse aufzulösen (forwarders).
- Soll der Nameserver selbst Anfragen starten, oder nur einen anderen Nameserver fragen.

Nachdem unser Nameserver nur als caching only Server betrieben werden soll, sind nur die wenigsten dieser Einstellungen für uns relevant. Eine einfache `/etc/named.boot` Datei könnte folgendermaßen aussehen:

```
;
; /etc/named.boot Datei fuer einen caching only Server
;
directory     /var/named
;
;             domain                    file
;-----------------------------------------------
cache         .                         named.root
primary       0.0.127.in-addr.arpa      named.local
```

Kommentare in dieser Datei werden durch einen Strichpunkt (;) eingeleitet. Die erste Anweisung, die kein Kommentar ist, gibt an, in welchem Verzeichnis alle Dateien gefunden werden, auf die wir uns im weiteren Verlauf der Datei beziehen. In unserem Fall ist das das Verzeichnis `/var/named`.

Die nächste (nicht-Kommentar) Zeile gibt die Datei an, die für den caching-only Server die wichtigste ist. Die Datei `named.root` wird - dank der letzten Anweisung - im Verzeichnis `/var/named` gesucht. Sie enthält die Informationen über die root-server im Internet und sollte nicht selbst verändert werden. Die Datei kann - falls sie nicht bei der Installation bereits vorhanden war - im Internet über anonymes FTP bezogen werden: Dateiname `/domain/named.root` auf dem Rechner `FTP.RS.INTERNIC.NET`.

Über diese Datei hat unser Nameserver jetzt Kontakt zum weltweiten DNS. Er kennt jetzt alle root-Server und kann somit jede Adresse im Internet auflösen. Er wird sich auch gerade aufgelöste Adressen merken, so daß nicht für jede Adresse erneut ein Lookup stattfinden muß.

Die letzte Zeile unserer Konfigurationsdatei verweist auf die Datei `/var/named/named.local`, die ausschließlich die Information zum *localhost* (127.0.0.1) also zu unserem eigenen Rechner enthält. Diese Datei entspricht vom Aufbau her der Form, die auch "echte" Informationsdateien des Nameservers hätten, wenn sie verantwortlich für eine Domain wären:

```
;
; /var/named/named.local       Reverse mapping of 127.0.0
;
@                   IN  SOA   ns.mydomain.de. (
                              root.mydomain.de.
                              1          ; serial
                              360000     ; refresh: 100 Std
                              3600       ; retry:   1 Std
                              3600000    ; expire:  42 Tage
                              360000     ; minimum: 100 Std
                              )
                    IN  NS    ns.mydomain.de.
```

```
1                   IN  PTR   localhost.
```

Diese Datei enthält 3 Datensätze. Der erste ist der sogenannte *Start of Authority* (SOA) Eintrag. Er definiert den Namen unseres Nameservers (`ns.mydomain.de`), die E-Mail Adresse des Verwalters, wobei statt dem @ Zeichen ein normaler Punkt verwendet wird (`root.mydomain.de`) und eine zusammengesetzte Seriennummer, die festlegt, wie lange Einträge des Nameservers gültig sein sollen.

Der zweite Eintrag definiert den Nameserver (NS) der verwendet werden soll - in unserem Fall ist das der Rechner `ns.mydomain.de` selbst.

Der dritte und letzte Eintrag ist die Angabe, daß die Adresse, die auf 1 endet (bei 127.0.0.1 immer der Localhost) eben der *localhost* ist.

Wenn diese Einträge jetzt existieren, kann der Nameserver gestartet werden. Normalerweise wird das über ein Init-Script (z.B. `/etc/init.d/named start`) erledigt. Bei den **bind4** Nameservern existierten meist auch noch zwei Hilfsprogramme **named.restart** und **named.reload**, mit denen ein Nameserver neu gestartet werden kann, oder gezwungen werden kann, die Konfigurationsdateien neu einzulesen.

Konfiguration eines caching-only Nameservers unter bind Version 8.0

Der wesentliche Unterschied bei den Konfigurationsdateien der beiden Nameserver-Generationen liegt in der zentralen Konfigurationsdatei. Diese Datei enthält auch bei **bind8** die selbe Information, wie bei **bind4**, jedoch heißt die Datei anders und hat einen völlig anderen Aufbau.

