HOWTO zum Test von Bedingungen in Shell-Skripten

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

$Id: shell-test-HOWTO.txt,v 1.9 2025/02/18 10:08:31 tsbirn Exp $

Dieses Dokument beschreibt die verschiedenen Möglichkeiten, die das Kommando "test" bzw. "[...]" bzw. "[[...]]" zum Bedingungs-Test in der Shell bietet.

INHALTSVERZEICHNIS

1) Das Kommando "test"
2) Mit "test" überprüfbare Bedingungen
2.1) Nur in der Bash vorhandene Bedingungen
2.2) Nur in [[...]] vorhandene Bedingungen

---

1) Das Kommando "test"

Das Kommando "test" überprüft eine Bedingung, die ihm in Form von Argumenten mitgegeben wird und gibt folgende Exit-Stati zurück:

0	Bedingung erfüllt ("wahr")
1	Bedingung NICHT erfüllt ("falsch")
2	Syntaxfehler

Die Syntax des "test"-Kommandos lautet (Kommando "[" ist Link auf "test"):

```
test COND ...     # Kommando-Form
[ COND ] ...      # Leerzeichen um eckige Klammern notwendig (analog "test")
[[ COND ]] ...    # Leerzeichen um eckige Klammern notwendig (modern, kann mehr)
```

Das Kommando "test" wird zusammen mit "if", "while", "&&" und "||" verwendet.

2) Mit "test" überprüfbare Bedingungen

Folgende Bedingungen COND sind möglich, die einzelnen Elemente einer Bedingung müssen mit Leerzeichen getrennt werden:

* Dateiexistenz- und Dateigröße-Vergleich:

```
-e FILE           # Datei FILE existiert [exists]
-s FILE           # Datei FILE existent und NICHT leer [size]
\! -s FILE        # Datei FILE existent und leer [not size]
```

* Dateityp-Vergleich (gleichzeitig Existenz-Test!):

```
-f FILE           # Datei FILE normale Datei [file]
-d FILE           # Datei FILE [directory]
-b FILE           # Datei FILE blockorientiert [block device]
-c FILE           # Datei FILE zeichenorientiert [character device]
-L FILE           # Datei FILE ein symbolischer Link [link]
-p FILE           # Datei FILE eine Named Pipe/FIFO [pipe]
-S FILE           # Datei FILE ein Socket [socket]
-t FILE           # Datei FILE ein Terminal [terminal]
```

