1.4 Anwendung/Berechtigungen

1.4.1 Berechtigungen

Beim Arbeiten mit Mehrbenutzersystemen sowie bei der Arbeit in Netzen kommt es darauf an, mit Berechtigungen für Dateien, Verzeichnisse und Geräte sicher und schnell umgehen zu können. Solche Berechtigungen unterliegen einem bestimmten Modell von den Geschehnissen am Computer (Abbildung 1). Ein Benutzer initiiert einen Prozess[1], und dieser Prozess versucht eine Resource

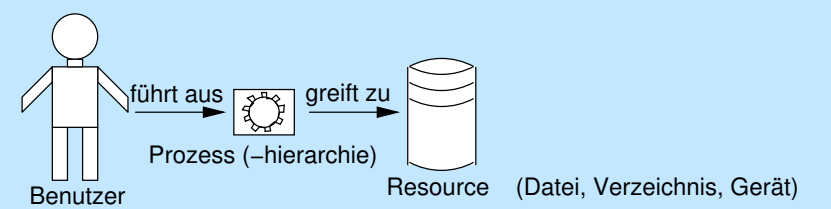

Abbildung 1: Modell für Berechtigungen

zu benutzen, zum Beispiel eine Datei zu lesen. An dieser Stelle greift das Betriebssystem ein. Entweder es erlaubt den Zugriff oder es verweigert ihn.

Damit das Betriebssystem in jedem Fall (und damit zu jeder Resource) eine Entscheidung treffen kann, wird in den meisten Betriebssystemen zu jeder Ressource gespeichert, *wer* auf die Ressource *mit welcher Aktion* zugreifen kann.

Als Beispiel soll ein System mit 100.000 Dateien angenommen werden, auf dem 100 Benutzerkennungen vorhanden sind. Auf eine Datei kann man 10 verschiedene Aktionen anwenden (Löschen, Umbenennen, Lesen, Überschreiben, Anhängen, Berechtigungen ändern, Kopieren, Verschieben, Ausführen, Verschlüsseln). Dann bedeutet das, dass für die Berechtigungen $100000 \cdot 100 \cdot 10 = 10^8$ Bits richtig gesetzt werden müssen. Ein Administrator, der jede Sekunde ein Bit setzt, braucht also bei ununterbrochener Arbeit etwa 116 Tage, um dieses System zu konfigurieren (Abbildung 2).

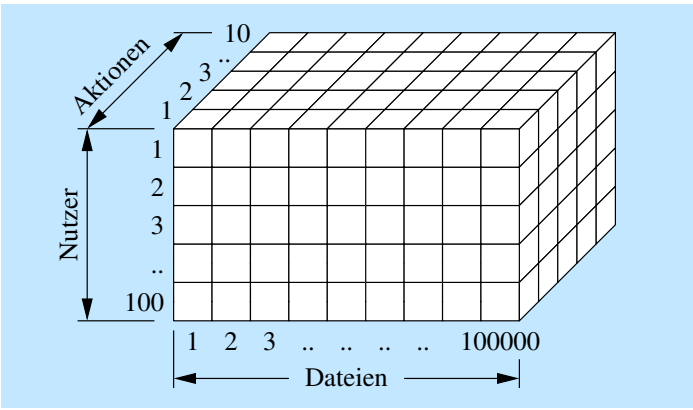

Abbildung 2: Berechtigungen allgemein

Wie kann man sich nun die Arbeit vereinfachen? Man kann

- Dateien zusammenfassen, z.B. in Verzeichnissen
- Nutzer zusammenfassen, z.B. in Gruppen
- Aktionen zusammenfassen, z.B. in leseartige, schreibartige Zugriffe und „Vollzugriff“

[1] dieser Prozess kann wiederum weitere Prozesse, eine ganze Prozesshierarchie begründen, was hier aber nicht abgebildet ist, weil es hier nicht interessiert.

- bei der Installation Standardwerte benutzen

Beim FAT-Dateisystem von Windows und DOS gibt es keine verschiedenen Benutzer (oder alle sind in einer Gruppe) und nur wenige für Berechtigungen wirksame Attribute (Abbildung 3):

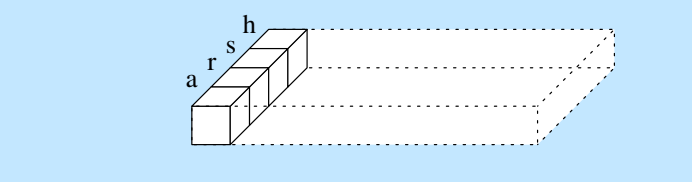

Abbildung 3: FAT-Berechtigungen

- R: *read-only*, die Datei darf nur gelesen werden
- A: *archive*, die Datei muss in einem Backup berücksichtigt werden
- S: *system*, es handelt sich um eine wichtige Datei des Betriebssystems
- H: *hidden*, die Datei soll von normalen Dienst- und Anwenderprogrammen nicht angezeigt werden

