Kurzakte Shellprogrammierung

Spezielle Shellvariablen

$?	Rückgabewert des letzten Kommandos
$$	PID der aktuellen Shell
$!	PID des letzten Hintergrundprozesses
$0	Name des gerade ausgeführten Shellskripts
$#	Anzahl der übergebenen Parameter
$n	n-ter Parameter (für n<9)
${n}	n-ter Parameter (für n>9)
$*	alle Parameter als „$1 $2 $3“
$@	alle Parameter als „$1“ „$2“ „$3“

Arithmetische Substitution

echo $[1+1]	Bash Versionen <2
echo $((1+1))	Bash ab Version 2
echo `expr 1 + 1`	
let variable=1+1; echo $variable	

Operator	Bedeutung
+ -	Einstelliger Operator (Vorzeichen)
! ~	Logische und bitweise Negation
**	Exponentialfunktion
* / %	Multiplikation, Division und Modulo
+ -	Addition, Subtraktion
<< >>	Bitweise Links-/Rechtsverschiebung
<= >= < >	Vergleiche
== !=	Gleichheit und Ungleichheit
&	Bitweises UND
^	Bitweises Exclusive ODER
|	Bitweises ODER
&&	Logisches UND
||	Logisches ODER
expr ? expr : expr	Bedingte Zuweisung
=, *=, /=, %=, +=, -= <<=, >>=, &=, ^=, |=	Zuweisungen

Bedingungen formulieren

entweder:	test Ausdruck
oder:	[Ausdruck]

test-Bedingungen auf Dateien: wahr, wenn

-r dat	dat existiert und ist lesbar
-w dat	dat existiert und ist beschreibbar
-x dat	dat existiert und ist ausführbar
-d dat	dat existiert und ist ein Verzeichnis
-s dat	dat existiert und ist nicht leer
-b dat	dat existiert und ist ein blockorientiertes Gerät
-c dat	dat existiert und ist ein zeichenorientiertes G.
-g dat	dat existiert und das SGID-Bit ist gesetzt
-k dat	dat existiert und das Sticky-Bit ist gesetzt
-u dat	dat existiert und das SUID-Bit ist gesetzt
-p dat	dat existiert und ist ein Named Pipe
-e dat	dat existiert
-f dat	dat existiert und ist eine reguläre dat
-L dat	dat existiert und ist ein symbolischer Link
-S dat	dat existiert und ist ein Socket
-O dat	dat existiert und gehört der eff. UID
-G datname	dat existiert und gehört der eff. GID
dat1 -nt dat2	dat1 ist neuer als dat2 (newer than)
dat1 -ot dat2	dat1 ist älter als dat2 (older than)
dat1 -ef dat2	dat1 und dat2 benutzen die gleiche i-node (equal file)

test-Bedingungen auf Zeichenketten: wahr, wenn

zk	zk nicht leer ist
-z zk	zk eine Länge von Null hat.
-n zk	zk eine Länge von größer als Null hat.
zk1 = zk2	zk1 gleich zk2
zk1 != zk2	zk1 ungleich zk2
zk1 \> zk2	zk1 lexikografisch nach zk2
zk1 \< zk2	zk1 lexikografisch vor zk2

test-Bedingungen auf Zahlen: wahr, wenn

z1 -eq z2 z1 gleich z2 (equal)
z1 -ne z2 z1 ungleich z2 (not equal)
z1 -gt z2 z1 größer z2 (greater than)
z1 -ge z2 z1 größer oder gleich z2 (greater or equal)
z1 -lt z2 z1 kleiner z2 (less than)
z1 -le z2 z1 kleiner oder gleich z2 (less or equal)

Verneinung und Kombination von test-Bedingungen

!Ausdruck	Logische Verneinung
Ausdruck -a Ausdruck	Logisches UND
Ausdruck -o Ausdruck	Logisches ODER

if-Anweisung (Alternation)

Mit if-Anweisungen steuert man in Abhängigkeit eines Ausdruckes den Programmablauf:

```
if <Bedingung1>; then
        <Kommando(s)>
[elif <Bedingung2>; then]
        <Kommando(s)>
[else]
        <Kommando(s)>
fi
```

case-Anweisung (Alternation)

vermeidet Verschachtelungen bei if-Konstrukten

```
case "Variable" in
        <Muster1> ) <Kommando(s)> ;;
        <Muster2> ) <Kommando(s)> ;;
        ...
        <Mustern> ) <Kommando(s)> ;;
esac
```

while-Schleife (test-Schleife)

so lange ausgeführt, bis Bedingung nicht mehr erfüllt:

```
while <Bedingung>; do
        <Kommando(s)>
done
```

until-Schleife (test-Schleife)

die Bedingung wird negiert formuliert:

```
until <Bedingung>; do
        <Kommando(s)>
done
```

for-Schleife (Iteration)

Abarbeitung von Listenelementen in ihrer Reihenfolge

```
for Variable [in Liste]; do
        <Kommando(s)>
done
```

Zählschleife adäquat zu anderen Programmiersprachen

```
for ((i=1; $i<=10; i++)); do
        <Kommando(s)>
done
```

Funktionen

[function] funktionsname() { kdoliste; } [Para1 Para2]
declare –f funktionsname Ausgabe der Anweisungen

Ausführen von Shellskripten

in einer Subshell:

./script.sh	mit x-Recht, im Arbeitsverzeichnis
bash script.sh	kein x-Recht nötig
bash < script.sh	kein x-Recht nötig
cat script.sh | bash	kein x-Recht nötig

in der aktuellen Shell:

. script.sh	kein x-Recht nötig

Debugging

set -x (trace) -n (Syntax-Check, no-exec) –v (verbose)

Erstellt von Dr. Jörg Meese www.doc-systems.de