Rechnen in UNIX-Shell-Prozeduren

Günter Partosch, HRZ; LOGIN 2/93, Juli 1993

Inhalt:

Problem 1
Eine Lösung in der Bourne-Shell 1
Eine zweite Lösung in der BourneShell 3
Eine Lösung in der Korn-Shell 3
INTEGER-Arithmetik in der Korn-Shell 4
INTEGER-Variablen in der Korn-Shell 6
Literatur 6

Wer unter dem Betriebssystem UNIX Shell-Prozeduren zu schreiben hat, wird über kurz oder lang vor der Notwendigkeit stehen, beispielsweise den Wert einer INTEGER-Variablen um 1 erhöhen oder die Werte zweier Variablen miteinander multiplizieren und das Produkt einer dritten Variablen zuweisen zu müssen. Wenn Sie in einer gängigen höheren Programmiersprache arbeiten, scheint das kein größeres Problem zu sein. So schreiben Sie z.B. in Pascal einfach

```
zaehler := zaehler + 1     (1)
```

bzw.

```
flaeche := breite * laenge (2)
```

oder in C:

```
zaehler ++
```

bzw.

```
flaeche = breite * laenge
```

In anderen Programmiersprachen gibt es entsprechende Konstrukte.

Problem

In einer UNIX-Shell-Prozedur können Sie jedoch das Problem so einfach nicht lösen. Wenn Sie nämlich beispielsweise

```
zaehler=$zaehler+1
```

schreiben, erhalten Sie `3+1` als neuen Inhalt der Shell-Variablen `zaehler` (wenn 3 der alte Wert gewesen war). Es wird also einfach eine Verknüpfung der Zeichenketten „`3`“, „`+`“ und „`1`“ vorgenommen. Sie könnten jetzt auf den Gedanken kommen, das Ganze mit eingestreuten Leerzeichen zu schreiben. Das funktioniert aber auch nicht, und Sie werden eine entsprechende Fehlermeldung erhalten. Also, so geht's nicht!

Natürlich gibt es eine Lösung. Diese unterscheidet sich danach, ob Sie mit der Bourne- oder mit der Korn-Shell arbeiten. Die C-Shell wollen wir bei unseren Betrachtungen ganz außer acht lassen.

Eine Lösung in der Bourne-Shell

Um die beiden oben genannten Zuweisungen (1) und (2) in der Bourne-Shell zu realisieren, bietet sich der UNIX-Befehl `expr` an. Beim Aufruf dieses Befehls können reguläre, Zeichen-

ketten-, arithmetische oder Vergleichsausdrücke als Parameter übergeben werden. Diese werden dann von `expr` ausgewertet:

```
expr $zaehler + 1
```

bzw.

```
expr $breite \* $laenge
```

In ersten Fall wird `4` auf dem Bildschirm ausgegeben (wenn 3 der alte Wert der Variablen `zaehler` war) und im zweiten Fall `12` (wenn die Werte der Variablen `breite` und `laenge` 3 und 4 waren). In beiden Beispielen sind die Leerzeichen zwischen den Parametern unbedingt anzugeben. Werden sie weggelassen, werden jeweils drei Zeichenketten miteinander verknüpft und dann `expr` angeboten. Im zweiten Beispiel ist zusätzlich der Backslash vor dem Operator `*` unbedingt zu schreiben, damit er nicht von der Shell interpretiert und expandiert wird (nämlich zu einer Liste aller Dateinamen im aktuellen Verzeichnis). Das Programm `expr` „weiß“ um diesen Umstand, erkennt den Fehler und gibt eine entsprechende Fehlermeldung aus.

Natürlich ist das noch nicht die Lösung unseres Problems, denn „unglücklicherweise“ wird das Ergebnis der Auswertung normalerweise auf die Standardausgabe (das ist üblicherweise der Bildschirm) geschrieben. Durch die folgende Konstruktion lässt sich jedoch diese Unzulänglichkeit leicht beheben:

```
zaehler=`expr $zaehler + 1`        (3)
```

bzw.

```
flaeche=`expr $breite \* $laenge`
```

Durch die Klammerung des Ausdrucks rechts vom Gleichheitszeichen durch zwei Gravis-Zeichen wird das, was durch die Anweisung `expr` normalerweise auf dem Bildschirm ausgegeben wird, an der entsprechenden Stelle in der Zuweisung eingesetzt und der linksstehenden Variablen zugewiesen. Mit Hilfe der UNIX-Anweisungen

```
echo $zaehler
```

und

```
echo $laenge, $breite, $flaeche
```

können Sie überprüfen, daß die Ergebnisse korrekt sind.

