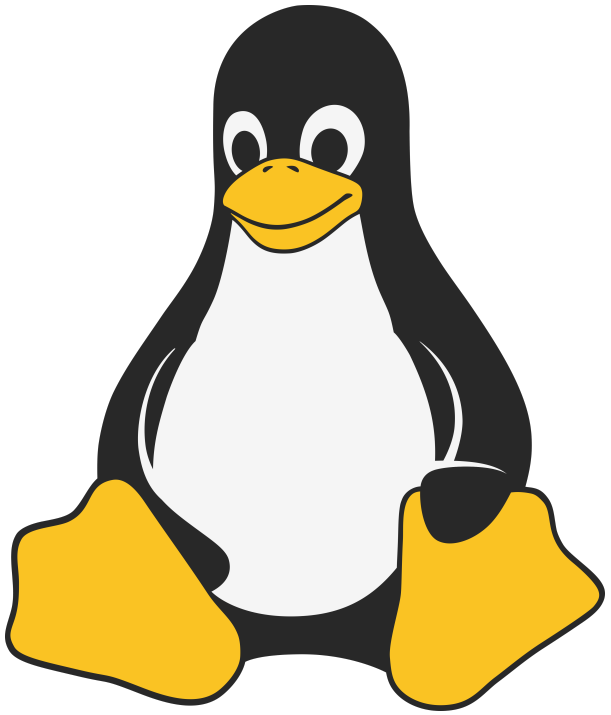
Shell-Praxis (Teil 12)



Prozesse und Kindprozesse

Inhaltsverzeichnis

Was sind Prozesse?

Ein Prozess ist ein laufendes Programm (ein CLI- oder ein GUI-Programm oder ein Dienst), das im Arbeitsspeicher des Computers geladen ist und vom Betriebssystem verwaltet wird.

Dies bedeutet nicht, dass ein Prozess immer aktiv ist. Ein Prozess kann z.B. auch schlafen (bis ihm vom Betriebssystem wieder Rechenzeit zugeteilt wird),
oder warten, bis ein Ereignis eintritt (z. B. bis Daten von der Festplatte oder aus dem Netzwerk gelesen wurden).

Zum Inhaltsverzeichnis ...

Prozess-Hierarchie

Jeder Prozess hat einen Elternprozess. Der Elternprozess ist der Prozess, der den Prozess erzeugt hat. Ein Prozess kann wiederum Kindprozesse erzeugen.

Z.B. wird bei jedem (externen) Kommando, das in einer Shell ausgeführt wird, ein neuer Prozess (Kindprozess) erzeugt. Der Prozess, der das Kommando ausgeführt hat, ist der Elternprozess. Rufen Sie z. B. das Kommando `ls -l` in einer Shell auf, dann ist der `bash`-Prozess der Elternprozess des `ls`-Prozesses. Oder umgekehrt: Der `ls`-Prozess ist ein Kindprozess des `bash`-Prozesses.

Zum Inhaltsverzeichnis ...

So entsteht eine Prozess-Hierarchie, die bis zum `init`-Prozess mit der Prozess-ID 1 zurückverfolgt werden kann.

Der `init`-Prozess ist der erste Prozess, der vom Linux-Kernel gestartet wird und ist der Vorfahre aller Prozesse. Bei diesem ist die PPID (Parent Process ID) 0 verzeichnet. D.h. der `init`-Prozess hat keine Elternprozess.

`ps` -Optionen (Auswahl)

- `ps` **ohne Option**: zeigt nur die Prozesse der aktuellen Terminal-Sitzung an in einem kompakten Format (Spalten: PID, TTY, TIME, CMD)

Steuerung des Ausgabeformats:

- `-f`: erweitert das Ausgabeformat um weitere Attribute (Spalten: UID, PID, PPID, C, STIME, TTY, TIME, CMD).

Prozessattribute - Kürzel der Spaltenüberschriften:

- `UID`: Benutzer-ID
- `PID`: Prozess-ID (systemweit eindeutige Prozessnummer)
- `PPID`: (parent process ID) Elternprozess-ID
- `C`: CPU-Auslastung in Prozent (ignorieren wir hier)
- `STIME`: Startzeit
- `TTY`: Gerätedatei des Terminals (`?` für Prozesse ohne Terminal)
- `TIME`: verbrauchte CPU-Zeit
- `CMD`: Kommandoaufruf mit Argumenten

Zum Inhaltsverzeichnis ...

