Jobkontrolle mit der Shell

`jobs`	Listet Ihre Jobs auf.
`&`	Führt einen Job im Hintergrund aus.
`^Z`	Unterbricht den aktuellen (Vordergrund-)Job.
`suspend`	Unterbricht eine Shell.
`fg`	Hebt die Unterbrechung eines Jobs auf und bringt ihn in den Vordergrund.
`bg`	Lässt einen unterbrochenen Job im Hintergrund weiterlaufen.

Alle Linux-Shells besitzen eine **Jobkontrolle**: die Fähigkeit, Programme im Hintergrund (Multitasking hinter den Kulissen) und im Vordergrund (als aktiven Prozess an Ihrem Shell-Prompt) auszuführen. Ein **Job** ist einfach die Arbeitseinheit der Shell. Wenn Sie einen Befehl interaktiv ausführen, verzeichnet Ihre aktuelle Shell ihn als einen Job. Wird der Befehl abgeschlossen, verschwindet der damit verknüpfte Job. Jobs bewegen sich auf einer höheren Ebene als Linux-Prozesse; das Linux-Betriebssystem hat von ihnen keine Ahnung. Sie sind lediglich Konstrukte der Shell. Hier sind einige wichtige Begriffe aus dem Bereich der Jobkontrolle:

Vordergrundjob
: Läuft in einer Shell und belegt den Shell-Prompt, sodass man keinen anderen Befehl ausführen kann.

Hintergrundjob
: Läuft in einer Shell, belegt aber den Shell-Prompt nicht, sodass man in derselben Shell einen weiteren Befehl ausführen kann.

Suspendieren
: Temporäres Anhalten eines Vordergrundjobs.

Fortsetzen (Resume)
: Einen angehaltenen Job veranlassen, im Vordergrund weiterzulaufen.

jobs

Der integrierte Befehl `jobs` listet die Jobs auf, die in Ihrer aktuellen Shell laufen.

```
→ jobs
[1]-  Running                 emacs meinedatei &
[2]+  Stopped                 ssh example.com
```

Der Integerwert auf der linken Seite ist die Jobnummer, und das Pluszeichen kennzeichnet den Standardjob, der von den Befehlen `fg` (Vordergrund) und `bg` (Hintergrund) beeinflusst wird.

&

An das Ende einer Kommandozeile platziert, sorgt der Ampersand dafür, dass der angegebene Befehl als Hintergrundjob läuft.

```
→ emacs meinedatei &
[2] 28090
```

Die Antwort der Shell enthält die Jobnummer (2) und die Prozess-ID des Befehls (28090).

^Z

Tippt man `^Z` in eine Shell ein, während ein Job läuft, wird dieser Job angehalten. Er stoppt ganz einfach, sein Zustand wird aber vermerkt.

```
→ sleep 10          Wartet 10 Sekunden.
^Z
[1]+  Stopped                 sleep 10
→
```

Jetzt können Sie `bg` eingeben, um den Befehl in den Hintergrund zu schieben, oder `fg`, um ihn im Vordergrund weiterlaufen zu lassen. Sie können ihn auch angehalten lassen und andere Befehle ausführen.

suspend

Der integrierte Befehl `suspend` hält die aktuelle Shell an, falls das möglich ist, so als hätten Sie `^Z` in die Shell selbst eingetragen. Wenn Sie z. B. mit dem Befehl `sudo` eine Administratoren-Shell geöffnet

haben und zu Ihrer ursprünglichen Shell zurückkehren wollen, probieren Sie Folgendes:

```
→ whoami
smith
→ sudo bash
Password: *******
# whoami
root
# suspend
[1]+  Stopped                 sudo bash
→ whoami
smith
```

bg

```
bg [%jobnummer]
```

Der eingebaute bg-Befehl sendet einen angehaltenen Job zur Ausführung in den Hintergrund. Ohne Argumente nimmt bg den zuletzt angehaltenen Job. Um einen bestimmten Job festzulegen (vom jobs-Befehl gezeigt), geben Sie die Jobnummer an, der Sie ein Prozentzeichen voranstellen:

```
→ bg %2
```

