

ps - Prozesse und Kindprozesse



BASH
THE BOURNE-AGAIN SHELL



Inhaltsverzeichnis

- [Was sind Prozesse?](#)
- [Prozess-Hierarchie](#)
- `ps` [-Optionen \(Auswahl\)](#)
- [Prozesse anzeigen mit](#) `ps`
- [Prozess-Hierarchie mit](#) `ps`
- [Prozesse an verschiedenen TTYs](#)
- [Prozesse eines Benutzers](#)

- Prozess-spezifische Umgebungsvariablen
- Vordergrund- und Hintergrund-Prozesse
- Key Takeaways
- Aufgaben
- Hintergrundprozesse (erweitert)
- Dämon-Prozesse

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-Hierarchie

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

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 keinen 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

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	573	572	0	13:44	pts/0	00:00:00	-bash
hermann	622	573	0	14:15	pts/0	00:00:00	ps -f

```
hermann@debian:~$ ps -f -p 572
```

UID	PID	PPID	C	STIME	TTY	TIME	CMD
hermann	572	549	0	13:44	?	00:00:00	sshd: hermann@pts/0

```
hermann@debian:~$ ps -f -p 549
```

UID	PID	PPID	C	STIME	TTY	TIME	CMD
root	549	519	0	13:44	?	00:00:00	sshd: hermann [priv]

```
hermann@debian:~$ ps -f -p 519
```

UID	PID	PPID	C	STIME	TTY	TIME	CMD
root	519	1	0	13:42	?	00:00:00	sshd: /usr/sbin/sshd -D [listener] ...

```
hermann@debian:~$ ps -f -p 1
```

UID	PID	PPID	C	STIME	TTY	TIME	CMD
root	1	0	0	13:42	?	00:00:00	/sbin/init

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 $USER
```

UID	PID	PPID	C	STIME	TTY	TIME	CMD
hermann	552	1	0	13:44	?	00:00:00	/lib/systemd/systemd --user
hermann	553	552	0	13:44	?	00:00:00	(sd-pam)
hermann	572	549	0	13:44	?	00:00:00	sshd: hermann@pts/0
hermann	573	572	0	13:44	pts/0	00:00:00	-bash
hermann	645	639	0	14:23	?	00:00:00	sshd: hermann@pts/1
hermann	646	645	0	14:23	pts/1	00:00:00	-bash
hermann	653	573	0	14:24	pts/0	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 die Shell ihre eigene PID ermitteln.
- **\$PPID**: die Prozess-ID des Elternprozesses. Damit kann die Shell ihre PPID (Parent Process ID) ermitteln.

(Dies gilt nicht nur für die Shell, sondern für alle Prozesse.)

```
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 unabhängig 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.

Kommando `sleep`

- Das Kommando `sleep seconds` wartet die angegebene Anzahl von Sekunden und beendet sich dann.

```
hermann@debian:~$ date; sleep 60; date; sleep 60; date
Mo 3. Feb 18:08:07 CET 2025
Mo 3. Feb 18:09:07 CET 2025
Mo 3. Feb 18:10:07 CET 2025
```

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.

- 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 und freiwillig.)

Hintergrundprozesse (erweitert)

Ein im Terminal gestarteter Hintergrundprozess bleibt **an das TTY gebunden**.

- Die Shell wartet nicht auf sein Ende,
- Er ist mit **Ctrl-C** nicht mehr zu unterbrechen (wie ein Vordergrundprozesse), aber ...
- Er schreibt seine Ausgaben und Fehlerausgaben auf das TTY. Dies lässt sich mit der Umlenkung der Ausgaben in Dateien verhindern.

- Die Standardeingabe bleibt mit dem TTY verbunden. Der Prozess wird angehalten, wenn er von der Tastatur liest. Dies lässt sich mit der Umlenkung der Standardeingabe aus einer Datei verhindern.
- Beim Beenden der Terminal-Sitzung (z.B. durch Abmelden oder Schließen des Terminal-Fensters) erhält er das Signal 1 (`SIGHUP`) und wird beendet. Dies lässt sich mit dem Kommando `nohup` verhindern, sodass der Prozess auch nach dem Beenden der Terminal-Sitzung weiterläuft. Auch jetzt ist der Hintergrundprozess noch an das TTY gebunden. Bei der Anzeige mit dem `ps`-Kommando wird das TTY, an dem er gestartet wurde, immer noch angezeigt.

- Beim Beenden der Shell in der Terminal-Sitzung geht allerdings für den mit `nohup` gestarteten Hintergrundprozess die **Bindung an das TTY verloren**. Da die Shell beendet wurde, gibt es das TTY nicht mehr und der Prozess verliert seinen Parent-Prozess. Der `init`-Prozess (mit der PID 1) "adoptiert" den Prozess und wird sein neuer Elternprozess. In der TTY-Spalte des `ps`-Kommandos wird ein `?` angezeigt.

Wir würden durch diese Maßnahmen auf sehr umständliche Weise erreichen, dass der Hintergrundprozess ein Dämon-Prozess wird. Ein Dämon-Prozess ist ein Hintergrundprozess, der keine Bindung an ein TTY hat.

Dämon-Prozesse

Ein Dämon-Prozess (engl. *daemon*) ist ein Hintergrundprozess, der unabhängig von der Shell und der Terminal-Sitzung läuft. Er ist **nicht an ein TTY gebunden**. Das `ps`-Kommando zeigt in der TTY-Spalte ein `?` an. Allerdings werden sie von vornherein ohne Bindung an ein TTY gestartet - typischerweise von `systemd` beim Systemstart.

Dämon-Prozesse werden für Dienste verwendet, die im Hintergrund laufen und auf Ereignisse warten, z.B. auf Netzwerkanfragen oder auf Timer-Ereignisse.

Beispiele für Dämon-Prozesse:

- `sshd`: der SSH-Dienst
- `httpd` oder `nginx`: Webserver
- `mysqld`: MySQL-Datenbank-Server
- `ntpd`: Network Time Protocol Daemon
- `cron`: der Cron-Daemon für zeitgesteuerte Aufgaben
- etc.