Beim Aufruf des UNIX-Befehls `expr` können neben den Operatoren für reguläre, Zeichenketten- oder Vergleichsausdrücke einige arithmetische Operatoren angegeben werden:

`/` ganzzahlige Division

`%` Divisionsrest

`*` Multiplikation

`+` Addition

`-` Subtraktion

Klammern können ebenfalls angegeben werden, dann aber nur mit einem Backslash davor, also:

`\(` und `\)`

Eine zweite Lösung in der BourneShell

Die oben vorgestellten Lösungen erscheinen Ihnen wahrscheinlich nicht sehr elegant. Als Alternative und als eine gewisse Erleichterung bietet es sich an, mit dem UNIX-Kommando `bc` zu arbeiten.

`bc` ist ein Quasi-Tischrechner, der auf der Standardeingabe (Tastatur) zeilenweise arithmetische Ausdrücke erwartet und das Ergebnis wiederum zeilenweise auf der Standardausgabe (Bildschirm) ausgibt. `bc` arbeitet im Gegensatz zum verwandten Programm `dc` mit der gewohnten Infix-Schreibweise und verlangt keine Leerzeichen zwischen Operatoren und Operanden. Zusätzlich muss auch nicht der arithmetische Operator `*` „vor der Shell geschützt" werden. Entsprechendes gilt auch für die runden Klammern `(` und `)`.

Um `bc` in einer ähnlichen Weise wie `expr` in (3) zu verwenden, sind zwei Dinge zu tun:

1. Wie oben schon erwähnt, erwartet `bc` normalerweise die Eingabe von der Standardeingabe. Um `bc` einen arithmetischen Ausdruck als Eingabe zu übergeben, könnten Sie den Ausdruck auf der Tastatur eintippen oder aber auch eine Eingabedatei mit dem entsprechenden Inhalt bereitstellen und dann `bc` mittels des Umleitungszeichens `<` veranlassen, von dieser Datei statt von der Standardeingabe zu lesen. Da das Ganze für unser Problem zu umständlich ist, soll hier eine andere Lösung vorgestellt werden:

```
2.
3.  bc <<Ende
4.  $zaehler+1
5.  Ende
```

   Die Zeichenkombination `<<` (in UNIX-Terminologie „here-document") veranlasst `bc`, beginnend bei der nächsten Zeile, solange Eingabezeilen aus dem „Bauch der Prozedur" zu lesen, bis die Ende-Zeile (hier die Zeile mit der Zeichenkette „`Ende`") erreicht ist.

6. In der bisher vorgestellten Teillösung wird das Ergebnis lediglich auf die Standardausgabe geschrieben. Um es der Variablen `zaehler` zuzuweisen, benutzen wir den gleichen Trick wie in (3):

```
7.
8.  zaehler=`bc<<Ende    (4)
9.  $zaehler+1
10. Ende
11. `
```

   Vergessen Sie nicht die beiden Gravis-Zeichen (`` ` ``), die dafür sorgen, daß das Ergebnis des `bc`-Aufrufs nicht auf die Standardausgabe geschrieben wird sondern hinter dem Gleichheitszeichen eingesetzt wird!

Eine Lösung in der Korn-Shell

Im Prinzip ist das, was bisher als Lösung vorgestellt wurde, auch in der Korn-Shell einsetzbar. Es ist aber noch zu umständlich und undurchsichtig. In der Korn-Shell kann jedoch das Ganze bis zu einem gewissen Grad vereinfacht werden:

Statt (3) wie in der Bourne-Shell können Sie in der Korn-Shell auf die Klammerung durch Gravis-Zeichen verzichten und stattdessen

```
zaehler=$(expr $zaehler + 1)       (5)
```

schreiben. Diese Schreibweise bietet mehrere Vorteile: Der Ausdruck ist einfacher, es gibt weniger Probleme beim Zusammentreffen mehrerer Quotierungsmechanismen und eine Schachtelung ist möglich. Es gelten aber immer noch die einschränkenden Bedingungen für den Einsatz von `expr`.

