Kurs: BATCH fuer Einsteiger

http://kickme.to/tiger/

==========

Kurs: BATCH fuer Einsteiger im FIDO-Echo BATCH.GER 1994

Der Batch-Kurs fuer Einsteiger soll die Grundkenntnisse zur BAT-Programmierung unter MS-DOS vermitteln.

Die 20 Lektionen dieses Kurses sind von Februar bis Mai 1994 im BATCH.GER Echo erschienen und wurden anschliessend noch einmal ueberarbeitet.

Fragen, Hinweise, Anregungen oder Reklamationen bitte im BAT.GER Echo oder per NetMail...

Auf wiederbatch!

Horst Schaeffer (2:2480/13.75)

Dank an Rainer Heuwes und Harald Gerber fuer ihre Mitwirkung!

*** 15.05.1994

---------- Uebersicht ----------

***

Batch fuer Einsteiger

========== BAT Ablaeufe ========== Lektion #1 ==

Die Grundidee der Batch-Ablaeufe ("Stapelverarbeitung") ist die automatische Abarbeitung von Programmen und Befehlen, z.B. um nach dem Einschalten des Computers eine Reihe von residenten Programme zu installieren (AUTOEXEC.BAT).

DOS bietet dazu die Moeglichkeit, Befehle aus einer ASCII-Datei zeilenweise abzuarbeiten, so als wuerden sie nacheinander auf der Befehlsebene eingegeben. Eine solche Datei muss den Namenszusatz "BAT" haben, und ist damit - neben COM- und EXE-Dateien - eine weitere Variante der "ausfuehrbaren" Dateien, die als Befehl auf der DOS-Ebene aufgerufen werden koennen.

Damit ein BAT-Ablauf mehr ist, als nur eine einfache Folge von Programm-Aufrufen, gibt es besondere Anweisungen und Eigenschaften, die einfache Funktionen einer Programmiersprache bieten:

* Befehlsparameter (%1..)
* Bildschirm-Ausgaben (ECHO)
* Halt (PAUSE)
* Variable (SET, Umgebungsvariable)
* Bedingte Ausfuehrung (IF.. GOTO, Labels)
* Schleifen (FOR..)
* CALL weiterer BAT-Ablaeufe
* Ein-/Ausgabe-Umleitungen

Die BAT-Sprache ist allerdings von Microsoft auesserst spaerlich ausgestattet worden, und sie unterstuetzt vor allem keinerlei Bildschirmdialoge, etwa fuer Eingaben oder zur Auswahl durch den Benutzer. (Seit Ver. 6.0 gibt's allerdings CHOICE.)

Aus diesem Grunde sind schon viele nuetzliche Erweiterungen der BAT-Sprache entwickelt worden (BAT Enhancements), die aber meist nur isolierte Befehle bieten, ohne die BAT-Sprache selbst (d.h. die Programm-Logik) zu verbessern.

Das Problem liegt darin, dass der BAT-Prozessor von DOS fester Bestandteil von COMMAND.COM ist, also kein separates Programm, das man so einfach durch ein besseres ersetzen koennte. Das oft zitierte 4DOS ersetzt COMMAND.COM komplett und kann daher mit einem eigenen, erheblich erweiterten BAT-Prozessor aufwarten.

Soviel zu BAT-Erweiterungen. Hier in diesem Einsteiger-Kursus soll es nur um das reine MD-DOS gehen.

Bevor in den naechsten Lektionen die einzelnen BAT-Befehle behandelt werden, hier noch einige Basis-Informationen:

BAT-Datei, ASCII

Um BAT-Dateien zu schreiben, braucht man einen ASCII-Editor. Wer nichts besseres hat, kann EDIT von MS-DOS benutzen. Das alte EDLIN frueherer DOS-Versionen sollte man sich allerdings nicht zumuten.

Eine BAT-Datei darf nur ASCII-Zeilen mit einer Laenge von maximal 127 Zeichen enthalten. Das Zeilen-Ende wird durch CR+LF (ASCII 13,10) bestimmt, wobei das LF auch fehlen darf. Das Datei-Ende kann durch den EOF-Code (Control-Z, ASCII 26) markiert werden, was aber heute kaum noch ueblich ist.

Alle uebrigen ASCII-Zeichen (1..255) duerfen verwendet werden.

Allerdings koennen Sonderzeichen eine besondere Bedeutung haben! Allgemein gilt: Sonderzeichen, die auch in Dateinamen gueltig sind, werden wie Buchstaben behandelt. Sonstige Sonderzeichen sowie "%" bitte nur ganz bewusst und mit Absicht verwenden! (Naeheres bei den jeweiligen Befehlen.)

Leerzeilen werden ignoriert (wie ein CR auf der Befehlsebene), koennen also zur besseren Lesbarkeit eingebaut werden.

Abarbeitung durch DOS

Zur Kontrolle des Ablaufs legt COMMAND.COM beim Start im Arbeitsspeicher einen BAT-Steuerblock an, der am Ende wieder entfernt wird (64 Bytes oder auch etwas mehr). In diesem Steuerblock wird unter anderem ein Zeiger auf die naechste auszufuehrende Zeile der BAT-Datei verwaltet.

Fuer jede einzelne Befehlszeile des BAT-Ablaufs wird von DOS die BAT-Datei geoeffnet, gelesen und geschlossen. Grundsaetzlich kann also eine BAT-Datei auch veraendert werden, waehrend sie laeuft, was man allerdings mit Vorsicht geniessen sollte. Auf jeden Fall sollte man vermeiden, in einem BAT-Ablauf den Editor aufzurufen, um eben diese BAT-Datei zu aendern. Das bringt die Abarbeitung mit hoher Wahrscheinlichlkeit durcheinander!

Fehler in BAT-Ablauf, Abbruch

Bei ungueltigen Befehlen oder Syntax-Fehlern gibt DOS eine Meldung aus und setzt die Verarbeitung (wenn moeglich) mit der naechsten Zeile fort.

Da DOS nicht die Zeilen-Nummer des Fehlers angibt, muss man notfalls ECHO ON schalten, um den Ablauf zu verfolgen (s. Kapitel ECHO).

Eine laufende BAT-Datei laesst sich normalerweise mit Control-C oder Control-BREAK (Strg-Unterbr.) abbrechen. DOS erwartet dann noch eine Bestaetigung:

> Stapelverarbeitung abbrechen? (J/N)

Ob sich ein gerde ausgefuehrtes PROGRAMM abbrechen laesst ist natuerlich eine andere Frage.

Der ECHO-Befehl hat zwei verschiedene (!) Funktionen:

1. Echo-Status einstellen/ausgeben
2. Textzeile ausgeben

Echo-Status einstellen/ausgeben

ECHO [ON|OFF]

Der Echo-Status legt fest, ob die folgenden Anweisungszeilen der BAT-Datei jeweils vor der Ausfuehrung auf dem Bildschirm ausgegeben werden (ON) oder nicht (OFF). Ausgaben erfolgen immer mit dem (jeweils definierten) DOS-Prompt.

Normalerweise wird das ECHO in BAT-Dateien gleich am Anfang OFF geschaltet und hoechstens streckenweise ON geschaltet, damit man sieht, welche Befehle gerade ausgefuehrt werden. Auch zum Testen ist ECHO ON manchmal ganz nuetzlich, um Fehler zu lokalisieren.

Der ECHO-Befehl ohne weitere Angaben veranlasst DOS, den derzeitigen Status nach folgendem Muster auszugeben:

```
ECHO ist eingeschaltet
```

Nach dem Wort ECHO kann optional ein "=" gesetzt werden. Leerzeichen sind erlaubt. Beispiel:

```
ECHO =  off
ECHO =
```

Dies ist allerdings nicht (?) dokumentiert, und man sollte vielleicht darauf verzichten, wenn man kompatibel bleiben will.

Text ausgeben

Bei allen ECHO-Befehlen, die nicht den obigen Bedingungen entsprechen, wird der Rest der Zeile auf dem Bildschirm ausgegeben (genauer gesagt: zum Standard Output Device). Beispiel:

```
ECHO Dies ist eine ECHO-Zeile
```

Durch Leerzeichen laesst sich der Text einruecken:

```
ECHO      Dieser Text ist eingerueckt
```

Das erste Leerzeichen nach ECHO gilt jedoch als Trennzeichen und gehoert nicht zum Text. Alternativ kann ein Punkt verwendet werden, z.B:

```
ECHO.Fertig
ECHO.OFF                        (OFF ist hier Text!)
```

Auf diese Weise kann auch eine leere Zeile, also ein extra Zeilenvorschub, produziert werden (ECHO allein geht nicht, s. ECHO-Status!). Beispiel:

```
ECHO.
```

Der Punkt muss ohne Leerzeichen anschliessen, sonst ist er Text!

Bei MS-DOS koennen anstelle des Punktes auch andere Sonderzeichen verwendet werden, sofern sie nicht in Befehlen bzw. Dateinamen gueltig sind (,:;+/). Probiert's mal auf der Befehlszeile!

Ganz wichtig:

> Textzeilen, die per ECHO ausgegeben werden sollen, duerfen nicht die Zeichen "<", ">" oder "|" enthalten, da diese fuer Umleitungen/PIPES reserviert sind. Wer mal nicht dran denkt, wird sich ueber auesserst merkwuerdige Fehlermeldungen oder unerwartet auftauchende Dateien wundern.

ECHO-Ausgaben duerfen auch Steuerzeichen (ASCII 1..31) enthalten, sofern sie nicht die Abarbeitung sabotieren (wie z.B. CR oder EOF) Wenn der Editor es erlaubt, kann man z.B. jederzeit ein Control-G (BELL, ASCII 7) einbauen. Ausprobieren!

Klammeraffe

Waehrend das ECHO ON ist, kann es bei Bedarf fuer einzelne Zeilen abgeschaltet werden, indem ein Klammeraffe an den Anfang der Zeile gesetzt wird (ab MS-DOS 3.3). Spezielles Beispiel:

```
@ECHO OFF
```

Ohne den Klammeraffen wuede der Befehl selbst noch ge-echot, bevor er wirksam wird (!)

ECHO auf der Befehlsebene

Der ECHO-Befehl kann grundsaetzlich auch auf der Befehlsebene verwendet werden, was aber nur in Sonderfaellen sinnvoll ist.

Achtung: ECHO OFF bewirkt, dass kein DOS-Prompt mehr erscheint!! Mit anderen Worten: wenn der Prompt mal auf unerklaerliche Weise verschwunden ist: einfach ECHO ON versuchen.

Weitere Informationen

Interresant wird der ECHO-Befehl erst mit Umleitungen/PIPES. Ausserdem koennen mit Hilfe der ANSI-Sequenzen Farben und Cursor-Steuerungen (u.a.) ins Spiel gebracht werden.

Mehr dazu in spaeteren Lektionen.

==========

Batch fuer Einsteiger

========== ECHO Umleitung ========== Lektion #3 ==

Das Thema "Umleitungen" soll eigentlich erst spaeter behandelt
werden, aber die ECHO-Umleitung wird schon mal vorweg genommen,
weil sie recht unproblematisch ist.

Ein ECHO-Text kann sehr einfach mit einem ">"-Zeichen (Pfeil nach
rechts) in eine Datei oder an ein anderes Geraet - wie LPT1 -
umgeleitet werden. Beispiel:

```
ECHO irgendwas> TEST.DAT
ECHO irgendwas> LPT1
```

Genau wie auf dem Bildschirm, wird auch in diesen Faellen der
Text mit einem Zeilenvorschub (CR,LF) ausgegeben. Eine leere
Zeile (also nur ein Zeilenvorschub) kann so ausgegeben werden:

```
ECHO.> TEST.DAT
```

Die angegebene Datei wird immer kommentarlos neu angelegt, d.h.
eine evtl. vorhandene Datei wird geloescht!