* Dateirechte-Vergleich:

```
  -r FILE           # Datei FILE lesbar [readable]
  -w FILE           # Datei FILE schreibbar [writable]
  -x FILE           # Datei FILE ausführbar [executable]
  -g FILE           # Datei FILE hat Group-ID Recht gesetzt [group]
  -k FILE           # Datei FILE hat Sticky-Bit gesetzt [sticky]
  -u FILE           # Datei FILE hat User-ID Recht gesetzt [user]

* Text-Vergleich:

  "TEXT"            # TEXT NICHT leer (nicht mit !/-a/-o kombinierbar!)
  -n "TEXT"         # TEXT NICHT leer (mit !/-a/-o kombinierbar!) [nonzero]
  -z "TEXT"         # TEXT leer [zero]
  "TEXT1" = "TEXT2"  # TEXT1 und TEXT2 gleich
  "TEXT1" != "TEXT2" # TEXT1 und TEXT2 verschieden

* Numerischer Vergleich (nur ganze Zahlen, kein Text oder leer erlaubt):

  NUM1 -eq NUM2     # Zahl NUM1 gleich NUM2 [equal]
  NUM1 -ne NUM2     # Zahl NUM1 NICHT gleich NUM2 [not equal]
  NUM1 -le NUM2     # Zahl NUM1 kleiner gleich NUM2 [less equal]
  NUM1 -lt NUM2     # Zahl NUM1 kleiner NUM2 [less than]
  NUM1 -ge NUM2     # Zahl NUM1 größer gleich NUM2 [greater equal]
  NUM1 -gt NUM2     # Zahl NUM1 größer NUM2 [greater than]

* Logische Verknüpfung (Vorrang von oben nach unten):

  \( ... \)         # Klammerung (quotiert wg. Shell!)
  \! EXPR           # Negation von EXPR [not]
  EXPR1 -a EXPR2    # EXPR1 und EXPR2 [and]
  EXPR1 -o EXPR2    # EXPR1 oder EXPR2 [or]


2.1) Nur in der Bash vorhandene Bedingungen
-------------------------------------------
* Dateiexistenz-Vergleich:

  -a FILE           # Datei FILE existiert (analog "-e") [available]

* Dateityp-Vergleich (gleichzeitig Existenz-Test):

  -h FILE           # Datei FILE ein symbolischer Link (analog "-L")

* Datei-Besitzer/Besitzergruppe-Vergleich:

  -G FILE           # Datei FILE gehört effektiver Group-ID [Group]
  -O FILE           # Datei FILE gehört effektiver User-ID [Owner]

* Dateialter-Vergleich bzgl. Inode-Vergleich:

  -N FILE1          # Datei FILE1 seit letztem Lesen verändert [New]
  FILE1 -nt FILE2   # Datei FILE1 neuer als FILE2 [newer than]
                    # (oder FILE1 existiert und FILE2 existiert NICHT)
  FILE1 -ot FILE2   # Datei FILE1 älter als FILE2 [older than]
                    # (oder FILE2 existiert und FILE1 existiert NICHT)
  FILE1 -ef FILE2   # Dateien haben gleiche Device + Inode-Nummer [equal file]

* Text-Vergleich:

  "TEXT1" == "TEXT2" # TEXT1 und TEXT2 gleich (analog "=")
  "TEXT1" \< "TEXT2" # TEXT1 alphabetisch kleiner als TEXT2 (quotiert wg. Shell]
  "TEXT1" \> "TEXT2" # TEXT1 alphabetisch größer als TEXT2 (quotiert wg. Shell]

* Shell-Variable testen:

  -v VAR            # Variable VAR gesetzt/hat einen Wert [variable]
  \! -v VAR         # Variable VAR NICHT gesetzt/undefiniert [not variable]
```

```
  -R VAR            # Variable VAR gesetzt und ist eine Namensreferenz [reference]

* Shell-Option testen:

  -o OPT            # Shell-Option OPT gesetzt [option]
  \! -o OPT         # Shell-Option OPT NICHT gesetzt [not option]


2.2) Nur in [[...]] vorhandene Bedingungen
------------------------------------------

* Vergleich mit Regulärem Ausdruck:

  "TEXT" =~ REGEX    # Regulärer Ausdruck passt zu Text (KEIN "..." oder /.../ um
REGEX!)



HINWEIS: Folgende Unterschiede zu "test" bzw. "[...]" gibt es:

* Innendrin findet keine Wort-Zerlegung und Pfadnamen-Expansion statt, aber
  Tilde-Expansion, Parameter + Variablen-Substitution, Arithmetische Auswertung,
  Kommando-Substitution, Prozess-Substitution, Entfernung der Quotierung
  (d.h. automatisch quotiert in "...").

* "<" und ">" sortiert LEXIKOGRAFISCH gemäß locale-Einstellung
  (statt gemäß ASCII-Reihenfolge).

* Shell-Option "nocasematch" schaltet Berücksichtigung von GROSS/kleinschreibung ab.

* Operator "=" und "==" machen Dateinamen-Matching (Globbing)
  als ob die Shell-Option "extglob" aktiv wäre.

* Innendrin Quotierung der Shell-Sonderzeichen * < > ! ( ) ... nicht nötig.

* Innendrin statt "-a" und "-o" auch "&&" und "||" nutzbar.
```