Steuerung des Filters (Auswahl der anzuzeigenden Prozesse):

- `-e`: zeigt alle Prozesse des Systems an
- `-p <pid,...>`: zeigt nur den Prozess mit den angegebenen Prozess-IDs an
- `-t <tty,...>`: zeigt nur die Prozesse an, die an die angegebenen Terminals gebunden sind
- `-u <user,...>`: zeigt nur die Prozesse der angegebenen Benutzer an. Zur Auswahl werden die effektiven Benutzer-IDs (EUID) verwendet.

Prozesse anzeigen mit `ps`

```
hermann@debian:~$ ps # processes of the current tty in short format
    PID TTY          TIME CMD
   1447 pts/1    00:00:00 bash
   1517 pts/1    00:00:00 ps
```

```
hermann@debian:~$ ps -f # processes of the current tty in full format
UID          PID    PPID  C STIME TTY          TIME CMD
hermann     1447    1446  0 01:01 pts/1    00:00:00 -bash
hermann     1518    1447 99 01:23 pts/1    00:00:00 ps -f
```

```
hermann@debian:~$ ps -fe # all processes of the system in full format
UID          PID    PPID  C STIME TTY          TIME CMD
root           1       0  0 00:02 ?        00:00:01 /sbin/init
root           2       0  0 00:02 ?        00:00:00 [kthreadd]
root           3       2  0 00:02 ?        00:00:00 [rcu_gp]
root           4       2  0 00:02 ?        00:00:00 [rcu_par_gp]
root           5       2  0 00:02 ?        00:00:00 [slub_flushwq]
root           6       2  0 00:02 ?        00:00:00 [netns]
root           8       2  0 00:02 ?        00:00:00 [kworker/0:0H-events_highpri]
...
hermann     1447    1446  0 01:01 pts/1    00:00:00 -bash
hermann     1520    1447  0 01:31 pts/1    00:00:00 ps -fe
```

Prozess-Hierarchie mit `ps`

```
hermann@debian:~$ ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
hermann     1013     946  0 00:02 pts/0    00:00:00 bash
hermann     1607    1013  0 01:39 pts/0    00:00:00 ps -f
```

```
hermann@debian:~$ ps -f -p 946
UID          PID    PPID  C STIME TTY          TIME CMD
hermann      946       1  0 00:02 ?        00:00:00 lxterminal
```

```
hermann@debian:~$ ps -f -p 1
UID          PID    PPID  C STIME TTY          TIME CMD
root           1       0  0 00:02 ?        00:00:01 /sbin/init
```

TTYs und Pseudo-TTYs

Ein TTY (Teletype Writer) ist ein Terminal, das früher aus einem physischen Bildschirm und einer physischen Tastatur bestand. Sie sind unter Linux als Gerätedateien im Verzeichnis `/dev` zu finden: `/dev/tty1`, `/dev/tty2` etc.

Heute sind TTYs meist virtuelle Terminals, die im Bildschirmfenster eines Terminal-Emulators (z.B. LXTerminal) dargestellt werden. Auch sie sind als Gerätedateien im Verzeichnis `/dev` zu finden: `/dev/pts/0`, `/dev/pts/1` etc. Auch bei einer SSH-Sitzung zu einem entfernten Rechner wird die Shell an ein Pseudo-TTY gebunden.

Alle interaktiven Shell-Sitzungen sind an ein TTY bzw. Pseudo-TTY gebunden. Ein Prozess, der an ein TTY gebunden ist, kann auch mit dem TTY kommunizieren.

Die Shell und ihre Kindprozesse lesen (wenn die Standardeingabe nicht umgeleitet wurde) von der Tastatur. Die Standardeingabe (STDIN) ist mit dem TTY verbunden.

Ebenso schreiben die Shell und ihre Kindprozesse (wenn die Standardausgabe und der Standardfehlerausgabe nicht umgeleitet wurden) auf den Bildschirm. Die Standardausgabe (STDOUT) und der Standardfehlerausgabe (STDERR) sind mit dem TTY verbunden.

