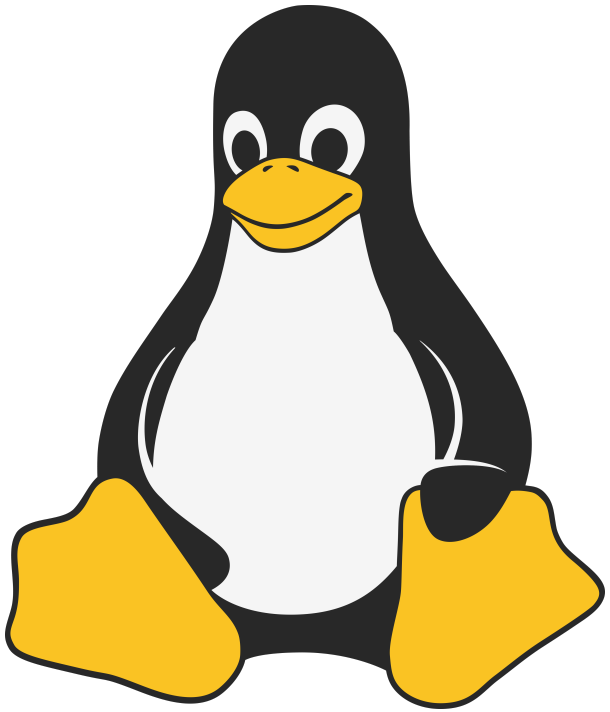
Prozesse und Signale



Inhaltsverzeichnis

- Prozesse anzeigen mit `ps`
- Prozesse anzeigen mit `top` und `htop`
- Vordergrund- und Hintergrundprozesse
- Signale
- Signale auflisten mit `kill -l`
- Die wichtigsten Signale für Shell-Anwender und Admins
- Signalreaktion mit `trap`
- Signale senden mit `kill`

Vorbemerkung

Dieser Foliensatz ist Erweiterung und Vertiefung des Foliensatzes "Prozesse und Kindprozesse" im Kapitel 04.

Dies ist optionaler Stoff zum Selbststudium und zur Vertiefung des Wissens über Prozesse und Signale in Linux gedacht.

Zum 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).

Prozess-Status (vereinfacht)

Ein Prozess kann sich in einem von fünf Zuständen befinden:

- **RUNNING & RUNNABLE (R)**: Der Prozess wird gerade ausgeführt oder wartet darauf, dass ihm (vom Scheduler) Rechenzeit zugeteilt wird.
- **UNINTERRUPTABLE_SLEEP (D)**: Der Prozess wartet auf ein Ereignis, das nicht durch ein Signal unterbrochen werden kann. Typischerweise ist dies ein Ereignis, das den Abschluss einer I/O-Operation signalisiert (z. B. die Verfügbarkeit von Daten von der Festplatte oder aus dem Netzwerk).

- **INTERRUPTABLE_SLEEP (S)**: Der Prozess wartet auf ein Ereignis, das durch ein Signal unterbrochen werden kann. Typischerweise ist dies ein Ereignis wie der Empfang von eines Netzwerk-Requests bei einem Server-Prozess (Web-Server, File-Server, etc.). Es kann auch ein Ereignis sein, das den Abschluss einer Tastatureingabe signalisiert, z. B. das Drücken einer Taste.
- **STOPPED (T)**: Der Prozess wurde angehalten (z. B. durch das Signal SIGSTOP oder SIGTSTP). Ein angehaltener Prozess kann mit dem Signal SIGCONT fortgesetzt werden.

- **ZOMBIE (Z)**: Der Prozess hat seine Arbeit beendet, aber der Endestatus des Prozesses wurde noch nicht vom Elternprozess abgerufen. Der Prozess bleibt in der Prozesstabelle des Systems verzeichnet, bis der Elternprozess den Endestatus abgerufen hat. Erst danach wird der Prozess aus der Prozesstabelle entfernt.

Prozess-Hierarchie

Jeder Prozess hat einen Elternprozess. Der Elternprozess ist der Prozess, der den Prozess erzeugt hat. Jeder Prozess kann selbst 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.

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.

Prozess-Attribute

Der Linux-Kernel speichert für jeden Prozess eine Vielzahl von Informationen, die als Prozess-Attribute bezeichnet werden. (Je nach Kernel-Version liegt deren Anzahl in der Größenordnung von 50 Attributen.)

