



# Benutzer- und Berechtigungskonzepte unter Linux

Autor: H. Degenhardt ([hede@pingos.org](mailto:hede@pingos.org))

Autor: J. Meinhold ([j.meinhold@imail.de](mailto:j.meinhold@imail.de))

Autor: M. Kleine ([kleine\\_matthias@gmx.de](mailto:kleine_matthias@gmx.de))

Layout: Matthias Hagedorn ([matthias.hagedorn@selflinux.org](mailto:matthias.hagedorn@selflinux.org))

Lizenz: GFDL

Dieser Text ist den grundlegenden Konzepten des Benutzer- und Berechtigungssystems von Linux gewidmet. Das Thema "Benutzerverwaltung" baut auf diesen Konzepten auf, wird hier aber nicht explizit behandelt. Vielmehr soll ein Kenntnis der wichtigsten Begriffe und ein Verständnis für das große Ganze vermittelt werden, welches in der Benutzerverwaltung seine Anwendung findet. Den praktischen Aspekten der Benutzerverwaltung ist ein [separates Kapitel](#) gewidmet.

## **Inhaltsverzeichnis**

### **1 Einleitung**

### **2 Was ist ein Benutzer?**

- 2.1 Benutzername und Paßwort
- 2.2 Benutzer-Id (UID) und Gruppen-Id (GID)
- 2.3 Nach der Anmeldung - die Shell
- 2.4 Das Heimatverzeichnis
- 2.5 Was ist nun also ein Benutzer?

### **3 Benutzertypen**

- 3.1 root
- 3.2 Systembenutzer
- 3.3 Standardbenutzer

### **4 Benutzerklassen: user, group und others**

### **5 Berechtigungsklassen: Lesen, Schreiben und Ausführen**

- 5.1 Lesen
  - 5.1.1 Leserecht für Dateien
  - 5.1.2 Leserecht für Verzeichnisse
- 5.2 Schreiben
  - 5.2.1 Schreibrecht für Dateien
  - 5.2.2 Schreibrecht für Verzeichnisse
- 5.3 Ausführen
  - 5.3.1 Ausführrecht für Dateien
  - 5.3.2 Ausführrecht für Verzeichnisse

### **6 Die zentralen Benutzerdateien**

- 6.1 Die Datei /etc/passwd
- 6.2 Die Datei /etc/shadow
- 6.3 Die Datei /etc/groups
- 6.4 Das Verzeichnis /etc/skel
- 6.5 Network Information Service (NIS)

## **1 Einleitung**

Die Nutzung von Informationssystemen ist üblicherweise mit einem Zugangssystem verbunden, welches die Verwendung des Systems auf eine bekannte Benutzergruppe beschränkt, Daten über die registrierten Benutzer speichert und die Verteilung der Ressourcen auf die Benutzer steuert. Häufig ist die Konzeption des Zugangssystems für den einzelnen Benutzer transparent - außer seinem Benutzernamen und einem Paßwort benötigt der Benutzer kaum weitere Kenntnisse, um das System in Anspruch zu nehmen. Für die Arbeit mit einem Linux-System sollten Sie sich dennoch einige elementare Kenntnisse über dessen Benutzer- und Berechtigungskonzept aneignen.

Die Notwendigkeit für diese Konzepte ergibt sich für Linux aus seiner Mehrbenutzerfähigkeit. Ein erster wichtiger Aspekt ist der Schutz des Systems vor den Handlungen seiner Benutzer. Weiterhin müssen auch die einen Benutzer vor den Handlungen der anderen geschützt werden. Und schließlich darf bei allem Schutz des Systems und der Benutzer voreinander das Miteinander-Arbeiten nicht allzusehr erschwert werden. Um all dies zu gewährleisten, bedarf es eines feinkörnigen Systems der Einschränkungen und Erlaubnisse. Dieses System ideal an die jeweiligen Gegebenheiten anzupassen, ist die Aufgabe des Systemverwalters. Es bleibt zu hoffen, daß er diese Aufgabe in Absprache mit den Benutzern vornimmt.

Dieser Text ist derzeit in fünf Hauptabschnitte gegliedert. Im ersten Abschnitt [Was ist ein Benutzer?](#) wird erläutert, was einen Benutzer unter Linux ausmacht. Der zweite Abschnitt [Benutzertypen](#) behandelt die Unterschiede zwischen Standardbenutzern, Systembenutzern und dem Superuser root. Abschnitt drei [Benutzerklassen: user, group und others](#) widmet sich den für das Berechtigungskonzept zentralen Benutzerklassen unter Linux. In Abschnitt vier [Berechtigungstypen: Lesen, Schreiben und Ausführen](#) werden die Berechtigungstypen für diese Benutzerklassen im Detail ausgeführt. Abschnitt fünf schließlich beschreibt überblicksweise die für die [Benutzerverwaltung zentralen Konfigurationsdateien](#).

Die Autoren sind sich der Tatsache bewußt, daß damit wesentliche Themen, die in dieses Kapitel gehören, noch nicht abgehandelt wurden. Es ist geplant, diese Abschnitte in der kommenden *SelfLinux-Release* nachzureichen.

## 2 Was ist ein Benutzer?

### 2.1 Benutzername und Paßwort

Ein Zugang oder **Account** besteht in der Informationstechnik aus einer Benutzername/Paßwort-Kombination. Von einer weiteren Verkomplizierung des Zugangssystems, etwa durch Einsatz von Verschlüsselungstechnologien aus Gründen erhöhter Sicherheitsanforderungen, soll in diesem Kapitel nicht die Rede sein.

Erwartungsgemäß melden Sie sich auch bei Linux durch die Angabe Ihres Benutzernamens und des zugehörigen Paßwortes an. Mehr muß der Anwender im allgemeinen nicht wissen. Aus Systemsicht sind jedoch noch einige weitere Attribute Ihres Accounts interessant.

