

# Linux – Prinzipien und Programmierung

Dr. Klaus Höppner

Hochschule Darmstadt – Sommersemester 2014

Benutzerkonzept

Passende Shell-Befehle

Dateisystem-Shellbefehle

bash

# Benutzerkonzept unter Linux

Linux ist ein *Multiuser*-Betriebssystem.

Hierbei wird unterschieden zwischen:

**root** Account des Systemadministrators, volle Rechte,

**Systembenutzer** Nutzer, unter deren User-Id Systemdienste gestartet werden, denen aber das Öffnen einer interaktiven Sitzung (Shell) nicht erlaubt ist,

**Normale Benutzer** Ein Benutzer, der sich normal einloggen und eine Shell öffnen darf.

## Nutzer und Gruppen

Linux kennt neben Nutzern auch Gruppen, wobei jeder Nutzer und jede Gruppe eindeutig durch eine Integer-Zahl identifiziert wird:

- User-Id (uid)  
Besonderheit: root hat die uid 0
- Group-Id (gid)  
Besonderheit: root ist in der gleichnamigen Gruppe mit der gid 0.
- Effective User-Id (euid)
- Effective Group-Id (egid)

Jeder Nutzer ist mindestens einer Gruppe (Primärgruppe) zugeordnet, kann aber noch weiteren Gruppen angehören (durch Eintrag in der Datei `/etc/group`).

## Die Datei `/etc/passwd`

Die auf dem System bekannten (lokalen) User sind in der Datei `/etc/passwd` aufgelistet.

Hier finden sich ursprünglich folgende Informationen:

- Username,
- User-Id und Group-Id (der Primärgruppe),
- optional der vollständige Name des Users und weitere Infos,
- Home-Verzeichnis,
- Login-Shell,
- früher das verschlüsselte Passwort (heute i. A. ausgelagert nach `/etc/shadow`).

Beispiel:

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

# Anlegen, Ändern, Löschen von Benutzern

Linux-Distributionen bieten meist eine eigene grafische Oberfläche zum Anlegen, Ändern bzw. Löschen von Accounts.

Zusätzlich stehen in der Regel drei Shell-Programme für diese Zwecke zur Verfügung:

- `useradd`,
- `usermod`,
- `userdel`

Beim Anlegen eines Users werden auf Wunsch die Dateien aus dem Verzeichnis `/etc/skel` in das neue Home-Verzeichnis kopiert.

Passwort-Änderungen erfolgen über den Befehl `passwd`.

## Zugriffsarten auf eine Datei

Linux kennt drei Zugriffsarten, bezeichnet mit den Buchstaben `r`, `w`, `x`:

`read` Datei darf gelesen werden,

`write` Datei darf überschrieben bzw. gelöscht werden,

`execute` Datei ist ausführbar, bzw. es darf in das Verzeichnis gewechselt werden.

Dabei ist jede Datei bzw. jedes Verzeichnis einem User (Owner) und einer Group zugeordnet.

# Die Befehle `chown` und `chgrp`

Mit den Befehlen

- `chown`
- `chgrp`

kann der Besitzer bzw. die Gruppe einer Datei oder eines Verzeichnisses geändert werden.

Frage: Warum darf nur `root` den Befehl `chown` aufrufen und nicht der Besitzer der Datei?



## Der Befehl `chmod`

Zugriffsrechte auf eine Datei/Verzeichnis können mit dem Befehl `chmod` geändert werden, und zwar bezogen auf den besitzenden User (u), die Gruppe (g), andere Nutzer (o) oder alle (a).

Die allgemeine Form zum Aufruf von `chmod` lautet dann:

```
chmod [ugoa][+~]rwxXst filename
```

Alternativ kann stattdessen auch

```
chmod number filename
```

benutzt werden, wobei `number` eine drei- oder vierstellige Oktalzahl ist. Die letzten drei Ziffern stehen dabei für die Zugriffsrechte für User, Group und Others.

## SetUID/SetGID-Bit und Sticky-Bit

Welche Bedeutung haben bei `chmod` die Buchstaben `st` bei den Zugriffsrechten (bzw. die erste Ziffer bei einer vierstelligen Oktalzahl)?