1.4.2 Linux- und UNIX-Berechtigungen

In Linux und anderen UNIX-artigen Systemen geht man anders vor (Abbildung 4): Hier werden

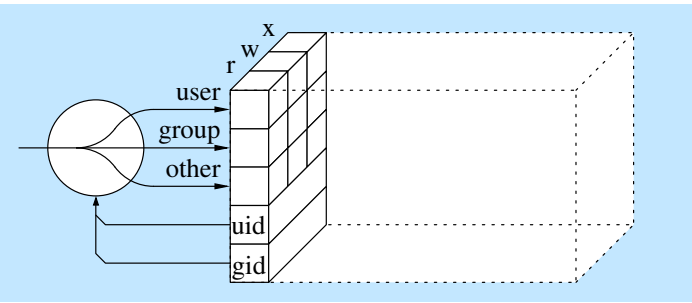

Abbildung 4: Linux-Unix-Berechtigungen

zu jeder Datei eine User-ID, eine Gruppen-ID und drei mal drei Berechtigungs-Bits gespeichert. Je nachdem, ob man zur angegebenen User-ID oder zur angegebenen Gruppen-ID oder zu keinem von beiden passt, wird man in eine von drei Benutzerklassen einsortiert. Und für jede Benutzerklasse gelten drei der Berechtigungsbits. Das sind im einzelnen:

r wie *read* (lesen)

w wie *write* (schreiben)

x wie *execute* (ausführen)

Wie hier bei der Ausgabe von `ls -l` zu sehen, werden hier zur Datei `preisliste.pdf` der Eigentümer (`willi`), die Eigentumsgruppe (`einkauf`) und neun Zeichen zu den Berechtigungs-Bits (`rwxrw-r--`) angegeben:

Terminal

```
schueler@debian964:~$ ls -l
-rwxrw-r-- 1 willi einkauf 34720 2015-02-25 13:02 preisliste.pdf
```

Die neun Zeichen zu den Berechtigungs-Bits teilen sich wie folgt auf:

a) Die drei linken Zeichen (`rwx`) gehören zur Benutzerklasse **Eigentümer** (=*user*, abgekürzt u).

b) Die drei mittleren Zeichen (`rw-`) gehören zur Benutzerklasse **Eigentumsgruppe** (=*group*, abgekürzt g).

c) Die drei rechten Zeichen (`rw-`) gehören zur Benutzerklasse **Andere** (=*other*, abgekürzt o).

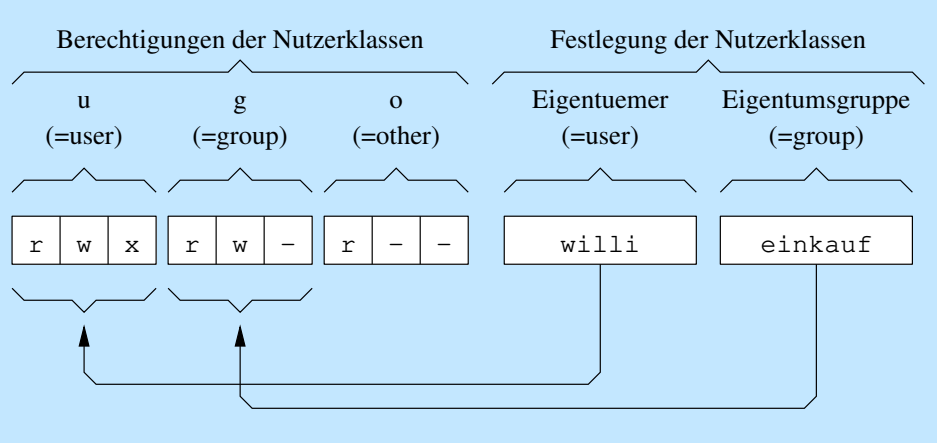

Abbildung 5: Aufteilung der neun Berechtigungs-Bits

Abbildung 5 zeigt noch einmal diese Aufteilung am Beispiel von `preislistel.pdf`.

Ein gesetztes Bit wird durch einen der Buchstaben r, w oder xgekennzeichnet, ein nicht gesetztes Bit durch ein Minuszeichen. Abbildung 6 zeigt das wiederum am Beispiel von `preislistel.pdf`.

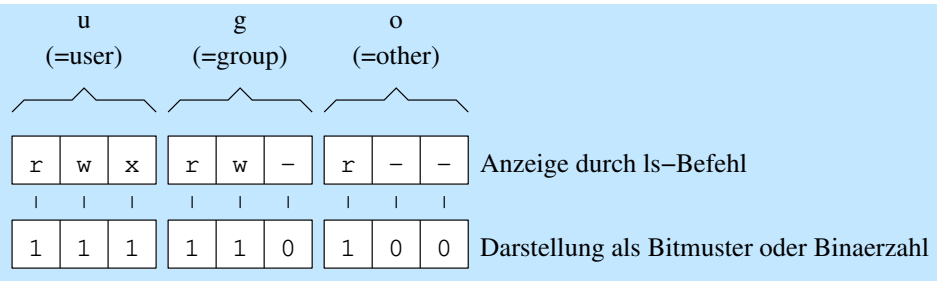

Abbildung 6: Gesetzte und nicht gesetzte Bits

Woher weiß man nun als Benutzer (oder als Prozess), zu welcher Benutzerklasse man gehört? Diese Zuordnung findet nach einem festen System statt (Abbildung 7):

a) Hat der Prozess, der den Zugriff anfordert, die User-ID null, wurde er also vom Administrator `root` gestartet, dann wird der Zugriff in jedem Fall gewährt[2].