### 2.2 Benutzer-Id (UID) und Gruppen-Id (GID)

Während Sie üblicherweise einfach einen Benutzernamen angeben, um sich auf einen bestimmten Benutzer zu beziehen, verwendet Linux intern lediglich eine Identifikationsnummer, die sogenannte **UID (User-Id)**. Betrachten Sie die (verkürzte) Ausgabe des folgenden Kommandos:

```
user@linux ~/ # id
uid=500(matthias)
```

In Klammern sehen Sie den vertrauten Benutzernamen, hier **matthias**. Die **UID** ist 500. Für interne Zwecke findet nahezu ausschließlich die **UID** Verwendung. Selbst wenn Sie Kommandos absetzen, welche den Benutzernamen als Parameter erwarten, findet zunächst ein Mapping des Namens auf die **UID** statt, bevor die gewünschte Aktion ausgeführt wird. Umgekehrt sollten Sie sich nicht täuschen lassen, wenn Ihnen ein Kommando den Benutzernamen anstelle der **UID** liefert. Betrachten Sie z.B. die folgende (wiederum etwas komprimierte) Ausgabe:

```
user@linux / # ps -aux
```

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.0	0.0	448	64	?	S	Jun21	0:07	init [3]
root	2	0.0	0.0	0	0	?	SW	Jun21	0:00	[keventd]
nobody	650	0.0	0.1	5680	716	?	S	Jun21	0:00	
/usr/sbin/in.identd	-e									
nobody	653	0.0	0.1	5680	716	?	S	Jun21	0:01	
/usr/sbin/in.identd	-e									
lp	746	0.0	0.1	1944	712	?	S	Jun21	0:00	lpd Waiting
matthias	15971	0.0	0.2	2528	1188	tty1	S	01:09	0:00	
/bin/sh/usr/X11R6/bin/startx										
matthias	16097	0.0	0.3	2816	1576	pts/1	S	01:09	0:00	/bin/bash

In der ersten Spalte listet **ps** bereitwillig die vertrauten oder weniger vertrauten Namen diverser Benutzer (root, nobody, lp, matthias) auf. Das System verwaltet allerdings die Prozesse nicht unter den Benutzernamen, sondern ausschließlich unter deren **UID's**. **ps** hat hier eigenständig ein Mapping von den **UID's** auf die Benutzernamen vorgenommen.

Neben der **UID** ist jedem Benutzer eine weitere Nummer zugeordnet, die sogenannte **Gruppen-ID (GID)**. Auch diese Nummer liefert das **id** Kommando (hier die etwas längere, aber immer noch verkürzte Ausgabe):

```
user@linux / # id
```

```
uid=500(matthias) gid=100(users)
```

Jeder Prozeß trägt *UID* und *GID* seines Erzeugers. Wie diese beiden Kennzahlen bei der Ermittlung von Berechtigungen verwendet werden, wird später im Detail erläutert.

### 2.3 Nach der Anmeldung - die Shell

Nach der Anmeldung möchte sich der Benutzer in einer Umgebung wiederfinden, welche ihm das Arbeiten ermöglicht. Erfolgt die Anmeldung nicht über ein grafisches Login-Werkzeug, so wird für gewöhnliche Benutzer eine Shell gestartet. Um welche Shell es sich handelt, wird dabei für jeden Benutzer einzeln festgelegt. In der Tat kann man für einzelne Benutzer auch festlegen, daß sie keine Shell erhalten - z.B. wenn man sie temporär vom System aussperren möchte oder wenn für einzelne Benutzer generell keine Shell-Umgebung ermöglicht werden soll. Wie später noch gezeigt wird, kann dies sehr sinnvoll sein und kommt durchaus häufig vor.

### 2.4 Das Heimatverzeichnis

Die Login-Shell startet im sogenannten Heimatverzeichnis des Benutzers. Dieses und alle seine Unterverzeichnisse "gehören" dem Benutzer. Was genau damit gemeint ist, wird in einigen Augenblicken erläutert. Im wesentlichen bietet das Heimatverzeichnis seinem Besitzer Platz für die Ablage von Dateien, welche meist oder ausschließlich von ihm verwendet werden. Neben den Arbeitsdateien und -verzeichnissen selbst liegen im Heimatverzeichnis noch eine Reihe (meist unsichtbarer) Dateien, welche applikationsspezifische Einstellungen des Benutzers enthalten. Es ist somit der Dreh- und Angelpunkt der Aktivitäten des Benutzers - eine gewohnte, vor anderen Benutzern geschützte Umgebung.

### 2.5 Was ist nun also ein Benutzer?

Benutzername und Paßwort, *UID*, Login-Programm (üblicherweise eine Shell) und Heimatverzeichnis sind es also, die zusammengenommen einen Benutzer auf einem Linux-System ausmachen. All diese Daten werden in einer zentralen Benutzerdatei verwaltet. Bevor diese näher beschrieben wird, folgt eine Einteilung der auf einem Linux-System möglichen Benutzer in Benutzertypen.

### 3 Benutzertypen

Hier nochmals die verkürzte Ausgabe eines `ps` Kommandos auf meinem System:

```
user@linux / # ps -aux
```

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	1	0.0	0.0	448	64	?	S	Jun21	0:07	init [3]
root	2	0.0	0.0	0	0	?	SW	Jun21	0:00	[keventd]
nobody	650	0.0	0.1	5680	716	?	S	Jun21	0:00	
/usr/sbin/in.identd	-e									
nobody	653	0.0	0.1	5680	716	?	S	Jun21	0:01	
/usr/sbin/in.identd	-e									
lp	746	0.0	0.1	1944	712	?	S	Jun21	0:00	lpd Waiting
matthias	15971	0.0	0.2	2528	1188	tty1	S	01:09	0:00	
/bin/sh/usr/X11R6/bin/startx										
matthias	16097	0.0	0.3	2816	1576	pts/1	S	01:09	0:00	/bin/bash

Die Benutzer **root**, **nobody**, **lp** und **matthias** sind als Inhaber der jeweiligen Prozesse gelistet. Ich darf Ihnen jedoch versichern, daß zum Zeitpunkt dieses Kommandos lediglich ein einziger Benutzer auf dem System angemeldet war, nämlich der Benutzer **matthias**. Die Tatsache, daß dennoch einige Prozesse auf dem System unter der Kennung anderer Benutzer laufen, zeigt bereits an, daß es verschiedene Typen von Benutzern geben muß. Gewöhnlichen Benutzern wäre es nämlich unmöglich, ohne vorherige Anmeldung einen Prozeß zu starten. Wir unterscheiden daher drei Benutzertypen: Erstens den Systemverwalter oder Superuser **root**, zweitens alle Standardbenutzer und drittens die Systembenutzer.

#### 3.1 root

Der Benutzer **root** ist mit allen Rechten ausgestattet, die ihm die Administration (bei Unachtsamkeit natürlich auch die Beschädigung!) des Systems erlauben. Diesem auch als **Superuser** bezeichneten Account ist immer die **UID 0** zugeordnet:

```
root@linux ~/ # id
```

uid=0(root)	gid=0(root)	Gruppen=0(root)	[...]
-------------	-------------	-----------------	-------

Dieser Account dient ausschließlich Eingriffen in die Konfiguration des Systems und sollte nur dann verwendet werden, wenn kein anderer Account die für eine Aufgabe notwendigen Rechte innehat. **Der unter Einsteigern beliebteste Fehler ist es, sich zunächst ausschließlich als root anzumelden.** Da nach der Systeminstallation ohnehin noch häufig administrative Aufgaben erledigt werden müssen, wird es als lästig empfunden, permanent zwischen einem Benutzer- und dem Superuser-Account hin- und herzuwechseln. Die Bequemlichkeit wird oft mit einer Beschädigung des Systems bezahlt.