Es ist aber auch moeglich, eine Zeile an eine vorhandene Datei
anzufuegen. Dazu werden einfach zwei Pfeile (">>") verwendet:

```
ECHO eine Zeile > TEST.DAT
ECHO noch eine Zeile >> TEST.DAT
```

Bei aelteren DOS-Versionen (-3.3) darf allerdings die bestehende
Datei keine Ende-Markierung (EOF, Control-Z) haben, sonst wird man
die zusaetzliche Zeile vergeblich suchen...

Uebrigens: eventuelle Leerzeichen vor dem Pfeil werden mit in die
Datei geschrieben!

Die ECHO-Umleitungen lassen sich gut auf der Befehlsebene testen.

==========

Batch fuer Einsteiger

========== PAUSE ========== Lektion #4 ==

Mit der PAUSE-Anweisung wird der BAT-Ablauf angehalten. DOS gibt
dazu standardmaessig folgende Aufforderung aus:

```
    Eine beliebige Taste druecken, um fortzusetzen
```

Ein Abbruch ist an dieser Stelle, wie ueblich, mit Control-C oder
Control-BREAK (Strg-Unterbr.) moeglich.

Soll mehr als nur die Standard-Meldung erscheinen, koennen vorher
geeignete ECHO-Anweisungen gemacht, werden, z.B.:

```
    ECHO Bitte Diskette wechseln!
    Pause
```

Nach dem PAUSE-Befehl werden weitere Angaben in der selben Zeile
ignoriert. Allerdings werden eventuelle Umleitungszeichen <|>
von DOS konsequent abgearbeitet, was selten Sinn mach - bis auf
diesen Fall:

```
PAUSE > NUL
```

Damit wird die Ausgabe der DOS-Aufforderung ins Nichts umgeleitet,
falls man etwas anderes formulieren moechte.

Hinweis:

Bis DOS Version 5.0 ist es nicht moeglich, die gedrueckte Taste
anschliessend abzufragen. Auch eine automatische Fortsetzung nach
einer gewissen Zeit ist nicht vorgesehen. Dafuer gibt es aber
kleine Utilities mit Namen wie ASK, INKEY, WAIT etc.. Ab DOS 6.0
wird das Hilfsprogramm CHOICE mitgeliefert (s. spaetere Lektion).

==========

Batch fuer Einsteiger

========== REM - Bemerkungen ========== Lektion #5 ==

REM dient zum Schreiben von Bemerkungszeilen.

Bitte auch hier keine Umleitungszeichen <|> verwenden, sofern
nicht wirklich beabsichtigt.

Alternativ koennen Bemerkungszeilen mit dem fuer Sprungmarken
vorgesehenen Doppelpunkt geschrieben werden, am besten mit Doppel-
Doppelpunkt (weitere Informationen dazu s. Kapitel LABEL):

```
::Das ist eine Bemerkung
```

Der Vorteil dieser Version besteht vor allem darin, dass dabei
bedenkenlos Umleitungszeichen verwendet werden, z.B.:

```
::Syntax: MOP <Quelldatei> <Zieldatei> [/M|/X]
```

Empfehlung daher: Vergesst REM fuer Bemerkungen.
Nur fuer einen speziellen Fall ist REM recht nuetzlich:
zum Anlegen einer leeren Datei per Umleitung,
z.B.:

```
REM > XXX.DAT
REM erzeuge eine leere > XXX.DAT
```

Da REM nichts ausgibt, wird auch nichts in die Datei umgeleitet.
Aber angelegt wird sie.

==========

Batch fuer Einsteiger

========== Befehlsparameter %1.. ========== Lektion #6 ==

In BAT-Ablaeufen sind oft variable Angaben erforderlich, die erst
mit dem jeweiligen Aufruf bestimmt werden sollen.

Dazu werden in der BAT-Datei "Platzhalter" eingebaut. Beim Start einer BAT-Datei werden dann die aktuellen Werte einfach als zusaetzliche Angaben in der Befehlszeile gemacht (Befehlsparameter).

Platzhalter bestehen aus einem Prozentzeichen mit der laufenden Nummer des Befehlsparameters, also %1.....%9 (nur _eine_ Ziffer moeglich).

Ein Beispiel (Datei EDR.BAT):

```
    attrib -R %1
    XEDIT %1
    attrib +R %1
Aufruf:
    EDR ANY.TXT
```

Hier wird vor dem Aufruf des Editors (XEDIT) der Schreibschutz der angegebenen Datei entfernt und anschliessend wieder eingeschaltet.

DOS ersetzt Platzhalter jeweils bevor eine Zeile interpretiert wird durch den entsprechenden Befehlsparamter. Auf diese Weise kann praktisch alles in einem BAT-Ablauf variabel gemacht werden, auch Befehle und Programm-Aufrufe.

Trennzeichen, Sonderzeichen

Zur Trennung von Befehlsparametern koennen auch Komma oder Semikolon verwendet werden. Sie werden praktisch durch Leerzeichen ersetzt. Leere Parameter koennen auf diese Weise NICHT uebergeben werden, z.B.:

```
XXX.BAT  A,,,B              ergibt: %1=A, %2=B
```

Auch die Uebergabe eines Parameters, in dem Leerzeichen enthalten sein sollen, ist nicht moeglich (Anfuehrungszeichen nutzen nichts) Beisipel:

```
AAA.BAT  "das Wort"         ergibt: %1="das, %2=Wort"
```

Achtung: Das Gleich-Zeichen (=) gilt ebenfalls als Trennzeichen, wird also wie ein Leerzeichen behandelt!

Zum Ausprobieren empfiehlt es sich, eine TEST.BAT zu schreiben, die einfach 9 ECHO-Befehle enthaelt:

```
    ECHO %1
    ECHO %2
    (etc.)
```

Parameter %0

Mit %0 kann man im BAT-Ablauf den Namen der BAT-Datei ansprechen, genauer gesagt, den Befehl, so wie er beim Aufruf angegeben wurde. Nur fuer spezielle Tricks zu gebrauchen.

SHIFT-Befehl

Kaum genutzt, aber der Vollstaendigkeit halber:
SHIFT verschiebt die Parameter-Liste nach links, d.h.

    %0 faellt raus,
    der bisherige %1 wird zu %0
    %2 wird zu %1
    und so weiter

Damit kann auch der zehnte Parameter erreicht werden (jetzt also
%9). SHIFT kann bei Bedarf wiederholt werden.

Weitere Themen

Variable Befehlsparameter koennen erst richtig eingesetzt werden,
wenn man abfragen und verzweigen kann. Dazu gibt's Bedingungen
(IF), GOTO und Labels.

Ein GOTO geht zu einer anderen Stelle im BAT-Ablauf. Dazu muss
die gewuenschte Stelle markiert werden, und zwar mit einer Sprung-
marke ("Label"). Eine Sprungmarke ist irgend ein Wort mit einem
vorangestellten Doppelpunkt. Beispiel:

    :WEITER

Mit dem Befehl GOTO WEITER wuerde also der Ablauf an dieser Stelle
fortgesetzt.

Ein GOTO wird natuerlich erst interessant, wenn er bedingt ein-
gesetzt werden kann, z.B.:

    IF not exist C:\COMMAND.COM GOTO FEHLER

Aber bevor wir auf die IF-Konstruktionen eingehen, zunaechst mal
ein nuetzliches Beispiel ganz ohne IF.

Beispiel: GO.BAT

Angenommen es gibt diverse Programme in verschiedenen Verzeich-
nissen, die jederzeit zu starten sein sollen, ohne dass man des-
wegen alle diese Verzeichnisse im PATH angeben will (der ist ja
eh' schon lang genug).

Man braucht also eine BAT-Datei, die den noetigen Verzeichnis-
wechsel (CD) und dann den Aufruf ausfuehrt. Fuer jedes Programm
eine eigene BAT-Datei zu schreiben ist kein Kunststueck. Aber wie
kann man verschiedene Aufrufe in _einer_ BAT-Datei steuern?

Ganz einfach. Nennen wir die BAT-Datei: GO.BAT
Sie enthaelt zunaechst die Befehle:

```
@echo off
GOTO %1
```

Der Ablauf springt also direkt zu der Marke, die zum GO-Befehl angegeben wird. Als Marken dienen die Programmnamen oder irgendwelche Abkuerzungen, die man sich leicht merken kann. Fuer jedes Programm wird nun eine kleine Aufruf-Routine geschrieben, z.B.:

```
:TERM                  (Marke, anzugeben)
CD \MODEM\FD           (Verzeichnis-Wechsel)
FD /T                  (Programm-Aufruf)
GOTO ENDE              (BAT-Datei beenden)
```

Natuerlich koennen auch weitere Befehle (z.B. Laden residenter Programme, SET..) in diese kleine Routine aufgenommen werden.

Ganz am Ende der BAT-Datei darf die Marke :ENDE nicht fehlen. Empfehlenswert auch ein anschliessendes CD \

Damit lassen sich nun alle moeglichen Programme nach diesem Muster aufrufen:

```
GO TERM
```

Falls man allerdings kein Ziel oder ein falsches angibt, meldet DOS "Sprungmarke nicht gefunden" und bricht ab.

Jetzt noch einige Hinweise, die fuer den Umgang mit Labels wichtig sind.

Gueltige Labels

Eine Sprungmarke darf beliebig lang sein, aber DOS ignoriert alles, was ueber 8 Stellen hinausgeht !! Falls mehrere Marken vorkommen, die (in den ersten 8 Stellen) gleich sind, wird immer nur die erste gefunden.

Wenn Sonderzeichen verwendet werden, bitte nur solche, die auch in Dateinamen gueltig sind (ausser "%"). Andere Sonderzeichen sind Trennzeichen. Vorsicht: Ein "?" ist z.B. gueltig, aber ein "*" ist absolut unbrauchbar. Am besten: keine dubiosen Sonderzeichen verwenden!

Nach dem Leerzeichen oder Trennzeichen wird der Rest der Zeile ignoriert. Dies sollte zur Kommentierung genutzt werden!

Ersatz fuer REM

Labels koennen hervorragend fuer Bemerkungen missbraucht werden, zumal dabei ausnahmsweise auch die Umleitungszeichen <|> bedenkenlos verwendet werden duerfen. Damit man gleich sieht, dass es sich

nicht um eine echte Sprungmarke handelt, hat sich der doppelte
Doppelpunkt eingebuergert, z.B.:

```
:: Syntax: MOP <Quelldatei> <Zieldatei> [/M|/X]
```

Sprungziel im GOTO-Befehl

Auch beim GOTO darf der angegebenen Sprungmarke ein Doppelpunkt
vorangestellt werden (Geschmacksache), z.B.:

```
GOTO :WEITER
```

==========

Batch fuer Einsteiger

========== Bedingungen, IF EXIST ========== Lektion #8 ==

Bedingungen werden durch IF-Ausdruecke formuliert. DOS bietet
drei Varianten:

```
IF [not] exist <Datei-Ausdruck>
IF [not] errorlevel <n>
IF [not] <Wort1>==<Wort2>
```

Nach einer Bedingung kann jede beliebige BAT-Anweisung stehen,
z.B. ein GOTO, ein DOS-Befehl oder ein Programm-Aufruf.
Die Anweisung wird nur ausgefuehrt, wenn die Bedingung WAHR ist.
Das optionale NOT bedarf wohl keiner weiteren Erklaerung.

