

HOWTO zur Shell-Kommando-Kombination

(C) 2006-2017 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 1.15 2017/08/07 15:49:28 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
- 6) ODER-Verknüpfung
- 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 einen ";"). Allerdings gibt es eine Vielzahl von Kommando-Kombinationen mit Hilfe bestimmter Sonderzeichen, durch die auch zwei (oder mehr) Kommandos in einer Zeile erlaubt sind.

### 0) Übersicht

Die Kommando-Kombinationen lassen sich folgendermaßen charakterisieren:

- \* Kommandos: Laufen parallel oder sequentiell ab
- \* Kommandos: Laufen abhängig oder unabhängig voneinander ab
- \* Kommandos: 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: Gemeinsam oder nicht (Environment)  
(Environment-Var.) (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	NEIN
	JA	JA	NEIN	JA	JA	NEIN	NEIN
'...' \$(...)	NEIN	JA	NEIN	JA	JA	NEIN	NEIN
&&	NEIN	JA	NEIN	NEIN	JA	JA	JA
	NEIN	JA	NEIN	NEIN	JA	JA	JA
(...)	NEIN	NEIN	JA	JA	JA	NEIN	NEIN
{...}	NEIN	NEIN	JA	NEIN	JA	JA	JA
<(...)	NEIN	NEIN	--	JA	NEIN	NEIN	NEIN

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

### 1) Kommandos nacheinander ausführen

Erst 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 die Abfrage ein neues Kommando darstellt, ist der Exit-Status anschließend

überschrieben. Soll er mehrfach verwendet werden, ist er zwischenspeichern:

```
ES="$?"
```

## 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 immer Exit-Status 0 (OK) zurück, falls das Kommando gestartet werden konnte. Falls das Kommando nicht gestartet werden konnte, liefert sie Fehlerstatus 126 oder 127 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 "|" bereit gestellten Puffer im Speicher (etwa 2-16 KByte), indem Kommando CMD1 nur dorthin schreibt, wenn Platz im Puffer vorhanden ist und CMD2 nur daraus liest, wenn Daten im Puffer vorhanden sind:

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

Der Exit-Status der Pipeline ist der Exit-Status des LETZTEN Kommandos. Falls ein Kommando mit Fehler abbricht, werden alle Kommandos der Pipeline beendet.

## 4) Kommando-Substitution

Zuerst das Kommando CMD2 ausführen, seine Ausgabe auf der Standard-Ausgabe in die Kommandozeile von CMD1 einfügen und dann Kommando CMD1 aufrufen:

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

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

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

## 5) UND-Verknüpfung

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

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

Exit-Status ist der Exit-Status des LETZTEN von links nach rechts ausgeführten Kommandos (ein Exit-Status ungleich 0 bricht die Verarbeitung ab).

## 6) ODER-Verknüpfung

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

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

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!
```

Exit-Status ist der Exit-Status des LETZTEN von links nach rechts ausgeführten Kommandos (ein Exit-Status gleich 0 bricht die Verarbeitung ab).

## 7) Subshell

-----  
 Kommandos CMD1 und CMD2 in einer gemeinsamen SUBSHELL hintereinander starten, die Ausgaben können gemeinsam umgelenkt werden:

```
( CMD1; CMD2 )           # Gemeinsam in Subshell starten
( CMD1; CMD2 ) > out 2> err # Gemeinsam Ausgabe umlenken
( CMD1; CMD2 ) &         # Gemeinsam in Hintergrund schicken
```

Exit-Status ist der Exit-Status des letzten in der Klammer ausgeführten Kommandos.

## 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; } &         # Im Hintergrund: Erzeugt trotzdem eine Subshell
```

Exit-Status ist der Exit-Status des letzten in der Klammer ausgeführten Kommandos.

ACHTUNG: Ein Strichpunkt ";" ist auch nach dem letzten Kommando notwendig und die Leerzeichen nach "{" und vor "}" sind ebenfalls 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 eines Dateinamens übergeben:

```
CMD1 <(CMD2)
```

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

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