
Dieser Batch-Kurs soll die Grundkenntnisse zur Batch-Programmierung unter MS-DOS vermitteln.

Der Kurs ist 1994 erstmals im FIDO Echo BATCH.GER erschienen, und wurde mehrmals bearbeitet, zuletzt unter Beruecksichtigung von WIN95 (MS-DOS 7.x). Historische DOS-Versionen vor 3.3 werden nicht mehr ber cksichtigt, da sie wohl kaum noch benutzt werden.

Grundlagen der Befehlseingabe unter MS-DOS (Command Prompt) und Kenntnisse der Eigenschaften von Dateinamen und Pfaden werden hier vorausgesetzt. Die Standardbefehle von MS-DOS sollten auch bekannt sein. Erl uuterungen k nnen bei Bedarf auf der Befehlsebene mit /? abgerufen werden.

Der vorliegende Text ist ein DOS-Text, also mit Umlauten gem. DOS-Zeichensatz.

Fragen, Hinweise, Anregungen bitte im FidoNet Echo BAT.GER oder per Mail:

Horst Schaeffer, FIDO 2:2480/13.75
horst@confusion.rmc.de

Lektion Thema

- 1 BAT Abl ufe, allgemeine Informationen
- 2 ECHO, @
- 3 ECHO Umleitung
- 4 PAUSE
- 5 REM - Bemerkungen
- 6 Befehlsparameter %1..
- 7 GOTO Label
- 8 Bedingungen, IF EXIST
- 9 IF Wort1==Wort2
- 10 IF ERRORLEVEL n
- 11 CALL oder nicht CALL
- 12 FOR (Schleifen)
- 13 Umgebungsvariable, SET
- 14 Umleitung: Ausgabe
- 15 Umleitung: Eingabe, Pipe
- 16 Errorlevel in Variable
- 17 ANSI Sequenzen (Bildschirm)
- 18 ANSI Sequenzen (Tastatur)

19 CHOICE - Auswahl
20 SETWORD.COM - Systemvariable

===== BAT Abl ufe ===== Lektion #1 =====

Die Grundidee der Batch-Abl ufe ("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 M glichkeit, Befehle aus einer ASCII-Datei zeilenweise abzuarbeiten, so als w rden sie nacheinander auf der Befehlsebene eingegeben. Eine solche Datei muá den Namenszusatz "BAT" haben, und ist damit, neben COM- und EXE-Dateien, eine weitere Variante der "ausf hrbaren" Dateien, die als Befehl auf der DOS-Ebene gestartet werden k nnen.

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 Ausf h rung (IF.. GOTO, Labels)
- * Schleifen (FOR..)
- * CALL weiterer BAT-Abl ufe
- * Ein-/Ausgabe-Umleitungen

Die BAT-Sprache ist allerdings von Microsoft u áerst sp rlich ausgestattet worden. Bildschirmdialoge, etwa f r Eingaben oder Auswahl durch den Benutzer sind nur in Ans tzen vorhanden.

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

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

Soviel zu Batch-Erweiterungen. Hier in diesem Einsteiger-Kurs soll es nur um das reine MS-DOS gehen, und zwar ab Version 3.3.

Itere DOS-Versionen sind im Funktionsumfang der BAT-Verarbeitung noch st rker eingeschr nkt und spielen heute keine Rolle mehr.

Bevor in den nächsten 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, z.B. den von MS mitgelieferten EDIT. Auch ein WIN Editor kann verwendet werden, allerdings werden z.B. bei NOTEPAD deutsche Umlaute nicht nach DOS-Standard gespeichert.

In einer BAT-Datei dürfen beliebige Zeichen verwendet werden, mit Ausnahme des EOF-Codes (Ctrl-Z, ASCII 26), der generell als Ende einer ASCII-Datei interpretiert wird. Jede Zeile wird durch CR+LF (ASCII 13,10) abgeschlossen.

Einige Sonderzeichen haben in BAT-Dateien eine besondere Funktion. Die Wichtigsten: das Prozentzeichen ist für Variable, die Größer- und Kleiner-Zeichen sowie das "!" sind für Ein-/Ausgabeumleitungen reserviert. Nehmes bei den jeweiligen Befehlen.

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

Die Länge der Batchzeilen ist bis MS-DOS 6.x auf 127 Zeichen begrenzt, und zwar nachdem eventuelle Variable eingesetzt wurden (s. Lektion über Variable). Ab Version 7 (WIN95) darf die Batchzeile 255 Zeichen lang sein, intern (nach Einsetzen der Variablen) sogar 1023 Zeichen (s.a. COMMAND Optionen /U und /L).

