HOWTO zur Shell-Kommando-Kombination

(C) 2006-2024 T.Birnthaler/H.Gottschalk <howtos(at)ostc.de>
OSTC Open Source Training and Consulting GmbH
http://www.ostc.de

$Id: shell-command-combination-HOWTO.txt,v 1.17 2025/02/18 10:08:31 tsbirn Exp $

Dieses Dokument beschreibt die verschiedenen Verfahren zur Kombination von Kommandos in der Shell.

INHALTSVERZEICHNIS

0) Übersicht
1) Kommandos nacheinander ausführen
2) Hintergrund-Kommandos
3) Pipe
4) Kommando-Substitution
5) UND-Verknüpfung (IF-AND)
6) ODER-Verknüpfung (IF-OR)
7) Subshell
8) Gruppierung/Codeblock
9) Temporäre Datei

Shells (sh, bash, ksh, zsh, csh, tcsh, ...) erlauben nur EIN Kommando pro Zeile, jedes Kommando muss durch einen Zeilenumbruch abgeschlossen werden (oder durch ";"). Allerdings gibt es eine Vielzahl von Kommando-Kombinationen mit Hilfe bestimmter Sonderzeichen, bei denen auch zwei (oder mehr) Kommandos in einer Zeile erlaubt sind.

0) Übersicht

Die Kommando-Kombinationen lassen sich folgendermaßen charakterisieren:

* Parallel: Laufen parallel oder seriell/sequenziell ab
* Abhängig: Laufen abhängig (synchronisiert) oder unab. voneinander ab
* Gemeinsam: Gemeinsam umlenkbar/in Hintergrund stellbar oder nicht
* Subshell: Zur Ausführung zusätzlich gestartet oder nicht
* Exit-Status: Abfragbar oder nicht
* Shell-Variablen: Gemeinsam oder nicht (nur Shell-Kommandos)
* Umgebungs-Variablen (Environment-Var.): Gemeinsam oder nicht (Environment) (an Kindprozess immer VereRBT, nicht zurückgegeben)

Kombination	Parallel	Abhängig	Gem.	Subsh	Exit	ShVar	EnvVar
NEWLINE ;	NEIN	NEIN	NEIN	NEIN	JA	JA	JA
&	JA	NEIN	NEIN	JA	NEIN	NEIN	VRBT
|	JA	JA	NEIN	JA	JA	NEIN	VRBT
`...` $(...)	NEIN	NEIN	NEIN	JA	NEIN	NEIN	VRBT
&& ||	NEIN	NEIN	NEIN	NEIN	JA	JA	JA
(...)	NEIN	NEIN	JA	JA	JA	NEIN	VRBT
{...}	NEIN	NEIN	JA	NEIN	JA	JA	JA
<(...) >(...)	NEIN	NEIN	NEIN	NEIN	NEIN	JA	JA

HINWEIS: Nach Pipe-Symbol "|", logischer Verknüpfung "&&" und "||" sowie nach "{" darf ein ZEILENUMBRUCH folgen. Ebenso darf innerhalb den Hochkommas "..." und '...' sowie `...` und den Klammern (...) und {...} beliebig Zeilenumbruch verwendet werden.

1) Kommandos nacheinander ausführen

Zuerst CMD1 ausführen und nach dessen Abschluß CMD2 ausführen,
die Kommandos sind NICHT miteinander verknüpft:

```
CMD1          oder        CMD1; CMD2
CMD2
```

Direkt nach jedem Kommando kann sein Exit-Status abgefragt werden:

```
CMD1          oder        CMD1; echo "$?"
echo "$?"
```

Da jede Abfrage ein neues Kommando darstellt, ist der Exit-Status anschließend
überschrieben. Soll er mehrfach verwendet werden, ist er zwischenzuspeichern:

```
ES="$?"   # Exit Status
```

2) Hintergrund-Kommandos

Kommandos GLEICHZEITIG (parallel) im Hintergrund ausführen, die Kommandos sind
NICHT miteinander verknüpft:

```
CMD1 &        oder        CMD1 & CMD2 &
CMD2 &
```

Der Exit-Status eines Hintergrund-Kommandos kann NICHT abgefragt werden. Die
Shell liefert sofort den Exit-Status "0" (OK) zurück, falls das Kommando
gestartet werden konnte.