Manche Arten von interaktiven Jobs können nicht im Hintergrund bleiben – etwa wenn sie auf eine Eingabe warten. Falls Sie es versuchen, hält die Shell den Job an und gibt Folgendes aus:

```
[2]+  Stopped                 Kommandozeile hier
```

Sie können den Job nun wieder aufnehmen (mit fg) und fortfahren.

fg

```
fg [%jobnummer]
```

Der eingebaute fg-Befehl bringt einen angehaltenen oder im Hintergrund laufenden Job wieder in den Vordergrund. Ohne Argumente wählt er einen Job aus, und zwar üblicherweise den zuletzt angehaltenen oder in den Hintergrund verschobenen. Um einen bestimm-

ten Job festzulegen (wie vom `jobs`-Befehl gezeigt), geben Sie die Jobnummer an, der Sie ein Prozentzeichen voranstellen:

```
→ fg %2
```

Mehrere Shells auf einmal laufen lassen

Mit der Jobkontrolle können Sie mehrere Befehle gleichzeitig nutzen, aber es kann immer nur einer im Vordergrund laufen. Sie können sogar mehrere Shells auf einmal laufen lassen, die jede einen Befehl im Vordergrund und eine beliebige Zahl von Befehlen im Hintergrund ausführt.

Läuft auf Ihrem Linux-Computer ein Fenstersystem wie KDE oder GNOME, können Sie problemlos mehrere Shells parallel laufen lassen, indem Sie einfach mehrere Shell-Fenster öffnen (siehe »Eine Shell ausführen« auf Seite 20). Zudem können manche Shell-Fenster-Programme, wie zum Beispiel `konsole` bei der KDE, mehrere Tabs in einem Fenster öffnen, in denen jeweils eine Shell läuft.

Auch ohne eine grafische Benutzeroberfläche – zum Beispiel über eine SSH-Verbindung – können Sie mehrere Shells gleichzeitig einsetzen. Das Programm `screen` nutzt ein normales ASCII-Terminal, um mehrere Fenster zu simulieren, in denen jeweils eine Shell läuft. Mit bestimmten Tastenkombinationen können Sie zwischen diesen Fenstern wechseln. (Ein ähnliches Programm ist `tmux`.) Um eine Session mit `screen` zu starten, verwenden Sie einfach:

```
→ screen
```

Sie bekommen eventuell ein paar einleitende Nachrichten zu sehen, dann sollte Ihr normaler Shell-Prompt erscheinen. Es sieht aus, als sei nichts passiert, aber es läuft nun eine neue Shell innerhalb eines virtuellen »Fensters«. Das `screen`-Programm bietet zehn solcher Fenster an, die mit 0 bis 9 bezeichnet sind.

Geben Sie einen einfachen Befehl wie `ls` ein und drücken Sie dann `^A^C` (Strg-A, Strg-C) Der Bildschirm wird sich leeren und Ihnen einen neuen Shell-Prompt präsentieren. Sie befinden sich jetzt in einem zweiten, unabhängigen »Fenster«. Führen Sie einen anderen Befehl aus (zum Beispiel `df`) und drücken Sie dann `^A^A`, um zum

ersten Fenster zurückzukehren, in dem Sie wieder die Ausgabe von `ls` sehen. Mit einer weiteren Eingabe von `^A^A` wechseln Sie erneut zum zweiten Fenster zurück. Die wichtigsten Tastenkürzel für `screen` sind hier aufgeführt (schauen Sie auch auf die Manpage oder drücken Sie `^A?` für eine direkte Hilfe):