Die wichtigsten Prozess-Attribute:

- **Prozess-ID (PID)**: Eindeutige Nummer, die den Prozess identifiziert
- **Eltern-Prozess-ID (PPID)**: Die Prozess-ID des Elternprozesses

- **Benutzer-ID (UID)**: Die Benutzer-ID des Benutzers, der den Prozess gestartet hat
- **Effektive Benutzer-ID (EUID)**: Die Benutzer-ID, die für die Zugriffskontrolle verwendet wird (unterscheidet sich z.B bei `sudo` von der UID)
- **Gruppen-ID (GID)**: Die Gruppen-ID des Benutzers, der den Prozess gestartet hat.
- **Effektiive Gruppen-ID (EGID)**: Die Gruppen-ID, die für die Zugriffskontrolle verwendet wird (unterscheidet sich z.B bei `sudo`)
- **Prozess-Status (S)**: Der aktuelle Status des Prozesses (siehe oben)

- **Priorität (PRI)**: Die Priorität des Prozesses (0-99, 0 = höchste Priorität). Sie wird vom Scheduler bei der Zuteilung von Rechenzeit berücksichtigt.
- **Nice-Wert (NI)**: Der Nice-Wert des Prozesses (-20 bis +19, 0 = Standardwert). Dieser Wert kann die Priorität des Prozesses beeinflussen. Damit signalisiert ein Prozess dem Scheduler, dass er Schlechter priorisiert werden kann. Ein Prozess mit einem hohen Nice-Wert wird seltener Rechenzeit zugeteilt. Ein root-Prozess kann auch einen negativen Nice-Wert verwenden, um eine höhere Priorität zu erhalten.
- **Hauptspeicher-Belegung (RSS)**: Die Größe des Hauptspeichers, der nicht ausgelagert wurde (Resident Set Size)

- **Größe des Prozess-Images (SZ)**: Die Größe des Prozess-Images im Hauptspeicher in Speicherseiten
- **Wait-Channel (WCHAN)**: Adresse der Kernelfunktion, in der der Prozess schläft, wenn er im Zustand `D` oder `S` ist. Dies zeigt an, auf welches Ereignis der Prozess wartet.
- **Startzeit (STIME)**: Die Zeit, zu der der Prozess gestartet wurde
- **CPU-Zeit (TIME)**: Die verbrauchte CPU-Zeit des Prozesses
- **Terminal (TTY)**: Das Terminal, an das der Prozess gebunden ist. Bei einem Prozess, der nicht an ein Terminal gebunden ist, steht hier `?` (z.B. Systemdienste)
- **Befehlszeile (CMD)**: Das Kommando (und seine Argumente), mit dem der Prozess gestartet wurde

Weitere Prozess-Attribute

- **Dateideskriptoren**: Kennnummern für geöffnete Dateien, Sockets (Netzwerkverbindungen) und Pipes (Kommunikationskanäle zwischen Prozessen)
- **Umgebungsvariablen**: Umgebungsvariablen, die vom Prozess verwendet werden
- **Arbeitsverzeichnis**: Das Arbeitsverzeichnis des Prozesses
- **Signale**: Die Signale, die der Prozess empfangen will und solche, die er blockiert

Zum Inhaltsverzeichnis ...

- **Signal-Handler**: Signal-Behandlungsroutinen, die beim Empfang eines Signals ausgeführt werden
- **Prozess-Gruppen-Informationen**: Informationen über die Prozessgruppe, zu der der Prozess gehört (Eine Prozessgruppe mit einer gemeinsamen Prozessgruppen-ID (PGID) wird verwendet, um Signale an alle Prozesse der Gruppe zu senden.)
- **Thread-Informationen**: Informationen über die Threads, die im Kontext des Prozesses laufen
- viele weitere ...

Geschichte des `ps`-Kommandos

Das `ps`-Kommando stammt aus den 70er Jahren und wurde ursprünglich für das Betriebssystem Unix entwickelt. In den 80er Jahren war Unix in viele Varianten zersplittert.

Die beiden Haptlinien der Unix-Entwicklung waren BSD Unix (Berkeley Software Distribution) und AT&T Unix (weiterentwickelt zu Unix System V). Jede Linie hatte ihre eigene Version des `ps`-Kommandos.