Falls ein Kommando nicht gestartet werden konnte, liefert sie Exit-Status 126
(gefunden, aber nicht ausführbar) oder 127 (nicht gefunden) zurück.

3) Pipe

Kommandos GLEICHZEITIG (parallel) starten und die Standard-Ausgabe von Kommando
CMD1 an die Standard-Eingabe von Kommando CMD2 übergeben. Die beiden Prozesse
SYNCHRONISIEREN sich über den von der Pipe "|" bereitgestellten Puffer im
Speicher (etwa 4-16 KByte), indem Kommando CMD1 nur dorthin schreibt, falls
Platz im Puffer vorhanden ist und CMD2 nur daraus liest, falls Daten im Puffer
vorhanden sind:

```
CMD1 | CMD2       auch       CMD1 | CMD2 | ... | CMD_N
```

Der Exit-Status einer Pipeline ist der Exit-Status des LETZTEN Kommandos in der
Pipeline. Bei gesetzter Option -o pipefail der ist es der Exit-Status des ERSTEN
Kommandos, bei dem ein Fehler auftritt oder 0 falls bei keinem Kommando ein
Fehler auftritt. Falls ein Kommando in einer Pipeline abbricht, werden alle
Kommandos der Pipeline automatisch beendet.

4) Kommando-Substitution

Kommando CMD1 zuerst ausführen, seine Standard-Ausgabe in die Kommandozeile
von CMD2 einfügen und dann Kommando CMD1 aufrufen:

```
CMD2  `CMD1`    # Alte Form    (Bourne-Shell sh)
CMD2 $(CMD1)    # Moderne Form (Bash- und Korn-Shell ksh)
```

Nur der Exit-Status von CMD2 kann abgefragt werden. Um den Exit-Status von
CMD1 und CMD2 abzufragen, folgende Konstruktion benutzen:

```
RESULT="$(CMD1)"   # Ausgabe von CMD1 abfangen
```

```
  echo "$?"          # Exit-Status von CMD1
  CMD2 "$RESULT"     # Ausgabe von CMD1 an CMD2 übergeben
  echo "$?"          # Exit-Status von CMD2
```

5) UND-Verknüpfung (IF-AND)

Nur dann Kommando CMD2 ausführen, falls Kommando CMD1 erfolgreich ablief
(d.h. einen Exit-Status von 0 = OK ergab):

```
  CMD1 && CMD2    # Bsp: [ -e FILE ] && rm FILE
```

Der Exit-Status einer UND-Verknüpfung ist der Exit-Status des LETZTEN von links
nach rechts ausgeführten Kommandos. Beim ERSTEN Kommando mit Exit-Status
"ungleich 0" bricht die Verarbeitung ab.

6) ODER-Verknüpfung (IF-OR)

Nur dann Kommando CMD2 ausführen, falls Kommando CMD1 NICHT erfolgreich ablief
(d.h. einen Exit-Status "ungleich 0" = FEHLER ergab):

```
  CMD1 || CMD2    # Bsp: [ -n "$VAR" ] || VAR=Default
```

Der Exit-Status einer ODER-Verknüpfung ist der Exit-Status des LETZTEN von
links nach rechts ausgeführten Kommandos. Beim ERSTEN Kommando mit Exit-Status
"gleich 0" bricht die Verarbeitung ab.

