Bedingungen formulieren

Das builtin-Kommando `test` formuliert eine Vielzahl von Bedinungen über Optionen, die in den nachfolgenden Tabellen aufgelistet werden. Diese werden vor allem für die bedingte Kommandoausführung sowie für die Kontrollstrukturen benötigt. Ist die Bedingung wahr, so liefert `test` einen `exit`-Wert von 0, ansonsten einen von 0 unterschiedlichen Wert. Sollten keine Argumente `test` übergeben werden, so liefert `test` immer einen Wert unterschiedlich von 0 (falsch). Für den Aufruf gibt es zwei Möglichkeiten:

```
test <ausdruck> oder [ ausdruck ]
```

Nach [und vor] muß mindestens ein Leer- oder Tabulatorzeichen angegeben werden.

test-Bedingungen auf Zeichenketten

Ausdruck	liefert wahr, wenn Zeichenkette
`zk`	nicht leer ist
`-z zk`	leer ist [0 Zeichen]
`-n zk`	nicht leer ist
`zk1 = zk2`	identisch sind
`zk1 !=` `zk2`	voneinander abweichen

test-Bedingungen auf Zahlen

Ausdruck	liefert wahr, wenn
`z1 -eq z2`	die Zahlen gleich sind
`z1 -ne z2`	die Zahlen ungleich sind
`z1 -gt z2`	`z1` größer `z2` ist
`z1 -ge z2`	`z1` größer gleich `z2` ist
`z1 -lt z2`	`z1` kleiner `z2` ist
`z1 -le z2`	`z1` kleiner gleich `z2` ist

test-Bedingungen auf Dateien (Auszug)

Ausdruck	liefert wahr, wenn dat
`-b dat`	`dat` eine blockspezif. Gerätedatei ist
`-c dat`	`dat` eine zeichenspezif. Gerätedatei ist
`-d dat`	`dat` ein Verzeichnis ist
`-e dat`	`dat` existiert
`-f dat`	`dat` eine einfache Datei ist
`-L dat`	`dat` ein symbolische Link ist
`-r dat`	`dat` gelesen werden darf
`-s dat`	`dat` mindestens 1 Byte lang ist
`-w dat`	`dat` verändert werden darf
`-x dat`	`dat` ausgeführt werden darf
`-O dat`	`dat` der effektiven UID gehört
`-G dat`	`dat` der effektiven GID gehört
`dat1 -nt dat2`	`dat1` neuer als `dat2` ist
`dat1 -ot dat2`	`dat1` älter als `dat2` ist

Kontrollstrukturen

`if`-Anweisung

```
if Bedingung1
 then
   then_kdoliste1
 [elif Bedingung2
  then
    elif_kdoliste2]
 :
 :
 [else
    else_kdoliste1]
fi
```

`case`-Anweisung

```
case wort in
   pattern1) kdoliste1;;
   pattern2) kdoliste2;;
   :
   :
   pattern*n*) kdoliste*n*;;
esac
```

`while`-Schleife

```
while kdoliste1
 do
   kdoliste2
done
```

`until`-Schleife

```
until kdoliste1
 do
   kdoliste2
done
```

`for`-Schleife

```
for laufvariable [in wort1  wortn]
 do
   kdoliste
 done
```

Funktionen

```
[function] funktionsname() {
   kdoliste;}
```