b) Hat der Prozess dieselbe User-ID wie die Datei, dann gehört er zur Benutzerklasse Eigentümer (=user, abgekürzt als u).

[2]Es gibt eine Ausnahme: Der Benutzer `root` kann eine Programmdatei nur dann ausführen, wenn mindestens eines der x-Bits der Datei gesetzt ist. Allerdings kann er das x-Bit selbst setzen.

c) Ein Prozess kann mehrere Gruppen-IDs haben, denn man kann als Benutzer in mehreren Gruppen sein. Ist eine dieser IDs gleich der Gruppen-ID der Datei, dann gehört der Prozess zur Benutzerklasse Eigentumsgruppe (=group, abgekürzt als g)

d) Wenn auch das nicht zutrifft, dann gehört er zur Benutzerklasse Andere (=other, abgkürzt als o).

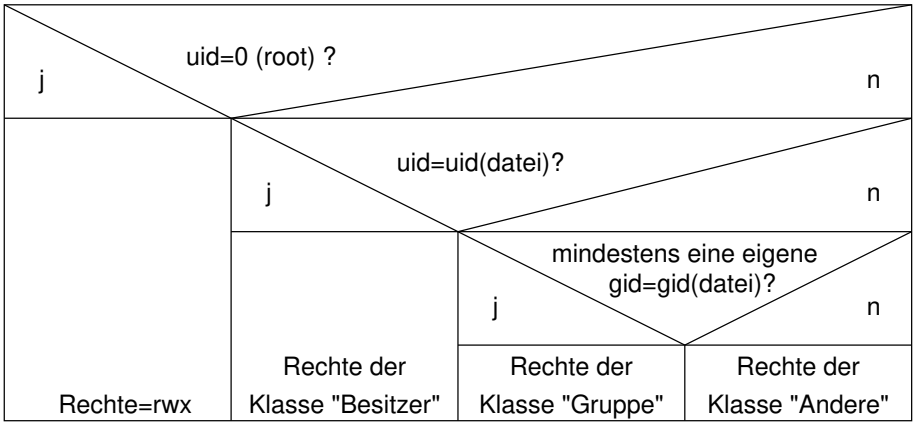

Abbildung 7: Zuordnung eines Nutzers oder Prozesses zu einer Benutzerklasse

1.4.3 Benutzer und Gruppen

Das Betriebssystem sieht die Welt so, dass es Benutzer gibt, die in Gruppen zusammengefasst werden können (Abbildung 8). Dabei ist jeder Benutzer genau einer *Primären Gruppe*[3] zugeordnet. Dies ist in der Datei `/etc/passwd` festgehalten.

Zudem kann jeder Benutzer in beliebig vielen *Sekundären Gruppen*[4] Mitglied sein. Dies kann man in der Datei `/etc/group` festlegen. Legt ein Benutzer eine Datei an, gehört sie standard-

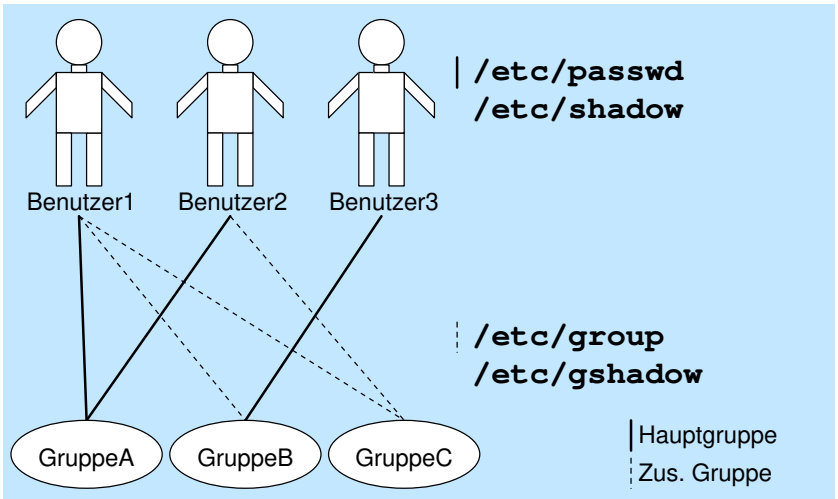

Abbildung 8: Benutzer und Gruppen in Linux und Unix

mäßig zu ihm als Eigentümer und zu seiner Primägruppe als Eigentumsgruppe.