Prozesse an verschiedenen TTYs

```
hermann@debian:~$ # session on TTY /dev/pts/0
hermann@debian:~$ tty
/dev/pts/0
hermann@debian:~$ ls -l $(tty)
crw--w---- 1 hermann tty 136, 0 29. Nov 02:12 /dev/pts/0
hermann@debian:~$ ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
hermann     1013     946  0 00:02 pts/0    00:00:00 bash
hermann     5584    1013  0 02:13 pts/0    00:00:00 ps -f
hermann@debian:~$ ps -f -t pts/1
UID          PID    PPID  C STIME TTY          TIME CMD
hermann     1447    1446  0 01:01 pts/1    00:00:00 -bash
```

```
hermann@debian:~$ # session on TTY /dev/pts/1
hermann@debian:~$ tty
/dev/pts/1
hermann@debian:~$ ls -l $(tty)
crw--w---- 1 hermann tty 136, 1 29. Nov 02:11 /dev/pts/1
hermann@debian:~$ ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
hermann     1447    1446  0 01:01 pts/1    00:00:00 -bash
hermann     5438    1447  0 02:11 pts/1    00:00:00 ps -f
hermann@debian:~$ ps -f -t pts/0
UID          PID    PPID  C STIME TTY          TIME CMD
hermann     1013     946  0 00:02 pts/0    00:00:00 bash
```

Prozesse eines Benutzers

```
hermann@debian:~$ ps -f -u hermann
UID          PID    PPID  C STIME TTY          TIME CMD
hermann      797       1  0 00:02 ?        00:00:00 /lib/systemd/systemd --user
hermann      815     792  0 00:02 ?        00:00:00 /usr/bin/lxsession -s LXDE -e LXDE
...
hermann      946       1  0 00:02 ?        00:00:04 lxterminal
...
hermann     1013     946  0 00:02 pts/0    00:00:00 bash
...
hermann     1446    1440  0 01:01 ?        00:00:00 sshd: hermann@pts/1
hermann     1447    1446  0 01:01 pts/1    00:00:00 -bash
hermann     7677    1447  0 02:30 pts/1    00:00:00 ps -f -u hermann
```

Prozess-spezifische Umgebungsvariablen

Zwei Shell-Variablen werden immer automatisch gesetzt, wenn ein Prozess gestartet wird:

- `$$`: die Prozess-ID des aktuellen Prozesses. Damit kann jeder Prozess (auch ein Shell-Prozess) seine eigene PID (Prozess-ID) ermitteln.
- `$PPID`: die Prozess-ID des Elternprozesses. Damit kann jeder Prozess (auch ein Shell-Prozess) die PID seines Elternprozesses ermitteln.

```
hermann@debian:~$ echo "process id (PID) of current shell: $$"
process id (PID) of current shell: 988
hermann@debian:~$ ps -f -p $$
UID          PID    PPID  C STIME TTY          TIME CMD
hermann      988     987  0 13:18 pts/1    00:00:00 -bash
```

```
hermann@debian:~$ echo "parent process id (PPID) of current shell: $PPID"
parent process id (PPID) of current shell: 987
hermann@debian:~$ ps -f -p $PPID
UID          PID    PPID  C STIME TTY          TIME CMD
hermann      987     981  0 13:18 ?        00:00:00 sshd: hermann@pts/1
```

Vordergrund- und Hintergrund-Prozesse

- **Vordergrund-Prozesse**: blockieren die Shell, die sie gestartet hat, bis sie beendet sind. Erst danach zeigt die Shell wieder den Prompt an und wartet auf die nächste Eingabe.
- **Hintergrund-Prozesse**: blockieren die Shell nicht. Sie laufen unabängig von der Shell, die sie gestartet hat, im Hintergrund weiter. Die Shell zeigt die PID (und die Jobnummer) des Hintergrund-Prozesses und danach den Prompt an und ist sofort wieder bereit für die nächste Eingabe. Hintergrund-Prozesse schreiben ihre Ausgaben und Fehlerausgaben auch auf den Bildschirm. Sie lesen aber nicht von der Tastatur.

Zum Inhaltsverzeichnis ...