`ps`-Options-Dschungel

Dies spiegelte sich in der Vielfalt der Optionen und der Ausgabeformate des `ps`-Kommandos bis heute wider. Es gab und gibt auch keine einheitliche Syntax für das `ps`-Kommando.

Unter Linux unterstützt das `ps`-Kommando zwei Gruppen von Schaltern (Optionen):

- BSD-Style-Optionen ohne führendes `-`
- AT&T-Style-Optionen mit führendem `-`

Die BSD-Style-Optionen sind die am häufigsten verwendeten. Welche Optionen man verwenden will, ist einerseits eine Geschmacksfrage, andererseits abhängig von der gewünschten Ausgabeform.

Beide Optionsstile können auch in einer Kommandoeingabe kombiniert werden.

Um den Einstieg zu erleichtern, verwenden wir im Folgenden eine Auswahl des Dozenten. Diese kleine Auswahl basiert auf den AT&T-Style-Optionen mit führendem `-`.

`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:

- `-fly`: Die Kombination der Optionen `-f`, `-l` und `-y` erweitert das Ausgabeformat um weitere Attribute (Spalten: S, UID, PID, PPID, C, PRI, NI, RSS, SZ, WCHAN, STIME, TTY, TIME, CMD).

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.
- `-U <user,...>`: zeigt nur die Prozesse der angegebenen Benutzer an. Zur Auswahl werden die realen Benutzer-IDs (UID) verwendet.

Prozesse anzeigen mit `ps`

Prozesse im aktuellen Terminal (Kurz- und Langformat)

```
hermann@debian:~$ ps # processes of the current tty in short format
    PID TTY          TIME CMD
   1459 pts/0    00:00:00 bash
   1487 pts/0    00:00:00 ps
```

```
hermann@debian:~$ ps -fly # processes of the current tty in long format
S UID          PID    PPID  C PRI  NI    RSS    SZ WCHAN  STIME TTY          TIME CMD
S hermann     1459    1458  0  80   0   5124  2059 do_wai 18:38 pts/0    00:00:00 -bash
R hermann     1490    1459  0  80   0   4508  2795 -      18:43 pts/0    00:00:00 ps -fly
```


Zum Inhaltsverzeichnis ...

Alle Prozesse des Systems (Kurz- und Langformat)

```
hermann@debian:~$ ps -e # all processes of the system in short format
    PID TTY          TIME CMD
      1 ?        00:00:00 systemd
      2 ?        00:00:00 kthreadd
...
      8 ?        00:00:00 kworker/0:0H-events_highpri
...
   1459 pts/0    00:00:00 bash
...
   1510 pts/0    00:00:00 ps
```

```
hermann@debian:~$ ps -fly -e # all processes of the system in long format
...
S UID          PID    PPID  C PRI  NI    RSS    SZ WCHAN  STIME TTY          TIME CMD
S root           1       0  0  80   0 12576 41988 -      16:16 ?        00:00:00 /sbin/init
S root           2       0  0  80   0     0     0 -      16:16 ?        00:00:00 [kthreadd]
I root           3       2  0  60 -20     0     0 -      16:16 ?        00:00:00 [kworker/0:0H
I root          10       2  0  60 -20     0     0 -      16:16 ?
...
S hermann     1459    1458  0  80   0  5124  2059 do_wai 18:38 pts/0    00:00:00 -bash
...
R hermann     1523    1459 99  80   0  4536  2795 -      18:54 pts/0    00:00:00 ps -fly -e
```