Da IF EXIST am einfachsten zu handhaben ist, soll zunaechst mal
mit diesem Ausdruck begonnen werden. Beispiele:

```
IF exist TEST.BAK del TEST.BAK
IF NOT exist TEST.DAT goto WEITER
```

IF EXIST liefert WAHR, wenn die betreffende Datei existiert.
Dabei koennen auch die Jokerzeichen "?" und "*" verwendet werden,
und natuerlich sind Laufwerk- und Pfadangaben erlaubt.

```
IF EXIST C:\TEMP\*.*       (irgendeine Datei in C:\TEMP ?)
```

Ungueltige Datei-Angaben

Bei Ungueltigen Datei-Angaben oder Pfaden gilt die Datei als NICHT
vorhanden. Es gibt also keine Fehlermeldung.

Verzeichnis Vorhanden?

Das Vorhandensein eines Verzeichnisses kann nicht direkt fest-
gestellt werden, da DOS nur nach Dateinamen sucht. Man koennte
zwar pruefen, ob Dateien im gewuenschten Verzeichnis sind, aber
damit wird ein evtl. leeres Verzeichnis nicht festgestelt.

Es gibt jedoch eine einfache Abhilfe: man sucht nach NUL.
NUL ist ein spezielles Geraet, das zwar nicht vorhanden, aber
ueberall definiert ist.

```
IF EXIST C:\TEMP\NUL
```

Dieser Ausdruck ist nur dann WAHR, wenn der Pfad C:\TEMP gueltig
ist, egal ob Dateien vorhanden sind oder nicht.

UND-Verknuepfung

Nach einer Bedingung koennen weitere Bedingungen in der selben
Zeile folgen. Das entspricht einer UND-Verknuepfung. Beispiel:

```
IF exist TEST.BAK  if not EXIST TEST.NEU  ren TEST.BAK TEST.NEU
```

Variabler Datei-Ausdruck

Mit IF exist koennen auch variable Angaben ueberprueft werden:

```
IF NOT exist %1 ECHO %1 ist nicht vorhanden
```

Allerdings sollte man sichergehen, dass der angegebene Befehls-
parameter nicht leer ist, denn dann bekaeme DOS folgende Zeile
zu lesen:

```
IF NOT exist ECHO ist nicht vorhanden
```

Da die Datei ECHO vermutlich nicht existiert, fuehrt DOS den
Befehl "ist" aus, sofern ein solches Programm vorhanden ist.
Andernfalls Fehler: "Befehl oder Dateiname nicht gefunden".

Dieses Problem besteht bei allen IF-Ausdruecken mit Variablen,
weil DOS den variablen Wert bereits VOR der Interpretation der
Zeile einsetzt. Bei leeren Variablen kommt so die gesamte Syntax
des Befehls durcheinander. Haeufigste Fehlermeldung daher:
"Syntax-Fehler".

Nicht vorhandene (leere) Variable bzw. Befehlsparameter lassen
sich mit der Vergleichsbedingung feststellen. Das Thema kommt in
der naechsten Lektion.

Wenn zwei Woerter verglichen werden sollen, so macht dies nur
Sinn, wenn mindestens eines davon eine Variable ist, also ein
Befehlsparameter oder eine "Umgebungsvariable". Hier soll es
zunaechst nur um Befehlsparameter gehen. (Bei Umgebungsvariablen
gibt es zusaetzliche Komplikationen, weil sie z.B. Leerzeichen
und Sonderzeichen enthalten koennen.)

Beispiel: IF %1==A: GOTO WEITER

Achtung: DOS besteht auf dem doppelten "=". Sonst Syntax-Fehler!
Leerzeichen vor und hinter den Gleich-Zeichen sind erlaubt.

Gross-/Kleinschreibung wird unterschieden. Wenn also im obigen Beispiel "a:" als Befehlsparameter angegeben wurde, ist die Bedingung NICHT wahr. Da hilft nur eines: beide Moeglichkeiten abfragen.

Leere Parameter

Bei der IF EXIST Abfrage wurde schon gezeigt, dass leere Befehlsparameter die Syntax durcheinander bringen.

Beispiel: IF %1 == A: GOTO WEITER
ergibt: IF == A: GOTO WEITER (wenn %1 leer)

Resultat: "Syntax-Fehler".

Die Loesung des Problems: Man erweitert beide Seiten der Gleichung einfach um das selbe Zeichen.

Beispiel: IF %1x == A:x GOTO WEITER
ergibt: IF x == A:x GOTO WEITER (wenn %1 leer)

Zu diesem Zweck kann man beliebige Buchstaben verwenden, auch Sonderzeichen, ausgenommen Trennzeichen wie Komma, Semikolon, da diese ebenso wie Leerzeichen ignoriert werden. Eine gute Wahl sind z.B. Anfuehrungszeichen. (Sie haben im BATCH nicht die sonst uebliche Bedeutung, sondern werden wie Buchstaben behandelt.)

Beispiel: IF "%1" == "A:" GOTO WEITER
ergibt: IF "" == "A:" GOTO WEITER (wenn %1 leer)

Jetzt ist auch klar, wie man leere Parameter abfragt:

Beispiel: IF %1x == x ECHO Bitte Laufwerk angeben!
oder: IF "%1" == "" ECHO Bitte Laufwerk angeben!

Fazit:

Bei IF-Abfragen immer die hier beschriebenen Verfahren anwenden, sofern auch nur die geringste Moeglichkeit besteht, dass Variable bzw. Befehlsparameter leer sein koennten.

Noch ein Tip:

Angenommen man moechte sicherstellen, dass die Befehlsparameter %2 und %3 leer sind. Das koennte so aussehen (UND-Verknuepfung):

```
IF "%2"=="" IF "%3"=="" goto OK
```

Einfacher geht's so:

```
IF "%2%3"=="" goto OK
```

Klaro?

==========

Batch fuer Einsteiger

========== IF ERRORLEVEL n ========== Lektion #10 =

Jedes Programm, das zum Beenden die empfohlene DOS-Funktion 4C benutzt, gibt einen RETURN-Code ("Beendigungscode") an COMMAND.COM bzw. an das aufrufende Programm zurueck. Dieser Return-Code ist ein Byte und kann daher die Werte 0...255 haben.

Um diesen Return-Code zu interpretieren, muss man natuerlich wissen, welche Werte fuer welchen Fall von dem berteffenden Programm vorgesehen sind. In den meisten Faellen wird der Return-Code benutzt, um mitzuteilen, ob ein Fehler aufgetreten ist: Null bedeutet "Programm ordnungsgemaess beendet", jeder andere Wert signalisiert einen Fehler. Wenn es "leichte" und "schwere" Fehler gibt, kann das Programm verschiedene Codes verwenden: je hoeher um so schwerwiegender der Fehler.

Auf dieser Basis wurde die IF ERRORLEVEL... Abfrage im BATCH konzipiert:

IF ERRORLEVEL n
bedeutet: IF Return-Code >= n

So kann man mit einer einzigen Abfrage (IF ERRORLEVEL 1) feststellen, ob ueberhaupt ein Fehler aufgetreten ist, ohne dass alle anderen (moeglicherweise gelieferten) Codes abgefragt werden muessen.

Bei allen ERRORLEVEL-Abfragen muss man sich also der "groesser/gleich"-Logik bewusst sein. Wenn auf verschiedene Return-Codes reagiert werden soll, ist (bei Verwendung von GOTO) eine Abfrage in absteigender Folge erforderlich.

Beispiel: Ein Programm kann die Return-Codes 0,1,3 und 9 zurueckgeben. Dann lautet die Abfrage:

```
IF errorlevel 9 goto Label-9
IF errorlevel 3 goto Label-3
IF errorlevel 1 goto Label-1
:: hier ist errorlevel 0
```

Uebrigens:
Die Abfrage "IF [not] ERRORLEVEL 0 goto ..." ist absolut witzlos, denn groesser oder gleich Null ist der Return-Code IMMER. Um nur den Return-Code 0 abzufragen verwendet man logischerweise:

```
IF not errorlevel 1 goto ...
```

Soll nur ein ganz bestimmter Code abgefangen werden, z.B. 100, dann geht das so:

```
IF errorlevel 100  IF NOT errorlevel 101 goto ....
```

Noch ein Hinweis:

Der Return-Code kann nur in dieser IF ERRORLEVEL Konstruktion angesprochen werden. Eine *direkte* Verwendung des Wertes (z.B. als Variable) ist nicht ohne weiteres moeglich.

Andere Verwendung des Return-Codes

Ein Programm kann natuerlich den Return-Code auch fuer andere Zwecke als nur fuer Fehler verwenden. Z.B. kann ein Mailer mit bestimmten Codes mitteilen, was gerade anliegt. Auf jeden Fall muss vereinbart sein, welche Codes fuer welchen Zweck verwendet werden. Dafuer gibt es keinerlei allgemeine Regeln.

DOS-Befehle

DOS-Befehle, also in COMMAND.COM integrierte Befehle ohne eigene Programmdatei, geben - soweit feststellbar - ueberhaupt keinen Return-Code zurueck. Auch nicht Null: der zuletzt produzierte Return-Code bleibt erhalten! Wer also auf die Idee kommt, z.B. nach COPY oder MD den ERRORLEVEL abzufragen, wird sich wundern!

Sonstige DOS-Gemeinheiten

Wo man vielleicht einen Return-Code erwarten sollte, geben DOS-Programme moeglicherweise generell Null zurueck, z.B. beim Datei-Vergleich. Also: zur Nutzung der ERRORLEVEL-Abfrage ist eine genaue Kenntnis der vom Programm uebergebenen Werte unumgaenglich.

Fuer einige DOS-Programme finden sich die Beendigungscodes im DOS-Handbuch, fuer andere gibt es jedoch keine Angaben, auch in Faellen, in denen Return-Codes tatsaechlich produziert werden.

==========

Batch fuer Einsteiger

========== CALL oder nicht CALL ========== Lektion #11 =

In einem BAT-Ablauf lassen sich nicht nur .COM und .EXE Programme aufrufen, sondern auch andere .BAT Ablaeufe. Verwendet man nun den Namen einer BAT-Datei als Befehl, dann sollte man meinen, dass DOS - wie bei Ausfuehrung eines Programmes - anschliessend in der aufrufenden BATCH weitermacht.

Das aber geht nur mit dem CALL-Befehl, sonst wird der (erste) BAT-Ablauf NICHT fortgesetzt!

Dazu muss man verstehen, dass es in den ersten DOS-Versionen (vor DOS 3.3) gar keine Verschachtelung von BAT-Ablaeufen gab. Die Beendigung eines Batch-Ablaufs bei einem weiteren Aufruf war also zwangslaeufig. Als DOS dann weiterentwickelt wurde, hat man halt den CALL erfunden. Also:

XXX.BAT setzt den Ablauf in XXX.BAT fort, OHNE zur aufrufenden Batch zurueckzukehren

CALL XXX.BAT kehrt nach Ausfuehrung von XXX.BAT zurueck.

Befehlsparameter koennen in jedem Falle uebergeben werden, auch weitergegeben werden, z.B.:

```
CALL update %1 A:
```

Nach Ausfuehrung einer BATCH mit CALL sind selbstverstaendlich die Befehlsparameter des ersten Ablaufs wieder verfuegbar, denn DOS fuehrt fuer jede weitere BAT-Ebene einen eigenen Steuerblock im Arbeitsspeicher, wo u.a. auch die jeweiligen Befehlsparameter abgelegt sind.

Sollen von der aufgerufenen Batch irgendwelche Werte zurueckgegeben werden, so sind dafuer Umgebungsvariable zu verwenden (s. spaerete Lektion). Der zuletzt von einem Programm erzeugte Return-Code kann aber noch nach der Rueckkehr in die aufrufende Batch per ERRORLEVEL abgefragt werden. (Ein BAT-Ablauf selbst produziert keinen Return-Code/Errorlevel.)