[3] Auch Hauptgruppe, Primärgruppe oder *primary group* genannt
[4] Auch Nebengruppen, zusätzliche Gruppen oder *supplementary groups* genannt

1.4.4 Befehle für Benutzer und Gruppen

Für die Arbeit mit Benutzern und Gruppen gibt es mehrere Befehle:

a) `who`
Mit dem Befehl `who` kann man sich anzeigen lassen, welche Benutzer gerade eingeloggt sind. Gibt man `who am i` ein, erhält man seinen eigenen Loginnamen.

b) `id`
Mit dem Befehl `id` erfährt man dazu noch die eigene User-ID sowie die eigenen Group-IDs und Gruppennamen sowohl von der Primär- als auch von den Sekundärgruppen.

c) `su`
Der Befehl `su` (*substitute user* oder *switch user*) erlaubt das Ändern des aktuellen Benutzers.

d) `sg` und `newgrp`
Die ähnlichen Befehle `sg` (*substitute group*) und `newgrp` erlauben das Ändern der aktuellen Gruppe; sie werden heute kaum noch gebraucht, da jeder Benutzer gleichzeitig vielen verschiedenen Gruppen angehören kann.

e) `sudo`
Der Befehl `sudo` erlaubt es, einen Befehl unter einer anderen Benutzerkennung auszuführen; er erfordert aber möglicherweise einige Konfiguration in der Datei `/etc/sudoers`. Merke: `sudo` ist nicht `su`!

f) `useradd`, `groupadd` und `usermod`
Die Befehle `useradd`, `groupadd` und `usermod` ermöglichen es, Benutzer und Gruppen anzulegen und zu modifizieren. Sie sind meistens nur für den Administrator verfügbar.

1.4.5 Befehle für Dateien und Verzeichnisse

Die Befehle für Dateien und Verzeichnisse werden viel häufiger benötigt:

a) `chown`
Mit dem Befehl `chown` kann der Administrator `root` den Eigentümer von Objekten ändern:

Terminal
```
root@debian964:~# chown mueller_heinz neu.dat irgendwas.dat tmp
```

Andere Nutzer können `chown` nicht verwenden.

b) `chgrp`
Für das Ändern der Eigentumsgruppe gibt es den Befehl `chgrp`. Der Administrator darf dabei alles, normale Benutzer dürfen nur ihre eigenen Objekte ändern und das auch nur dann, wenn sie Mitglied der angegebenen Ziel-Gruppe sind:

Terminal
```
schueler@debian964:~$ chgrp versandabtg neu.dat irgendwas.dat tmp
```

c) `chmod`
Mit dem Befehl `chmod` darf der Eigentümer eines Objekts die neun Berechtigungsbits ändern.

1.4.6 Einzelbits setzen und löschen mit chmod

Mit dem Befehl `chmod` kann man gezielt ein einzelnes Bit eines Objektes oder mehrerer Objekte verändern:

Im folgenden Beispiel wird bei allen angegebenen Dateien und Verzeichnissen für die Benutzerklasse u (Eigentümer) das w-Bit (w=schreiben) gesetzt (Pluszeichen):

Terminal
```
schueler@debian964:~$ chmod u+w neu.dat tmp /etc
```

Wenn das w-Bit des Objekts vorher auf null war, wird es auf eins gesetzt; andernfalls bleibt es auf eins.

Und hier wird für die Benutzerklassen g (=Eigentumsgruppe) das x-Bit (Ausführen) gelöscht (Minuszeichen):

Terminal
```
schueler@debian964:~$ chmod g-x neu.dat tmp /etc
```

Wenn das x-Bit des Objekts vorher auf eins war, wird es auf null gesetzt; andernfalls bleibt es auf null.

Man kann das auch für mehrere Benutzerklassen machen, z. B. hier für die Benutzerklassen g und o. Es wird das x-Bit für g gelöscht und das x-Bit für o ebenfalls:

Terminal
```
schueler@debian964:~$ chmod go-x neu.dat tmp /etc
```

Will man ein Bit für alle Benutzerklassen setzen oder löschen, kann man das abkürzen:

Terminal
```
schueler@debian964:~$ chmod ugo-x neu.dat tmp /etc # lange Form
schueler@debian964:~$ chmod a-x neu.dat tmp /etc   # lange Form
```

Hier bedeutet das a alle Benutzerklassen.

Man kann auch mehrere Setz- oder Löschanweisungen kombinieren. Die Anweisungen werden dann durch jeweils ein Komma (kein Leerzeichen!) getrennt. In diesem Beispiel wird für u das w-Bit gesetzt. Für alle (=a) wird das r-Bit gesetzt. Und für g werden das w- und das x-Bit gelöscht:.