Die zentrale Konfigurationsdatei der modernen Nameserver heißt `/etc/named.conf` und ist ähnlich aufgebaut, wie ein C-Programm. Logische Blöcke werden in geschweifte Klammern zusammengefasst. Nach einer geschlossenen Klammer folgt - wie auch nach jeder Anweisung - ein Strichpunkt. Die Konfigurationsdate eines caching only Servers könnte folgendermaßen aussehen:

```
// Konfigurationsdatei für einen caching-only Nameserver
// /etc/named.conf

options {
        directory "/var/named";
};

zone "." {
        type hint;
        file "root.hints";
};

zone "0.0.127.in-addr.arpa" {
        type master;
        file "local.zone";
};
```

Der erste Block (`options`) definiert globale Optionen. In unserem einfachen Beispiel findet sich hier nur die Information, in welchem Verzeichnis die Nameserver-Dateien zu finden sind, auf die sich der Rest der Einträge hier bezieht. Der Eintrag entspricht also genau der entsprechenden Angabe der alten Konfigurationsdatei.

Die einzelnen Zonen werden jetzt auch als Blöcke verwaltet, nicht mehr in einer Zeile, wie oben. Jede Zone hat mindestens einen Typ und eine Angabe einer Datei, die die Informationen über diese Zone beinhaltet. Wie oben haben wir wieder zwei Einträge, einmal für die Zone "." (die Wurzel des DNS-Baums) und einmal für unsere lokale Zone, das lokale Netz. Wie oben schon wird diese Angabe umgekehrt geschrieben, also statt `127.0.0` steht hier `0.0.127.in-addr.arpa`.

Die einzelnen Dateien in `/var/named` unterscheiden sich nicht, außer in ihrem Namen. Und der wäre prinzipiell

frei wählbar. Immerhin können wir zumindest bei der Zonendatei `local.zone` den SOA-Eintrag etwas verständlicher gestalten:

```
;       Zonendatei für den localhost
@               IN      SOA     ns.mydomain.de. root.mydomain.de. (
                                1       ; Serial
                                8H      ; Refresh
                                2H      ; Retry
                                1W      ; Expire
                                1D)     ; Minimum TTL
                        NS      ns.mydomain.de.
1                       PTR     localhost.
```

Die Angaben über die Zeiten können hier mit Prefixen versehen werden, die die Zeiteinheiten bezeichnen, statt sie immer in Sekunden angeben zu müssen. Die Prefixe `H, D, W` stehen für Stunde (hour), Tag (day) und Woche.

Auch hier wird der Nameserver über ein Init-Script gestartet, allerdings existieren die beiden Programme **named.restart** und **named.reload** nicht mehr. Stattdessen kann dem Nameserverprozeß ein HUP-Signal geschickt werden, das ihn zwingt, seine Konfigurationsdatei neu einzulesen.

Beide Nameserver-Versionen geben Diagnosemeldungen an den Syslog Daemon weiter. Ein Studium der Datei `/var/log/messages` sollte also im Fehlerfall immer der erste Schritt zur Problemlösung sein.

1.113.7 - Einrichten von Secure Shell (OpenSSH)

Beschreibung: Prüfungskandidaten sollten in der Lage sein, OpenSSH zu installieren und zu konfigurieren. Dieses Lernziel beinhaltet grundlegende Installation und Problemlösung von OpenSSH sowie die Konfiguration von **sshd** für den automatischen Start beim Booten.

Die wichtigsten Dateien, Bezeichnungen und Anwendungen:

- `/etc/hosts.allow`
- `/etc/hosts.deny`
- `/etc/nologin`
- `/etc/ssh/sshd_config`
- `/etc/ssh_known_hosts`
- `/etc/sshrc`
- **sshd**
- **ssh-keygen**

Die Techniken zum Einloggen in einen fremden Rechner mittels **telnet** und **rlogin** haben einen großen Nachteil, der in einem lokalen (vertrauenswürdigen) Netz keine große Rolle spielt, im Internet aber fatal sein kann. Bei beiden Protokollen werden alle Übertragungen unverschlüsselt realisiert. Das heißt, daß sowohl die Übertragung von Username und Passwort, als auch die der eigentlichen Sitzung von Lauschern mitgelesen werden kann.