Prozess-Hierarchie

```
hermann@debian:~$ ps -fly # get PPID of bash process
S UID          PID    PPID  C PRI  NI    RSS    SZ WCHAN  STIME TTY          TIME CMD
S hermann     1570    1567  0  80   0  5096  2061 do_wai 19:10 pts/0    00:00:00 bash
R hermann     1615    1570 99  80   0  4500  2795 -      19:15 pts/0    00:00:00 ps -fly
```

```
hermann@debian:~$ ps -fly -p 1567
S UID          PID    PPID  C PRI  NI    RSS    SZ WCHAN  STIME TTY          TIME CMD
S hermann     1567     878  0  80   0 39728 101042 do_sys 19:10 ?        00:00:00 lxterminal
```

```
hermann@debian:~$ ps -fly -p 878
S UID          PID    PPID  C PRI  NI    RSS    SZ WCHAN  STIME TTY          TIME CMD
S hermann      878     778  0  80   0 54512 182695 do_sys 16:16 ?        00:00:01 pcmanfm --desktop --profile LXDE
```

```
hermann@debian:~$ ps -fly -p 778
S UID          PID    PPID  C PRI  NI    RSS    SZ WCHAN  STIME TTY          TIME CMD
S hermann      778     756  0  80   0 17728 63998 do_sys 16:16 ?       00:00:00 /usr/bin/lxsession -s LXDE -e LXD
```

```
hermann@debian:~$ ps -fly -p 756
S UID          PID    PPID  C PRI  NI    RSS    SZ WCHAN  STIME TTY          TIME CMD
S root         756     663  0  80   0  7864 40627 -      16:16 ?       00:00:00 lightdm --session-child 15 18
```

```
hermann@debian:~$ ps -fly -p 663
S UID          PID    PPID  C PRI  NI    RSS    SZ WCHAN  STIME TTY          TIME CMD
S root         663       1  0  80   0  7636 77217 -      16:16 ?       00:00:00 /usr/sbin/lightdm
```

```
hermann@debian:~$ ps -fly -p 1
S UID          PID    PPID  C PRI  NI    RSS    SZ WCHAN  STIME TTY          TIME CMD
S root           1       0  0  80   0 12576 41988 -      16:16 ?       00:00:01 /sbin/init
```

Prozesse in verschiedenen Terminals

```
hermann@debian:~$ tty # working in terminal pts/0
/dev/pts/0
hermann@debian:~$ # start 3 'sleep' processes in bg and a 'ping' in fg
hermann@debian:~$ sleep 300 & sleep 400 & sleep 500 & ping localhost
hermann@debian:~$ sleep 300 & sleep 400 & sleep 500 & ping localhost
[1] 1076
[2] 1077
[3] 1078
PING localhost(localhost (::1)) 56 data bytes
64 bytes from localhost (::1): icmp_seq=1 ttl=64 time=0.033 ms
64 bytes from localhost (::1): icmp_seq=2 ttl=64 time=0.058 ms
64 bytes from localhost (::1): icmp_seq=3 ttl=64 time=0.076 ms
...
```

```
hermann@debian:~$ tty # working in terminal pts/1
/dev/pts/1
hermann@debian:~$ # check the processes in terminal pts/0 and pts/1
hermann@debian:~$ hermann@tuxp14:~$ ps -fly -t pts/0,pts/1
S UID          PID    PPID  C PRI  NI    RSS    SZ WCHAN  STIME TTY          TIME CMD
S hermann     1038    1037  0  80   0  5168  2059 do_wai 19:34 pts/1      0:00 -bash
S hermann     1052    1051  0  80   0  5876  2407 do_wai 19:35 pts/0      0:00 -bash
S hermann     1076    1052  0  80   0   876  1366 hrtime 19:43 pts/0      0:00 sleep 300
S hermann     1077    1052  0  80   0   868  1366 hrtime 19:43 pts/0      0:00 sleep 400
S hermann     1078    1052  0  80   0   908  1366 hrtime 19:43 pts/0      0:00 sleep 500
S hermann     1079    1052  0  80   0  1096  1884 -      19:43 pts/0      0:00 ping localhost
R hermann     1080    1038 50  80   0  4536  2795 -      19:44 pts/1      0:00 ps -fly t pts/
```

Prozesse eines Benutzers

```
hermann@debian:~$ ps -u hermann
    PID TTY          TIME CMD
    739 ?        00:00:00 systemd
...
    756 ?        00:00:00 lxsession
...
    883 ?        00:00:00 lxterminal
...
    954 pts/0    00:00:00 bash
   1085 pts/0    00:00:00 ps
hermann@debian:~$ ps -u hermann | wc -l
36
```

Prozesse anzeigen mit `top` und `htop`

`top` und `htop` sind interaktive Prozess-Überwachungsprogramme, die eine Liste der laufenden Prozesse anzeigen und Informationen über die Systemauslastung bereitstellen. Sie zeigen die Prozesse in Echtzeit an und aktualisieren die Anzeige regelmäßig (standardmäßig alle 2 Sekunden (`top`) bzw. alle 1 Sekunde (`htop`)).