Terminal
```
schueler@debian964:~$ chmod u+w,g-wx,a+r neu.dat tmp /etc
```

1.4.7 Alle Rechte-Bits einer Benutzerklasse festlegen mit chmod

Mit dem Gleichheitszeichen bestimmt man die drei Bits einer

Benutzerklasse gleichzeitig. Hier werden für die Benutzerklasse u das r- und das w-Bit gesetzt und zugleich das (nicht aufgeführte!) x-Bit gelöscht:

Terminal
```
schueler@debian964:~$ chmod u=rw neu.dat tmp /etc
```

Die Regeln für das Gleichheitszeichen lauten:

- Alle genannten Bits werden gesetzt (auf eins).
- Alle nicht genannten Bits werden gelöscht (auf null).

Falls kein Bit gesetzt werden soll, wird eben keins genannt. Hier wird für die Benutzerklassen g und o kein Bit gesetzt und alle Bits gelöscht:

Terminal
```
schueler@debian964:~$ chmod go= neu.dat tmp /etc
```

1.4.8 chmod mit Oktalzahl

Wenn man alle neun Berechtigungs-Bits gleichzeitig bestimmen will, müsste man eigentlich das ganze Bitmuster auf einmal eingeben:

Terminal
```
schueler@debian964:~$ chmod u=rwx,g=rx,o=x tmp
```

Das ist jedoch etwas umständlich. Stattdessen benutzt man dazu meistens eine dreistellige Oktalzahl:

Terminal
```
schueler@debian964:~$ chmod 751 tmp
```

Was bedeutet das nun?

Die erste Ziffer gehört dabei zur Benutzerklasse u, die zweite Ziffer gehört zur Benutzerklasse g und die dritte zur Benutzerklasse o. u bekommt hier den Wert 7, g den Wert 5 und o den Wert 1.

Woher kommen nun die Ziffern 7, 5 und 1? Aus den drei Bits einer Benutzerklasse wird eine Dualzahl gebildet mit den folgenden Stellenwerten:

a) Das r-Bit hat den Wert 4.

b) Das w-Bit hat den Wert 2.

c) Das x-Bit hat den Wert 1.

Die Ziffer berechnet man dann aus der Summe der drei Werte. Tabelle 1 zeigt die (acht) möglichen Kombinationen. Abbildung 9 zeigt diese Zuordnung am Beispiel von `preisliste.pdf`. Der

Bits	Oktal	Berechnung
421		
---	0	$0\cdot 4+0\cdot 2+0\cdot 1$
--x	1	$0\cdot 4+0\cdot 2+1\cdot 1$
-w-	2	$0\cdot 4+1\cdot 2+0\cdot 1$
-wx	3	$0\cdot 4+1\cdot 2+1\cdot 1$
r--	4	$1\cdot 4+0\cdot 2+0\cdot 1$
r-x	5	$1\cdot 4+0\cdot 2+1\cdot 1$
rw-	6	$1\cdot 4+1\cdot 2+0\cdot 1$
rwx	7	$1\cdot 4+1\cdot 2+1\cdot 1$

Tabelle 1: Zuordnung Berechtigungsbits – Oktalziffer

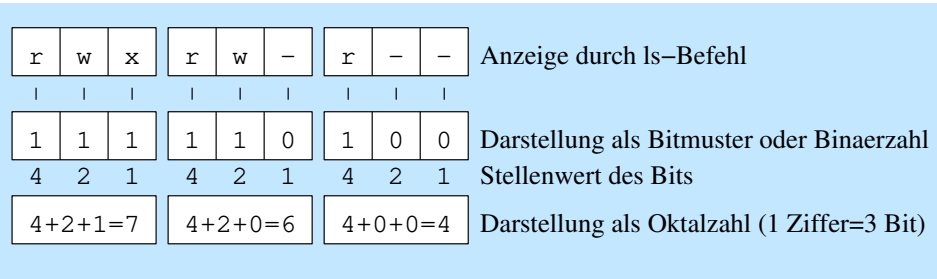

Abbildung 9: Gesetzte und nicht gesetzte Bits als Oktalziffern

Befehl `chmod 751 /tmp` bedeutet also: Die Benutzerklasse u hat die Berechtigungs-Bits rwx, die Benutzerklasse g hat die Berechtigungs-Bits rx und die Benutzerklasse o hat das Berechtigungs-Bit x:

Terminal
```
schueler@debian964:~$ chmod 751 tmp
schueler@debian964:~$ ls -l
drwxr-xr--x 2 schueler schueler 4096 .....
```

1.4.9 Bedeutung der Berechtigungs-Bits bei Dateien

Was bedeuten nun die drei Bits für den Umgang mit einer Datei?

- Ist das r-Bit gesetzt, darf eine Datei gelesen werden.
- Ist das w-Bit gesetzt, darf sie verändert, überschrieben und ergänzt werden.
- Ist das x-Bit gesetzt, darf eine Binärdatei (z.B. ein compiliertes C++-Programm) ausgeführt werden.
- Eine Scriptdatei dagegen kann nur dann ausgeführt werden, wenn sie außerdem auch lesbar ist (also r- und x-Bit gesetzt sind). Das liegt daran, dass ein Interpreter die Datei erst lesen muss, um den in ihr enthaltenen Code auszuführen.