Rekursion

Grundsaetzlich kann auch die selbe BATCH per CALL (rekursiv) aufgerufen werden, entweder direkt oder auf Umwegen ueber weitere CALLs, - aber dann sollte man schon was davon verstehen, sonst pflastert DOS den Arbeitsspeicher mit Steuerbloecken voll, von anderen moeglichen Overflows ganz zu schweigen.

Manchmal geschieht so etwas unbeabsichtigt. Uebersichtliche CALL-Strukturen sind daher immer eine gute Investition. (Es gelten die allgemeinen Regeln der Programmierkunst!)

Rekursionen koennen auch auftreten, wenn aus einem Programm heraus (z.B. Mailer) eine weitere DOS-Shell mit Batch-Ablauf gestartet wird. Ohne saubere Trennung der einzelnen BAT-Ebenen kommt es hier leicht zu Ueberraschungseffekten...

QUIT

Ein spezieller BAT-Aufruf ohne CALL hat sich als recht nuetzlich zum Abbruch einer BATCH erwiesen: Es wird einfach eine leere BAT-Datei, z.B. mit dem Namen QUIT.BAT, gerufen. Dies geht normalerweise schneller als ein GOTO ans Ende der laufenden BAT-Datei (was daran liegt, dass DOS keine sonderlich effiziente Batchverarbeitung hat). Beispiel:

```
IF errorlevel 3 QUIT
```

Anstelle der leeren QUIT.BAT kann auch folgende einzeilige BAT-Datei verwendet werden:

```
%1 %2 %3 %4 %5 %6 %7 %8 %9
```

Damit wird aus allen Angaben (bis zu 9), die nach QUIT folgen, eine Befehlszeile produziert und ausgefuehrt. Oft laesst sich

auf diese Weise eine extra GOTO-Konstruktion einsparen.
Benutzungsbeispiele:

```
QUIT cls
QUIT echo Datei %1 ist nicht vorhanden - Abgebrochen!
QUIT del *.TMP
```

Abbruch bei GOTO-Fehler

An dieser Stelle noch ein Hinweis auf eine besondere Tuecke von
DOS: Sollte bei einem GOTO das Sprungziel nicht gefunden werden,
beendet DOS nicht nur die laufende BATCH mit einer Fehlermeldung,
sondern saemtliche per CALL entstandene Verschachtelungsebenen!

==========

Batch fuer Einsteiger

========== FOR (Schleifen) ========== Lektion #12 =

Die FOR-Konstruktion ermoeglicht die mehrfache Ausfuehrung eines
Befehls mit einem variablen Argument. Die Argumente werden nach-
einander aus einer Liste entnommen. Beispiel:

```
FOR %%a IN (X Y Z) DO echo %%a
        -------    --------
        Liste      Befehl
```

Das hat die gleiche Wirkung wie:

```
echo X
echo Y
echo Z
```

Die Schluesselwoerter "IN" und "DO" sind vorgeschrieben.
Die Argument-Liste muss immer in Klammern gesetzt werden.
Und jetzt zu diesem "%%a":

Erstens: Es kann jeder beliebige Buchstabe verwendet werden, nur
keine Ziffer (fuer Befehlsparameter reserviert). Dass ueberhaupt
verschiedene Buchstaben moeglich sind, macht eigentlich keinen
Sinn, denn diese Variable ist nur innerhalb der FOR-Zeile gueltig,
und ein mehrfaches FOR (Schachtelung) ist nicht zulaessig.

Zweitens: Das doppelte %-Zeichen ist in BAT-Dateien Vorschrift.
Auf der Befehlsebene (wo die FOR-Konstruktion auch moeglich ist)
darf jedoch nur EIN %-Zeichen verwendet werden.

Anmerkung: DOS ersetzt in einer BAT-Zeile grundsaetzlich doppelte
%-Zeichen durch ein einfaches, und versucht in diesem Falle nicht,
Umgebungsvariable oder Befehlsparameter einzusetzen. Danach sieht
also eine FOR-Zeile genauso aus wie auf der Befehlsebene.

Als Befehl in einer FOR-Konstruktion sind beliebige BAT-Befehle
(auch CALL), DOS-Befehle oder Programmaufrufe moeglich. Nur ein
weiteres FOR ist nicht moeglich.

Noch ein Beispiel:

```
FOR %%x in (DER.TXT DIE.DAT DAS.BLA) do COPY %%x A:
```

Argumente mit Joker-Zeichen

Sobald DOS in der Argument-Liste Fragezeichen oder Sternchen findet, wird das Argument als Dateiname verstanden (ggfs. mit Laufwerk und Pfad). Der Befehl wird dann fuer jeden Dateinamen ausgefuehrt, auf den der "Match-Code" passt. Beispiele:

```
FOR %%a in (C:\*.BAT) do type %%a

FOR %%a in (*.TXT *.DAT) do echo %%a
```

Im zweiten Beispiel werden alle Dateinamen mit den Zusaetzen .TXT und .DAT auf dem Bildschirm ausgegeben.

```
FOR %%f in (A:*.*) do DEL %%f
```

Hier wird nicht etwa der Befehl "DEL A:*.*" ausgefuehrt, sondern ein DEL-Befehl fuer jede einzelne Datei!

Trennzeichen in der Liste

Ausser Leerzeichen koennen Komma, Semikolon oder sogar das Gleich-Zeichen verwendet werden. Ein Argument darf also diese Zeichen nicht enthalten. Etwas ganz Merkwuerdiges geschieht beim Schraeg-strich. Ausprobieren: FOR %%a in (TEST/L12) do ECHO %%a

Befehl mit IF

Der auszufuehrende Befehl darf auch bedingt sein. Beispiel:

```
FOR %%a in (100,70,10,5,3,1) do IF ERRORLEVEL %%a GOTO L-%%a
```

bewirkt:

```
IF ERROELEVEL 100 goto L-100
IF ERROELEVEL  70 goto L-70
(u.s.w....)
```

Dazu muss gesagt werden, dass eine FOR-Schleife (zwangslaeufig) mit dem ersten ausgefuehrten GOTO beendet wird.

Generell lassen sich alternative Bedingungen (ODER) mit einer FOR-Schleife einfacher darstellen als durch mehrfache IF-Zeilen, z.B.:

```
FOR %%x in (A: a: B: b:) do IF !%1==!%%x goto OK
QUIT echo Ungueltiges Laufwerk!
:OK
```

Hier wird getestet, ob der Befehlsparameter %1 ein gueltiges Diskettenlaufwerk enthaelt. (Das "!" verhindert Syntaxfehler, falls %1 leer ist. QUIT s. Lektion #11.)

Ein bedingtes FOR kann ebenfalls verwendet werden, z.B.:

```
if not !%1==! FOR %%a in (A: a: B: b:) do IF %1==%%a goto OK
```

Leere Argumente

Sind einzelne Argumente leer, wird der Befehl dafuer nicht ausgefuehrt, z.B. hier:

```
FOR %%a in (%1,%2,%3,%4,%5) do ECHO %%a
```

Ist die ganze Argument-Liste leer, wird gar nichts ausgefuehrt.
Es gibt auch keine Fehlermeldung, wenn hier z.B. %1 leer ist:

```
FOR %%a in (%1)  do irgendwas
```

Vorsicht!

Fuer die Variable koennen Gross- oder Kleinbuchstaben verwendet werden, aber innerhalb einer FOR-Zeile muss man bei der Schreibweise bleiben! Das hier funktioniert nicht:

```
FOR %%a in (*.*) do ECHO %%A
    ^               ^
```

FOR mit CALL

Was eigentlich nicht geht, naemlich FOR-Verschachtelungen und Ausfuehrung MEHRERER Befehle, laesst sich evtl. durch CALL einer weiteren BAT-Datei im FOR-Befehl realisieren.

==========

Batch fuer Einsteiger

========== Umgebungsvariable, SET ========== Lektion #13 =

In BAT-Ablaeufen lassen sich Variable benutzen, die als sogenannte "Umgebungsvariable" in einem besonderen Speicherbereich gefuehrt werden. Auf Konzeption und Bedeutung dieser Umgebungsvariablen-Speicher soll hier nicht weiter eingegangen werden. Wichtig ist zunaechst nur, dass dieser Bereich eine (variable) Liste von Zuweisungen enthaelt.

Jede Zuweisung besteht aus einem Variablennamen und einer Zeichenfolge (string), getrennt durch das Gleich-Zeichen, z.B.:

```
PROMPT=$p$g
TEMP=C:\WORK\TEMP
```

Der aktuelle Inhalt des Speichers kann jederzeit mit dem Befehl SET (ohne Angaben!) auf der DOS-Befehlsebene geprueft werden.

COMMAND.COM bedient sich dieses Speichers, um bestimmte Infor-

mationen zu ermitteln, z.B. die Verzeichnispfade, in denen Programme gesucht werden sollen (PATH) oder die Darstellung der Eingabeaufforderung auf der Befehlsebene (PROMPT). Aber auch andere Programme koennen diese Informationen nutzen.

Die wichtigste Verwendung bietet sich jedoch in BAT-Dateien:

* Mit der SET-Anweisung koennen neue Zuweisungen aufgenommen, bestehende geaendert oder geloescht werden (moeglich auch auf der Befehlsebene).

  Beispiel: SET WORT=das (neue Zuweisung oder Aenderung)
  SET WORT= (Loeschung: Leer-Zuweisung)

* Durch Angabe des Variablennamens, eingeschlossenen in Prozentzeichen, kann die aktuelle Zuweisung an jeder beliebigen Stelle der BAT-Datei eingesetzt werden (NICHT jedoch auf der Befehlsebene).

  Beispiel: ECHO Die Variable WORT enthaelt %WORT%

Im BAT-Ablauf untersucht DOS jede Zeile vor der Ausfuehrung auf %-Zeichen, um Variable durch die zugewiesene Zeichenfolge zu ersetzen. Dabei ist es gleichgueltig, ob sich die Variable in einem Text, einer Anweisung, einem GOTO-Ziel oder in sonstigen Angaben befindet. Ausgenommen sind nur Labels. Ist eine Variable nicht vorhanden, wird nichts eingesetzt, d.h. der %...% Ausdruck wird einfach entfernt.

Hinweis:
Um das Prozentzeichen in einem Text zu zeigen, muss es verdoppelt werden. Beispiel: ECHO Zuzueglich 15%% MwSt.

Regeln fuer Variablen-Namen

Bei Variablen-NAMEN wird Groá-/Kleinschreibung NICHT unterschieden (es wird generell in Groábuchstaben umgewandelt). Die zugewiesene Zeichenfolge wird dagegen immer unveraendert gespeichert.

Variablen-Namen duerfen nicht mit einer Ziffer beginnen, denn %0...%9 sind fuer Befehlsparameter reserviert.

Bitte nicht versehentlich Variablennamen verwenden, die bereits von DOS oder von Programmen benutzt werden (wie z.B. PATH oder PROMPT).

Beim SET-Befehl werden im Variablen-Namen auch Sonderzeichen und sogar Leerzeichen akzeptiert (kein Witz!). Um Problemen aus dem Wege zu gehen, sollten aber nur Zeichen verwendet werden, die auch in Dateinamen gueltig sind (ausgenommen "%"). Eine besondere Falle fuer Ahnungslose ist das Leerzeichen am Ende des Namens:

```
SET TEST = JA
      ^
```

Hier ist der Variablen-Name nicht "TEST", sondern "TEST " (!)

Also: zwischen Name und "=" bitte KEIN Leerzeichen!

Inhalt der zugewiesenen Zeichenfolge

Mit SET wird der Variablen der gesamte Rest der Zeile zwischen dem Gleichzeichen und CR zugewiesen. Die Zeichenfolge kann also auch aus mehreren Woertern bestehen.