`^A?`	Hilfe: Zeigt alle Tastenkürzel an.
`^A^C`	Erzeugt ein Fenster.
`^A0`, `^A1` ... `^A9`	Wechselt zu den Fenstern 0 bis 9.
`^A'`	Fragt nach einer Fensternummer (0 bis 9) und wechselt dann dorthin.
`^A^N`	Wechselt zum (numerisch) nächsten Fenster.
`^A^P`	Wechselt zum (numerisch) vorigen Fenster.
`^A^A`	Wechselt zum zuletzt am häufigsten genutzten Fenster (wechselt damit zwischen zwei Fenstern).
`^A^W`	Gilt alle Fenster aus.
`^AN`	Gibt die aktuelle Fensternummer aus (beachten Sie, dass es sich hier um ein großes `N` handelt).
`^Aa`	Schickt ein echtes `^A` an Ihre Shell, das von `screen` ignoriert wird. In bash bewegt `^A` normalerweise den Cursor an den Anfang der Befehlszeile. (Beachten Sie, dass das zweite `a` kleingeschrieben ist.)
`^D`	Beendet die aktuelle Shell. Das ist das normale »End of File«-Tastenkürzel, das in »Eine Shell beenden« auf Seite 52 beschrieben ist und eine jede Shell schließt.
`^A\`	Schließt alle Fenster und beendet `screen`.

Nutzen Sie einen Texteditor in einem `screen`-Fenster, sollten Sie darauf achten, dass `screen` all Ihre `^A`-Tastendrücke abfängt, auch wenn sie eigentlich zum Bearbeiten gedacht sind. Schicken Sie daher ein `^Aa`, um Ihrer Anwendung ein `^A` zukommen zu lassen.

Einen laufenden Befehl beenden

Falls Sie von der Shell, die im Vordergrund läuft, einen Befehl gestartet haben, den Sie auf der Stelle beenden wollen, tippen Sie `^C`. Die Shell erkennt die Bedeutung von `^C` als: »Terminiere sofort den aktuellen Vordergrundbefehl.« Falls Sie also eine sehr lange

Datei anzeigen (z. B. mit dem cat-Befehl) und sie das stoppen wollen, tippen Sie ^C:

```
→ cat grossedatei
Lorem ipsum dolor sit amet, consectetur adipiscing
odio. Praesent libero. Sed cursus ante dapibus diam.
quis sem at nibh elementum blah blah blah ^C
→
```

Um ein Programm zu beenden, das im Hintergrund läuft, können Sie es mit fg in den Vordergrund holen und dann ^C eingeben:

```
→ sleep 50 &
[1] 12752
→ jobs
[1]-  Running     sleep 50 &
→ fg %1
sleep 50
^C
→
```

Alternativ benutzen Sie den Befehl kill (siehe »Prozesse steuern« auf Seite 147).

Das Eintippen von ^C ist keine freundliche Methode, um ein Programm zu beenden. Sofern das Programm seine eigene Methode zum Beenden mitbringt, sollten Sie diese nach Möglichkeit nutzen; Näheres erfahren Sie im Kasten.

Das Beenden überleben

Das Beenden eines Vordergrundprogramms mit ^C könnte Ihre Shell in einen seltsamen Zustand versetzen, möglicherweise reagiert sie nicht oder zeigt nicht mehr das an, was Sie eintippen. Das liegt daran, dass das beendete Programm keine Möglichkeit hatte, hinter sich aufzuräumen. Falls das bei Ihnen passiert:

1. Drücken Sie ^J, um einen Shell-Prompt zu erhalten. Dies erzeugt dasselbe Zeichen wie die Enter-Taste (ein Newline), funktioniert aber auch dann, wenn Enter es nicht mehr tut.
2. Tippen Sie den Shell-Befehl reset ein (auch wenn die Buchstaben beim Tippen nicht angezeigt werden) und drücken Sie ^J erneut, um diesen Befehl auszuführen. Das sollte die Shell wieder in den Normalzustand zurückbringen.

^C funktioniert nur mit Shells. Es wird wahrscheinlich in einer Anwendung, die kein Shell-Fenster ist, keine Wirkung zeigen. Darüber hinaus sind manche Befehlszeilenprogramme so geschrieben, dass sie ^C »abfangen« und ignorieren: Ein Beispiel dafür ist der Texteditor emacs.

Eine Shell beenden

Um eine Shell zu beenden, führen Sie entweder den `exit`-Befehl aus oder tippen ^D.[8]

→ **`exit`**

Das Shell-Verhalten anpassen

Um alle Ihre Shells so zu konfigurieren, dass sie auf eine bestimmte Weise funktionieren, bearbeiten Sie die Dateien *.bash_profile* und *.bashrc* in Ihrem Home-Verzeichnis. Diese Dateien werden jedes Mal ausgeführt, wenn Sie sich anmelden (*~/.bash_profile*) oder eine Shell öffnen (*~/.bashrc*). Sie können Variablen und Aliase setzen, Programme ausführen, Ihr Horoskop ausgeben oder was auch immer.