Aus diesem Grund existiert eine sichere Lösung, die sogenannte *Secure Shell* oder kurz **ssh**. Dieses Protokoll besteht, genauso wie die beiden anderen oben genannten, aus einem Client-Server Paar. Auf dem Rechner, auf dem man sich sicher einloggen soll, läuft der Server (**sshd**), der User, der sich dort einloggen will, startet den Client (**ssh**).

Der Server **sshd** erlaubt nicht nur das sichere Einloggen, sondern ist auch noch ein Ersatz für **rsh**, das heißt, er kann auch einfach Befehle ausführen, und er ist auch in der Lage, Dateien sicher über das Netz zu transportieren, wenn der User statt **ssh** das Programm zum Kopieren (**scp**) aufruft.

Der **sshd** Daemon wird in der Regel als Stand-Alone Dienst gestartet, also über ein Init-Script und nicht - wie seine Vorgänger - über inetd. Für jede eingehende Verbindung wird dann ein neuer Prozeß gestartet. Dieser neue Prozeß kümmert sich dann um die Schlüsselübergabe, die Verschlüsselung, Authentifizierung, Kommandoausführung und den Datentransfer.

Jeder Rechner hat einen rechnerspezifischen RSA-Schlüssel (normalerweise 1024 Bit breit) der ihn eindeutig identifiziert. Der Server erstellt zusätzlich in regelmäßigen Abständen einen speziellen Serverschlüssel (meist 768 Bit), der niemals abgespeichert, sondern immer nur im Speicher gehalten wird.

Sobald ein Client eine Anfrage startet, schickt der Server ihm seine beiden Public-Keys (Server und Host Schlüssel). Der Client überprüft dann, ob der Hostschlüssel des Servers mit seinem gespeicherten übereinstimmt und hat so die Gewißheit, daß es sich tatsächlich um den gewünschten Server handelt und nicht um eine Fälschung. Anschließend erzeugt der Client eine Zufallszahl und verschlüsselt sie mit den beiden Schlüsseln des Servers. Diese verschlüsselte Zahl wird an den Server geschickt. Nur dieser Server, der den passenden Secret-Key hat, kann die Zahl wieder entschlüsseln.

Die Zahl wird im weiteren Verlauf benützt, um die gesamte Kommunikation zwischen Client und Server zu verschlüsseln. Da diese Zahl schon verschlüsselt übertragen wurde, ist es auch einem Lauscher nicht möglich, sie zu verwenden um die Kommunikation abzuhören.

Optional kann **sshd** auch die Mechanismen der **r-Befehle** benutzen, die auf die Datei `~/.rhosts` basieren. Standardmäßig ist dieses Feature aber abgeschaltet, da es die Sicherheit kompomittieren kann.

Konfiguration des Daemons

Der **sshd** wird über die Datei `/etc/ssh/sshd_config` konfiguriert. Diese Datei erlaubt alle notwendigen Einstellungen für den Server. Eine genaue Beschreibung aller Direktiven dieser Datei ist in der Handbuchseite **sshd_config**(5) nachzulesen. Wichtige Einstellungen sind:

Port *N*
: Die Angabe auf welchem Port **sshd** laufen soll. Normalerweise ist es Port 22.

HostKey *Dateiname*
: Spezifiziert die Datei, in der der Host-Schlüssel des Servers abgelegt ist. Normalerweise ist das die Datei `/etc/ssh/ssh_host_key`. Diese Datei darf nicht für alle Welt lesbar sein, ansonsten verweigert sich **sshd** die Datei zu verwenden.

ServerKeyBits *Zahl*
: Hier wird die Breite des Server-Schlüssels angegeben. Normalerweise 768.

KeyRegenerationInterval *Zeit*
: Die Anzahl der Sekunden, nach denen der Server-Schlüssel neu erstellt werden soll. Standardmäßig sollte hier 3600 (eine Stunde) stehen. Steht hier eine 0, so wird der Schlüssel niemals neu erstellt.

PermitRootLogin *yes|no*
: Ist es erlaubt, daß sich der Systemverwalter über **ssh** einloggen kann?

IgnoreRhosts *yes|no*
: Sollen die Dateien `~/.rhosts` und `~/.shosts` ignoriert werden?

Sind diese Einstellungen erledigt, so kann der Daemon gestartet werden.

Weitere Sicherheitseinstellungen

Obwohl **sshd** nicht über inetd gestartet wird, arbeitet er trotzdem mit den TCP-Wrappern. Diese Technik wurde bereits an anderer Stelle detailiert dargestellt. Der Servername für die Steuerung von **sshd** in den Dateien `/etc/hosts.allow` und `/etc/hosts.deny` ist einfach **sshd**.