Es gibt Prozesse, die immer unter der Kennung des Superusers laufen und auch laufen müssen. Das einfachste Beispiel ist der Prozeß **init**, der auch in der obigen Ausgabe erscheint. **init** ist der erste Prozeß, der nach dem Booten des Kernels die Kontrolle übernimmt und wird daher auch als "Vater aller Prozesse" bezeichnet. Dies drückt sich in der Prozeß-ID 1 aus. Da zum Zeitpunkt des Startens von **init** freilich noch kein Benutzer auf dem System angemeldet sein kann, andererseits aber jedem Prozeß eine gültige Benutzerkennung zugeordnet sein muß, und da **init** des weiteren zur Erledigung seiner Aufgaben mit weitreichenden Rechten ausgestattet sein muß, läuft **init** unter der Kennung des Superusers **root**. Daß dies nicht nur für **init** gilt, zeigt das folgende Kommando:

```
root@linux ~/ # ps aux | grep root

root          1  0.0  0.0   448   76 ?        S    Oct25   0:07 init
[...]
```

UID	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	4	0.0	0.0	0	0		SW	Oct25	0:01	[keventd]
root	602	0.0	0.1	1356	552		S	Oct25	0:01	/sbin/syslogd
root	1651	0.0	0.6	4856	3232		S	Oct25	0:08	/usr/sbin/cupsd
root	2063	0.0	0.0	1260	4	tty1	S	Oct25	0:00	/sbin/mingetty

```
--noclear tty1
[...]
```

Diese nach Prozeßnummern geordnete, verkürzte Liste zeigt im oberen Bereich zunächst den Vater aller Prozesse **init**. Danach folgen kernennahe Prozesse, welche bereits früh während des Bootvorganges gestartet werden. Später kommen einige Dienst- und Serverprozesse hinzu, darunter der Log-Dämon **syslogd**, der Druckdienst **cupsd** sowie einige Terminalprozesse (**mingetty**'s), welche das Einloggen auf den verschiedenen Konsolen ermöglichen. All diese Prozesse wurden nicht etwa von einem eingeloggtten Benutzer **root** gestartet, sondern automatisch beim Hochfahren des Systems - allerdings unter der Kennung von **root**, d.h. mit **UID 0**.

### 3.2 Systembenutzer

Je nach System kann eine Vielzahl von Prozessen und Diensten erwünscht sein, die bereits beim Hochfahren des Systems verfügbar sein sollen. Nicht jeder dieser Prozesse benötigt jedoch die volle Rechteeinstattung des Superusers. Man möchte natürlich so wenige Prozesse wie nur möglich unter einer **root** Kennung starten, da die weitreichenden Rechte solcher Prozesse unnötige Möglichkeiten für Mißbrauch und Beschädigung des Systems liefern.

Ein Systembenutzer-Account ist in diesem Sinne ein Benutzer-Account, der jedoch (nahezu) ausschließlich zur Ausführung von Programmen unter einer speziellen Benutzerkennung verwendet wird. Kein menschlicher Benutzer meldet sich normalerweise unter einem solchen Account an. Die oben bereits gezeigte Ausgabe eines **ps** Kommandos zeigt zwei häufige Beispiele: Der Drucker-Dämon **lpd** wurde unter der Benutzerkennung **lp** gestartet. Zwei Prozesse werden unter der Kennung des Benutzers **nobody** gelistet. **nobody** wird generell dann von Prozessen als Benutzerkennung verwendet, wenn nur ein Minimum an Rechten vergeben werden soll. Da **nobody** (laut Konvention, aber keineswegs notwendigerweise) keiner Gruppe angehört, wird er gewöhnlich der Benutzerklasse **others** angehören und somit die geringstmöglichen Rechte besitzen. Mehr zu [Benutzerklassen](#) folgt unten.

Ein Blick in die zentrale Benutzerdatei (Details zu dieser Datei folgen später) zeigt, daß vielen Systembenutzern explizit keine Shell zugeordnet wird:

```
root@linux ~/ # cat /etc/passwd | grep false

firewall:x:41:31:Firewall account:/var/lib/firewall:/bin/false
postfix:x:51:51:Postfix daemon:/var/spool/postfix:/bin/false
mysql:x:60:2:MySQL database admin:/var/lib/mysql:/bin/false
dpbox:x:61:56:DpBox account:/var/spool/dpbox:/bin/false
zope:x:64:2:Zope daemon:/var/lib/zope:/bin/false
vscan:x:65:65534:Vscan account:/var/spool/vscan:/bin/false
wnn:x:66:100:Wnn system account:/var/lib/wnn:/bin/false
pop:x:67:100:POP admin:/var/lib/pop:/bin/false
perforce:x:68:60:Perfoce admin:/var/lib/perforce:/bin/false
```

Das Programm **/bin/false** beendet sich ohne weitere Arbeit selbst, sodaß ein gewöhnlicher Login als einer der aufgeführten Benutzer nicht zu einer Shellsession führen kann. Prozesse unter dieser Kennung werden somit nicht von

einer Benutzershell gestartet, sondern über andere Mechanismen, die noch zu besprechen sein werden.

Es soll jedoch nochmals ausdrücklich erwähnt werden, daß die Unterscheidung zwischen Systembenutzern und Standardbenutzern willkürlich ist und nicht durch das Linux-Rechesystem selbst festgelegt wird. Es hilft jedoch beim Verständnis diverser Rechetkonzepte, wenn man sich der Tatsache bewußt ist, daß es zahlreiche Benutzer-Accounts gibt, welche ausschließlich im Zusammenhang mit bestimmten Diensten verwendet werden.

### 3.3 Standardbenutzer

Dies ist der **normale** Benutzer-Account, unter welchem jeder überlicherweise arbeiten sollte.



## 4 Benutzerklassen: user, group und others

Aus der Sicht des Systems existieren drei Benutzerklassen, wenn entschieden werden soll, ob die Berechtigung für einen Dateizugriff existiert oder nicht. Soll beispielsweise eine Datei gelöscht werden, so muß das System ermitteln, ob der Prozeß, welcher die Datei löschen möchte, das erforderliche Recht besitzt:

```
user@linux ~/ # rm testdatei

rm: Entfernen (unlink) von "testdatei" nicht möglich: Keine Berechtigung
```

In diesem Fall wurde dem `rm` Kommando der beabsichtigte löschende Zugriff auf die Datei verwehrt - der ausführende Benutzer hatte nicht das Recht, die Datei zu löschen. Um diese Entscheidung zu treffen, verwendet das System das Konzept der Benutzerklassen. Drei Benutzerklassen werden unterschieden: **user**, **group** und **others**. Jede dieser Benutzerklassen ist wiederum in ein Lese-, Schreib- und Ausführrecht unterteilt. Diese werden im folgenden als Berechtigungsklassen bezeichnet. Somit ergibt sich folgende Körnung für die einfachen Zugriffsrechte einer Datei:

<b>user-read</b>	Leserecht für Dateieigentümer
<b>user-write</b>	Schreibrecht für Dateieigentümer
<b>user-execute</b>	Ausführrecht für Dateieigentümer
<b>group-read</b>	Leserecht für Gruppe des Dateieigentümers
<b>group-write</b>	Schreibrecht für Gruppe des Dateieigentümers
<b>group-execute</b>	Ausführrecht für Gruppe des Dateieigentümers
<b>other-read</b>	Leserecht für alle anderen Benutzer
<b>other-write</b>	Schreibrecht für alle anderen Benutzer
<b>other-execute</b>	Ausführrecht für alle anderen Benutzer

Tabelle 1: Einfache Zugriffsrechte für Dateien.

