Arithmetik in der `tcsh`

- Variablen speichern Zeichenketten (also Strings/Wörter)
- `@` statt `set` $\rightsquigarrow$ Interpretation als arithmetische Ausdrücke (aus Ziffern, `(`, `)`, `+`, `-`, `*`, `/`, `%` bestehend)

  Beispiele:

```
@ var = (3 + 4) * 5   ⇝  var = 35
@ var++               ⇝  var = 36
@ modulo = $var % 5   ⇝  modulo = 1
@ modulo--            ⇝  modulo = 0
```

- Es gelten die üblichen Vorrangregeln.
- Operanden müssen von Operatoren durch *whitespace* voneinander getrennt sein.

Wiederholung von Kommandos

1. **repeat**-Anweisung: `repeat` *count Kommando*

   - *Kommando* wird genau *count* mal ausgeführt
   - *Kommando* darf nicht Alias, Pipeline oder Kommandoliste sein!

2. **for**-Schleife:

```
foreach Variable Wortliste
   Kommando
   Kommando
      ⋮
   Kommando
end
```

   - Wiederholung der Kommandos so oft, wie die *Wortliste* lang ist
   - Bei der k-ten Wiederholung nimmt die *Variable* den Wert des k-ten Wortes der *Wortliste* an.

3. **while**-Schleife:

```
while (Ausdruck)
   Kommando
   Kommando
      ⋮
   Kommando
end
```

- Wiederholung der Kommandos solange, bis der *Ausdruck* den Wert `0` annimmt
  - **arithmetischer** *Ausdruck*
  - **boolescher** *Ausdruck* mit Werten `1` für `true` oder `0` für `false`
    - durch Vergleich arithmetischer Ausdrücke mit `==`, `!=`, `<=`, `>=`, `<`, `>`
    - durch Vergleich zweier Zeichenketten mit `==`, `!=`
  - ein **Kommando** in `{ ... }` ergibt genau dann `1` (`true`), falls es fehlerfrei ausgeführt wurde (*Exit-Status 0*)

- **Datei-Tests:**

`-e` *datei*	1 gdw. *datei* existiert
`-f` *datei*	1 gdw. *datei* ist gewöhnliche Datei
`-d` *datei*	1 gdw. *datei* ist Verzeichnis
`-z` *datei*	1 gdw. *datei* hat Größe null
`-s` *datei*	1 gdw. *datei* hat nicht Größe null
`-o` *datei*	1 gdw. User ist Eigentümer von *datei*
`-r` *datei*	1 gdw. User hat Leserecht
`-w` *datei*	1 gdw. User hat Schreibrecht
`-x` *datei*	1 gdw. User hat Ausführungsrecht

- **logische Verknüpfungen:**

`&&`	und-Verknüpfung
`||`	oder-Verknüpfung
`!`	Negation

Bedingte Ausführung von Kommandos

1. **if**-Anweisung: `if` (*Ausdruck*) *Kommando*

   - *Kommando* wird genau dann ausgeführt, wenn der *Ausdruck* den Wert `1` (`true`) annimmt

   - *Kommando* darf nicht Alias, Pipeline oder Kommandoliste sein!

   - wichtige *Kommando*s:
     - `break` $\rightsquigarrow$ führt zum vorzeitigen (sofortigen) Verlassen der (direkt umgebenden) Schleife

       Geschachtelte Schleifen können verlassen werden, indem entsprechend viele `break`-Anweisungen aufgerufen werden.

     - `continue` $\rightsquigarrow$ führt zum Abbruch des aktuellen und Beginn des nächsten Schleifendurchlaufs

2. **if-then-else**-Anweisung:

```
if (Ausdruck) then
  Kommando
  Kommando
    ⋮
  Kommando
else
  Kommando
  Kommando
    ⋮
  Kommando
endif
```

- Die Kommandos nach `then` werden ausgeführt, falls der *Ausdruck* den Wert `1` (`true`) annimmt, sonst die Kommandos nach `else`.
- `else` und `endif` müssen die ersten Wörter einer Zeile sein.
- Der `else`-Teil ist optional.

Shell-Programme (Shell-Skripte)

- Textdatei, deren Zeilen Shell-Kommandos sind
- Ausführen eines Skripts bedeutet zeilenweises Ausführen der enthaltenen Kommandos von oben nach unten
- `goto` $label$ $\rightsquigarrow$ Fortsetzung mit einer Zeile, deren erstes Wort $label$: ist
- leere Zeilen und Zeileninhalte nach # werden ignoriert

Methoden zum Starten eines Kommandos/Skripts

Shellausführung `csh` *Skriptdatei*

Das System startet eine Subshell (`csh`), welche die *Skriptdatei* liest und ein Kommando nach dem anderen ausführt.

Start mit Dateiname *Skriptdatei*

Das System startet eine Subshell (Bourne-Shell), welche die *Skriptdatei* liest und die enthaltenen Kommandos ausführt.

Vorraussetzung ist, dass die *Skriptdatei* lesbar und ausführbar ist!

Soll das Skript in einer `tcsh` ausgeführt werden, so muss die erste Zeile im Skript lauten:

```
#!  /bin/tcsh -f
```

Start mit source `source` *Skriptdatei*

Die Ausführung erfolgt in der aufrufenden Shell. Damit sind auch alle lokalen Variablen für das Skript verfügbar.

Nach Beendigung der Ausführung wird in das aufrufende Programm zurückgekehrt.

Start mit exec `exec` *Skriptdatei*

wie `source`, aber nach Beendigung der Ausführung wird das aufrufende Programm beendet.

Nach `exec` können auch einzelne UNIX-Kommandos anstelle der *Skriptdatei* stehen.

Spezielle Variablen

`$argv[n]`	`$n`	n. Argument aus der Kommandozeile
`$#argv`	`$#`	Anzahl der Argumente aus der Kommandozeile
`$argv[$#argv]`		letztes Argument aus der Kommandozeile
`$argv[*]`	`$*`	alle Argumente aus der Kommandozeile
`$0`		Name des Programms, das diesen Wert abfragt
`$$`		Prozess-Nummer (`PID`) der ausführenden Shell
`$?`*`variable`*		liefert 1 falls *variable* definiert ist, 0 sonst
`$<`		eine Zeile der Standard-Eingabe `stdin`

Wie üblich, können die Variablennamen in geschweifte Klammern eingeschlossen werden, etwa um sie von nachfolgenden Zeichen zu trennen.