Beim Ausführen eines Programms wird dieses effektiv mit der User- und Group-Id des Aufrufers ausgeführt. Manchmal benötigen Programme jedoch erweiterte Rechte (z. B. das Programm `passwd`, das die Systemdatei mit den Passwörtern ändert, aber jeder User ausführen kann). Zu diesem Zweck kann mit dem SetUID- bzw. SetGID-Bit festgelegt werden, dass die effektive User- bzw. Group-ID beim Ausführen derjenigen des Owners bzw. der besitzenden Gruppe entspricht.

Das Sticky-Bit ist heute nur noch bei Verzeichnissen sinnvoll. Darf normalerweise eine Datei von jedem gelöscht werden, der in dem Verzeichnis schreiben darf, wird dies durch das Sticky-Bit für das betreffende Verzeichnis auf die jeweiligen Besitzer eingeschränkt.

## Passende C-API

```
#include <unistd.h>  
#include <sys/types.h>
```

```
uid_t getuid();  
uid_t geteuid();
```

```
gid_t getgid();  
gid_t getegid();
```

Unterscheide: User- und Group-Id von demjenigen, der den Prozess ausführt, und *effektive* User- und Group-Id, die auf Grund des SetUID- bzw. SetGID-Bits unterschiedlich sein kann.

## Anzeigen von Dateien in Verzeichnissen

Der Inhalt eines Verzeichnisses wird mit `ls verzeichnisname` aufgelistet. Wird das Verzeichnis weggelassen, wird das aktuelle genommen.

Dabei gibt es folgende wichtige Optionen:

- a Alle Dateien, auch solche die mit einem Punkt beginnen,
- l Ausführliche Liste mit Dateiinformationen,
- R Rekursiv alle Unterverzeichnisse anzeigen

# Dateieigenschaften

Die Anzeige von `ls -l` sieht etwa so aus:

```
user@linux ~/temp$ ls -al
total 8
drwxr-xr-x  2 hede      users      4096 Jul 10 07:25 .
drwxr-xr-x  8 hede      users      4096 Jul 10 07:25 ..
-rw-r--r--  1 hede      users              0 Jul 10 07:25 test.txt
```

Angezeigt werden Owner/Gruppe, Änderungsdatum, Zugriffsrechte, sowie ein *Linkzähler*

⇒ Was sind Hard Links bzw. Soft Links?

## „Punkt“-Dateien

Eine besondere Rolle spielen Dateien, die mit einem Punkt beginnen.

Im Gegensatz zu Windows kennt Linux keine versteckten Dateien, aber bei dem Befehl `ls` werden Punkt-Dateien normalerweise nicht angezeigt. Diese enthalten i. A. Konfigurationseinstellungen.

Beispiele:

- `.profile`
- `.bashrc`
- `.vimrc`

## Sonstige Shell-Befehle

- `id` Anzeigen von UID, GID
- `groups` Welchen Gruppen gehöre ich an?
- `w, who` Wer ist angemeldet?
  - `ps` Laufende Prozesse (je nach Optionen für aktuellen User oder alle)

## Sonstige Shell-Befehle (Forts.)

- `ln` Link erzeugen (hard oder soft, `ln -s`),
- `cp, mv, rm` Datei kopieren, verschieben bzw. löschen,
- `mkdir, rmdir` Verzeichnis anlegen oder löschen,
- `cat` Inhalt der Datei ausgeben,
- `more, less` Inhalt seitenweise anzeigen (bei letzterem mit mehr Komfort),
- `file` Informationen über die Dateiarart anzeigen,
- `touch` Änderungsdatum der Datei auf aktuelle Zeit setzen (falls Datei noch nicht existiert, wird eine leere der Größe 0 angelegt).



## Zeitstempel von Dateien

Linux unterscheidet zwischen drei verschiedenen Zeitstempeln, die einer Datei zugeordnet sind:

**Zugriffszeit** (access time, atime) Zeitpunkt des letzten Zugriffs,

**Änderungszeit** (modification time, mtime) Zeitpunkt der letzten Modifikation der Datei,

**Inode-Änderungszeit** (change time, ctime) Zeitpunkt der letzten Änderungen der Metadaten im Inode, z. B. Änderung der Zugriffsrechte.