Beispiel: SET TEXT=Bitte Taste druecken!

Auch Leerzeichen nach dem "=" sowie eventuelle Leerzeichen am Ende der Zeile (falls der Editor sowas nicht unterdrueckt) sind Teil der zugewiesenen Zeichenfolge!

Es gibt also keinerlei Begrenzungszeichen. DOS nimmt praktisch jedes Zeichen, sogar Control-Codes. Nur ein "=" in der Zeichenfolge wird reklamiert (Syntax-Fehler).

Eine Variable kann auch eine ganze Pararmeter- oder Argument-Liste enthalten, die sich dann mit CALL oder FOR wieder zerlegen laesst. Hier ein Beispiel (!):

```
FOR %%a in (%PATH%) do ECHO %%a
```

Verkettung, Kombinationen

Natuerlich kann die zugewiesene Zeichenfolge ihrerseits Variable enthalten, besser gesagt: die aktuellen Zuweisungen von Variablen. denn diese werden ja vor Interpretation der Zeile eingesetzt. Beispiele:

```
SET X=%X%/D
SET COM=%BEF% %QUELLE% %ZIEL%
SET PATH=%PATH%;C:\XYZ
FOR %%x in (A a B b) do  IF %LW%==%%x  set LW=%LW%:
                                  ^
```

Variable und IF

Solange eine Variable nur ein einfaches Wort enthaelt, koennen IF-Abfragen wie bei Befehlsparametern gemacht werden. Aber auch hier gelten die Grundregeln:

* Gross-/Kleinschreibung wird unterschieden
* Leere Variable gefaehrden die Syntax

Immer daran denken, dass DOS erst dann anfaengt, eine Zeile zu interpretieren, nachdem alle Variablen eingesetzt sind.

Zur Abfrage auf leere Variable bzw. von moeglicherweise leeren Variablen siehe Lektion #9. Beispiele:

```
IF "%LW%"=="A:" goto OK
IF !%LW%==!     goto FEHLER
```

Wenn allerdings eine Variable mehrere Woerter enthaelt (getrennt durch Leerzeichen, Komma, Semikolon), dann wird's gefaehrlich.

* Mehrere Woerter NACH dem "==" bringen die Syntax durcheinander (verglichen wird ohnehin nur das erste Wort).

* Bei mehreren Woertern VOR dem "==" gibt es zwar keinen Fehler, aber es wird nur das erste Wort zum Vergleich herangezogen!! (Ausprobieren: IF DAS NICHT==DAS echo Vergleich positiv)

Mit anderen Worten: IF-Abfragen sind in diesen Faellen tabu! Einzige Ausnahme: die Abfrage auf leere Variable funktioniert trotzdem (mal nachdenken, warum).

Verfuegbarer Speicherplatz

Der Speicherplatz fuer Umgebungsvariable ist begrenzt. Es wird daher dringend empfohlen, am Ende einer BAT-Datei alle nicht mehr benoetigten Variablen zu loeschen, damit kein Muell angehaeuft wird. Mit FOR geht das ganz elegant - Beispiel:

```
FOR %%a in (DATEI EL X Y Z) do SET %%a=
```

Wieviel Platz noch frei ist, kann man DOS leider nicht entlocken. (Dafuer gibt es aber kleine Utilities.)

Da der Speicherplatz COMMAND.COM zugeordnet ist, kann die Groesse des Speichers nur beim Laden des Befehlsprozessors beeinflusst werden. Dazu dient die Option /E der SHELL-Anweisung in der CONFIG.SYS - Beispiel (800 Bytes gewuenscht):

```
SHELL=C:\COMMAND.COM /P /E:800
```

Schaut mal ins DOS-Handbuch.

Weitere COMMAND-SHELL

Beim Laden einer weiteren COMMAND-SHELL wird ein eigener Speicherbereich fuer Umgebungsvariable angeglegt. In diesen werden alle Zuweisungen der ersten SHELL kopiert. Aber umgekehrt wird nach Verlassen der SHELL (per EXIT) nichts uebernommen!

Standardmaessig wird von DOS nur recht wenig freier Speicher in einer weiteren COMMAND-Shell zur Verfuegung gestellt. Bei Bedarf kann auch hier Option /E verwenden werden, wobei natuerlich der bereits verbrauchte Speicher (s.o.) zu beruecksichtigen ist.

Normalerweise wird jedoch die COMMAND-Shell von einem Programm bereitgestellt, das dann auch fuer die Speichergroesse zustaendig ist. Gute Programme schaffen soviel freien Umgebungsvariablenspeicher wie in der ersten Shell noch frei war, oder lassen die Groesse per Setup einstellen. (Falls z.B. jemand WINDOWS benutzt: SYSTEM.INI, [NonWindowsApp], CommandEnvSize=....)

System-Variable

Von DOS bzw. COMMAND.COM werden folgende Variable benutzt:

PATH Suchpfade fuer ausfuehrbare Dateien
TEMP Verzeichnis fuer temporaere Dateien
COMSPEC Befehlsprozessor (COMMAND.COM) mit vollstaendigem Pfad
PROMPT Eingabeaufforderung auf der Befehlsebene
DIRCMD fuer DIR Optionen (ab DOS 5.0)

Zum Schluss nochmal der Hinweis: ERRORLEVEL ist keine Variable (sondern Schluesselwort in einem IF-Ausdruck). Wie man aber den Errorlevel in eine Variable kriegt, wird in einer der folgenden Lektionen erklaert.

==========

Batch fuer Einsteiger

========== Umleitungen: Ausgabe > ========== Lektion #14 =

Umleiten lassen sich Bildschirm-Ausgaben und Tastatur-Eingaben durch Verwendung folgender Symbole:

> Ausgabe in Datei oder an ein Geraet umleiten
< Eingabe aus Datei oder von einem Geraet holen
| Ausgabe direkt in eine Eingabe leiten

Umleitungen sind sowohl auf der Befehlsebene als auch in BAT-Dateien moeglich. In dieser Lektion soll es zunaechst um die Ausgabe-Umleitungen gehen. Das Prinzip wurde bereits in Lektion #3 (ECHO-Umleitung) beschrieben. Beispiele:

```
ECHO Dieser Satz wird in die Datei TEST geschrieben > TEST
ECHO Dieser Satz wird in der Datei TEST angefuegt >> TEST
```

Eine eventuell vorhandene Datei wird durch den einfachen Pfeil geloescht, waehrend beim doppelten Pfeil am Ende angehaengt wird.

Grundsaetzlich lassen sich auch die Bildschirm-Ausgaben von anderen Programmen oder DOS-Befehlen umleiten, z.B.:

```
FIND "49-89-" NODE\REGION24.001 > MUC.LST
```

Das Ergebnis der Suche, das standardmaessig auf dem Bildschirm erscheint, wird hier in die Datei MUC.LST geschrieben.

WICHTIG: Nicht alle Bildschirm-Ausgaben koennen umgeleitet werden.

Voraussetzung ist, dass der Output vom Programm ueber die DOS-STANDARD-AUSGABE vorgenommen wird. Nicht umleiten lassen sich:

* Ausgaben ueber das BIOS oder direkt in den VIDEO-RAM-Speicher
* Ausgaben ueber die STANDARD-FEHLER-AUSGABE

Die STANDARD-FEHLER-AUSGABE geht immer auf den Bildschirm, damit der Benutzer auch bei umgeleiteten Ausgaben ueber eventuelle Fehler sofort informiert wird. Dazu das folgende Beispiel:

```
copy  DIESE.DAT A:  > AUSGABE.TXT
```

Die Bestaetigung von DOS, "... Datei(en) kopiert", wird hier in die Datei AUSGABE.TXT umgeleitet. Dies gilt auch fuer den Fall, dass die zu kopierende Datei nicht existiert. Die zusaetzliche Fehlermeldung "Datei nicht gefunden" erscheint dann jedoch auf dem Bildschirm.

Ob und welche Ausgaben eines Programmes sich umleiten lassen, muss man gegebenenfalls durch Ausprobieren herausfinden.

Umleitung an andere Geraete

Geraete-Namen koennen unter DOS wie Dateinamen verwendet werden. Die Ausgabe kann also auch z.B. an den Drucker geschickt werden Beispiel:

```
type DIESEN.TXT > PRN       (= copy DIESEN.TXT PRN)
arj l ARCHIV > LPT1         (Inhaltsverzeichnis an Drucker)
```

Umleitung > NUL

NUL ist ein nicht existierendes Geraet, das aber benutzt werden kann, um Ausgaben ins Nichts zu schicken. Damit lassen sich oft unerwuenschte Ausgaben in BAT-Ablaeufen unterdruecken, sofern (!) die Standard-Ausgabe verwendet wird.

Stoerende Fehlermeldungen, z.B. bei DEL ("Datei nicht gefunden") lassen sich uebrigens am besten durch geeignete Abfragen unterdruecken:

```
IF exist %DATEI% del %DATEI%
```

Anfuegen in Dateien

Bei Verwendung des doppelten Pfeiles bitte beachten:

* Wenn die Datei noch nicht existiert, wirkt der doppelte Pfeil wie ein einfacher.

* Beim ersten Mal, z.B in einer FOR-Schleife, sollte die Datei einen definierten Zustand haben (evtl. vorher loeschen), z.B.:

```
del LISTE.TMP
for %%a in (*.TXT) do ECHO %%a >> LISTE.TMP
```

* Falls die vorhandene Datei ein EOF-Zeichen (Control-Z) am Ende hat, wird dieses entfernt (zumindest bei neueren DOS-Versionen)

Eingaben bei Ausgabe-Umleitung

Wenn ein Programm bei umgeleiteter Ausgabe eine Eingabe erwartet, steht man oft im Dunklen, weil die Eingabeaufforderung ja auch umgeleitet wird.

Dies kann z.B. bei CHKDSK /F passieren oder auch bei DIR, wenn da jemand die Umgebungsvariable DIRCMD=/P gesetzt hat...

Wie sich die Eingabe umleiten (also aus einer Datei holen) laesst, kommt in der naechsten Lektion.

Hinweise
---

* Wenn es nichts umzuleiten gibt, wird trotzdem die angegebene Datei erstellt, und zwar mit Null Bytes (!) Das kann man gezielt nutzen, z.B.: REM > DATEI

* Normalerweise werden auf dem Bildschirm Zeilen ausgegeben, so dass auch in der Datei Zeilen mit CR+LF am Ende gespeichert werden. Das muss aber nicht so sein und haengt vom jeweiligen Programm ab.

* Der Umleitungsausdruck, also Pfeil(e) mit Dateiname/Geraet, muss nicht unbedingt hinter dem Befehl stehen. DOS nimmt vor der Verarbeitung eines Befehls den Umleitungsausdruck heraus, ganz egal, wo er gefunden wird. In den folgenden Beispielen ist das Ergebnis immer das selbe:

  ```
  ECHO Das ist ein Test> DATEI
  >DATEI ECHO das ist ein Test
  ECHO Das> DATEI ist ein Test
  ```

  Auch bei versehentlichen Umleitungszeichen geht DOS ganz konsequent vor, ob es nun Sinn macht oder nicht...