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 nächste auszuführende Zeile der BAT-Datei verwaltet.

Für jede einzelne Befehlszeile des Batch-Ablaufs wird von DOS die BAT-Datei geöffnet, gelesen und geschlossen. Grundsätzlich kann also eine BAT-Datei auch verändert werden, während sie läuft, was man allerdings mit Vorsicht genießen sollte. Auf jeden Fall sollte man vermeiden, in einem Batch-Ablauf den Editor aufzurufen, um eben diese BAT-Datei zu ändern. Das bringt die Abarbeitung mit hoher Wahrscheinlichkeit durcheinander!

Fehler in BAT-Ablauf, Abbruch

Bei ungültigen Befehlen oder Syntax-Fehlern gibt DOS eine Meldung aus und setzt die Verarbeitung (wenn möglich) mit der nächsten

Zeile fort. Bei schweren Fehlern wird der BAT-Ablauf abgebrochen.

Da DOS nicht die Zeilen-Nummer des Fehlers angibt, muß man notfalls die einzelnen Befehlszeilen auf dem Bildschirm ausgeben lassen, um den Ablauf zu verfolgen (s. Lektion ECHO). Ab MS-DOS 6.x ist auch ein Einzelschritt-Modus verfügbar. Dazu wird die Batchdatei mit folgendem Aufruf gestartet (s. COMMAND Optionen):

```
COMMAND /Y/C batch.BAT
```

COMMAND zeigt dann jede einzelne Zeile und fragt, ob sie ausgeführt werden soll oder nicht.

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

```
Stapelverarbeitung abbrechen? (J/N)
```

Ob und wie sich ein gerade ausgeführtes PROGRAMM abbrechen läßt, ist natürlich eine andere Frage.

===== ECHO, @ ===== Lektion #2 =====

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 Ausführung auf dem Bildschirm ausgegeben werden (ON) oder nicht (OFF). Ausgaben erfolgen immer mit dem DOS-Prompt, genau so wie es auf der Befehlsebene aussehen würde.

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

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

```
ECHO=off  
ECHO = on
```

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

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

ECHO ist eingeschaltet

Das braucht man so gut wie nie, und wenn diese Meldung auf dem Bildschirm erscheint, dann meistens unbeabsichtigt (s.u.).

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-Ausgabe-Gerät). Beispiel:

ECHO Dies ist eine ECHO-Zeile

Durch Leerzeichen läßt sich der Text einrücken:

ECHO Dieser Text ist eingerückt

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

ECHO.Fertig

ECHO.OFF (OFF ist hier Text!)

Auf diese Weise läßt sich auch eine leere Zeile, also ein extra Zeilenvorschub, produzieren (ECHO allein bewirkt was anderes! s. ECHO-Status). Beispiel:

ECHO.

Der Punkt muß ohne Leerzeichen anschließén, sonst ist er Text!

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

Ganz wichtig:

ECHO Texte dürfen nicht die Zeichen "" oder "" enthalten, da diese für Umleitungen/Pipes reserviert sind.

Wer mal nicht dran denkt, wird sich über äußerst merkwürdige Fehlermeldungen oder unerwartet auftauchende Dateien wundern.

ECHO-Ausgaben dürfen 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 (at-sign)

Während das ECHO ON ist, kann es bei Bedarf für einzelne Zeilen abgeschaltet werden, indem ein Klammeraffe an den Anfang der Zeile gesetzt wird. Spezielles Beispiel:

```
@ECHO OFF
```

Ohne den Klammeraffen würde der Befehl selbst noch ge-echo't, bevor er wirksam wird (!) Dieser Befehl steht üblicherweise am Anfang jeder Batch-Datei.

ECHO auf der Befehlsebene

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

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

Weitere Informationen

Interessant wird der ECHO-Befehl erst mit Umleitungen/Pipes. Außerdem können mit Hilfe der ANSI-Sequenzen Farben und Cursor-Steuerungen (u.a.) ins Spiel gebracht werden. Mehr dazu später.

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

Das Thema "Umleitungen" soll eigentlich erst später 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 Gerät - wie LPT1, also an den Drucker - umgeleitet werden. Beispiel:

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

Genau wie auf dem Bildschirm, wird auch in diesen Fällen 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 neu angelegt, d.h. wenn diese Datei bereits vorhanden war, wird sie ohne Warnung überschrieben!!

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

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

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

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

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

Mit der PAUSE-Anweisung wird der BAT-Ablauf angehalten. DOS gibt dazu standardmäßig diese (oder eine ähnliche) Aufforderung aus:

Weiter mit beliebiger Taste . . .