`top` bietet außer dem interaktiven Modus mit der Option `-b` auch einen Batch-Modus, mit dem die Ausgabe in eine Datei oder über eine Pipe in ein anderes Programm umgeleitet werden kann. Mit der zusätzlichen Option `-n <count>` kann die Anzahl der Iterationen festgelegt werden.

Der Vorzug von `htop` gegenüber `top` ist die bessere Benutzeroberfläche, die farbige Darstellung und die erweiterten interaktiven Funktionen zur Sortierung und Filterung der Prozesse. `htop` hat stellt jedoch keine Batch-Modus zur Verfügung.

`top` im Batch-Modus

```
hermann@debian:~$ top -b -n 1
top - 22:38:03 up  3:15,  3 users,  load average: 0,00, 0,00, 0,00
Tasks: 125 total,   1 running, 124 sleeping,   0 stopped,   0 zombie
%CPU(s):  0,0 us,100,0 sy,  0,0 ni,  0,0 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
MiB Spch:   2008,9 total,   1153,6 free,    539,1 used,    544,6 buff/cache
MiB Swap:    976,0 total,    976,0 free,      0,0 used.   1469,7 avail Spch

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     ZEIT+ BEFEHL
      1 root      20   0  102472  12328   9208 S   0,0   0,6   0:00.82 systemd
      2 root      20   0       0      0      0 S   0,0   0,0   0:00.01 kthreadd
      3 root       0 -20       0      0      0 I   0,0   0,0   0:00.00 rcu_gp
      4 root       0 -20       0      0      0 I   0,0   0,0   0:00.00 rcu_par_gp
      5 root       0 -20       0      0      0 I   0,0   0,0   0:00.00 slub_flushwq
      6 root       0 -20       0      0      0 I   0,0   0,0   0:00.00 netns
      8 root       0 -20       0      0      0 I   0,0   0,0   0:00.00 kworker/0:0H-events+
...
```

Vordergrund- und Hintergrundprozesse

- Ein Prozess, der von einer Shell gestartet wird, wird standardmäßig im Vordergrund ausgeführt. Das bedeutet, dass die Shell auf die Beendigung des Prozesses wartet, bevor sie den Prompt wieder anzeigt.
- Ein Prozess kann auch im Hintergrund ausgeführt werden, indem man ein Kommando mit einem `&`-Zeichen abschließt. Der Prozess wird dann im Hintergrund gestartet und die Shell wartet nicht auf seine Beendigung. Nach dem Start des Hintegrundprozesses wird zunächst seine Prozess-ID (und die Jobnummer in eckigen KLammern) angezeigt und danach der Shell-Prompt.

Signale

Siehe auch: Unix-Signale auf Wikipedia

Ein Signal ist eine Benachrichtigung, die an einen Prozess oder an einen Prozessgruppe gesendet wird, um ihm eine bestimmte Information zu übermitteln oder um ihn zu einer bestimmten Aktion zu veranlassen.

Signal-Sender ist ein Prozess, das Betriebssystem oder ein Benutzer (über die Tastatur, z. B. mit `Ctrl-C`).

Signal-Empfänger ist immer ein Prozess (oder eine Prozessgruppe).

Der Signal-Empfänger kann grundsätzlich auf unterschiedlich Weise auf das Eintreffen eines Signals reagieren:

- **Ignorieren**: Der Prozess reagiert nicht auf das Signal.
- **Verarbeiten**: Der Prozess reagiert mit einer Standardaktion auf das Signal (z. B. Anhalten, Fortsetzen, Beenden mit oder ohne Speicherabbild (Core-Dump)).
- **Blockieren**: Der Prozess blockiert das Signal zeitweise, bis er die Blockierung wieder aufhebt.
- **Abfangen**: Der Prozess führt eine spezielle, durch den Programmierer definierte Funktion (Signal-Handler) aus, wenn das Signal eintrifft.