Benutzerklassen sind also eng mit der Eigentümerschaft von Dateien verbunden. Jede Datei und jedes Verzeichnis ist sowohl einem Benutzer (einer *UID*) als auch einer Gruppe (einer *GID*) zugeordnet. *UID* und *GID* gehören zur elementaren Verwaltungsinformation von Dateien und Verzeichnissen und werden in der sogenannten **Inode** gespeichert.

Beim Zugriff auf eine Datei werden nun *UID* und *GID* des zugreifenden Prozesses mit *UID* und *GID* der Datei verglichen. Ist **other-read** gesetzt, darf jeder Benutzer lesend zugreifen und ein weiterer Vergleich erübrigt sich. Ist lediglich **group-read** gesetzt, muß der Zugreifende mindestens der Gruppe des Dateieigentümers angehören, d.h. eine identische *GID* aufweisen. Ist ausschließlich **user-read** gesetzt, so darf nur der Eigentümer selbst die Datei lesen. **root** ist von dieser Einschränkung freilich ausgenommen. ("Ich bin root, ich darf das!").

## 5 Berechtigungsklassen: Lesen, Schreiben und Ausführen

Da unter Unix letztlich alle Ein- und Ausgabeoperationen mit denselben Systemrufen vorgenommen werden, hat sich der Ausspruch **Alles ist eine Datei!** etabliert. Beispielsweise sind auch Verzeichnisse letztlich - wie alle anderen Einträge im Dateisystem - eine besonderer Typ Datei. Weitere Typen sind etwa Character- und Blockdevices, benannte Pipes, reguläre Dateien, symbolische Links und Sockets. Für jeden dieser Dateitypen haben die unterschiedlichen Rechte (Lesen, Schreiben und Ausführen) eine unterschiedliche Bedeutung.

Im Sinne einer minimalistischen Philosophie wurde dennoch die Abstraktion vorgenommen, alle diese unterschiedlichen Operationen unter dem Begriff der Datei zusammenzufassen und einheitliche Zugriffsmethoden bereitzustellen. Sowohl für den Entwickler wie auch für den Administrator bedeutet diese Abstraktion eine Vereinfachung seiner Aufgaben. Beispielsweise erfolgt die Vergabe des Leserechtes für reguläre Dateien auf exakt dieselbe Weise wie die Vergabe des Leserechtes für ein Sounddevice.

Da die Bedeutung der Lese-, Schreib- und Ausführrechte für die einzelnen Dateitypen im einzelnen besser bei den verschiedenen Spezialthemen besprochen werden kann, sollen im folgenden lediglich die unterschiedlichen Bedeutungen dieser Rechte für reguläre Dateien und Verzeichnisse erläutert werden. Reguläre Dateien sind Dateien mit definierten Dateiformaten, darunter etwa gewöhnliche Textdateien, Bildformate, Sounddateien, aber auch Programmdateien (Executables). Die Anzahl an Dateiformaten füllt ganze Enzyklopädien. Verzeichnisse sind Dateien, die einen Katalog von Dateien und Unterverzeichnissen enthalten können.

### 5.1 Lesen

#### 5.1.1 Leserecht für Dateien

Für Datendateien aller Art, wie z.B. Textdateien, Bilder usw., leuchtet das Leserecht unmittelbar ein: Die Datei kann zur Ansicht geöffnet und ihr Inhalt angezeigt oder abgespielt werden. Für Programmdateien ist das Leserecht weniger intuitiv verständlich. Manchen mag es überraschen, daß für die Ausführung eines Programms nicht das Leserecht gesetzt sein muß:

```
user@linux ~/ # su
Password: (Eingabe)
root@linux ~/ # chmod a-r /bin/echo
root@linux ~/ # ls -l /bin/echo

--wx--x--x    1 root      root          7064 2002-09-09 20:05 /bin/echo

root@linux ~/ # exit
user@linux ~/ # echo hallo

hallo
```

Mittels des `chmod` Kommandos wurde der Programmdatei des `echo` Kommandos temporär das Leserecht entzogen. Dennoch ist `echo` weiterhin verwendbar. Für manchen erscheint dies widersprüchlich, da zur Ausführung eines Programmes schließlich die Programmdatei **eingelesen** werden muß. Dies ist jedoch nicht notwendig. Das Laden eines Programmes ist sowohl technisch als auch konzeptionell ein völlig anderer Vorgang als das Lesen einer Datei. Daß **root** eine Sonderstellung einnimmt, zeigen übrigens die folgenden Kommandos:

```
user@linux ~/ # wc /bin/echo

wc: /bin/echo: Keine Berechtigung

user@linux ~/ # su
```

```
Password: (Eingabe)
root@linux ~/ # wc /bin/echo

    36      234      7064 /bin/echo

root@linux ~/ # chmod a+r /bin/echo
root@linux ~/ # exit
user@linux ~/ # wc /bin/echo

    36      234      7064 /bin/echo
```

Trotz fehlendem Leserecht durfte **root** mittels des **wc** Kommandos die Datei **lesen**, nämlich die Anzahl der Zeilen, Worte und Zeichen in der Datei zählen (eine nicht unbedingt sinnvolle Aktion, die hier nur zur Demonstration der Sonderstellung von root durchgeführt wurde). Der angemeldete Standardbenutzer durfte **wc** nicht auf die Datei anwenden und erhielt einen Berechtigungsfehler.