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

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

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

Beim PAUSE-Befehl werden weitere Angaben in der selben Zeile ignoriert. Allerdings werden eventuelle Umleitungszeichen von DOS konsequent abgearbeitet, was selten Sinn macht - bis auf diesen Fall:

```
PAUSE > NUL
```

Damit wird die Ausgabe der Standardaufforderung ins Nichts umgeleitet, falls man etwas anderes per ECHO formuliert hat.

Hinweis:

Bis DOS Version 5.0 war es nicht möglich, die gedrückte Taste anschließend abzufragen. Auch eine automatische Fortsetzung nach einer gewissen Zeit war nicht vorgesehen. Ab DOS 6.0 wird dafür

das Hilfsprogramm CHOICE mitgeliefert (s. Lektion CHOICE).

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

REM dient zum Schreiben von Bemerkungszeilen, also von Kommentaren, die nicht ausgegeben werden.

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

Alternativ können Bemerkungszeilen mit dem für Sprungmarken vorgesehenen Doppelpunkt geschrieben werden, am besten mit Doppelpunkt (weitere Informationen dazu s. Lektion LABEL).
Beispiel:

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

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

```
:: Syntax: MOP [/M/X]
```

Empfehlung daher: Vergeßt REM für Bemerkungen. Die Doppelpunkt-Lösung ist sowieso schneller, auch wenn das heute praktisch keine Rolle mehr spielt...

Nur für einen speziellen Fall ist REM wirklich nützlich: zum Anlegen einer leeren Datei per Umleitung, z.B.:

```
REM > XXX.DAT  
REM mach mal eine leere > XXX.DAT
```

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

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

In BAT-Abläufen werden oft variable Angaben gebraucht, die erst mit dem jeweiligen Aufruf festgelegt werden sollen.

Dazu werden in der BAT-Datei "Platzhalter" eingebaut. Beim Start einer BAT-Datei werden dann die aktuellen Werte einfach zusätzlich in der Befehlszeile angegeben (Befehlsparameter).

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

Ein Beispiel (Datei EDR.BAT):

```
attrib -R %1  
EDIT %1  
attrib +R %1
```

Aufruf:
EDR ANY.TXT

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

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

Trennzeichen, Sonderzeichen

Zur Trennung von Befehlsparametern können auch Komma oder Semikolon verwendet werden. Sie werden praktisch durch Leerzeichen ersetzt. (Auch das Gleich-Zeichen "=" wird auf die selbe Weise behandelt.) Leere Parameter können auf diese Weise NICHT übergeben werden. Beispiel:

```
XXX.BAT eins,,,vier ergibt: %1=eins  
%2=vier
```

Auch die Übergabe eines Parameters, in dem Leerzeichen enthalten sein sollen, ist bis einschließlich MS-DOS 6.x nicht möglich, denn Anführungszeichen werden wie ganz normale Zeichen behandelt. Beispiel (bis MS-DOS 6.xx):

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

Ab MS-DOS 7.0 werden (bedingt durch die langen Dateinamen) die Anführungszeichen interpretiert. Sie werden aber nicht entfernt. Beispiel (ab MS-DOS 7):

```
AAA.BAT "Eigene Dateien" ergibt: %1="Eigene Dateien"
```

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

```
@echo off  
ECHO.%1  
ECHO.%2  
(etc.)
```

Dann kann diese BAT-Datei mit allen möglichen Kombinationen getestet werden. (Der Punkt nach ECHO sorgt dafür, daß bei leeren Parametern nicht die Meldung kommt "ECHO ist ausgeschaltet".)

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 für spezielle Tricks zu gebrauchen.

SHIFT-Befehl

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

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

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

Weitere Themen

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

===== GOTO Label ===== Lektion #7 =====

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

```
:WEITER
```

Mit dem Befehl GOTO WEITER irgendwo in der BAT-Datei würde also der Ablauf an dieser Stelle fortgesetzt.

Ein GOTO wird natürlich erst interessant, wenn er bedingt eingesetzt werden kann, z.B.:

```
IF not exist C:\COMMAND.COM goto FEHLER
```

Aber bevor wir auf die IF-Konstruktionen eingehen, zun chst mal ein n tzliches GOTO-Beispiel ganz ohne IF.

Beispiel: GO.BAT

Angenommen es gibt Programme in verschiedenen Verzeichnissen, die jederzeit zu starten sein sollen, ohne daá man deswegen alle diese Verzeichnisse im PATH angeben will (der ist ja eh' schon lang genug).