Die Darstellung (5) kann in der Korn-Shell nochmals vereinfacht werden, indem Sie

```
zaehler=$((zaehler+1))      (6)
```

schreiben. Zusätzlich müssen Variablennamen nicht mehr mit einem vorangestellten Dollarzeichen aufgeführt werden, und es müssen auch keine Leerzeichen zwischen Operanden und Operatoren wie beispielsweise in (5) eingefügt werden.

Schließlich kann (6) in der Korn-Shell weiter vereinfacht werden zu

```
((zaehler=zaehler+1))       (7)
```

bzw.

```
((zaehler+=1))     (7´)
```

Die Darstellungen (6), (7) und (7´) benutzen die eingebaute Anweisung `let` und sind daher deutlich schneller als die Lösungen mit `expr`. Sie sind deshalb auch beim Arbeiten mit der Korn-Shell auf jeden Fall zu bevorzugen, zumal sie auch noch andere Vorteile bieten.

INTEGER-Arithmetik in der Korn-Shell

Mit Hilfe der Korn-Shell-Anweisung `let` können arithmetische Ausdrücke (und Zuweisungen) elegant und schnell ausgewertet werden:

```
let ausdruck
```

Beispiele für den Einsatz von `let` sind:

```
let zaehler=zaehler+1
```

bzw.

```
let flaeche=breite*laenge
```

Bemerkenswert ist dabei,

- dass keine zusätzlichen Leerzeichen zwischen Operanden und Operatoren eingestreut werden müssen,
- dass `*`, `(` und `)` nicht „vor der Shell geschützt“ werden müssen und
- dass Shell-Variablen auf der rechten Seite der Zuweisung ohne vorangestelltes Dollarzeichen aufgeführt werden dürfen.

Wenn Sie mit einem `let`-Aufruf mehrere Ausdrücke gleichzeitig auswerten lassen oder wenn Sie der besseren Lesbarkeit wegen Leerzeichen einstreuen wollen, sind die Ausdrücke jeweils in doppelte Anführungszeichen einzuschließen, wie beispielsweise in

```
let "z = z + 1" "f = b * l"
```

Für `let` existiert eine zweite Darstellung

```
((ausdruck))
```

die äquivalent zu

```
let "ausdruck"
```

ist. Die beiden letzten Darstellungen haben den schönen Nebeneffekt, daß einige Operatoren, die ansonsten Metazeichen der Korn-Shell sind (wie z.B. < oder >), nicht vorzeitig von der

Shell interpretiert werden. Im übrigen sind die folgenden Operatoren (mit der in C üblichen Bedeutung) bei `let` zulässig:

Operator	Bedeutung
`! ~ -`	logische Negation, bit-weise Negation, arithmetische Negation
`* / %`	Multiplikation, Division, Divisionsrest
`+ -`	Addition, Subtraktion
`<< >>`	Links-Shift, Rechts-Shift
`< <= >= > == !=`	Vergleiche
`&`	bit-weises UND
`^`	bit-weises exklusives ODER
`|`	bit-weises ODER
`&&`	bit-weises UND
`||`	bit-weises ODER
`= += -= *= /= %= &= |= ^= <<= >>=`	Zuweisungen

Die Anweisung `let` liefert einen Error-/Exit-Status, der in Bedingungen verwendet werden kann. Die folgende kleine Korn-Shell-Prozedur zur Berechnung der ersten zwölf Fibonacci-Zahlen verdeutlicht das:

```
[1] #!/bin/ksh
[2] zaehler=2
    an=1
    anplus1=1
[3] while ((zaehler <=12)) ; do
[4]     ((anminus1=an))
        ((an=anplus1))
        ((anplus1=an+anminus1))
[5]     echo $zaehler : $anplus1
[6]     ((zaehler+=1))
[7]     done
```

Die Zahlen in den eckigen Klammern sind nicht Bestandteil der Prozedur.