Hintergrund-Prozesse starten mit `&`

```
hermann@debian:~$ # start 3 bg processes at 16:33:20
hermann@debian:~$ date; sleep 20 & sleep 30 & sleep 40 &
Fr 29. Nov 16:33:20 CET 2024
[1] 19340
[2] 19341
[3] 19342
```

Hintergrund-Prozesse anzeigen/kontrollieren mit `ps -f`

```
hermann@debian:~$ # show processes at 16:33:23 - 3 bg processes are running
hermann@debian:~$ date; ps -f
Fr 29. Nov 16:33:23 CET 2024
UID          PID    PPID  C STIME TTY          TIME CMD
hermann      988     987  0 13:18 pts/1    00:00:00 -bash
hermann    19340     988  0 16:33 pts/1    00:00:00 sleep 20
hermann    19341     988  0 16:33 pts/1    00:00:00 sleep 30
hermann    19342     988  0 16:33 pts/1    00:00:00 sleep 40
hermann    19344     988 99 16:33 pts/1    00:00:00 ps -f
```

```
hermann@debian:~$ # show processes at 16:33:41 - 2 bg processes are running
hermann@debian:~$ date; ps -f
Fr 29. Nov 16:33:41 CET 2024
[1]   Fertig                  sleep 20
UID          PID    PPID  C STIME TTY          TIME CMD
hermann      988     987  0 13:18 pts/1    00:00:00 -bash
hermann    19341     988  0 16:33 pts/1    00:00:00 sleep 30
hermann    19342     988  0 16:33 pts/1    00:00:00 sleep 40
hermann    19386     988  0 16:33 pts/1    00:00:00 ps -f
```

```
hermann@debian:~$ # show processes at 16:33:51 - 1 bg process is running
hermann@debian:~$ date; ps -f
Fr 29. Nov 16:33:51 CET 2024
[2]-  Fertig                   sleep 30
UID          PID    PPID  C STIME TTY          TIME CMD
hermann      988     987  0 13:18 pts/1    00:00:00 -bash
hermann    19342     988  0 16:33 pts/1    00:00:00 sleep 40
hermann    19408     988  0 16:33 pts/1    00:00:00 ps -f
```

```
hermann@debian:~$ # show processes at 16:34:03 - no bg process is running
hermann@debian:~$ date; ps -f
Fr 29. Nov 16:34:03 CET 2024
[3]+  Fertig                   sleep 40
UID          PID    PPID  C STIME TTY          TIME CMD
hermann      988     987  0 13:18 pts/1    00:00:00 -bash
hermann    19431     988  0 16:34 pts/1    00:00:00 ps -f
```

Ein- und Ausgabe von Hintergrund-Prozessen

STDOUT und STDERR von Hintergrund-Prozessen

Hintergrund-Prozesse schreiben ihre Ausgaben und Fehlerausgaben auf den Bildschirm. Diese Ausgaben können optisch mit den Ausgaben der Shell und ihrer Kindprozesse, die im Vordergrund laufen, interferieren. Das kann für den Benutzer störend sein.

Um das zu verhindern, kann man die Ausgaben und Fehlerausgaben von Hintergrund-Prozessen in Dateien umlenken.

STDIN von Hintergrund-Prozessen

Ein Hintergrund-Prozess liest nicht von der Tastatur. Die Standardeingabe bleibt mit dem TTY verbunden. Der Prozess jedoch wird angehalten. So wird verhindert, dass dem Vordergrund-Prozess, der von der Tastatur liest, die Eingaben "weggefressen" werden.

Auch hier kann das Problem mit der Umlenkung der Standardeingabe aus einer Datei gelöst werden. Liest der Hintergrund-Prozess aus einer Datei, dann wird er nicht angehalten.

Key Takeaways

- Ein Prozess ist ein laufendes Programm, das im Arbeitsspeicher des Computers geladen ist und vom Betriebssystem verwaltet wird.
- Prozesse haben sehr viele Attribute, von denen wir nur einige kennen gelernt und mit `ps -f` angezeigt haben. Die wichtigsten …
  - *PID*: systemweit eindeutige Prozess-ID
  - *PPID*: PID des Elternprozesses
  - *UID*: Benutzer-ID des Prozesses (wichtig für die Zugriffsrechte)
  - *TTY*: Gerätedatei des Terminals, an das der Prozess gebunden ist
  - etc. (Es gibt viele weitere Attribute.)