Existiert die Datei `/etc/nologin`, so verweigert **sshd** jeden Einlogvorgang außer dem von root (sofern root-Eingänge in der Konfigurationsdatei erlaubt waren). Der Inhalt der Datei wird dem Client übermittelt. Ein Systemverwalter kann also diese Datei anlegen und eine kurze Erklärung hineinschreiben, warum der Zugang gerade verboten ist.

Die Dateien `/etc/ssh/ssh_known_hosts` und `$HOME/.ssh/known_hosts` enthalten die Public-Keys aller bekannten Rechner. Die globale Datei wird vom Systemverwalter administriert, die userbezogene wird automatisch angelegt und erweitert, sobald der User sich von einem bisher unbekannten Rechner einloggt.

Jede Zeile dieser Dateien enthält die folgenden durch Leerzeichen getrennten Felder:

```
Hostnamen Bits Exponent Modulus Kommentar
```

Hostnamen ist eine durch Kommas getrennte Liste von Namen oder Mustern (* und ? dürfen benutzt werden). Jedes Muster wird mit dem vollständigen Hostnamen des Rechners verglichen, der sich anmelden will.

Bits, Exponent und Modulus werden direkt den Host-Schlüsseln entnommen, die in den Dateien `/etc/ssh/ssh_host_*_key.pub` gespeichert sind. Das optionale Kommentarfeld wird nicht ausgewertet.

Über diesen Mechanismus können Hosts als bekannt (known) definiert werden. Jeder neue Host, wird - sobald er sich eingeloggt hat, als bekannter Host eingetragen. So können Versuche enttarnt werden, wenn ein fremder Rechner vorgibt, ein anderer - bekannter - zu sein.

Der Login-Vorgang

Wenn ein User sich über **ssh** erfolgreich angemeldet hat, dann erledigt **sshd** folgende Schritte:

- Wenn es sich um ein terminalbasiertes Login (nicht um eine Kommandoausführung) handelt, dann gibt **sshd** aus, wann der letzte Login war und zeigt den Inhalt der Datei `/etc/motd` an. Anschließend wird die Zeit des aktuellen Logins abgespeichert (um beim nächsten wieder sagen zu können, wann der letzte war).
- Wenn die Datei `/etc/nologin` existiert, wird ihr Inhalt ausgegeben und die Verbindung getrennt (außer bei einem root-Login).
- **sshd** wechselt zur Benutzerkennung des Users, der sich eingeloggt hat.
- **sshd** erzeugt eine grundlegende Umgebung (Shellvariablen wie PATH)
- **sshd** ließt die Datei `$HOME/.ssh/environment` wenn sie existiert und nimmt die dort gemachten Einstellungen in die Umgebung auf.
- **sshd** wechselt in das Homeverzeichnis des Users.
- Wenn `$HOME/.ssh/rc` existiert, wird es abgearbeitet. Ansonsten wird die Datei `/etc/ssh/sshrc` gesucht und falls gefunden ausgeführt.
- Startet die Shell (oder das angegebene Kommando)

Schlüsselerzeugung mit ssh-keygen

Der Befehl **ssh-keygen** erzeugt und verwaltet die RSA und DSA Schlüssel für **ssh** Verbindungen. Normaluser können sich damit einen Schlüssel erzeugen, der Systemverwalter kann auch den Host-Schlüssel damit anlegen.

Diese Schlüssel sind nicht zwingend für den Gebrauch von **ssh** erforderlich, bestimmte Server verlangen ihn aber.

Es stehen zwei verschiedene Schlüsseltypen zur Verfügung, **rsa** und **dsa**. Mit dem Befehl

```
ssh-keygen -t Typ
```

wird ein neues Schlüsselpaar angelegt. Speicherort und Name werden beim Anlegen erfragt. Als *Typ* stehen **rsa** und **dsa** zur Auswahl.

Normalerweise wird dieses Programm hauptsächlich vom Init-Script aufgerufen, wenn das erste Mal der SSH-Server oder -Client betrieben wird, um einen Host-Schlüssel zu erzeugen.

Es ist aber auch möglich, über diese Schlüsselpaare eine Authentifizierung zu steuern, so daß ein User beim Einloggen via ssh kein Passwort mehr angeben muß.