## Passende C-API

```
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
```

```
int stat(const char *path, struct stat *buf);
```

mit

```
struct stat {
    dev_t    st_dev;        /* ID of device containing file */
    ino_t    st_ino;       /* inode number */
    mode_t   st_mode;      /* protection */
    nlink_t  st_nlink;     /* number of hard links */
    uid_t    st_uid;       /* user ID of owner */
    gid_t    st_gid;       /* group ID of owner */
    dev_t    st_rdev;      /* device ID (if special file) */
    off_t    st_size;      /* total size, in bytes */
    blksize_t st_blksize;  /* blocksize for filesystem I/O */
    blkcnt_t st_blocks;    /* number of blocks allocated */
    time_t   st_atime;     /* time of last access */
    time_t   st_mtime;     /* time of last modification */
    time_t   st_ctime;     /* time of last status change */
};
```

## Wechseln von Verzeichnissen

**cd** Wechsel in das angegebene Verzeichnis (falls keins angegeben: `$HOME`). Hierbei kann in der bash das Homeverzeichnis als `~` abgekürzt werden, so bezeichnet z. B. `~/Documents` ein Unterverzeichnis `Documents` im Home-Verzeichnis.

**pwd** Ausgabe des aktuellen Verzeichnisses.

**pushd, popd** Verzeichniswechsel mit „Merken“ der Verzeichnisse auf einem Verzeichnis-Stack (mit **pushd** wechseln und neuen Stackeintrag anlegen, mit **popd** zurück und letzten Stackeintrag entfernen).

# Finden

- grep** Suche nach Zeichenketten innerhalb einer oder mehrerer Dateien.
- find** Rekursive Suche nach Dateien anhand des Namens, Datums etc.
- locate** Globale Suche nach einer Datei anhand ihres Namens (innerhalb einer Datenbank-Datei, die regelmäßig per **updatedb** aktualisiert werden sollte).
- which** Angabe des Pfades zu einem ausführbaren Programm, das innerhalb von **\$PATH** liegt.

# Vorteile der bash als Standard-Shell

- Großer Funktionsumfang
- Command Completion mit der TAB-Taste
- History-Funktionen:
  - Blättern
  - Suchen
  - Wiederholen alter Kommandos

## Ausführen von Kommandos

In der Bash können folgende Befehle eingegeben werden:

- Interner Bash-Befehl (z. B. `pushd`),
- Programm, das innerhalb der in `$PATH` enthaltenen Verzeichnisse gefunden wird,
- Programm mit (absoluter oder relativer) Pfadangabe.

Kommandos können kombiniert werden:

`cmd1; cmd2` `cmd1` und `cmd2` werden hintereinander ausgeführt,

`cmd1 && cmd2` `cmd2` wird dann ausgeführt, wenn `cmd1` erfolgreich war,

`cmd1 || cmd2` `cmd2` wird dann ausgeführt, wenn `cmd1` nicht erfolgreich war.

# Beispiele

```
test -d ~/foo && rm -rf ~/foo
```

Falls das Verzeichnis `~/foo` existiert, wird es (rekursiv) gelöscht.

```
test -d ~/foo || mkdir ~/foo
```

Das Verzeichnis `~/foo` wird angelegt, falls es noch nicht existiert.

Alternative zu `test -d ~/foo`:

```
[ -d ~/foo ]
```

## Befehle im Hintergrund

Mit einem einzelnen `&` werden Befehle im Hintergrund ausgeführt, z. B.

`cmd1 &`

Das heißt, man kann direkt neue Befehle in der Shell ausführen, auch wenn `cmd1` noch nicht beendet ist.

Kommandos im Zusammenhang mit Hintergrundprozessen:

- `jobs` Befehle im Hintergrund auflisten,
- `fg %nr` Hintergrundprozess mit angegebener Job-Nr. in den Vordergrund holen,
- `Ctrl-Z` Aktuellen Vordergrundprozess schlafen legen (der arbeitet dann nicht mehr!),
- `bg` Schlafenden Prozess im Hintergrund weiter ausführen (so dass er wieder weiterarbeitet).



**X**