Tabelle 2 gibt eine Übersicht.

	Bits	Erlaubte Aktionen
0	`---`	Nichts
1	`--x`	Ausführen von Binärprogrammen
2	`-w-`	Schreiben (incl. Anhängen)
3	`-wx`	Wie 2+1
4	`r--`	Lesen
5	`r-x`	Lesen und Ausführen aller Programme
6	`rw-`	Wie 4+2
7	`rwx`	Wie 5+2

Tabelle 2: Bedeutung der Berechtigungsbits bei Dateien

1.4.10 Bedeutung der Berechtigungs-Bits bei Verzeichnissen

Bei einem Verzeichnis haben die Berechtigungsbits eine Bedeutung für den Zugriff auf die Inhalte des Verzeichnisses:

- Dateien in diesem Verzeichnis
- Unterverzeichnisse und deren Inhalte

Was bedeuten nun die drei Bits bei einem Verzeichnis?

1.4.10.1 Das r-Bit bei einem Verzeichnis Ist das r-Bit gesetzt, darf man den Verzeichnisinhalt (die Liste aller Objektnamen) lesen. Der `ls`-Befehl gibt dann die Liste aller Objektnamen aus, allerdings ohne weitere Angaben:

Terminal

```
schueler@debian964:~$ mkdir neuverz
schueler@debian964:~$ touch neuverz/neu.txt
schueler@debian964:~$ echo 123 > neuverz/neu.txt # schr. 123 i.d.Datei
schueler@debian964:~$ chmod 444 neuverz
schueler@debian964:~$ ls -l neuverz
ls: Zugriff auf 'neuverz/neu.txt' nicht möglich: Keine Berechtigung
insgesamt 0
-????????? ? ? ? ?            ? neu.txt
schueler@debian964:~$ cat neuverz/neu.txt
cat: neuverz/neu.txt: Keine Berechtigung
```

Man bekommt also nur zu sehen, dass dort eine Datei `neu.txt` liegt. Man bekommt aber keine Eigenschaften zu sehen und man kann die Datei auch nicht lesen.

1.4.10.2 Das x-Bit bei einem Verzeichnis Ist das x-Bit gesetzt, darf man auf die Elemente dieses Verzeichnisses zugreifen:

- Man darf Dateien lesen und schreiben, falls deren Datei-Berechtigungen das erlauben
- Man darf Unterverzeichnisse durchsuchen, falls deren Verzeichnis-Berechtigungen das erlauben.
- Man darf in das Verzeichnis wechseln.

Terminal

```
schueler@debian964:~$ chmod 111 neuverz
schueler@debian964:~$ ls -l neuverz
ls: Öffnen von Verzeichnis 'neuverz/' nicht möglich: Keine
                                                  Berechtigung
schueler@debian964:~$ ls -l neuverz/neu.txt
-rw-r--r-- 1 schueler schueler 4 2. Mai 21:50 neuverz/neu.txt
schueler@debian964:~$ cat neuverz/neu.txt
123
schueler@debian964:~$ echo Hallo > neuverz/neu.txt#schr.Hallo i.d.Dat.
schueler@debian964:~$ ls -l neuverz/neu.txt
-rw-r--r-- 1 schueler schueler 6 2. Mai 21:52 neuverz/neu.txt
```

Das x-Bit wirkt also anders als das r-Bit: Man kann das Verzeichnis eben nicht einfach auflisten. Wenn man aber weiß, wie die Datei heißt, kann man sich alle ihre Eigenschaften anzeigen lassen. Man kann sie lesen und man kann sie sogar schreiben (wenn ihre Datei-Berechtigungen das zulassen). Man kann sie aber nicht löschen, verschieben oder umbenennen:

Terminal

```
schueler@debian964:~$ mv neuverz/neu.txt neuverz/ganzneu.txt
mv: das Verschieben von 'neuverz/neu.txt' nach 'neuverz/ganzneu.txt'
    ist nicht möglich: Keine Berechtigung
```

Ebenso kann man auch keine neue Datei anlegen:

Terminal

```
schueler@debian964:~$ touch neuverz/neu.py
touch: 'neuverz/neu.py' kann nicht berührt werden: Keine Berechtigung
```

1.4.10.3 Das w-Bit bei einem Verzeichnis Ist das w-Bit allein gesetzt, kann man damit noch nichts machen. Wenn aber das w-Bit und das x-Bit *beide* gesetzt sind, dann darf man den Verzeichnisinhalt (=die Liste aller Dateinamen) verändern: Nun darf man im Verzeichnis neue Einträge für Dateien oder Unterverzeichnisse anlegen und löschen:

Terminal

```
schueler@debian964:~$ chmod 333 neuverz
schueler@debian964:~$ mv neuverz/neu.txt neuverz/ganzneu.txt
schueler@debian964:~$ cp neuverz/ganzneu.txt neuverz/alt.txt
schueler@debian964:~$ rm neuverz/ganzneu.txt
schueler@debian964:~$ ls -l neuverz/alt.txt
-rw-r--r-- 1 schueler schueler 6  2. Mai 21:54 neuverz/alt.txt
```

Warum reicht das w-Bit alleine nicht zum Anlegen und Löschen von Objekten im Verzeichnis aus? Beim Anlegen müsste ein I-Node erstellt werden. Für den Zugriff auf den I-Node zu einem Verzeichniseintrag brauchte man aber das x-Bit. Ebenso beim Löschen: Der Link-Counter im I-Node, der zum Eintrag gehört, müsste dekrementiert werden; dazu brauchte man aber schon wieder das x-Bit.

