HOWTO zur Shell-Quotierung

$Id: shell-quoting-HOWTO.txt,v 1.20 2025/02/18 10:08:31 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 für sich selbst stehen, sondern von ihr speziell interpretiert werden und dann bestimmte Funktionen auslösen (26 Stück!):

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

Die 3 WHITESPACE-Zeichen <Space>, <Tabulator> und <Newline> dienen in der Kommandozeile als Trennzeichen und sind ebenfalls als Shell-Sonderzeichen zu betrachten (auch wenn man sie nicht sieht).

2) Quotierung

Sollen die Shell-Sonderzeichen nicht ihre Sonderbedeutung haben, sondern als ganz normaler Text interpretiert werden, sind sie vor der Shell zu "schützen", der Fachausdruck dafür ist "quotieren" (zitieren).

Die Shell kennt 3 verschiedene Möglichkeiten des Schützens von Zeichen, die für unterschiedliche Anwendungsfälle benötigt werden:

```
  +-------+-------------+-------------------------------------------+
  | Typ   | Name        | Bedeutung                                 |
  +-------+-------------+-------------------------------------------+
  | '...' | single quote | ALLE Sonderzeichen in ... abschalten     |
  | "..." | double quote | ALLE S. BIS AUF $ $(...) `...` \ ! abschalten |
  | \Z    | backslash   | EIN (das folgende) Sonderzeichen Z abschalten |
  +-------+-------------+-------------------------------------------+
```

Verwandt damit sind folgende Sonderzeichen der Shell, sie aber eine andere Aufgabe haben:

```
  +--------+----------+-----------------------------------------------+
  | `CMD`  | back tick | Keine Quotierung, sondern Kommando-Substitution |
  | $(CMD) |           | Ersatz dafür (ähnlich zur Variablen-Syntax)   |
  +--------+----------+-----------------------------------------------+
```

Die Quotierung per "..." wird auch als "SCHWACH" (weak) bezeichnet, da sie NICHT alle Sonderzeichen abschaltet. Die Quotierung per '...' und \Z wird als "START" (strong) bezeichnet, da sie ALLE Sonderzeichen abschaltet.

--> Dieses unterschiedliche Verhalten ist ABSICHT!

Alle 3 Arten der Quotierung haben ihre Berechtigung, da sie

* genau zugeschnittene unterschiedliche Wirkung
* für eine Vielzahl unterschiedlicher Situationen haben
* und durch Kombination von 2 oder mehr Quotierungsarten auch
  Kommandos per SSH an andere Rechnern geschickt werden können.

Hier eine Übersicht über die von den einzelnen Quotierungszeichen geschützten (*) bzw. NICHT geschützten (-) Sonderzeichen (SP=Space, TAB=Tabulator, NL=Newline, Hst=Kmdo-History, Glob=Dateinamen-Globbing, E=Ende, W=wörtlich übernommen, I=ignoriert):

Typ	$VAR	Kmdo-Subst. `...` $(...)	Hst !	Hst \	Glob *?[]	Glob "	Glob '	Whitespace (SP/TAB/NL)	N L	Rest
"..."	-	-	-	-	*	E	*	*	W	*
'...'	*	*	*	*	*	*	E	*	W	*
\Z	*	*	*	*	*	*	*	*	I	*

Der Grund für den Unterschied zwischen den beiden Quotierungs-Varianten "..." und '...' ist es, zu unterscheiden, ob man den WERT einer Variablen (oder die AUSGABE eines Kommandos) nach dem Einsetzen in einer Kommandozeile ERNEUT von der Shell interpretieren lassen möchte oder nicht. Bei "..." wird erneut interpretiert, bei '...' nicht. Beispiel:

```
VAR=*         # Variable VAR den Text "*" (STERN) zuweisen (ok)
VAR="*"       # Variable VAR den Text "*" (STERN) zuweisen (besser)
VAR='*'       # Variable VAR den Text "*" (STERN) zuweisen (besser)
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) haben mit Quotierung nichts zu tun, sondern führen das dazwischen stehende Kommando aus und setzen das ERGEBNIS (die Standard-Ausgabe des Kommandos) an dieser Stelle ein. Dabei entstehende Sonderzeichen in der Ausgabe sind automatisch per \ geschützt:

```
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)
```

In der Bash ist statt `...` auch $(...) zur Kommando-Substitution verwendbar (leichter lesbar + ohne Quotierungs-Tricks verschachtelbar):

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

Neben dem Quotieren von Texten oder Dateinamen mit Sonderzeichen

```
VAR="Hallo welt"
echo "*** $VAR ***"
echo '###'
ls -l datei\ name.txt
```

ist Quotieren auch notwendig, um Sonderzeichen der Shell an Kommandos wie "grep", "find", "awk", ... "durchzureichen", damit erst diese sie für ihre Zwecke interpretieren:

```
find -name "*.pdf"                       # Dateinamen-Muster in "..."
grep '^a.*b$' datei.txt                  # Regulärer Ausdruck in '...'
```

```
  awk '/^$/ { print("hallo Leerzeile"); }'   # Awk-Skript in '...'
```

Durch die niedrige Anzahl an verfügbaren ASCII-Zeichen "überlappen" sich leider die Sonderzeichen der Regulären Ausdrücke mit denen der Shell. Auch hier ist grundsätzlich per Quotieren zu entscheiden, ob die Shell oder das Kommando diese Sonderzeichen sehen soll:

```
  find $HOME -regex 'abc.*xzy\.pdf'
  grep "echo '\*\*\*'" datei.txt
```

3) Beispiel

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

```
  echo * $TERM `date` TEXT > xxx &          Kommando echo + 7 Argumente, ALLE
      ^^^     ^^    ^^    ^^^   ^^          Sonderzeichen werden ausgewertet,
      +++-----++----++----+++---++----------(insbesondere 7x das Leerzeichen)
                                            d.h. alle Dateinamen des akt. Verz.,
                                            Wert der Variablen TERM, das akt.
                                            Datum und Text TEXT landen in
                                            Datei "xxx" und Kommando "echo"
                                            wird im Hintergrund ausgeführt

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

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

  echo \* \$TERM \`date\` TEXT \> xxx \&    Kommando "echo" + 7 Argumente
      ^  ^      ^        ^    ^  ^   ^      KEIN Sonderzeichen ausgewertet
      +--+------+--------+----+--+---+------(außer 7x das Leerzeichen)

  echo \*\ \$TERM\ \`date\`\ TEXT\ \>\ xxx\ \&    Kommando "echo" + 1 Argument
      ^                                           KEIN Sonderzeichen ausgewertet
      +-------------------------------------------(außer 1x das Leerzeichen)
                                                  (analog Variante mit '...')

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

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