* Eine Umleitung beim Aufruf einer BAT-Datei ist wirkungslos. Wer auf diese Weise ALLE Ausgaben einer BAT-Datei umleiten will, muss dazu eine weitere COMMAND-Shell starten, z.B.:

  ```
  COMMAND /C X.BAT > DATEI       (Option /C s.DOS-Handbuch.)
  ```

==========

Batch fuer Einsteiger

========== Umleitung: Eingabe, Pipe ========== Lektion #15 =

Die Eingabe-Umleitung ist das Gegenstueck zur Ausgabe-Umleitung und verwendet den Pfeil nach links. Dabei werden Eingaben, die eigentlich von der Tastatur erwartet werden, aus einer Datei eingelesen. Voraussetzung ist natuerlich, dass die angegebene Datei mit dem erforderlichen Inhalt existiert. Beispiel:

```
DEL A:*.* < J.TXT
```

Da DOS hier nachfragt "Sind Sie sicher (J/N)?" muss die Datei J.TXT ein "J" und ein CR enthalten, also eine Zeile mit "J" und dem ueblichen Zeilenabschluss.

Falls die Datei weitere Daten enthaelt, werden diese ignoriert, da der DEL-Befehl ja keine weiteren Eingaben erwartet. Grundsaetzlich koennen aber auch mehrfache Eingaben fuer ein Programm nacheinander aus den einzelnen Zeilen einer Datei entnommen werden. Dies laesst sich z.B. beim Programm DEBUG nutzen, um ein kleines Programm als ASCII-Datei zu verschicken. Mit dem Befehl

```
DEBUG < Datei
```

werden dann alle Zeilen der Datei als DEBUG-Befehle abgearbeitet. Noch ein Beispiel:

```
MORE < SOME.TXT
```

Das DOS-Programm MORE gibt die Eingabezeilen auf dem Bildschirm aus, und zwar mit Stop nach jeder vollen Seite. (Ohne Umleitung der Eingabe macht MORE gar keinen Sinn.)

Standard Input

Entsprechend der Ausgabe-Umleitung ist auch bei der Eingabe eine Umleitung nur moeglich, wenn der Input des Programms ueber die STANDARD-EINGABE angefordert wird. Dass es sich hierbei um die Tastatur handelt, ist klar, aber das Programm muss die ueblichen DOS-Funktionen benutzen. Bei Verwendung von BIOS-Funktionen kann der Input nicht umgeleitet werden.

Quelle fuer eine Input-Umleitung kann auch ein Geraet sein, z.B. COM1.

Ausgabe direkt in Eingabe leiten (PIPE)

Im obigen DEL-Beispiel wurde angenommen, dass die Datei J.TXT fuer die Eingabe-Umleitung bereits vorhanden ist. Natuerlich kann sie auch unmittelbar vorher per ECHO-Umleitung produziert werden:

```
ECHO J> J.TXT
DEL A:*.* < J.TXT
del J.TXT
```

Das ist jedoch ziemlich umstaendlich, da auch noch die Hilfsdatei hinterher geloescht werden muss. Einfacher geht's mit der direkten Methode:

```
ECHO J| del A:*.*
```

Das "|" bewirkt, dass der Output von der linken Seite als Eingabe fuer die rechte Seite uebernommen wird. DOS produziert dazu zwar auch eine Zwischendatei (im temporaeren Verzeichnis %TEMP%), aber man braucht sich um deren Namen und um das anschliessende Loeschen

nicht zu kuemmern.

Solche "PIPES" sind eigentlich dafuer gedacht, ganze Text-Dateien
zu verarbeiten. Das Programm SORT z.B. macht wie MORE (s.o.) auch
nur Sinn, wenn es mit Umleitungen benutzt wird. Beide Programme
erwarten (standardmaessig) den Input von der Tastatur und senden
den Output auf den Bildschirm.

Der folgende Befehl sortiert die Datei ASCII.TXT mit SORT /R in
absteigender Folge und gibt das Resultat seitenweise auf dem
Bildschirm aus (MORE):

```
SORT /R < ASCII.TXT | MORE
```

Anordnung im Befehl

Es koennen mehrere und verschiedene Umleitungen in einem Befehl
kombiniert werden. Fuer die richtige Reihenfolge der Angaben ist
nur zu beachten, dass mit "|" (Pipe) verbundene Programme immer
von links nach rechts abgearbeitet werden. Befehlsparameter und
Optionen muessen unmittelbar auf den zugehoerigen Befehl folgen.

Umleitungsausdruecke mit Pfeil nach links/rechts sind zwangs-
laeufig dem ersten bzw. letzten Programm (Befehl) zugeordnet,
ganz gleich, wo sie plaziert werden.

Hinweise

Vorsicht bei der Eingabe-Umleitung fuer Programme, die ganz
bestimmte Eingaben erwarten: Falls die Eingabedatei falsche oder
gar keine Angaben enthaelt, laesst sich auch mit der Tastatur
nichts mehr korrigieren. Mit etwas Glueck geht vielleicht noch
ein Control-BREAK, sonst ist ein Warmstart faellig.

Und: bitte nicht fuer Eingabe- und Ausgabe-Umleitung die selbe
Datei verwenden, z.B.:

```
Befehl < DATEI > DATEI
```

Da sich DOS beeilt, die Ausgabedatei neu anzulegen, ist die
Eingabedatei hin, und dann.. s.o.

Achtung: Bug!

Anweisungen mit PIPE ("|") duerfen nicht bedingt sein, z.B.:

```
IF %1==X echo.|date
```

MS-DOS meldet "undefinierte Zugriffsnummer", wenn SHARE vorhanden
ist, sonst passiert einfach gar nichts! Bei SORT kann sogar die
Fehlermeldung "Zuwenig Speicherplatz auf dem Datentraeger" kommen.
Falls noetig, also eine IF...GOTO Konstruktion verwenden!

Bedingte Anweisungen mit Pfeil nach links/rechts sind dagegen

problemlos moeglich.

==========

Batch fuer Einsteiger

========== Errorlevel in Variable ========== Lektion #16 =

DOS erlaubt es leider nicht, den Errorlevel z.B. per ECHO direkt
auf dem Bildschirm auszugeben. Dazu muss diese Information zu-
naechst auf irgend eine Weise in eine Variable uebernommen werden.

Zur Loesung des Problems gibt es einmal die brutale Methode:
255 IF-Abfragen mit GOTO zu einem von 255 Labels, wo jeweils der
entsprechende SET-Befehl ausgefuehrt wird.

Auf die etwas feinere Art werden FOR-Schleifen benutzt, mit denen
das Ergebnis schrittweise aus drei Ziffern zusammengesetzt wird.

Um zu verstehen, wie das funktioniert, soll erst mal der Error-
level von 0...9 (bzw. 10) ermittelt werden:

```
---------------------------------------------------------------
FOR %%e in (0 1 2 3 4 5 6 7 8 9) do IF Errorlevel %%e set EL=%%e
::
IF errorlevel 10 set EL=10 oder hoeher
ECHO Errorlevel: %EL%
---------------------------------------------------------------
```

Der Variablen EL werden hier nacheinander die Werte 0,1,2,3...
zugewiesen, aber nur so lange wie die Errorlevel-Abfrage WAHR ist.
Wenn also z.B. der Errorlevel 5 ist, dann passiert in der Schleife
ab 6 nichts mehr, und in der Variablen bleibt 5 haengen.

Im Gegensatz zur IF Errorlevel Abfrage mit GOTO muessen hier also
die Werte in aufsteigender Folge abgefragt werden.

Uebrigens - Immer daran denken: IF Errorlevel n
bedeutet: IF Return-Code >= n

Um zwei Ziffern (also 00..99) zu ermitteln, braucht man auch zwei
Schleifen:

```
---------------------------------------------------------------
:: Zehner ermitteln
FOR %%z in (0 1 2 3 4 5 6 7 8 9) do IF Errorlevel %%z0 set EL=%%z
:: Einer anhaengen
FOR %%e in (0 1 2 3 4 5 6 7 8 9) do IF Errorlevel %EL%%%e set EL=%EL%%%e
::
IF errorlevel 100 set EL=100 oder hoeher
ECHO Errorlevel: %EL%
---------------------------------------------------------------
```

Die vielen %-Zeichen sind etwas verwirrend, darum hier im Detail:

In der ersten Schleife wird Errorlevel %%z0 abgefragt, also %%z
(aus der Liste) mit einer angehaengten Null - ergibt 00,10,20....
In EL wird aber zunaechst nur die Zehner-Ziffer gespeichert.

In der zweiten Schleife wird Errorlevel %EL%%%e abgefragt, also zusammengesetzt aus %EL% (der bereits produzierte Zehner) und %%e (Ziffer aus der Liste).

Dazu ein Hinweis: Umgebungsvariable in einer FOR-Zeile werden von DOS nur einmalig vor Ausfuehrung der Schleife eingesetzt. Obwohl also in der zweiten Schleife EL staendig veraendert wird, hat dies keinen Einfluss mehr auf den dabei verwendeten Wert %EL%.

Die Hunderter koennte man ja nun auf die gleiche Weise ermitteln. Aber da wird die Sache etwas komplizierter, weil dann zwangslaeufig Errorlevels ueber 255 abgefragt werden - und dabei gibt es Probleme: DOS subtrahiert kommentarlos 256.

Die Abfrage IF Errorlevel 260
bewirkt also: IF Errorlevel 4

Aus diesem Grund muss extra dafuer gesorgt werden, dass die Ziffern-Liste nur (0 1 2 3 4 5) enthaelt, wenn die Schleife ueber 255 hinausgehen wuerde. Dazu wird fuer die Ziffern von 6 bis 9 eine Variable (%!%) verwendet, die bei Bedarf leer bleibt.

Dies ist die komplette Routine:

```
----------------------------------------schnipp----------
@echo off
set !=
:: Hunderter
FOR %%h in (0 1 2) do IF Errorlevel %%h00 set EL=%%h
:: & Zehner
IF not Errorlevel 200 set !=6 7 8 9
FOR %%z in (0 1 2 3 4 5 %!%) do IF Errorlevel %EL%%%z0 set EL=%EL%%%z
:: & Einer
IF not Errorlevel 250 set !=6 7 8 9
FOR %%e in (0 1 2 3 4 5 %!%) do IF Errorlevel %EL%%%e set EL=%EL%%%e
::
ECHO Errorlevel: %EL%
set !=
----------------------------------------schnapp----------
```

Wer das Ganze zu kompliziert findet, kann ja einfach die Routine ausschneiden und daraus eine BAT-Datei, z.B. ELEVEL.BAT, machen. Diese kann dann jederzeit nach einem Programm per CALL aufgerufen werden, um den Errorlevel-Wert auf dem Bildschirm zu zeigen oder um %EL% sonstwie zu verwenden.

Bitte beachten: die Variablen EL und ! sollten in der aufrufenden Batch nicht fuer andere Zwecke verwendet werden. EL muss spaeter noch geloescht werden.

==========

Batch fuer Einsteiger

========== ANSI Sequenzen (Bildschirm) ========== Lektion #17 =

Mit den beruehmten ANSI-Sequenzen lassen sich Farben und Cursor-

Steuerungen in BAT-Dateien realisieren. Ausserdem koennen Tasten umbelegt bzw. mit Befehlen belegt werden (naechste Lektion) und der Video-Modus gewaehlt werden (Naeheres s. DOS-Handbuch).

Voraussetzung ist der Konsol-Treiber ANSI.SYS (oder ein Ersatz), der in der CONFIG.SYS installiert wird. Ueber diesen Treiber gehen alle DOS-Bildschirmausgaben und -Tastatureingaben, so dass hier bestimmte Bildschirm-Aktionen oder Tasten-Umwandlungen gesteuert werden koennen.

Die dazu erforderlichen Befehle sind irgend wann einmal vom American National Standards Institute ("ANSI"), also einer Art DIN-Behoerde der USA, definiert worden.

Die Befehle werden als "ANSI-Sequenzen" per DOS-Ausgabe an den Bildschirm geleitet und vom Konsol-Treiber abgefangen. Damit ANSI.SYS einen Befehl erkennt, muss dieser mit den zwei Code-Bytes ESCAPE und "[" beginnen. Danach folgen die Befehlsinformationen, und als letztes immer ein Buchstabe, der die Art des Befehls bestimmt.