Man braucht also eine BAT-Datei, die jeweils den Verzeichniswechsel (CD) und dann den Aufruf ausf hrt. F r jedes Programm eine eigene BAT-Datei zu schreiben ist kein Kunstst ck. Aber wie kann man verschiedene Aufrufe in *einer* BAT-Datei steuern?

Ganz einfach. Nennen wir die BAT-Datei: GO.BAT
Sie enth lt zun chst 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 Abk rzungen, die man sich leicht merken kann. F r jedes Programm wird nun eine kleine Aufruf-Routine geschrieben, z.B.:

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

Nat rlich k nnen auch weitere Befehle (z.B. das Laden residenter Programme) in diese kleine Routine aufgenommen werden.

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

Damit lassen sich nun alle m glichen Programme nach diesem Muster aufrufen:

```
GO TERM
```

Falls allerdings kein Ziel oder ein falsches angegeben wird, meldet DOS "Sprungmarke nicht gefunden" und bricht ab. Das ist aber kein Problem. Eine aufwendige Pr fung auf g ltige Befehle w rde am Ende auch nichts anderes machen...

Jetzt noch einige wichtige Hinweise f r den Umgang mit Labels.

G ltige Labels

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

Groß-/Kleinschreibung ist egal. Auch z.B. "goto weiter" findet die Marke ":WEITER". Umlaute sollten allerdings vermieden werden, weil dabei die Umwandlung nur in eine Richtung funktioniert (Bug).

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

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

Ersatz für REM

Labels können hervorragend für Bemerkungszeilen mißbraucht werden, zumal dabei ausnahmsweise auch die Umleitungszeichen bedenkenlos verwendet werden dürfen. Damit man gleich sieht, daß es sich nicht um eine echte Sprungmarke handelt, hat sich der doppelte Doppelpunkt eingebürgert, z.B.:

:: Syntax: MOP [/M/X]

Sprungziel des GOTO-Befehls

Auch im GOTO-Befehl darf der angegebenen Marke ein Doppelpunkt vorangestellt werden (Geschmacksache). Beispiel:

GOTO :WEITER

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

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

IF [not] exist
IF [not] errorlevel
IF [not] ==

Nach einer solchen Bedingung kann jede beliebige BAT-Anweisung stehen, z.B. ein GOTO, ein DOS-Befehl oder ein Programm-Aufruf. Die Anweisung wird nur ausgeführt, wenn die Bedingung WAHR ist.

Das optionale NOT bedarf wohl keiner weiteren Erklärung.

Da IF EXIST am einfachsten zu handhaben ist, soll zunächst 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 können auch die Jokerzeichen "?" und "*" verwendet werden, und natürlich sind Laufwerk- und Pfadangaben erlaubt.

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

Unter DOS 7 sind Anführungszeichen erlaubt, unter WIN95 natürlich auch lange Dateinamen:

```
IF EXIST "\Eigene Dateien\Word\bla bla.doc" ...
```

Ungültige Datei-Angaben

Bei Ungültigen Datei-Angaben oder Pfaden gilt die Datei als NICHT vorhanden. Es gibt also keine Fehlermeldung. Allerdings gibt's bei Laufwerken mit nicht eingelegtem Datenträger die übliche Meldung:

Nicht bereit beim Lesen von Laufwerk ...

Verzeichnis Vorhanden?

Das Vorhandensein eines Verzeichnisses kann leider nicht direkt festgestellt werden, da DOS nur nach Dateinamen sucht. Man könnte zwar prüfen, ob Dateien im gewünschten Verzeichnis sind, aber damit wird ein evtl. leeres Verzeichnis nicht ermittelt.

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

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

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

UND-Verknüpfung

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

IF exist TEST.BAK if not EXIST TEST.NEU ren TEST.BAK TEST.NEU
(und..) (dann..)

Variabler Datei-Ausdruck

Mit IF exist können auch variable Angaben überprüft werden:

IF NOT exist %1 ECHO %1 ist nicht vorhanden

Allerdings sollte man sichergehen, daß der angegebene Befehlsparameter nicht leer ist, denn dann bekommt man folgende Zeile zu lesen:

IF NOT exist ECHO ist nicht vorhanden

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

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

Nicht vorhandene (leere) Variable bzw. Befehlsparameter lassen sich mit der Vergleichsbedingung feststellen (nächste Lektion).

===== IF Wort1==Wort2 ===== Lektion #9 =====

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

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 ein kleines "a:" als Befehlsparameter angegeben wurde, ist die Bedingung NICHT wahr. Da hilft nur eines: beide Möglichkeiten abfragen.

Leere Parameter

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

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

Resultat: "Syntax-Fehler".