Diese beiden Dateien sind Beispiele für **Shell-Skripte**: ausführbare Dateien, die Shell-Befehle enthalten. Wir behandeln diese Eigenschaft näher in »Mit Shell-Skripten programmieren« auf Seite 231.

Damit beschließen wir unseren Überblick über Linux und die Shell. Wir wenden uns nun den Linux-Befehlen zu und beschreiben die nützlichsten Befehle zum Arbeiten mit Dateien, Prozessen, Benutzern, Netzwerken, Multimedia und mehr.

8 Strg-D sendet ein »Dateiende«-Signal an jedes Programm, das von der Standardeingabe liest. In diesem Fall ist das Programm die Shell selbst, die sich dann beendet.

Grundlegende Dateioperationen

`ls`	Listet Dateien in einem Verzeichnis auf.
`cp`	Kopiert eine Datei.
`mv`	Benennt eine Datei um (»verschiebt« die Datei).
`rm`	Löscht (»entfernt«) eine Datei.
`ln`	Erzeugt Links (alternative Namen) für eine Datei.

Eines der ersten Dinge, die Sie auf einem Linux-System tun werden, ist das Manipulieren von Dateien: Kopieren, Umbenennen, Löschen usw.

---

ls stdin **stdout** -datei **--opt** **--help** **--version**

`ls [`*optionen*`] [`*dateien*`]`

Der `ls`-Befehl listet die Attribute von Dateien und Verzeichnissen auf. Sie können die Dateien im aktuellen Verzeichnis auflisten:

→ **`ls`**

in bestimmten Verzeichnissen:

→ **`ls verz1 verz2 verz3`**

oder einzeln:

→ **`ls datei1 datei2 datei3`**

Die wichtigsten Optionen sind `-a`, `-l` und `-d`. Standardmäßig verbirgt `ls` Dateien, deren Namen mit einem Punkt beginnen, wie im Kasten »Punktdateien« auf Seite 35 erläutert wurde. Die Option `-a` zeigt alle Dateien an.

```
→ ls
meinedatei1   meinedatei2
→ ls -a
.verborgene_datei   meinedatei1   meinedatei2
```

Die Option -l erzeugt eine lange Liste:

```
→ ls -l meinedatei1
-rw-r--r--   1 smith users   149 Oct 28  2015 meinedatei1
```

Von links nach rechts enthält sie folgende Informationen: Dateiberechtigungen (`-rw-r--r--`), Anzahl der Hard Links (`1`), Besitzer (`smith`), Gruppe (`users`), Größe (`149` Byte), Datum der letzten Änderung (`Oct 28 2015`) und den Namen. Mehr Informationen über die Dateiberechtigungen finden Sie in »Schutz für Dateien« auf Seite 31.

Die Option `-d` listet Informationen über das Verzeichnis selbst auf, anstatt in das Verzeichnis herunterzusteigen, um die Liste mit dessen Dateien anzuzeigen.

```
→ ls -ld meinverz1
drwxr-xr-x   1 smith users   4096 Oct 29  2015 meinverz1
```

Nützliche Optionen

`-a`	Listet alle Dateien auf, auch die, deren Namen mit einem Punkt beginnen.
`-l`	Lange Liste, einschließlich der Dateiattribute. Fügen Sie die Option `-h` hinzu (von Menschen lesbar), um die Dateigrößen in Kilobyte, Megabyte und Gigabyte auszugeben anstatt in Byte.
`-G`	Lange Liste, aber ohne Gruppenangabe.
`-F`	Verziert bestimmte Dateinamen mit bedeutungsvollen Symbolen, die deren Typ anzeigen. Hängt »/« an Verzeichnisse, »*« an ausführbare Dateien, »@« an symbolische Links, »|« an benannte Pipes und »=« an Sockets an. Diese Indikatoren sind rein informativ und nicht Teil der Dateinamen!
`-S`	Sortiert die Dateien anhand ihrer Größe.
`-t`	Sortiert die Dateien anhand ihres letzten Bearbeitungszeitpunkts.
`-r`	Dreht die Sortierreihenfolge um.
`-R`	Listet beim Auflisten eines Verzeichnisses dessen Inhalt rekursiv auf.
`-d`	Beim Auflisten eines Verzeichnisses wird nicht dessen Inhalt, sondern nur das Verzeichnis selbst angeführt.