- Jeder Prozess hat einen Elternprozess, der ihn erzeugt hat. So entsteht eine Prozess-Hierarchie, die bis zum `init`-Prozess zurückverfolgt werden kann.
- TTYs sind Gerätedateien, die Terminals repräsentieren. Pseudo-TTYs sind virtuelle Terminals (in Terminal-Emulatoren oder bei SSH-Sitzungen).
- Die Umgebungsvariablen `$$` und `$PPID` enthalten die Prozess-ID des aktuellen Prozesses und die PID des Elternprozesses.
- Hintergrund-Prozesse blockieren die Shell nicht und werden mit dem `&`-Symbol am Ende des Kommandos gestartet.

Aufgaben

- Stellen Sie die Beispiele nach und überprüfen Sie die Ausgaben. Achten Sie dabei auf die Prozess-Hierarchie (PID und PPID) und die TTYs.
- Welchen Dateityp haben die TTYs `/dev/pts/0` und `/dev/pts/1` in der ersten Spalte der Ausgabe von `ls -l`? Was bedeutet das?

- Der Dozent stellt Ihnen ein Shell-Skript `script-pid` zur Verfügung. Kopieren Sie dieses Skript in ihr Verzeichnis `~/bin-trainer` und machen Sie es ausführbar. Ermitteln Sie die PID der `bash` ihrer aktuellen Sitzung mit `echo $$` und führen Sie dann das Skript auf drei verschiedene Arten aus:
  - `script-pid`
  - `bash ~/bin-trainer/script-pid`
  - `source script-pid`
- Was ist der Unterschied zwischen den drei Varianten? Achten Sie jeweils auf die PID und PPID der Skript-Shell!

- Der Dozent stellt Ihnen ein weiteres Shell-Skript zur Verfügung: `process-hierarchy`. Führen Sie das Skript aus und übergeben Sie ihm als Argument die PID eines Prozesses des Systems, die sie zuvor mit `ps` ermittelt haben. Was geschieht, wenn Sie keine PID als Argument übergeben?
- Sehen Sie sich das Skript `process-hierarchy` an, z.B. mit `nl -ba process-hierarchy`. Versuchen Sie zu verstehen, wie das Skript funktioniert. Welche Befehle werden verwendet? Es werden einige Shell-Techniken verwendet, die noch nicht besprochen wurden. (Diese Aufgabe ist fortgeschritten und deshalb optional.)

Signale senden mit `kill`

Mit dem Kommando `kill` können Signale an Prozesse gesendet werden. Der Prozess, der das Signal empfängt, kann auf unterschiedliche Weise darauf reagieren:

- Der Prozess kann das Signal ignorieren.
- Der Prozess kann das Signal abfangen und darauf reagieren.
- Der Prozess fängt das Signal nicht ab und wird beendet.

Von den 64 Signalen, die in einem modernen Linux-System zur Verfügung stehen, sind die folgenden die wichtigsten:

Zum Inhaltsverzeichnis ...

Die wichtigsten Signale für Shell-Anwender und Admins

- `SIGINT` **(Signal 2)**: Interrupt-Signal, das durch die Tastenkombination `Ctrl-C` vom Terminal-Benutzer erzeugt wird. Standardaktion: Beenden des Prozesses. Es kann auch abgefangen oder ignoriert werden.
- `SIGTERM` **(Signal 15)**: Terminierungs-Signal, das durch `kill`-Kommando erzeugt wird. Standardaktion: Beenden des Prozesses. Das Signal kann auch abgefangen oder ignoriert werden.
- `SIGKILL` **(Signal 9)**: Kill-Signal, das durch `kill -9`-Kommando erzeugt wird. Standardaktion: Sofortiges Beenden des Prozesses. !!! Dieses Signal kann nicht abgefangen oder ignoriert werden !!!

Wie beendet man als Terminal-Benutzer einen Prozess?

Signal 2 (SIGINT) senden:

Blockiert ein Vordergrund-Prozess das Terminal (z. B. weil er in einer Endlosschleife hängt), kann man ihn meist mit der Tastenkombination `Ctrl-C` beenden. `Ctrl-C` sendet das Signal 2 an den Prozess und bricht ihn damit in der Regel ab. Hintergrundprozesse sind mit `Ctrl-C` nicht erreichbar. Man kann ihnen jedoch das Signal 2 mit dem `kill`-Kommando senden: `kill -2 <pid>` oder `kill -INT <pid>`. Signal 2 mit dem `kill`-Kommando zu senden, ist eher unüblich. Meist nimmt man mit dem `kill`-Kommando die Signale 15 oder 9.