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

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

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. Klammern oder Anführungszeichen (sie haben im BATCH nicht die sonst übliche Bedeutung und werden wie Buchstaben behandelt).

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

Jetzt ist auch klar, wie man leere Parameter abfragt:

Beispiel: IF [%1] == [] 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 Möglichkeit besteht, daß Variable bzw. Befehlsparameter leer sein könnten.

Anmerkung:

In der Praxis ist alles halb so wild, weil oft Variable abgefragt werden, die man selbst vorher gesetzt hat. Damit kann man auch selbst dafür sorgen, daß die Variable nicht leer ist, und die Schreibweise (groß/klein) ist natürlich auch bekannt.

Hinweis zu MS-DOS 7:

Zwar dürfen Parameter in Anführungszeichen (lange Dateinamen) auch Leerzeichen enthalten, aber der IF.. Vergleich nimmt darauf leider keine Rücksicht. Nehmes beim Thema Variable.

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

Jedes Programm gibt einen Return-Code ("Beendigungscode") an das aufrufende Programm zurück, normalerweise also an COMMAND.COM. Dieser Return-Code ist ein Byte und kann daher die Werte 0...255 haben.

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

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

```
IF ERRORLEVEL n
```

bedeutet: IF Return-Code \geq n (größer oder gleich n)

So kann mit einer einzigen Abfrage (IF ERRORLEVEL 1) festgestellt werden, ob überhaupt ein Fehler aufgetreten ist, ohne daß alle anderen (möglicherweise gelieferten) Codes abgefragt werden müssen.

Bei allen ERRORLEVEL-Abfragen muß man sich also der "größer/gleich" Logik bewußt sein. Wenn auf verschiedene Return-Codes reagiert werden soll, ist eine Abfrage in absteigender Folge erforderlich.

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

```
IF errorlevel 9 goto Label-9
IF errorlevel 3 goto Label-3
IF errorlevel 1 goto Label-1
:: hier bleibt nur noch errorlevel 0
```

brigens:

Die Abfrage "IF [not] ERRORLEVEL 0 ..." ist absolut witzlos, denn größer 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 so ohne weiteres möglich.

Andere Verwendung des Return-Codes

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

DOS-Befehle

DOS-Befehle, also in COMMAND.COM integrierte Befehle ohne eigene Programmdatei, geben überhaupt keinen Return-Code zurück, auch nicht Null. Der zuvor produzierte Return-Code bleibt erhalten!

Das hat den Vorteil, daß Befehle wie ECHO den letzten Return-Code nicht verändern, aber leider auch den Nachteil, daß man auf diese Weise nicht feststellen kann, ob z.B. ein COPY oder DEL erfolgreich war.

Sonstige DOS-Dienstprogramme

Der FIND-Befehl gibt den erwarteten Errorlevel zurück: 0=gefunden, 1=nicht gefunden - allerdings erst seit MS-DOS Version 6.xx. Wer ähnliches z.B. von FC (Dateivergleich) erwartet, sieht sich leider getuschelt.

Grundsätzlich sollte man sich also genau informieren, welche Return-Codes geliefert werden, wenn Errorlevel-Abfragen benutzt werden. Entsprechende Informationen sind teils in alten DOS-Handbüchern, teils auch in der HELP-Funktion (ab MS-DOS 6.xx), meist aber gar nicht zu finden. Notfalls also selbst testen. Hinweise dazu in späteren Lektionen.

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

In einem BAT-Ablauf lassen sich nicht nur .COM und .EXE Programme aufrufen, sondern auch andere .BAT Abläufe. Verwendet man nun den Namen einer BAT-Datei als Befehl, dann sollte man meinen, daß DOS - wie bei Ausführung 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-Abläufen gab. Die Beendigung eines Batch-Ablaufs bei einem weiteren Aufruf war also zwangsläufig. 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 zurückzukehren

CALL XXX.BAT kehrt nach Ausführung von XXX.BAT zurück.

Befehlsparameter können in jedem Falle übergeben werden, auch weitergegeben werden, z.B.:

CALL update.bat %1 A:

Nach Ausführung einer BATCH mit CALL sind selbstverständlich die Befehlsparameter des ersten Ablaufs wieder verfügbar, denn DOS führt für jede BAT-Ebene einen eigenen Steuerblock im Arbeitsspeicher, wo u.a. auch die jeweiligen Befehlsparameter abgelegt sind.

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

Rekursion

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

Manchmal geschieht so etwas unbeabsichtigt. Übersichtliche CALL-Strukturen sind daher immer eine gute Investition.

Rekursionen können 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 Überraschungseffekten...