Signale auflisten mit `kill -l`

```
hermann@debian:~$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL       5) SIGTRAP
 6) SIGABRT      7) SIGBUS       8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```


Zum Inhaltsverzeichnis ...

Die wichtigsten Signale für Shell-Anwender und Admins

- `SIGINT` **(Signal 2)**: Interrupt-Signal, das durch die Tastenkombination `Ctrl-C` vom Terminal-Benuzter erzeugt wird. Standardaktion: Beenden des Prozesses.
- `SIGTERM` **(Signal 15)**: Terminierungs-Signal, das durch `kill`-Kommando erzeugt wird. Standardaktion: Beenden des Prozesses.
- `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 !!!

- `SIGCHLD` **(Signal 17)**: Child-Signal, das an den Elternprozess gesendet wird, wenn ein Kindprozess beendet wurde. Standardaktion: Ignorieren.
- `SIGCONT` **(Signal 18)**: Continue-Signal, das einen angehaltenen Prozess fortsetzt. Standardaktion: Fortsetzen des Prozesses.
- `SIGSTOP` **(Signal 19)**: Stop-Signal, das einen Prozess anhält. Standardaktion: Anhalten des Prozesses. !!! Dieses Signal kann nicht abgefangen oder ignoriert werden !!!
- `SIGTSTP` **(Signal 20)**: Terminal-Stop-Signal, das durch `Ctrl-Z` vom Terminal-Benuzter erzeugt wird. Standardaktion: Anhalten des Prozesses.

Wie beendet man als Terminal-Benutzer einen Prozess?

- Blockiert ein Prozess das Terminal (z. B. weil er in einer Endlosschleife hängt), kann man ihn meist mit `Ctrl-C` beenden.
- Wenn `Ctrl-C` nicht funktioniert, 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.

- Wenn auch `kill -15` nicht funktioniert, dann bleibt als letzter Ausweg `kill -9`, bzw. `kill -SIGKILL`. 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 `kill -15` oder `kill -9` beendet werden.

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

Signalreaktion mit `trap`

Mit dem `trap`-Kommando kann man in einem Shell-Skript eine Signal-Behandlungsroutine (Signal-Handler) definieren. Ist die Signal-Behandlungsroutine eine leere Zeichenkette, dann wird das Signal ignoriert.

Syntax:: `trap [Signal-Handler] [Signal-Nummer]`

Signale senden mit `kill`

Für Signale 2 (SIGINT) und 15 (SIGTERM) wird ein leerer Signal-Handler definiert. D.h. der Prozess ignoriert diese Signale.

```
hermann@debian:~$ tty
/dev/pts/0
hermann@debian:~$ (trap '' 2 15; sleep 300) # not interruptible with SIGINT and SIGTERM
hermann@debian:~$ (trap '' 2 15; sleep 300)
^C
Getötet
hermann@debian:~$
hermann@debian:~$
```

- Der Prozess kann nicht mit `Ctrl-C` beendet werden, weil das Signal 2 (SIGINT) ignoriert wird.

```
hermann@debian:~$ tty
/dev/pts/1
hermann@debian:~$ ps -t pts/0 # show processes in terminal pts/0
    PID TTY          TIME CMD
   1052 pts/0    00:00:00 bash
   1914 pts/0    00:00:00 sleep
hermann@debian:~$ kill -2 1914 # send SIGINT to sleep -> sleep not interrupted
hermann@debian:~$ kill -15 1914 # send SIGTERM to sleep -> sleep not interrupted
hermann@debian:~$ kill -9 1914 # send SIGKILL to sleep -> sleep interrupted in pts/0
```

Im Terminal pts/0 kann der sleep-Prozess in Terminal pts/0 nicht durch Senden der Signale 2 (SIGINT) und 15 (SIGTERM) beendet werden. `kill -2` und `kill -15` haben keine Wirkung. Der Prozess kann nur durch Senden von SIGKILL mit `kill -9` beendet werden.

```
hermann@debian:~$ (trap 'echo "You pressed Ctrl+C at $(date) !!!"' 2; date; sleep 30)
Sun Nov 24 02:01:22 CET 2024
^CYou pressed Ctrl+C at Sun Nov 24 02:01:31 CET 2024 !!!
```

In diesem Beispiel wird ein Signal-Handler für das Signal 2 (SIGINT) definiert. Wenn der Benutzer `Ctrl-C` drückt, wird die angegebene Meldung ausgegeben.

Hinweis: Das erste Datum ist das Datum, das beim Start des Prozesses ausgegeben wird. Das zweite Datum ist das Datum, das beim Drücken von `Ctrl-C` (in diesem Fall um 9 Sekunden zeitversetzt) ausgegeben wird.