Beim ESCAPE-Zeichen gibt's in kleines Problem: In der BAT-Datei muss ein Byte mit dem Wert 27 (dez) bzw. 1B (hex) gespeichert werden. Wie das eingegeben wird, haengt vom jeweiligen Editor ab. Eines der folgenden Verfahren fuehrt normalerweise zum Erfolg:

* ALT gedrueckt halten und Dezimal-Code (27) im Ziffernblock der Tastatur eingeben

* Prefix Control+P, um zu signalisieren, dass der anschliessende Control-Code (ESC-Taste oder ALT-Prozedur wie oben) als Zeichen gespeichert werden soll.

Was dann auf dem Bildschirm erscheint, ist ein kleiner Pfeil nach links (von IBM wurden auch den Control-Codes bestimmte Zeichen zugeordnet). Manche Editoren zeigen auch ^[- also eine 2-Byte-Sequenz, wobei die eckige Klammer zufaellig das selbe Symbol ist, das als naechstes Byte der ANSI-Sequenz folgen muss.

!! Da hier in diesem Text das ESC-Zeichen nicht verwendet
!! werden kann, wird in allen folgenden Beispielen das Zeichen
!! "#" an dessen Stelle gesetzt, also z.B.:

```
#[40;37m
^---------- hier ESC-Code!
```

Grundsaetzlicher Aufbau einer ANSI-Sequenz

Nach den beiden Erkennungsbytes - #[- folgen dezimale Zahlen-angaben (0...255), und zwar entweder mehrere, nur eine oder auch gar keine. Falls mehrere Zahlenangaben erforderlich sind, muessen sie durch ein Semikolon getrennt werden. Die gesamte Sequenz darf keine Leerzeichen enthalten.

Den Abschluss der ANSI-Sequenz bildet ein Buchstabe. Gross-/Klein-schreibung wird hier unterschieden - also aufpassen!

Beispiele: #[K
        #[3A
        #[30;47;1m

Die ANSI-Sequenzen koennen an beliebiger Stelle in Textausgaben eingesetzt werden, normalerweise im ECHO-Befehl. Grundsaetzlich koennen die Sequenzen aber in DOS-Bildschirmausgaben von jedem Programm verwendet werden.

ANSI-Sequenzen sind nicht auf der DOS-Befehlsebene nutzbar, da ESCAPE hier nicht als Zeichen akzeptiert wird.

Funktionen: Cursor
---

* Cursor relativ bewegen, ausgehend von der aktuellen Position:

  #[1A 1 Zeile aufwaerts
  #[3B 3 Zeilen abwaerts
  #[25C 25 Spalten nach rechts
  #[12D 12 Spalten nach links

  Bei zu grossen Werten wird der Cursor an die aeusserste moegliche Position gesetzt (!). Bei fehlenden Werten gilt 1.

  #[80D an den Zeilenanfang (egal von wo)
  #[A 1 Zeile aufwaerts

* Cursor absolut setzen (Zeile;Spalte).

  #[0;0H Zeile 0, Spalte 0 (linke obere Ecke)
  #[12;33H Zeile 12, Spalte 33

  Bei ungueltigen Angaben wird der Befehl ignoriert.
  Fehlende Angaben werden durch Null ersetzt, z.B.:

  #[H Cursor Home (linke obere Ecke)

  Zu allen Cursor-Positionierungen bitte beachten: Eine ECHO-Ausgabe wird immer mit CR+LF abgeschlossen. Der Cursor geht also an den Anfang der naechsten Zeile. Soll ein Text an eine bestimmte Stelle positioniert werden, muss er unmittelbar auf den Cursor-Befehl folgen, z.B.:

  ECHO #[0;30Hhier ist Position 30 in der oberen Zeile
       -----

* Position speichern/wiederherstellen

  #[s ANSI merkt sich die aktuelle Position
  #[u Gespeicherte Position wird wiederhergestellt

  Wenn der Bildschirm zwischendurch scrollt, stimmt die Position natuerlich nicht mehr.

Funktionen: Loeschen

```
#[2J        Bildschirm loeschen und Cursor Home (0;0)
            (die "2" ist eine Konstante!)
#[K         Zeile ab aktueller Position loeschen
            (grosses "K")
```

Funktion: Farben

Fuer Farben und "Attribute" gibt es folgende Zahlen-Codes:

Zeichen:	Grund:	Farbe:	Attribute
30	40	schwarz	0 weiss auf schwarz
31	41	rot	1 helle Zeichen
32	42	gruen	5 blinkend
33	43	braun/gelb	7 schwarz auf weiss
34	44	blau	8 schwarz auf schwarz
35	45	magenta (violett)	
36	46	zyan (tuerkis)	
37	47	weiss	

Alle Farb-Sequenzen werden durch ein kleines "m" abgeschlossen.
Beispiele:

```
#[0m          normalisieren (weiss auf schwarz)
#[0;1m        hell weiss auf schwarz
#[37;40;1m    das selbe
#[31;40m      rot auf schwarz
#[31;44;1m    hell rot auf blau
```

Alle Farben sowie die Attribute 1 und 5 koennen auch einzeln angegeben werden, wenn die uebrigen Werte unveraendert bleiben sollen:

```
#[37m         weisse Zeichen, Grund und Helligkeit bleiben
#[1m          helle Zeichen, Farben bleiben
```

Achtung!
Die Funktionsweise der Attribute ist nicht besonders logisch.

* Attribut 0 (Null) setzt nicht nur alle anderen Attribute zurueck, sondern auch die Farben (weiss auf schwarz).

* Um 1 (hell) oder 5 (blinkend) zurueckzusetzen, muessen also bei Bedarf die Farben erneut angegeben werden (und zwar NACH der Null), z.B.:

```
#[0;36;40m    normal tuerkis auf schwarz
```

* Attribut 7 soll offiziell "invertieren", macht aber immer schwarz auf weiss, egal was vorher war. Die gesetzten Werte fuer hell und blinkend bleiben erhalten.

* Attribut 8 soll "verstecken" (unsichtbar machen), aber falls Attribut 1 gesetzt war, wird *hell* schwarz auf schwarz produziert (und das ist nicht unsichtbar).

Die ANSI-Farbbefehle gelten fuer alle folgenden Zeichen-Ausgaben sowie fuer Loeschbefehle und neue Zeilen beim Scrolling. (Mit dem ANSI-Befehl selbst wird also zunaechst auf dem Bildschirm nichts veraendert). Beispiel:

```
#[37;44;1m#[2J  hiermit wird der Bildschirm fuer hell weisse
                Zeichen auf blauem Grund vorbereitet und dann
                geloescht.
```

ECHO off

Bei Verwendung von ECHO-Befehlen mit ANSI-Sequenzen sollte darauf geachtet werden, dass ECHO OFF ist, sonst werden alle Befehle doppelt an den Konsol-Treiber gegeben.

ANSI-Grafik

Fuer aufwendige Bildschirm-Gestaltungen empfiehlt es sich, Text und ANSI-Sequenzen in einer separaten ASCII-Datei unterzubringen, die dann per TYPE ausgegeben wird. Das macht die Batch-Datei uebersichtlicher und geht ausserdem schneller. Fuer ganz Eilige gibt es auch TYPE-Ersatzprogramme, die noch um ein Vielfaches schneller sind als DOS.

Uebrigens: ANSI-Grafiken gibt's massenweise in den Mailboxen.

Konsol-Treiber

Als Ersatz fuer ANSI.SYS sind diverse Alternativen im Umlauf, die - wie koennte es anders sein - schneller sind, zusaetzliche Funktionen bieten und weniger Speicher brauchen.

Prompt

Auch bei der Definition des Prompts (PROMPT-Anweisung) koennen ANSI-Sequenzen eingebaut werden. Hier wird jedoch anstelle des ESCAPE-Codes die Zeichenfolge "$e" verwendet. Naeheres s. DOS-Handbuch.

Zum Schluss noch ein Goody:

```
@echo off
echo #[0;5mÛÛ#[7mÛÛ#[0m bitte warten!
::        ^^----^^---------------------- ASCII Code 219
pause > nul  (zum Testen)
echo #[A#[K
```

==========

Batch fuer Einsteiger

========== ANSI Sequenzen (Tastatur) ========== Lektion #18 =

In Lektion 17 wurde bereits darauf hingewiesen, dass sich mit ANSI-Sequenzen auch Tasten umdefinieren oder mit Befehlen belegen lassen. Die Anweisung dazu wird, ebenso wie alle anderen ANSI-Befehle, per DOS-Bilschirmausgabe an den Konsoltreiber geschickt.

Abschluss-Buchstabe der Sequenz fuer Tasten-Umbelegung ist das keine "p".

Nochmal der Hinweis (s. Lektion 17):

!! Da hier in diesem Text das ESC-Zeichen nicht verwendet
!! werden kann, wird in allen folgenden Beispielen das Zeichen
!! "#" an dessen Stelle gesetzt, also z.B.:

```
#[36;156p
^---------- hier ESC-Code!
```

In obigen Beispiel wird der ASCII-Tasten-Code 36 ("$") durch das "œ"-Zeichen (Code 156) ersetzt. Anstelle der dezimalen Zahlen koennen hier aber auch die Zeichen selbst verwendet werden, und zwar in (einfachen oder doppelten) Anfuehrungsstrichen:

```
#["$";'œ'p
```

Funktionstasten

Funktionstasten und sonstige Tasten, die kein ASCII-Zeichen abgeben (z.B. Cursor-Tasten) werden durch zwei Zahlen bestimmt, von denen die erste immer Null ist. Beispiel:

```
#[0;59;0;80p    Die Funktionstaste F1 (Code 59) wird mit
  ---- ----    Cursor abwaerts (Code 80) belegt.
```

Die Tasten-Codes sind in entsprechenden Tabellen (u.a. im DOS-Handbuch) zu finden. Es gibt auch kleine Utilities, die den Code jeder gedrueckten Taste zeigen.

Tastenfolge zuweisen

Die neue Tasten-Belegung darf auch mehrere Zeichen oder Tasten-Codes enthalten, und zwar als Folge von dezimalen Werten oder als String in Anfuehrungszeichen (auch gemischt).

```
#[0;59;"chkdsk";13p
  ---- -----------
```

Die Taste F1 ist hier mit dem Befehl CHKDSK belegt, inklusive anschliessendem ENTER (CR: ASCII Code 13).

Umbelegung rueckgaengig machen

Wenn keine neue Tasten-Zuweisung angegeben wird, stellt ANSI.SYS den Original-Code der Taste wieder her.

```
#["$"p        Originalzustand der $-Taste
#[0;59p       F1-Taste liefert wieder den Code 59
```

Edit-Tasten bei DOS-Eingabe

Funktionstasten sind unter DOS standardmaessig mit bestimmten
Editier-Funktionen belegt, z.B. liefert die Taste F1 schrittweise
die Zeichen des letzten eingegebenen Befehls.

Diese Standard-Funktionen werden natuerlich durch Umbelegung per
ANSI-Sequenz aufgehoben, koennen aber wieder aktiviert werden,
indem der Taste der Original-Code zugewiesen wird.

```
#[0;59;"copy "p        eine Umbelegung
#[0;59;0;59p           DOS-Editierfunktion wiederherstellen
#[0;59p               KEINE Belegung (Taste liefert F1-Code)
```

DOS- und BIOS-Input

Die Tasten-Umbelegungen per ANSI sind nur wirksam, wenn DOS-Input-
Funktionen benutzt werden. Viele Programme verwenden jedoch BIOS-
Aufrufe, und diese gehen NICHT ueber ANSI.SYS.