Signal 15 (SIGTERM) senden:

Wenn `Ctrl-C` bzw. das Senden von Signal 2 nicht funktioniert (weil dieses Signal ignoriert wird), dann kann man sich an einem anderen Terminal anmelden und den Prozess mit `kill -15` oder `kill -TERM` beenden. Der Prozess reagiert darauf normalerweise mit einer geregelten Beendigung. (Er speichert ggf. seine Daten und gibt alle belegten Ressourcen frei, bevor er sich beendet.) Der Prozess kann aber auch so programmiert sein, dass er Signal 15 (SIGTERM) ignoriert. Dann ist die Beendigung auf diesem Weg nicht möglich.

Signal 9 (SIGKILL) senden:

- Wenn auch `kill -15` nicht funktioniert, dann bleibt als letzter Ausweg `kill -9`, bzw. `kill -KILL`. Dieses Signal kann nicht ignoriert oder abgefangen werden. Der Prozess wird sofort beendet, ohne dass er die Möglichkeit hat, sich kontrolliert zu beenden oder Ressourcen freizugeben. Dies kann zu Datenverlust führen.

Hintergrundprozesse sind mit `Ctrl-C` nicht erreichbar. Sie können nur mit dem `kill`-Kommando beendet werden.

Die Signalnummer 15 ist beim `kill`-Kommando der Standardwert. Sie kann weggelassen werden.

Senden von Signal 2 (SIGINT) mit `Ctrl+C` oder `kill -2`

```
hermann@debian:~$ tty
/dev/pts/1
hermann@debian:~$ sleep 60 # fg process
^C
hermann@debian:~$ # fg process was interrupted with Ctrl-C
hermann@debian:~$ sleep 60 & # bg process
[1] 37617
hermann@debian:~$ kill -2 37617 # send SIGINT to bg process
hermann@debian:~$
[1]+  Unterbrechung           sleep 60
```

Statt `kill -2 <pid>` kann auch die symboische Signalbezeichnung `kill -INT <pid>` verwendet werden.

Senden von Signal 15 (SIGTERM) mit `kill -15`

```
hermann@debian:~$ sleep 60 & # bg process
[1] 37620
hermann@debian:~$ kill -15 37620
hermann@debian:~$
[1]+  Beendet                 sleep 60
```

Statt `kill -15 <pid>` kann auch `kill -TERM <pid>` verwendet werden. Man kann die Signalnummer 15 auch weglassen, da sie der Standard-Signalwert ist.

Senden von Signal 9 (SIGKILL) mit `kill -9`

```
hermann@debian:~$ sleep 60 & # bg process
[1] 37626
hermann@debian:~$ kill -9 37626
hermann@debian:~$
[1]+  Getötet                 sleep 60
```

Statt `kill -9 <pid>` kann auch `kill -KILL <pid>` verwendet werden.
Das Signal 9 kann keinesfalls ignoriert oder abgefangen werden.

Key Takeaways - Signale

- Wenn sich ein Vordergrund-Prozess im Terminal (TTY) "festgefahren" hat, dann versuchen Sie ihn zuerst mit `Ctrl-C` zu beenden.
- Wenn das nicht funktioniert, oder wenn es sich um einen Hintergrund-Prozess handelt oder um einen Prozess, der nicht an ein TTY gebunden ist, dann versuchen Sie es mit `kill -15 <pid>`. Damit geben Sie dem Prozess die Chance, sich kontrolliert zu beenden. (Er kann dann z.B. noch Daten speichern oder belegte Ressourcen freigeben. Er könnte aber auch so programmiert sein, dass er das Signal 15 ignoriert.)

- Lässt sich der Prozess auch mit `kill -15` nicht beenden, dann bleibt als letzter Ausweg das Senden von Signal 9 mit `kill -9 <pid>`. Dieses Signal führt sofort zum bedingungslosen Abbruch des Prozesses. Der Prozess hat keine Möglichkeit mehr, sich kontrolliert zu beenden. Es kann zu Datenverlust kommen.