	Bits	Erlaubte Aktionen
0	`---`	Nichts
1	`--x`	Wechseln ins Verzeichnis, Benutzen der Elemente
2	`-w-`	Wie 0
3	`-wx`	Wie 1, dazu das Anlegen, Löschen und Umbenennen der Elemente
4	`r--`	Auflisten der Elementnamen
5	`r-x`	Wie 4+1
6	`rw-`	Wie 4
7	`rwx`	Wie 4+3

Tabelle 3: Bedeutung der Berechtigungsbits bei Verzeichnissen

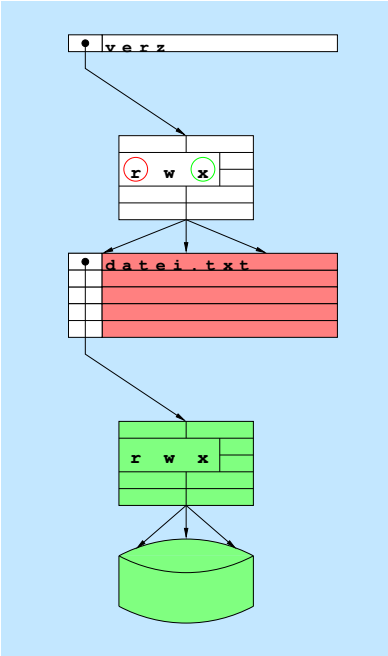

Abbildung 10: x- und r-Bit bei Verzeichnissen

1.4.10.4 Übersicht Tabelle 3 führt zusammen, welche Aktionen man mit welcher Verzeichnis-Berechtigung ausführen kann. Das r-Bit eines Verzeichnisses ermöglicht also nur das Auflisten (das Ansehen aller Elementnamen), während mit dem x-Bit der Zugriff auf die I-Nodes erlaubt wird – und damit der Zugriff auf die Attribute und Inhalte der Elemente (Abbildung 10).

1.4.11 SUID-Bit für ausführbare Dateien

Wie kann man eigentlich mit dem Programm `passwd` sein eigenes Passwort ändern, wenn man die Datei `/etc/shadow`, in der das Passwort steht, nicht lesen und schon gar nicht schreiben kann?

Terminal
```
schueler@debian964:~$ ls -l /etc/shadow
-rw-r----- 1 root shadow 2132 Feb 25 16:23 /etc/shadow
```

Das Geheimnis liegt in den Berechtigungen des Programms selbst.

Terminal
```
schueler@debian964:~$ ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 34567 Feb 27 2014 /usr/bin/passwd
```

Das Programm `/usr/bin/passwd` hat besondere Rechte, oktal ausgedrückt 04755. Das so genannte SUID-Bit (oktal 4000 bzw. u+s) bewirkt, dass das Programm `passwd` mit der UID des *Programmeigentümers* `root` ausgeführt wird und nicht wie sonst mit meiner UID.

Bei eigenen, eventuell nicht vollständig ausgetesteten Programmen ist das SUID-Bit gefährlich, ebenso bei Scripten! Deshalb wird es bei Linux für Scripte ignoriert.

1.4.12 SGID-Bit für Dateien

Weiterhin gibt es das SGID-Bit für ausführbare Dateien. Es hat den oktalen Wert 2000 (bzw. g+s) und zeigt sich im Verzeichnis-Listing so:

Terminal
```
schueler@debian964:~$ ls -l /bin/X
-rwxr-sr-x 1 root root 8364 Okt 24  2011 X
```

Wenn dieses Bit gesetzt ist, läuft das Programm mit der Group-ID der Gruppe *der ausführbaren Datei*. Das SGID-Bit wird eher selten benutzt, es stellt eine ähnliche Gefahr wie das SUID-Bit dar.

Bei nicht ausführbaren Dateien dient das SGID-Bit dem Datei-Locking.

Terminal
```
schueler@debian964:~$ chmod 2644 neu.txt
schueler@debian964:~$ ls -l neu.txt
-rw-r-Sr-- 1 heinz versand 0 Feb 25 18:26 neu.txt
```

Man erreicht damit exklusive Schreib- und Lese-Operationen. Während der Bearbeitung durch einen Prozess ist die Datei für andere Prozesse gesperrt.