cp

stdin stdout -datei --opt --help --version

```
cp [optionen] dateien (datei | verzeichnis)
```

Der cp-Befehl kopiert normalerweise eine Datei:

→ **cp einedatei anderedatei**

oder er kopiert mehrere Dateien in ein Verzeichnis:

→ **cp datei1 datei2 datei3 datei4** *zielverzeichnis*

Mithilfe der Optionen -a oder -r können Sie Verzeichnisse auch rekursiv kopieren.

Nützliche Optionen

-p	Kopiert nicht nur den Dateiinhalt, sondern auch die Dateiberechtigungen, Zeitstempel und, falls Sie die passenden Rechte dazu haben, ihren Eigentümer und die Gruppe. (Normalerweise gehören die Kopien Ihnen, haben den Zeitstempel der aktuellen Zeit, und die Berechtigungen werden gesetzt, indem Ihr umask auf die ursprünglichen Berechtigungen angewandt wird.)
-a	Kopiert eine Verzeichnishierarchie rekursiv, wobei alle Dateiattribute und -links beibehalten werden.
-r	Kopiert eine Verzeichnishierarchie rekursiv. Diese Option behält die Attribute der Dateien, wie Berechtigungen und Zeitstempel, nicht bei. Symbolische Links dagegen werden bewahrt.
-i	Interaktiver Modus. Fragt, bevor Zieldateien überschrieben werden.
-f	Erzwingt die Kopie. Falls eine Zieldatei existiert, wird diese bedingungslos überschrieben.

mv

stdin stdout -datei --opt --help --version

```
mv [optionen] quelle ziel
```

Der mv-Befehl (move) kann eine Datei umbenennen:

→ **mv einedatei anderedatei**

oder Dateien und Verzeichnisse in ein Zielverzeichnis verschieben:

→ **mv datei1 datei2 verz3 verz4** *zielverzeichnis*

Nützliche Optionen

`-i` Interaktiver Modus. Fragt vor dem Überschreiben der Zieldateien.

`-f` Erzwingt das Verschieben. Falls eine Zieldatei existiert, wird diese bedingungslos überschrieben.

---

rm

stdin stdout -datei --opt --help --version

```
rm [optionen] dateien | verzeichnisse
```

Der `rm`-Befehl (remove) kann Dateien löschen:

```
→ rm datei1 datei2 datei3
```

oder rekursiv Verzeichnisse löschen:

```
→ rm -r verz1 verz2
```

Nützliche Optionen

`-i` Interaktiver Modus. Fragt vor dem Löschen der einzelnen Dateien.

`-f` Erzwingt das Löschen, ignoriert dabei alle Fehler- und Warnmeldungen.

`-r` Entfernt rekursiv ein Verzeichnis und seinen Inhalt. Benutzen Sie das mit Vorsicht, vor allem in Kombination mit der Option `-f`, da dies alle Ihre Dateien löschen könnte.

---

ln

stdin stdout -datei --opt --help --version

```
ln [optionen] quelle ziel
```

Ein **Link** ist eine Referenz auf eine andere Datei, die durch den Befehl `ln` erzeugt wurde. Links geben derselben Datei mehrere Namen, wodurch es ihr möglich wird, an zwei (oder mehr) Orten gleichzeitig zu existieren.

Es gibt zwei Arten von Links. Ein **symbolischer Link** (auch **Symlink** oder **Softlink**) verweist anhand seines Pfads auf eine andere Datei, fast wie ein Windows- oder Mac OS X-Alias. Um einen symbolischen Link anzulegen, nutzen Sie die Option `-s`:

```
→ ln -s meinedatei meinsoftlink
```