### 5.1.2 Leserecht für Verzeichnisse

Da Verzeichnisse keine Daten im eigentlichen Sinne enthalten, sondern lediglich Information über Dateien und Unterverzeichnisse, hat das Leserecht hier freilich eine andere Bedeutung. Das klassische Kommando zum Auslesen von Verzeichnisinformation ist **ls**. Betrachten wir ein Verzeichnis **testdir**, das eine Textdatei **testfile** und das Programm **tipptrainer** enthält.

```
user@linux ~/ # ls -l | grep testdir
drwxr-xr-x    2 matthias users          104 2002-11-05 23:43 testdir

user@linux ~/ # ls -l testdir

insgesamt 408
-rw-r--r--    1 matthias users           0 2002-11-05 23:43 testfile
-rwxr-xr-x    1 matthias users       417072 2002-11-05 23:43 tipptrainer
```

Das erste Kommando zeigt die aktuellen Berechtigungen für das Testverzeichnis. Die Leserechte sind für alle drei Benutzerklassen gesetzt. Folglich ist das zweite Kommando beim Auslesen des Verzeichnisses erfolgreich und gibt den Verzeichnisinhalt aus. Das Entfernen des Leserechtes hat ebenfalls den erwarteten Effekt:

```
user@linux ~/ # chmod a-r testdir
user@linux ~/ # ls -l testdir

ls: testdir: Keine Berechtigung
```

Es ist jedoch wichtig festzuhalten, daß damit keineswegs das Leserecht für die enthaltene Dateien entfernt wurde:

```
user@linux ~/ # ls -l testdir/testfile
-rw-r--r--    1 matthias users           0 2002-11-05 23:43 testdir/testfile

user@linux ~/ # cat testdir/testfile

Dies ist eine Testdatei.
```

```
user@linux ~/ # testdir/tipptrainer &
[1] 7761
```

Es dürfen sowohl die Berechtigungen der Testdatei wie auch ihr Inhalt ausgelesen werden. Das Programm `tipptrainer` läßt sich ebenfalls problemlos starten. Das Entfernen des Leserechtes für ein Verzeichnis wirkt sich also keineswegs auf die Dateien und Unterverzeichnisse aus, welche in dem Verzeichnis abgelegt sind. Es ist wichtig, dies zu verstehen, da ansonsten die Illusion entstehen könnte, mit dem Entfernen des Leserechtes für ein Verzeichnis schütze man auch dessen Inhalt vor dem Zugriff.

Es ist hilfreich, sich ein Verzeichnis als einen Katalog vorzustellen: Sein Inhalt ist eine Liste der Knoten, die sich innerhalb des Dateibaumes unterhalb des Verzeichnisses befinden. Das Leserecht ermöglicht das Auslesen der Kataloginformation, beispielsweise mittels des `ls` Kommandos. Ein Entfernen des Leserechtes verbietet zwar das Auslesen des Kataloges, nicht aber den Zugriff auf die katalogisierten Inhalte.

Das Leserecht eines Verzeichnisses hat auch keinerlei Auswirkung darauf, ob Verzeichnisinhalte gelöscht oder angelegt werden dürfen. Da bei diesen Operationen kein lesender, sondern ein schreibender Zugriff auf den "Kataloginhalt" erfolgt, spielt das Leserecht hier keine Rolle:

```
user@linux / # rm testdir/testfile
user@linux / # touch testdir/testfile2
```

Eine interessante Ausnahme bildet die Verwendung von Wildcards:

```
user@linux ~/ # rm testdir/*

rm: Entfernen von "testdir/*" nicht möglich: Datei oder Verzeichnis nicht
gefunden

user@linux ~/ # chmod a+r testdir
user@linux ~/ # rm testdir/*
user@linux / # ls -l testdir

insgesamt 0
```

Um den `*` durch Dateinamen zu ersetzen, welche schließlich dem `rm` Kommando übergeben werden, muß die Shell lesend auf das Verzeichnis zugreifen können. Da kein Leserecht gesetzt war, liefert dieser Zugriff kein Ergebnis und das `rm` Kommando schlägt mangels übergebener Argumente (d.h. Dateinamen) fehl. Nach Vergabe des Leserechtes wird der `*` durch die Dateinamen im Testverzeichnis ersetzt und diese an das `rm` Kommando zum Löschen übergeben. Die Verwendung von Wildcards zur Dateinamensubstitution erfordert folglich ein Leserecht für das betroffene Verzeichnis.

## 5.2 Schreiben

### 5.2.1 Schreibrecht für Dateien

Das Schreibrecht für reguläre Dateien ist ebenso intuitiv verständlich wie das Leserecht. Ist dieses Recht gesetzt, darf die Datei überschrieben oder weiterer Inhalt an sie angehängt werden. Das Schreiben auf Spezialdateien wie z.B. Sockets, Framebuffer oder Gerätedateien erfordert ebenfalls ein hundsgemeines Schreibrecht. Insbesondere wenn man solche Dateien selbst erzeugt hat (z.B. um ein ungewöhnliches Gerät in das System zu integrieren) sollte man nicht vergessen, das Schreibrecht zu setzen - ein trivialer Umstand, der schon so manche Arbeitsstunde gekostet hat.

### 5.2.2 Schreibrecht für Verzeichnisse

Erwartungsgemäß bezieht sich das Schreibrecht für Verzeichnisse auf das Anlegen und Löschen von Dateien in Verzeichnissen. Ohne Schreibrecht ist weder das eine noch das andere möglich.

```
user@linux ~/ # ls -l | grep testdir
drwxr-xr-x    2 matthias users          48 2002-11-06 00:08 testdir

user@linux ~/ # touch testdir/testfile
user@linux ~/ # chmod a-w testdir
user@linux ~/ # rm testdir/testfile

rm: Entfernen von "testdir/testfile" nicht möglich: Keine Berechtigung

user@linux ~/ # touch testdir/testfile2

touch: Erzeugen von "testdir/testfile2": Keine Berechtigung
```

Eine andere Bedeutung kommt dem Schreibrecht für Verzeichnisse nicht zu. Insbesondere benötigt man kein Schreibrecht in einem Verzeichnis, um eine darin enthaltene Datei oder auch nur deren Rechte zu ändern. Da diese Information direkt in die Datei bzw. deren Inode geschrieben wird, ist das Schreibrecht des Verzeichnisses ohne Belang:

```
user@linux ~/ # echo hallo > testdir/testfile
user@linux ~/ # chmod +r testdir/testfile
```

## 5.3 Ausführen

### 5.3.1 Ausführrecht für Dateien

Programme und Skripte sind es, die ausgeführt werden können. Programme liegen in Binärformaten vor - unter Linux hat sich das **Executable and Linking Format** (ELF) durchgesetzt, aber auch andere Formate werden unterstützt. Skripte werden von Interpretern ausgeführt und liegen in Textformat vor.

Bei Programmen, d.h. Dateien in einem ausführbaren Binärformat, liegt die Sache einfach. Ist das Ausführrecht gesetzt, darf das Programm aufgerufen und ausgeführt werden. Zunächst wird die Berechtigung geprüft und danach versucht, das Programm zu laden. Diese Reihenfolge zeigt der Versuch, eine Datei eines nicht ausführbaren Binärformates auszuführen, hier ein Gif-Bild:

```
user@linux ~/ # ls -l ./test.gif
-rw-r--r--    1 matthias users       15568 2002-11-07 15:03 ./test.gif

user@linux ~/ # test.gif

bash: ./cervisia-einstellungen.gif: Keine Berechtigung

user@linux ~/ # chmod +x cervisia-einstellungen.gif
user@linux ~/ # ./test.gif

bash: ./test.gif: cannot execute binary file
```

Bei Skripten muß feiner differenziert werden. Welcher Interpreter für ein Skript gestartet wird, ist durch die erste Zeile eines Skriptes hinter dem sogenannten Shebang (amerikanisch "the whole shebang" für der ganze Plunder) definiert. Die Bezeichnung "Shebang" ist vermutlich von "shell bang" abgeleitet. Es handelt sich um die Zeichenfolge `#!`, z.B.