QUIT

Ein spezieller BAT-Aufruf ohne CALL hat sich als recht nützlich zum Beenden eines BATCH-Ablaufs 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 eine einzeilige BAT-Datei mit folgendem Inhalt 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 ausgeführt. Oft läßt sich auf diese Weise eine extra GOTO-Konstruktion einsparen. Benutzungsbeispiele:

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

Abbruch bei GOTO-Fehler

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

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

Die FOR-Konstruktion ermöglicht die mehrfache Ausführung eines Befehls mit einem variablen Argument. Die Argumente werden nacheinander 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 Schlüsselwörter "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 (für Befehlsparameter reserviert). Da überhaupt verschiedene Buchstaben möglich sind, macht eigentlich keinen Sinn, denn diese Variable ist nur innerhalb der FOR-Zeile gültig, und ein mehrfaches FOR (Schachtelung) ist nicht zulässig.

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

Anmerkung: DOS ersetzt in einer BAT-Zeile vorab grundsätzlich 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 möglich. Nur ein weiteres FOR ist nicht möglich.

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 für jeden Dateinamen ausgeführt, 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 Zusätzen .TXT und .DAT auf dem Bildschirm ausgegeben.

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

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

Trennzeichen in der Liste

Außer Leerzeichen können Komma, Semikolon oder sogar das "=" Zeichen verwendet werden. Ein Argument darf also diese Zeichen nicht enthalten.

Etwas ganz Merkwürdiges geschieht beim Schrägstrich.
Ausprobieren: FOR %%a in (TEST/L12) do ECHO %%a.

Dies wurde als undokumentiertes "Feature" für allerlei Tricks genutzt, funktioniert aber seit MS-DOS 7.x nicht mehr. Vielleicht war es doch nur ein Bug.

Befehl mit IF

Alternative Bedingungen (ODER) lassen sich 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 Laufwerk %1 ist ung ltig!
:OK
```

Hier wird getestet, ob der Befehlsparameter %1 ein g ltiges Diskettenlaufwerk enth lt. (Die Anf hrungszeichen verhindern 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 dafr nicht ausgef hrt, z.B. hier:

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

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

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

Vorsicht!

F r die Variable k nnen Groá- oder Kleinbuchstaben verwendet werden, aber innerhalb einer FOR-Zeile muá man bei der Schreibweise bleiben! Das hier funktioniert nicht:

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

FOR mit CALL

Was eigentlich nicht geht, n mlich FOR-Verschachtelungen oder Ausf hrung MEHRERER Befehle, l ást sich durch CALL einer weiteren BAT-Datei in der FOR-Schleife realisieren.

Bug: Errorlevel & GOTO

Eine Errorlevel-Abfrage nach einer Programmausf hrung sieht so (hnlich) aus:

```
IF ERRORLEVEL 10 goto L-10
IF ERRORLEVEL 7 goto L-7
IF ERRORLEVEL 5 goto L-5
(u.s.w....)
```

Zur Erinnerung: wegen der Größer/Gleich-Logik müssen Errorlevels absteigend abgefragt werden!

Wäre es nicht auch möglich, dies per FOR-Schleife zu erledigen?
Naheliegende Lösung:

```
FOR %%a in (10,7,5,3,2,1,0) do IF ERRORLEVEL %%a GOTO L-%%a
```

Wer es probiert, wird feststellen: es funktioniert nicht, und das hat folgenden Grund:

Bei einem GOTO wird von DOS zwar sofort die jeweilige Sprungmarke aufgesucht, aber diese Stelle wird nur vorgemerkt, und der Ablauf wird dort erst dann fortgesetzt, wenn die Ausgangszeile komplett abgearbeitet ist (!)

Leider kommt DOS nicht auf die Idee, daß eine FOR-Schleife mit dem ersten GOTO abgebrochen werden sollte. Statt dessen werden munter Sprungmarken gesucht (und die Adresse zur Fortsetzung des Ablaufs immer wieder beschrieben), solange die Bedingung WAHR ist - im obigen Beispiel also bis ganz zuletzt.

Entgegen der üblichen Logik muß hier also die Reihenfolge der Abfrage umgekehrt werden:

```
FOR %%a in (0,1,2,3,5,7,10) do IF ERRORLEVEL %%a GOTO L-%%a
```

Wenn nun z.B. der Errorlevel 3 ist, dann ist die Bedingung bei 3 zum letzten Male WAHR, und die dabei gefundene Sprungmarke bleibt bis zur kompletten Abarbeitung der Schleife gültig.

Aus diesem Grunde ist hier auch die Abfrage auf Errorlevel 0 ausnahmsweise sinnvoll, ja sogar unverzichtbar.

WIN95, LFNFOR Bug:

Bei Verwendung von Jokerzeichen in der Argumentliste werden nur DOS-Dateinamen und Pfade erkannt und behandelt. Es gibt zwar den neuen Befehl LFNFOR ON/OFF, um die Behandlung langer Dateinamen in einer FOR-Schleife ein/aus zu schalten, aber diese Funktion hat (zumindest bei DOS 7.0) einen Bug:

Wenn die Klammer einen Verzeichnispfad enthält, wird nur für die erste Instanz (gefundene Datei) der Pfad mitgeliefert - die weiteren kommen OHNE Pfad.

Verwendbar ist LFNFOR also nur im aktuellen Verzeichnis (also bei Dateibegriff ohne Pfad), oder aber wenn nur die erste Instanz gebraucht bzw. nur eine erwartet wird.

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

In BAT-Abläufen lassen sich Variable benutzen, die als sogenannte "Umgebungsvariable" in einem besonderen Speicherbereich geführt werden. Auf Konzeption und Bedeutung dieser Umgebungsvariablen-Speicher soll hier nicht weiter eingegangen werden. Wichtig ist zunächst nur, daß dieser Bereich eine Liste von Zuweisungen enthält.

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

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

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

COMMAND.COM bedient sich der Umgebungsvariablen, um bestimmte Informationen 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 können diese Informationen nutzen.

Die wichtigste Verwendung bietet sich jedoch in BAT-Dateien:

* Mit der SET-Anweisung können neue Zuweisungen aufgenommen, bestehende geändert oder gelöscht werden (möglich auch auf der Befehlsebene).

Beispiel: SET WORT=das (neue Zuweisung oder Änderung)
SET WORT= (Lösung: Leer-Zuweisung)

* Durch Angabe des Variablennamens, eingeschlossen in Prozentzeichen, kann die aktuelle Zuweisung an jeder beliebigen Stelle der BAT-Datei eingesetzt werden. (Auf der Befehlsebene können diese Variablen erst seit MS-DOS 7 verwendet werden.)

Beispiel: ECHO Die Variable WORT enthält "%WORT%"

Im BAT-Ablauf untersucht DOS jede Zeile vor der Ausführung auf %-Zeichen, um Variable durch die zugewiesene Zeichenfolge zu ersetzen. Dabei ist es gleichgültig, 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 Zuzuglich 7%% MwSt.

Regeln für Variablen-Namen

Bei Variablen-NAMEN wird Groß-/Kleinschreibung NICHT unterschieden (es wird generell in Großbuchstaben umgewandelt). Die zugewiesene Zeichenfolge wird dagegen immer unverändert gespeichert.

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

Bitte nicht versehentlich Variablenamen 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. Um Problemen aus dem Wege zu gehen, sollten aber nur Zeichen verwendet werden, die auch in Dateinamen gültig sind (ausgenommen "%"). Eine besondere Falle für Ahnungslose ist das Leerzeichen am Ende des Namens:

```
SET TEST = JA
```

^

Hier ist der Variablen-Name nicht "TEST", sondern "TEST ", und die Verwendung von %TEST% funktioniert natürlich nicht... Also: zwischen Name und "=" bitte KEIN Leerzeichen!

Hinweis:

WINDOWS setzt einige Variable mit klein geschriebenem Namen, z.B. windir=C:WINDOWS. Damit können diese Variable mit dem SET-Befehl weder verändert noch gelöscht werden. Seit MS-DOS 7 können diese Variablen aber wenigstens verwendet werden, wobei die Schreibweise (groß/klein) egal ist (%windir" oder %WINDIR%).

Inhalt der zugewiesenen Zeichenfolge

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

Beispiel: SET TEXT=Bitte Taste drücken!

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

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

Eine Variable kann auch eine ganze Parameter- oder Argument-Liste enthalten, die sich dann mit CALL oder FOR wieder zerlegen lässt. %PATH% ist so eine Liste. Beispiele:

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

```
FOR %%a in (%PATH%) do if exist %%a\ARJ.EXE set ARJ_PATH=%%a
```

Anmerkung: Bitte an die Länge der PATH-Variablen denken!. Die Zeile darf NACH dem Einsetzen der Variablen maximal 127 Zeichen lang sein, jedenfalls bis MS-DOS 6.xx. Seit MS-DOS 7 sind bis zu 1023 Zeichen möglich (sofern nicht ein kleinerer Wert mit COMMAND Option /L festgelegt wurde).

Verkettung, Kombinationen

Natürlich 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 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 enthält, können IF-Abfragen wie bei Befehlsparametern gemacht werden. Aber auch hier gelten die Grundregeln:

- * Groß-/Kleinschreibung wird unterschieden
- * Leere Variable gefährden die Syntax

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

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