1.4.13 SUID-, SGID- und Sticky-Bit für Verzeichnisse

Wenn bei einem Verzeichnis das SUID-Bit gesetzt ist, gehören alle neu angelegten Dateien in diesem Verzeichnis dem Verzeichniseigentümer. Dieses Bit ist bei Linux und einigen anderen Unix-artigen Systemen nicht wirksam[5], bei Free-BSD dagegen schon.

Wenn bei einem Verzeichnis das SGID-Bit gesetzt ist, werden neu angelegte Dateien in diesem Verzeichnis automatisch der gleichen Gruppe wie das Verzeichnis selbst zugeordnet. Ein normaler Nutzer kann dieses Bit kann nur für ein Verzeichnis setzen, das zu einer Gruppe gehört, bei der er selbst Mitglied ist. Nutzen kann das Bit jeder. Dieses Bit ist bei Linux wirksam, wird aber selten benutzt (am ehesten noch bei gemeinsamen Dateien für Gruppen).

Das Sticky-Bit hat den oktalen Wert 1000. Wenn bei einem Verzeichnis das Sticky-Bit gesetzt ist, dann können Dateien in diesem Verzeichnis nur von ihrem Eigentümer und vom Verzeichniseigentümer gelöscht oder umbenannt werden. Das wichtigste Beispiel dafür ist wohl das `/tmp`-Verzeichnis:

Terminal
```
schueler@debian964:~$ ls -ld /tmp
drwxrwxrwt 15 root root 4096 Nov 10 16:46 /tmp
```

1.4.14 Standard-Berechtigungen für neu angelegte Objekte mit umask

Wenn ein Benutzer ein Dateisystemobjekt (also eine Datei, ein Verzeichnis, einen Link usw.) anlegt, gehört ihm dieses Objekt selbst. Der Administrator (root) könnte das nachträglich ändern, aber erst einmal ist die Eigentümer-UID des Objekts gleich der UID desjenigen, der das Objekt erstellt:

Terminal
```
schueler@debian964:~$ whoami
schueler
schueler@debian964:~$ rm a.txt; touch a.txt
```

[5] weil Eigentumsrechte an Dateien nur durch root geändert werden dürfen

```
schueler@debian964:~$ ls -l a.txt
-rw-r--r-- 1 schueler schueler 0 Feb 24 10:00 a.txt
```

Ebenso ist es mit der Gruppe: Das Objekt gehört zur gleichen Eigentumsgruppe wie derjenige, der das Objekt erstellt hat. Und zwar wird die Hauptgruppe genommen, die Hilfsgruppen spielen hier keine Rolle.

Eigentümer und Eigentumsgruppe sind also standardmäßig richtig eingerichtet. Wie aber sieht es mit den neun Berechtigungsbits (Modus) aus? Dazu hat jeder Prozess nicht nur eine UID und eine GID, sondern auch eine Datei-Erstellungsmaske, die so genannte Umask. Die Umask ist eine Zahl zwischen 0 und (oktal) 0777, und sie gibt an, welche Berechtigung eine neu angelegte Datei oder ein neu angelegtes Verzeichnis haben. Standardmäßig liegt ihr Wert auf (oktal) 022. Mit der eingebauten Shell-Funktion `umask` kann man den aktuellen Wert anzeigen:

Terminal
```
schueler@debian964:~$ umask
0022
```

Wenn man der eingebauten Shellfunktion `umask` als Parameter eine Oktalzahl zwischen 0000 und 0777 angibt, ändert man damit die Umask:

Terminal
```
schueler@debian964:~$ umask 640
schueler@debian964:~$ umask
0640
```

Wie legt man aber damit die Berechtigungen fest? Sind die Berechtigungen einer neu angelegten Datei gleich der Umask?

Terminal
```
schueler@debian964:~$ rm c.txt; touch c.txt
schueler@debian964:~$ ls -l c.txt
-----w-rw- 1 schueler schueler 0 Feb 24 10:03 c.txt
```

Offenbar ist das Gegenteil der Fall: Die Bits, die in der Umask *gesetzt* sind, werden in der angelegten Datei *gelöscht*. Mathematisch ausgedrückt:

$$\text{Modus} = 0666 \text{ AND NOT Umask}$$

Merke: **Hohe Umask – niedrige Berechtigungen**.

Terminal
```
schueler@debian964:~$ umask 027
schueler@debian964:~$ rm z.py; touch z.py
schueler@debian964:~$ ls -l z.py
-rw-r----- 1 schueler schueler 0 Feb 24 10:06 z.py # 0640=0666&~0027
```

Bei Verzeichnissen ist es einfacher: Hier wird standardmäßig das x-Bit gesetzt. Daher ist die mathematische Formel:

$$\text{Modus} = 0777 - \text{Umask}$$

Terminal
```
schueler@debian964:~$ umask 027
schueler@debian964:~$ rmdir z; mkdir z
schueler@debian964:~$ ls -ld z
drwxr-x--- 2 schueler schueler 4096 Feb 24 10:07 z # 0750=0777-0027
```