==========

Batch fuer Einsteiger

========== CHOICE (DOS 6.x) ========== Lektion #19 =

Seit MS-DOS Version 6.0 wird ein Hilfsprogramm mitgeliefert, das
in dieser oder aehnlicher Form (meist unter anderem Namen) bereits
seit etlichen Jahren im Umlauf ist: CHOICE ("Auswahl").

CHOICE haelt den Ablauf an und wartet auf eine Tasten-Eingabe,
wobei die erlaubten Tasten im CHOICE-Befehl vorgegeben werden.
Entsprechend der gedrueckten Taste kann dann per Errorlevel-
Abfrage verzweigt werden. Als Option gibt's die Moeglichkeit,
den Ablauf nach nn Sekunden automatisch fortzusetzen.

Syntax (Quote aus DOS-Hilfe, deutsche Version) ----------

CHOICE [/C[:]Tasten] [/N] [/S] [/T[:]c,nn] [Text]

/C[:]Tasten Angabe der zul„ssigen Tasten. Standard ist JN.
/N   Keine Anzeige eines ?-Zeichens am Ende der Meldung.
/S   Groá-/Kleinschreibung f r Tasten wird beachtet.
/T[:]c,nn   Nach nn Sekunden wird Standardauswahl c ausgef hrt.
Text   Meldung, die angezeigt wird.

ERRORLEVEL ist auf die Position der gedr ckten Taste aus der
Tastenauswahl gesetzt.

Beispiel: CHOICE /C:JN weiter
           if errorlevel 2 goto ENDE

DOS gibt folgenden Prompt aus:

           weiter [J,N]?

^^^^^^-------- wird gem. /C Angaben eingesetzt

Da "J,N" Standard ist, kann in diesem Falle die Option /C auch
weggelassen werden. Je nach gedrueckter Taste wird der Errorlevel
entsprechend der Position (1...) in der /C Option gesetzt. Mit
"N" wird also im obigen Beispiel Errorlevel 2 gesetzt.

Errorlevel 0 wird uebrigens nie zurueckgegeben.

Bei Syntax-Fehlern wird ERRORLEVEL 255 zurueckgegeben, und der
Ablauf wird ohne Pause fortgesetzt. Also: entweder zusaetzlich
Errorlevel 255 abfragen oder keine Fehler machen!

Weitere Erlaeuterungen:

/C[:]Tasten

Die Zahl der zugelassenen Tasten ist nicht begrenzt. Es koennen
aber nur ASCII-Zeichen als Tasten angegeben werden, ohne Trenn-
zeichen dazwischen. Beispiel:

```
CHOICE /C:NWA nochmal, weiter oder abbrechen
```

Control-Codes sind moeglich, wenn sie als Zeichen dargestellt
werden koennen, z.B. ESCAPE wie bei den ANSI-Sequenzen. ENTER und
Leertaste sind nicht verwendbar.

Text
/N
Was nicht mit einem Schraegstrich beginnt, wird als Text (Prompt)
auf dem Bildschirm ausgegeben. Natuerlich darf der Text keine
Schraegstriche enthalten.

Die zulaessigen Zeichen in eckigen Klammern (plus Fragezeichen)
werden automatisch hinzugefuegt, sofern nicht Option /N angegeben
wird. (Option /N empfiehlt sich z.B. bei Verwendung von ESC.)

/T[:]c,nn

Nach nn Sekunden (maximal zweistellige Angabe!) wird automatisch
fortgesetzt. Die Angabe c dient dazu, den Errorlevel so zu setzen,
als wuerde diese Taste gedrueckt. Beispiel:

```
CHOICE Weiter suchen /T:N,5
```

Menues

Mit ECHO-Zeilen kann zuvor ein Menue ausgegeben werden, z.B.:

```
echo   1  Mailer
echo   2  Editor
echo   3  Packer
echo   4  Terminal
echo.
choice  Auswahl (Ziffer), ESC=Ende  /C:1234# /N
::      hier ESC-Code (27) ----------------^
```

Durch zusaetzliche Rahmen und ANSI-Farben laesst sich damit ein
ganz passables Bild produzieren. Auch der Text in der CHOICE-Zeile
darf ANSI-Sequenzen enthalten!

ENTER-Taste als Default

Mit der /C Option laesst sich zwar die ENTER-Taste nicht angeben,
aber man kann die ENTER-Taste per ANSI-Sequenz voruebergehend um-
funktionieren. Beispiel (# fuer ESC-Code):

```
echo #[13;"J"p
CHOICE /C:JN weiter (ENTER)
echo #[13p
```

Damit gibt die ENTER-Taste voruebergehend den Buchstaben "J" ab.
Vorsicht! Falls mit Control+C abgebrochen wird, gibt es keine
ENTER-Taste mehr! (auch ALT + Ziffern-Eingabe hilft nichts). Es
empfiehlt sich daher, fuer den Notfall zuvor eine Funktionstaste
mit dem ENTER-Code zu belegen (z.B. F12: #[0;134;13p).

==========

Batch fuer Einsteiger

========== SETWORD.COM - Systemvariable ========== Lektion #20 =

Haeufigstes Problem bei der Programmierung von Batch-Ablaeufen
ist die Verwendung von System- und Datei-Informationen: Tagesdatum
und Uhrzeit, Volume Label, aktuelles Verzeichnis oder Laufwerk,
Verzeichnispfad einer Datei, Dateidatum und so weiter.

DOS bietet hier leider keine fertigen Loesungen an. Sofern man
nicht mit 4DOS arbeitet, gibt es folgende Moeglichkeiten:

(1) reine Batchloesungen, die meist recht aufwendig und schwer
    zu durchschauen sind,

(2) kleine Utilities, von denen sich mit der Zeit eine ganze
    Menge ansammeln,

(3) Multifunktionsprogramme, also eine Zusammenfassung vieler
    Utilities in einem Programm.

Hier soll eine Loesung vorgestellt werden, die eine Mischung aus
allen dreien ist. Dazu wird ein kleines Programm (SETWORD.COM)
in Verbindung mit normalen DOS-Funktionen und Umleitungen/PIPEs
eingesetzt, so dass auch ein Lerneffekt fuer diesen Batchkurs
dabei abfaellt. Das noetige Programmm wird (als "DEBUG-Skript",
s.u.) gleich mitgeliefert.

SETWORD.COM liest per PIPE (|) den DOS-Output von Befehlen wie
DIR, VOL, DATE, TIME etc. und entnimmt daraus das soundsovielte
Wort - auch ueber mehrere Zeilen hinweg! Die Position des Wortes
(1...) wird zu SETWORD angegeben - ohne Angabe gilt 1. Beispiel:

```
VOL A: | SETWORD 6
```

Der Befehl VOL liefert z.B. (bitte nachzaehlen):

```
Datentraeger in Laufwerk A ist FD-BACKUP
```

Damit nun das Wort im BAT-Ablauf verwendet werden kann, schreibt SETWORD einen entsprechenden SET-Befehl (per > Umleitung) in eine temporaere BAT-Datei, die dann per CALL ausgefuehrt wird. Dabei wird noch der gewuenschte Name der Variablen als Befehlsparameter uebergeben. Das Ganze sieht dann so aus:

```
VOL A: | SETWORD 6 > TMP.BAT
call TMP.BAT LABEL
::          ^^^^^--------- Name der Variablen
```

Nun kann %LABEL% abgefragt oder sonstwie verwendet werden.

Der SET-Befehl wird von SETWORD natuerlich immer mit %1 anstelle des Variablen-Namens produziert, z.B.:

```
SET %1=FD-BACKUP
```

Ohne Umleitung der Ausgabe erscheint der SET-Befehl auf dem Bildschirm - zum Testen genau das Richtige. Ausprobieren:

```
VOL | SETWORD 4              (aktuelles Laufwerk)
CD | SETWORD 1               (aktuelles Verzeichnis)
DIR Datei | SETWORD 11       (Pfad einer Datei)
TRUENAME Datei | SETWORD     (Datei mit komplettem Pfad)
```

Und was passiert, wenn SETWORD keine Eingabe per PIPE erhaelt? Dann wird die Eingabe halt von der Tastatur eingelesen (!), z.B.:

```
SETWORD > TMP.BAT
call TMP.BAT INPUT
```

Verzeichnis der TMP.BAT

In den Beispielen wurde die TMP.BAT im aktuellen Verzeichnis angelegt, und sollte daher anschliessend noch geloescht werden. Das kann man sich sparen, wenn immer das Verzeichnis %TEMP% verwendet wird, z.B.:

```
CD | SETWORD > %temp%\TMP.BAT
call %temp%\TMP UDIR
```

Zum Programm SETWORD

Als Trennzeichen zwischen Woertern in der Eingabedatei gelten Leerzeichen, Komma, Semikolon und CR,LF (Zeilenabschluss). Max. Groesse der Datei: 60 KB, Position gueltig von 1 bis 65535 ;-)

Weitere Anwendungen

DIR liefert eine Menge Informationen. Neben dem aktuellen Laufwerk

kann fuer jede beliebige Datei z.B. der Verzeichnispfad, der Name
ohne Extension, nur Extension, Datei-Datum und -Groesse ermittelt
werden. Bei Bedarf laesst sich auch eine bestimmte Zeile per FIND
herausziehen (hier um die Anzahl Dateien zu ermitteln):

```
DIR *.GIF | FIND "Datei(en)" | SETWORD 1
```

Bei DATE und TIME haelt DOS an und wartet auf eine Eingabe oder
RETURN. Dieses RETURN wird daher per ECHO an DATE/TIME gepiped:

```
ECHO.| DATE | SETWORD 4
```

DOS-Version

Die Beispiele oben gelten fuer MS-DOS 5.0 (deutsch). Abweichungen
im Output bei anderen Versionen moeglich, also bitte ausprobieren
und abzaehlen...

Und hier ist das Proggi:

```
----------------------------------------------schnipp----------------
nSETWORD.COM
e0100 BE 81 00 AC 3C 20 74 FB 4E 2B C0 2A FF B9 0A 00
e0110 8A 1C 46 80 EB 30 80 FB 09 77 06 F7 E1 01 D8 73
e0120 EF 85 C0 74 01 48 50 BE A2 01 2B DB 89 F2 B9 00
e0130 F0 B4 3F CD 21 8B D8 C6 00 1A BA 9B 01 BB 01 00
e0140 B9 07 00 B4 40 CD 21 5D 2B C9 AC 3C 20 74 FB 3C
e0150 2C 74 F7 3C 3B 74 F3 3C 0D 74 EF 3C 0A 74 EB 4E
e0160 89 F2 80 3C 1A 74 1E 46 8A 04 3C 20 74 10 3C 2C
e0170 74 0C 3C 3B 74 08 3C 0D 74 04 3C 0A 75 E4 3B CD
e0180 74 03 41 EB C5 3B CD 75 08 8B CE 29 D1 B4 40 CD
e0190 21 B4 3E CD 21 2A C0 B4 4C CD 21 73 65 74 20 25
e01A0 31 3D
rCX
A2
w
q
----------------------------------------------schnapp----------------
```

Das Skript enthaelt den kompletten Programm-Code als Folge von
(hex) Bytes mit den zugehoerigen Eingabebefehlen fuer DEBUG.

Zur Erstellung der COM-Datei bitte wie folgt vorgehen:

* Skript zwischen den Trennlinien ausschneiden und in eine Datei
  kopieren, z.B. SETWORD.SCR

* Auf der Befehlsebene eingeben: DEBUG < SETWORD.SCR

Die Eingabe-Datei wird damit von DEBUG abgearbeitet, und das
Resultat wird in die Datei SETWORD.COM (ins aktuelle Verzeichnis)
geschrieben.

Das war's

==========