HOWTO zur Shell-Quotierung

(C) 2006-2017 T.Birnthaler/H.Gottschalk <howtos(at)ostc.de>
OSTC Open Source Training and Consulting GmbH
http://www.ostc.de

$Id: shell-quoting-HOWTO.txt,v 1.19 2020/02/24 06:36:18 tsbirn Exp $

Dieses Dokument beschreibt die verschiedenen Verfahren zur Quotierung von Shell-Sonderzeichen.

INHALTSVERZEICHNIS

---

1) Shell-Sonderzeichen

Die Shell kennt eine Reihe von Sonderzeichen, die nicht fÃ¼r sich selbst stehen, sondern von ihr speziell interpretiert werden und dann bestimmte Funktionen auslÃ¶sen (26 StÃ¼ck!):

*	?	[	]	=	$	<	>	|	&	\		
"	'	`	;	(	)	{	}	~	^	!	#	
			<Space>		<Tabulator>		<Return>					

2) Quotierung

Sollen diese Zeichen nicht ihre Sonderbedeutung haben, sondern als ganz normaler Text interpretiert werden, so sind sie vor der Shell zu "schÃ¼tzen", der Fachausdruck dafÃ¼r ist "quotieren" (zitieren). Die Shell kennt 3 verschiedene MÃ¶glichkeiten des SchÃ¼tzens von Zeichen, die fÃ¼r unterschiedliche AnwendungsfÃ¤lle benÃ¶tigt werden:

Typ	Bedeutung
'...'	ALLE Sonderzeichen in ... abschalten [tick, single quote]
"..."	ALLE Sz. BIS AUF $, $(...), `...`, \, ! abschalten [double quote]
\Z	GENAU EIN (folgendes) Sonderzeichen Z abschalten [backslash]

Die Quotierung per "..." wird auch als "schwach" (weak) bezeichnet, da sie nicht alle Sonderzeichen abschaltet. Die Quotierung per '...' und \ wird als "stark" (strong) bezeichnet, weil sie ALLE Sonderzeichen abschaltet.

Hier eine Ãœ-^\bersicht Ã¼ber die von den Quotierungszeichen geschÃ¼tzten (*) bzw. NICHT geschÃ¼tzten (-) Sonderzeichen (CR=Zeilenende, Hst=Kmdo-History, e=Ende, w=wÃ¶rtlich Ã¼bernommen, i=Ignoriert):

Typ	$VAR	Kmdo-Substit `...` $(...)	Hst !	Hst \	Glob *?[]	"	'	Whitespace (TAB/NL/SP)	C R	Rest
"..."	-	-	-	-	*	e	*	*	w	*
'...'	*	*	*	*	*	*	e	*	w	*
\Z	*	*	*	*	*	*	*	*	i	*

Zweck der beiden Quotierungs-Varianten "..." und '...' ist es, zu unterscheiden, ob man den WERT einer Variablen (oder die AUSGABE eines Kommandos) nach dem Einsetzen in eine Kommandozeile NOCHMALS von der Shell interpretieren lassen mÃ¶chte oder nicht. Bei "..." wird nochmal interpretiert, bei '...' nicht. Beispiel:

```
VAR="*"       # Variable VAR den Text "*" (STERN) zuweisen
echo $VAR     # => Alle Dateinamen des akt. Verz. (2x ausgewertet)
echo "$VAR"   # => *                              (1x ausgewertet)
echo '$VAR'   # => $VAR                           (0x ausgewertet)
echo \$VAR    # => $VAR                           (0x ausgewertet)
```

HINWEIS: Die Zeichen `...` (backtick, left quote) fÃ¼hren das dazwischen stehende Kommando aus und setzen das Ergebnis (die Standard-Ausgabe des Kommandos) an dieser Stelle ein:

```
  DATE=`date +%D`         # Aktuelles Datum zuweisen (yy/dd/mm)
  DATE=`date +%d.%m.%Y`   # Aktuelles Datum zuweisen (dd.mm.yyyy)
  TIME=`date +%T`         # Aktuelle Uhrzeit zuweisen (HH:MM:SS)

Sonderzeichen in der erzeugten Ausgabe werden automatisch per \ geschÃ¼tzt.
In der Bash ist statt `...` auch $(...) zur Kommando-Substitution verwendbar:

  DATE=$(date +%D)         # Aktuelles Datum zuweisen (yy/dd/mm)
  DATE=$(date +%d.%m.%Y)   # Aktuelles Datum zuweisen (dd.mm.yyyy)
  TIME=$(date +%T)         # Aktuelle Uhrzeit zuweisen (HH:MM:SS)


3) Beispiel
-----------
Hier ein Beispiel fÃ¼r die Kombination vieler Sonderzeichen in einem
Shell-Kommando (statt `...` kann in der bash auch $(...) zur
Kommando-Substitution verwendet werden):

  echo * $TERM `date` > xxx &               Kommando echo + 6 Argumente, alle
      ^^^     ^^    ^^^^   ^^               Sonderzeichen werden ausgewertet,
      +++-----++----++++---++---------------(insbesondere 6x das Leerzeichen)
                                            d.h. alle Dateinamen des akt. Verz.,
                                            der Wert der Variablen TERM und
                                            das aktuelle Datum landen in
                                            Datei "xxx" und Kommando "echo"
                                            wird im Hintergrund ausgefÃ¼hrt

  echo "* $TERM `date` > xxx &"             Kommando "echo" + 1 Argument,
      ^^  ^     ^    ^        ^             NUR $TERM und `date` ausgewertet
      ++--+-----+----+--------+-------------(und 1x das Leerzeichen)

  echo '* $TERM `date` > xxx &'             Kommando "echo" + 1 Argument,
      ^^                      ^             KEIN Sonderzeichen wird ausgewertet
      ++----------------------+-------------(auÃM-^_er 1x das Leerzeichen)

  echo \* \$TERM \`date\` \> xxx \&         Kommando "echo" + 6 Argumente
      ^  ^      ^        ^  ^   ^           KEIN Sonderzeichen wird ausgewertet
      +--+------+--------+--+---+-----------(auÃM-^_er 6x das Leerzeichen)

  echo \*\ \$TERM\ \`date\`\ \>\ xxx\ \&    Kommando "echo" + 1 Argument
      ^                                     KEIN Sonderzeichen wird ausgewertet
      +-------------------------------------(auÃM-^_er 1x das Leerzeichen)
                                            (analog 3. Befehlszeile)

  echo\ \*\ \$TERM\ \`date\`\ \>\ xxx\ \&   1 Kommando (1. Wort) => Fehler
                                            "bash: ...: command not found"

  'echo * $TERM `date` > xxx &'             (analog vorherige Befehlszeile)
```