Beispiel
<pre>#!/bin/sh # kommando1 kommando2 ...</pre>

In der ersten Zeile findet sich der Shebang nebst Angabe des zu verwendenden Interpreters. Im obigen Fall ist `/bin/sh` definiert, es könnten dort auch andere Shells oder Interpreter verschiedener Skriptsprachen wie *Perl* oder *Tcl* verwendet werden.

Weshalb wird dies hier überhaupt erläutert? Der Grund ist, daß Skripte auf verschiedene Weisen aufgerufen werden können und es von dieser Aufrufart abhängt, in welcher Weise sich das Ausführrecht auswirkt. Als Beispiel soll das folgende Skript dienen:

```
user@linux ~/ # pwd
/home/matthias

user@linux ~/ # cat testscript.sh
#!/bin/sh
echo "Hallo!"

user@linux ~/ # ls -l testscript.sh
-rw-r--r--  1 matthias users      24 2002-11-07 23:04 testscript.sh
```

Wie zu sehen, referenziert das Skript auf `/bin/sh` als Interpreter, gibt im Falle einer Ausführung die Zeichenfolge "Hallo!" aus und besitzt derzeit keinerlei Ausführrechte. Trotzdem kann es auf verschiedene Weisen ausgeführt werden:

```
user@linux ~/ # sh testscript.sh
Hallo!

user@linux ~/ # source testscript.sh
Hallo!

user@linux ~/ # . testscript.sh
Hallo!
```

Versucht man jedoch, das Skript namentlich aufzurufen, scheitert dies an der mangelnden Berechtigung. Hier die drei verschiedenen Möglichkeiten, das zu tun:

```
user@linux ~/ # testscript.sh
```

```
bash: ./testscript.sh: Keine Berechtigung
user@linux ~/ # ./testscript.sh
bash: ./testscript.sh: Keine Berechtigung
user@linux ~/ # /matthias/testscript.sh
bash: /home/matthias/testscript.sh: Keine Berechtigung
```

Zuerst durch simple Angabe des Namens (das `.` Verzeichnis muß hierfür in `PATH` aufgeführt sein), dann relativ, dann absolut. In allen drei Fällen fehlt das Ausführrecht.

Der Unterschied kann so erklärt werden. Geben Sie ein Kommando ein, so prüft die Shell, ob für dieses Kommando die Berechtigung zur Ausführung besteht. Dabei stellt jeweils das erste Wort Ihrer Eingabezeile das Kommando dar, die restlichen Worte bilden die Parameter. In den drei Beispielen unter Verwendung von `sh`, `source` und `.` wird also die Berechtigung dieser drei Kommandos geprüft und nicht diejenige des Skriptes selbst. Der Skriptname wird dann nur noch als Parameter an das Kommando übergeben und von diesem entsprechend behandelt. In diesem Fall muß nur noch das Leserecht gesetzt sein, denn das Kommando muß die Datei natürlich zumindest einlesen können:

```
user@linux ~/ # chmod -r testscript.sh
user@linux ~/ # sh testscript.sh

testscript.sh: testscript.sh: Keine Berechtigung
```

Referenzieren Sie hingegen das Skript in einer der drei genannten Arten direkt unter der Ausnutzung des Shebang-Mechanismus, prüft die Shell das Ausführrecht und verweigert u.U. die Ausführung. Das Leserecht muß freilich auch hier bestehen - Ausführen impliziert für Skripte (im Gegensatz zu Programmdateien) vorheriges Einlesen!

### 5.3.2 Ausführrecht für Verzeichnisse

Das Ausführrecht für Verzeichnisse bezeichnet das elementare Recht, dieses Verzeichnis zu betreten. Hier das grundlegende Beispiel:

```
user@linux ~/ # chmod -x testdir
user@linux ~/ # ls -l | grep testdir

drw-r--r--    2 matthias users          48 2002-11-07 23:50 testdir
user@linux ~/ # cd testdir
bash: cd: testdir: Keine Berechtigung
user@linux ~/ # chmod +x testdir
user@linux ~/ # cd testdir
user@linux ~/testdir/ # pwd

/home/matthias/testdir
```

Das "Betreten" eines Verzeichnisses ist jedoch allgemeiner zu verstehen als das bloße Wechseln des **current working directory**. Es ist vielmehr die grundlegende Voraussetzung für alle weiteren Operationen auf dem Verzeichnis und

seinen Inhalten. Lesen von Dateien, Anlegen und Löschen von Dateien und auch Ausführen von Dateien in einem Verzeichnis erfordern ein Ausführrecht auf diesem Verzeichnis. Dies gilt übrigens rekursiv auch für alle Unterverzeichnisse und deren Inhalte.

### Auslesen des Verzeichnisses:

```
user@linux ~/ # ls testdir
testscript.sh

user@linux ~/ # chmod -x testdir
user@linux ~/ # ls testdir

ls: testdir/testscript.sh: Keine Berechtigung
```

### Lesen einer Datei:

```
user@linux ~/ # chmod +x testdir
user@linux ~/ # echo "Neue Testdatei." > testdir/lesetest.txt
user@linux ~/ # cat testdir/lesetest.txt

Neue Testdatei.

user@linux ~/ # chmod -x testdir
user@linux ~/ # cat testdir/lesetest.txt

cat: testdir/lesetest.txt: Keine Berechtigung
```

### Anlegen und Löschen einer Datei:

```
user@linux ~/ # touch testdir/testfile

touch: Erzeugen von "testdir/testfile": Keine Berechtigung

user@linux ~/ # rm testdir/lesetest.txt

rm: Aufruf von lstat für "testdir/lesetest.txt" nicht möglich: Keine
Berechtigung

user@linux ~/ # chmod +x testdir
user@linux ~/ # touch testdir/testfile
user@linux ~/ # rm testdir/lesetest.txt
```

### Ausführen einer Datei:

```
user@linux ~/ # ls -l testdir/testscript.sh

-rwxr-xr-x    1 matthias users          25 2002-11-07 23:56
testdir/testscript.sh

user@linux ~/ # ./testdir/testscript.sh

Hallo!

user@linux ~/ # chmod -x testdir
```



```
user@linux ~/ # ./testdir/testscript.sh
bash: ./testdir/testscript.sh: Keine Berechtigung
```