HINWEIS: && und || NICHT mischen, da die Bedeutung derartiger Verknüpfungen
sehr schwer verständlich ist (obwohl sie klar festgelegt ist):

```
  CMD1 && CMD2 || CMD3   # CMD1 Fehler? -> CMD2+CMD3 nicht ausführen
                         # CMD1 Ok?     -> CMD2 Ok?     -> CMD3 nicht ausführen!
                         # CMD1 Ok?     -> CMD2 Fehler? -> CMD3 ausführen!
  CMD1 || CMD2 && CMD3   # CMD1 Ok?     -> CMD2+CMD3 nicht ausführen
                         # CMD1 Fehler? -> CMD2 Fehler? -> CMD3 nicht ausführen!
                         # CMD1 Fehler? -> CMD2 Ok?     -> CMD3 ausführen!
```

7) Subshell

Kommandos CMD1 und CMD2 in einer SUBSHELL hintereinander starten und die
Ausgaben der Kommandos zusammenfassen (können gemeinsam umgelenkt werden) oder
sie gemeinsam im Hintergrund ausführen:

```
  ( CMD1; CMD2 )               # Gemeinsam in Subshell starten
  ( CMD1; CMD2 ) > out 2> err  # Gemeinsam Ausgabe auf Datei umlenken
  ( CMD1; CMD2 ) | CMD3        # Gemeinsam Ausgabe in weiteres Kmdo pipen
  ( CMD1; CMD2 ) &             # Gemeinsam in Hintergrund schicken
```

Der Exit-Status stammt vom letzten in der Klammer ausgeführten Kommando.

ACHTUNG: In (...) veränderte Variable sind außerhalb nicht verändert.

8) Gruppierung/Codeblock

Kommandos CMD1 und CMD2 in der aktuellen Shell hintereinander starten und die
Ausgaben der Kommandos zusammenfassen (können gemeinsam umgelenkt werden) oder
sie gemeinsam im Hintergrund ausführen:

```
  { CMD1; CMD2; }               # Analog CMD1; CMD2; ohne {...}
  { CMD1; CMD2; } > out 2> err  # Gemeinsam Ausgabe umlenken (in akt. Shell)
  { CMD1; CMD2; } | CMD3        # Gemeinsam Ausgabe in weiteres Kmdo pipen
  { CMD1; CMD2; } &             # Im Hintergrund: Erzeugt trotzdem Subshell!
```

Der Exit-Status stammt vom letzten in der Klammer ausgeführten Kommando.

ACHTUNG: In {...} veränderte Variable sind außerhalb auch verändert.

ACHTUNG: Ein Strichpunkt ";" ist auch nach dem letzten Kommando notwendig,
ebenso sind die Leerzeichen nach "{" und vor "}" notwendig!

HINWEIS: Geschweifte Klammern "{" und "}" sind aus historischen Gründen
Shell-Schlüsselworte --> Werden nur als 1. Symbol einer Zeile oder nach ";"
erkannt und Leerzeichen davor/dahinter sind notwendig!

9) Temporäre Datei

Die Ergebnisausgabe eines Kommandos als temporäre Datei ablegen und einem
anderen Kommando in Form des Dateinamens übergeben:

```
  CMD2 <(CMD1)  # A) Ausgabe von CMD1 mit CMD2 lesen
  CMD1 >(CMD2)  # B) Ausgabe von CMD1 mit CMD2 lesen
```

A) Nur der Exit-Status von CMD2 kann abgefragt werden. Um den Exit-Status von
CMD1 und CMD2 abzufragen, folgende Konstruktion benutzen:

```
  RESULT="$(CMD1)"           # Ausgabe von CMD1 abfangen
  echo "$?"                  # Exit-Status von CMD1
  echo -n "$RESULT" | CMD2   # Ausgabe von CMD1 an CMD2 übergeben
  echo "$?"                  # Exit-Status von CMD2
```

B) Nur der Exit-Status von CMD1 kann abgefragt werden. Um den Exit-Status von
CMD1 und CMD2 abzufragen, folgende Konstruktion benutzen:

```
  RESULT="$(CMD1)"           # Ausgabe von CMD1 abfangen
  echo "$?"                  # Exit-Status von CMD1
  echo -n "$RESULT" | CMD2   # Ausgabe von CMD1 an CMD2 übergeben
  echo "$?"                  # Exit-Status von CMD2
```