Interessant ist der Teil der Prozedur, in dem die Schleifenbedingung (Zeile [3]) abgeprüft wird: Ist die Bedingung `((zaehler<=12))` erfüllt, wird der Status auf 0 (entspricht „wahr“) gesetzt und damit die `while`-Schleife weiter durchlaufen. Ist das nicht der Fall, hat der Status einen Wert ungleich 0 und die Schleife wird nicht mehr durchlaufen.

Erläuterung (der übrigen Anweisungen):

[1] Diese Zeile sorgt dafür, daß diese Prozedur auf jeden Fall in der Korn-Shell ausgeführt wird.

[2] In dieser Zeile und in den beiden folgenden werden einige Vorbesetzungen vorgenommen.

[3] Alles zwischen Zeile [3] und [7] wird in einer Schleife ausgeführt.

[4] Die Zuweisungen a[n-1]←a[n], a[n]←a[n+1] und a[n+1]←a[n]+a[n-1] werden vorgenommen..

[5] Die Werte der Variablen `zaehler` und `anplus1` (an+1) werden ausgegeben.

[6] Der Wert der Zählervariablen `zaehler` wird um 1 erhöht.

INTEGER-Variablen in der Korn-Shell

In den bisher gezeigten Beispielen wurden Zeichenkettenvariablen verwendet, die INTEGER-Zahlen darstellten. Zusätzlich haben Sie in der Korn-Shell die Möglichkeit, mit Hilfe der Anweisung

```
integer variable
```

explizit eine `integer`-Variable *variable* zu vereinbaren. Diese Variable wird intern als binäre Zahl und nicht als Zeichenkette gespeichert. Damit werden auch alle Operationen, die mit Hilfe der Anweisung `let` an solchen Variablen vorgenommen werden, schneller.

Übrigens: Der Versuch, einer Variablen, die explizit durch die Anweisung `integer` deklariert wurde, einen Zeichenkettenwert zuzuweisen, wird mit einer Fehlermeldung quittiert.

Die `integer`-Variablen in der Korn-Shell bieten noch einige andere Möglichkeiten (z.B. das Arbeiten mit verschiedenen Zahlensystemen), die aber hier in diesem Beitrag nicht vorgestellt werden sollen.

Literatur

Beim Erstellen dieses Beitrag standen mir die folgenden Bücher über UNIX bzw. über die Korn-Shell zur Verfügung:

Bolsky, Morris I.; Korn, David G.: *Die Korn-Shell. Beschreibung und Referenzhandbuch zur Befehls- und Programmiersprache*; Hanser/Prentice-Hall International, München, Wien, London; 1991; ISBN 3-446-16188-0 und 0-13-515651-3

Herold, Helmut: UNIX-Shells. *Bourne-Shell, Korn-Shell, C-Shell*; Reihe: UNIX und seine Werkzeuge; Addison-Wesley, Bonn, München, Paris, Reading (Massachusetts), Menlo Park (California), New York, Don Mills (Ontario), Wokingham (England), Amsterdam, Milan, Sydney, Tokyo, Singapore, Madrid, San Juan, Seoul, Mexico City, Taipei (Taiwan); ISBN 389319-381-2

Kochan, Stephen; Wood, Patrick: *UNIX Shell Programmierung. Programmierumgebung und Tools der Bourne- und Korn-Shell*; te-wi-Verlag; 19912; München; ISBN 3-89362-084-2

Stapelberg, Stefan: *UNIX System V.4 für Einsteiger und Fortgeschrittene. Bourne-Shell, Korn-Shell, C-Shell, TCP/IP, UUCP, E-Mail & News, X Window System*; Addison-Wesley, Bonn, Paris, Reading (Massachusetts), Menlo Park (California), New York, Don Mills (Ontario), Wokingham (England), Amsterdam, Milan, Sydney, Tokyo, Singapore, Madrid, San Juan, Seoul, Mexico City, Taipei (Taiwan); 1993; ISBN 3-89319-433-9

Valley, John: *UNIX Desktop Guide to the Korn Shell. The Fast, Easy Refernce to the Korn Shell. Clear, concise coverage of the Korn and Bourne Shells. Productivity examples, illustrations, and tables. Complete reference to shell scripts, syntax, and UNIX commands*; Hayden Books, Carmel (Indiana); 1992; ISBN 0-672-48513-3