**Auch für Unterverzeichnisse die Einschränkungen wirksam:**

```
user@linux ~/ # mkdir testdir/subdir
user@linux ~/ # touch testdir/subdir/testfile
user@linux ~/ # ls testdir/subdir

testfile

user@linux ~/ # chmod -x testdir
user@linux ~/ # ls testdir/subdir

ls: testdir/subdir: Keine Berechtigung
```

Damit soll die ausführliche Behandlung der Dateirechte hier abgeschlossen werden. Wie zu erkennen ist, ergeben sich aus dem an sich einfachen Konzept aus drei Benutzerklassen (**user**, **group**, **others**) und drei Berechtigungsklassen (**read**, **write**, **execute**) durchaus komplexe Zusammenhänge und Möglichkeiten zur Abstufung von Berechtigungen. Die Kombination der verschiedenen Rechte und ihre Anwendung auf unterschiedliche Dateitypen (wobei hier bereits eine Einschränkung auf reguläre Dateien und Verzeichnisse vorgenommen wurde) bietet ein breites Experimentierfeld und ist immer wieder für Überraschungen gut.

Am besten spielen Sie selbst einmal mit den vielfältigen Möglichkeiten, um eine gewisse Intuition im Umgang mit den Rechten zu gewinnen. Für die Zusendung besonders interessanter Beispiele sind die Autoren dankbar und werden sie gerne in dieses Kapitel aufnehmen.

Richten Sie nun - nach einer angemessenen Pause - Ihre Aufmerksamkeit auf die zentralen Benutzerdateien im Rahmen der Benutzerkonzeption.

## 6 Die zentralen Benutzerdateien

Um das Kapitel abzurunden und Ihnen einen Einblick in die Registratur von Benutzern unter Linux zu geben, werden im folgenden die zentralen Benutzerdateien beschrieben, welche alle notwendige Informationen über Benutzernamen, Paßwörter, Gruppenzugehörigkeiten und andere Benutzerattribute enthalten. Den Abschluß bildet ein Verweis ein System zur zentralen Benutzerverwaltung in Netzwerken, das NIS (Network Information System).

Die Dateien zur Benutzerverwaltung finden Sie unter Linux im Verzeichnis `/etc`. Es handelt sich dabei um die Dateien `/etc/passwd`, `/etc/shadow` und `/etc/group`.

An dieser Stelle sei nochmals darauf hingewiesen, daß die meisten Linux-Distributionen komfortable Werkzeuge zur Benutzerverwaltung mitliefern und es auch eine Reihe von Befehlen gibt, die für die Benutzerverwaltung gedacht sind. Diese werden in diesem Kapitel nicht erläutert. Hier soll grundlegendes Wissen über den Aufbau, Inhalt und Funktionen der Dateien erläutert werden, die für die Benutzerverwaltung unter Linux von Bedeutung sind.

### 6.1 Die Datei `/etc/passwd`

Die Datei `/etc/passwd` ist die zentrale Benutzerdatenbank.

Mit `cat /etc/passwd` können Sie einen Blick in diese zentrale Benutzerdatei werfen. Hier werden jetzt alle Benutzer des Systems aufgelistet. Zu beachten ist, daß hier alle Benutzertypen eingetragen sind, also vom **root** über die Systembenutzer bis zu den Standardbenutzern.

Der Eintrag eines Benutzers hat generell folgende Syntax:

`Username : Password : UID : GID : Info : Home : Shell`

<b>Username</b>	das ist der Benutzername in druckbare Zeichen, meistens in Kleinbuchstaben
<b>Password</b>	hier steht verschlüsselt das Paßwort des Benutzers (bei alten Systemen); heutzutage steht da ein <code>x</code> , das bedeutet daß das Paßwort in der Datei <code>/etc/shadow</code> steht; möglich ist es noch, den Eintrag leer zu lassen, dann erfolgt der Login ohne Paßwortabfrage (in der Datei <code>/etc/shadow</code> muß dann an Stelle des verschlüsselten Paßwortes ein <code>*</code> stehen)
<b>UID</b>	<b>User-Id</b> des Benutzers; die Zahl hier sollte größer als 100 sein, weil die Zahlen unter 100 für Systembenutzer vorgesehen sind, weiterhin muss die Zahl aus technischen Gründen kleiner als 64000 sein
<b>GID</b>	<b>Group-Id</b> des Benutzers; auch hier muss die Zahl wie bei der UID kleiner als 64000 sein
<b>Info</b>	hier können mehr Einträge stehen, wie z.B. der vollständige Name des Benutzers und persönliche Angaben (Telefonnummer, Abteilung, Gruppenzugehörigkeit u.ä.)
<b>Home</b>	das Heimatverzeichnis des Benutzers bzw. das Startverzeichnis nach dem Login
<b>Shell</b>	die "vorgeschlagene" Shell, die nach dem Login gestartet wird, bleibt dieses Feld frei, dann wird die Standardshell <code>/bin/sh</code> gestartet

Hier ein Beispiel für einen Systembenutzer:

```
uucp:x:10:14:Unix-to-Unix CoPy system:/etc/uucp:/bin/bash
```

Der Benutzer heißt `uucp`, das Paßwort ist in der Datei `/etc/shadow` gespeichert (`x`), die `UID` ist 10, die `GID` 14, als Bezeichnung trägt es den Namen "Unix-to-Unix CoPy system", das Startverzeichnis nach dem Login ist `/etc/uucp`, und die vorgeschlagene Shell ist die `bash`.

### 6.2 Die Datei /etc/shadow

Bei früheren Versionen von Linux speicherte man die die Paßwörter direkt in die `passwd`-Datei. Allerdings war es durch einen sogenannten **Wörterbuchangriff** und der Hilfe des linux-eigenen Programmes `crypt` möglich, diese Paßwörter in vielen Fällen zu entschlüsseln und auszulesen.

Deshalb hat man die Datei `/etc/shadow` eingeführt, in der die Angaben über die Paßwörter durch ein spezielles System besser geschützt waren.

Der Eintrag in diese Datei erfolgt nach einem ähnlichen Schema, wie in der Datei `/etc/passwd`:

Username : Password : DOC : MinD : MaxD : Warn : Exp : Dis : Res

<b>Username</b>	das ist der Benutzername in druckbare Zeichen, meistens in Kleinbuchstaben
<b>Password</b>	hier steht verschlüsselt das Paßwort des Benutzers; wenn hier ein * (Sternchen) steht, dann bedeutet das, das kein Passwort vorhanden bzw. eingetragen ist
<b>DOC</b>	<b>Day of last change</b> ; der Tag an dem das Paßwort zuletzt geändert wurde; Besonderheit hier: es wird ab dem 1.1.1970 gezählt
<b>MinD</b>	Minimale Anzahl der Tage, die das Paßwort gültig ist
<b>MaxD</b>	Maximale Anzahl der Tage, die das Paßwort gültig ist
<b>Warn</b>	die Anzahl der Tage vor Ablauf der Lebensdauer, ab der vor dem Verfall des Paßwortes zu warnen ist
<b>Exp</b>	hier wird festgelegt, wieviele Tage das Paßwort trotz Ablauf der MaxD noch gültig ist
<b>Dis</b>	bis zu diesem Tag (auch hier wird ab dem 1.1.1970 gezählt) ist der Benutzer-Account gesperrt
<b>Res</b>	<b>Reserve</b> , dieses Feld hat momentan keine Bedeutung

Es folgt wieder ein Beispiel:

```
selflinux:/heSIGnYDr6MI:11995:1:99999:14:::
```

Der Benutzer heißt `selflinux`, das Paßwort lautet verschlüsselt `/heSIGnYDr6MI`, es wurde zuletzt geändert, als 11995 Tage seit dem 1.1.1970 vergangen waren. Das Paßwort ist minimal 1 Tag gültig, maximal 99999 Tage (was man als "immer" deuten kann - 99999 Tage sind ca. 274 Jahre). Es soll ab 14 Tage vor Ablauf des Paßwortes gewarnt werden. Die anderen Werte sind vom Administrator nicht definiert und bleiben daher leer.

### 6.3 Die Datei /etc/groups

In dieser Datei finden Sie die Benutzergruppen und ihre Mitglieder. In der Datei `/etc/passwd` wird mit der `GID` eigentlich schon eine Standardgruppe für den Benutzer festgelegt. Hier in der `/etc/group` können Sie weitere Gruppenzugehörigkeiten definieren. Das hat in der Praxis vor allem in Netzwerken eine große Bedeutung, weil Sie so in der Lage sind, z.B. Gruppen für Projekte oder Verwaltungseinheiten zu bilden. Für diese Gruppen kann man dann entsprechend die Zugriffsrechte einstellen. Dies hat dann wiederum den Vorteil, daß man die Daten gegen eine unbefugte Benutzung absichern kann.

Der Eintrag einer Gruppe in die Datei sieht so aus:

Groupname : Password : GID : Usernames (Mitgliederliste)

<b>Groupname</b>	der Name der Gruppe in druckbare Zeichen, auch hier meistens Kleinbuchstaben
<b>Password</b>	die Besonderheit hier ist, wenn das Paßwort eingerichtet ist, können auch Nichtmitglieder der Gruppe

	Zugang zu den Daten der Gruppe erhalten, wenn ihnen das Paßwort bekannt ist. Ein <b>X</b> sagt hier aus, das das Paßwort in <code>/etc/shadow</code> abgelegt ist. Der Eintrag kann auch entfallen, dann ist die Gruppe nicht durch ein Paßwort geschützt und in die Gruppe kann aber auch ein Benutzer wechseln, der nicht in diese Gruppe eingetragen ist.
<b>GID</b>	<b>Group-Id</b> der Gruppe
<b>Usernames</b>	hier werden die Mitglieder der Gruppe eingetragen. Diese sind durch ein einfaches Komma getrennt.

Für einen korrekten Eintrag in die `/etc/group` reicht eigentlich der Groupname und die *GID* aus. Damit ist die Gruppe dem System bekannt gemacht. Die Felder für das Paßwort und die Usernames können frei bleiben.

Wenn allerdings ein Benutzer in mehr als einer Gruppe (außer in seiner Standardgruppe, welche in der `/etc/passwd` festgelegt wurde) Mitglied sein soll, so muß er in die entsprechenden Gruppen eingetragen werden. Wollen Sie, daß mehrere Mitglieder in einer Gruppe zusammenarbeiten und diese Gruppe ist nicht die Standardgruppe dieser Mitglieder, dann müssen Sie ebenfalls jedes dieser Mitglieder in die gewünschte Gruppe eintragen.

Nochmal zur besseren Veranschaulichung mit anderen Worten. Soll der Benutzer nur in seiner Standardgruppe bleiben, ist kein Eintrag in die `/etc/group` notwendig. Hier reicht der Eintrag in die `/etc/passwd` völlig aus, weil ja dort die Standardgruppe schon mit angegeben wird. Nur wenn der Benutzer in weiteren bzw. mehreren Gruppen Mitglied sein soll, dann muss das in die `etc/group`-Datei eingetragen werden.

Für Paßwörter gilt das oben in der Tabelle gesagte.

Hier sehen Sie ein Beispiel für einen Eintrag:

```
dialout:x:16:root,tatiana,steuer,selflinux
```

Sie sehen eine Gruppe mit der *GID* "16" und den Namen *dialout*. (Zur Information: *dialout* erlaubt es normalen Benutzern einen **ppp**-Verbindungsaufbau zu starten, normalerweise hat nur **root** dieses Recht). Das **x** bedeutet hier, daß das Paßwort in `/etc/shadow` abgelegt ist. Da in `/etc/shadow` hier bei Password ein **\*** steht, ist also kein Passwort für die Gruppe vorhanden (Das bedeutet wiederum, das nur die eingetragenen Mitglieder Zugang zu dieser Gruppe haben). Mitglieder der Gruppe sind: *root, tatiana, steuer, selflinux*.

### 6.4 Das Verzeichnis `/etc/skel`

Dieses Verzeichnis hat mit der Benutzerverwaltung im engeren Sinn nichts zu tun. Es soll hier aber trotzdem erwähnt werden, weil in diesem Verzeichnis haben Sie die Möglichkeit, die "Erstausrüstung" an Konfigurationsdateien, die ein neuer Benutzer erhalten soll, festzulegen. Das heißt dann, wenn Sie einen neuen Benutzer einrichten, können Sie durch einfaches Kopieren des Verzeichnisses `/etc/skel` dem neuen Nutzer eine vorgefertigte, einheitliche Umgebung bereit stellen.

In der Praxis wird von dem `/etc/skel`-Verzeichnis sehr oft Gebrauch gemacht, denn sie müssen nur einmal dieses Verzeichnis anlegen und können dann eine große Anzahl von Benutzern auf einfache Weise einrichten. Bei den meisten Linux-Distributionen wird dieses Verzeichnis schon standardmäßig angelegt und kann dann nach den eigenen Wünschen verändert werden.

### 6.5 Network Information Service (NIS)

Wenn mehrere Linux- und Unix-Systeme in einem Netzwerk auf gemeinsame Ressourcen zurückgreifen wollen, muß dann sichergestellt sein, daß die Benutzer- und Gruppenkennungen aller Rechner in diesem Netzwerk miteinander harmonisieren und es zu keinen Konflikten kommt. Das ist die Aufgabe von dem Network Information Service (NIS).

Sie können NIS als Datenbanksystem verstehen, der Zugriff auf die Dateien `/etc/passwd`, `/etc/shadow` und `/etc/group` in dem gesamten angeschlossenen Netzwerk ermöglicht.