```
IF "%LW%"=="A:" goto OK
IF [%LW%]==[] goto FEHLER
```

Wenn allerdings eine Variable mehrere Wörter enthält (getrennt

durch Leerzeichen, Komma, Semikolon), dann wird's gef hrlich.

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

* Bei mehreren W rtern 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 F llen tabu!

Einzigste Ausnahme: die Abfrage auf leere Variable funktioniert trotzdem, z.B.:

```
IF [%VAR%]==[] goto ...
```

Mal nachdenken warum.

Verf gbarer Speicherplatz

Der Speicherplatz f r Umgebungsvariable ist begrenzt. Es wird daher dringend empfohlen, am Ende einer BAT-Datei alle nicht mehr ben tigten Variablen zu l schen, damit kein M ll angeh uft 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 so ohne weiteres entlocken. (Daf r gibt es aber kleine Utilities.)

Da der Speicherplatz COMMAND.COM zugeordnet ist, kann die Gr ae des Speichers nur beim Laden des Befehlsprozessors beeinfluat werden.

Dazu dient die Option /E der SHELL-Anweisung in der CONFIG.SYS - Beispiel (800 Bytes gew nscht):

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

Schaut mal bei HELP COMMAND nach...

Weitere COMMAND-SHELL

Beim Laden einer weiteren COMMAND-SHELL wird ein eigener Speicherbereich f r Umgebungsvariable angelegt. In diesen werden alle Zuweisungen der ersten SHELL kopiert. Aber umgekehrt wird nach Verlassen der SHELL (mit dem EXIT-Befehl) nichts b ernommen!

Standardm a ig wird von DOS nur recht wenig freier Speicher in einer weiteren COMMAND-Shell zur Verf gung gestellt. Erst seit MS-DOS 7 wird die Gr ae des Bereichs b ernommen, so daa die gleiche Menge an

freiem Speicher verfügbar ist.

Bei Bedarf kann aber auch bei jeder weiteren Shell die Option /E verwenden werden, wobei natürlich der bereits verbrauchte Speicher (s.o.) zu berücksichtigen ist.

Oft wird jedoch die COMMAND-Shell von einem Programm (z.B. NC) bereitgestellt, das dann auch für die Speichergröße zuständig ist. Gute Programme schaffen ausreichenden Speicherplatz, oder lassen die Größe per Setup einstellen.

Für die COMMAND-Shell unter WIN 3.x läßt sich die Größe des Bereichs in der SYSTEM.INI, [NonWindowsApp], CommandEnvSize=.... einstellen. Bei WIN95: s. Eigenschaften/Speicher.

System-Variable

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

PATH Suchpfade für ausführbare Dateien
TEMP Verzeichnis für temporäre Dateien
COMSPEC Befehlsprozessor (COMMAND.COM) mit vollständigem Pfad
PROMPT Eingabeaufforderung auf der Befehlsebene
DIRCMD für DIR Optionen (ab DOS 5.0)
CMDLINE (WIN95)
windir
winbootdir

===== Umleitung: Ausgabe ===== Lektion #14 =====

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

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

Umleitungen sind sowohl auf der Befehlsebene als auch in BAT-Dateien möglich. In dieser Lektion soll es zunächst um die Ausgabeumleitung 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 an die Datei TEST angehängt >> TEST

Eine eventuell vorhandene Datei wird durch den einfachen Pfeil gelöscht, während beim doppelten Pfeil am Ende angehängt wird.

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

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

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

WICHTIG: Nicht alle Bildschirm-Ausgaben können umgeleitet werden.

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

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

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

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

Die Bestätigung von DOS, "n Datei(en) kopiert", wird hier in die Datei AUSGABE.TXT umgeleitet. Dies gilt auch für den Fall, daß die zu kopierende Datei nicht existiert: "0 Datei(en) kopiert". Die zusätzliche 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 Geräte

Geräte-Namen können 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 Gerät, das aber benutzt werden kann, um Ausgaben ins Nichts zu schicken. Damit lassen sich oft unerwünschte Ausgaben in BAT-Abläufen unterdrücken, sofern(!) es sich um eine Standard-Ausgabe handelt.

Störende Fehlermeldungen, z.B. bei DEL ("Datei nicht gefunden") verschwinden damit nicht. Sie lassen sich am besten durch geeignete

Abfragen unterdrücken:

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

Anfragen 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 löschen), 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 von DOS entfernt.

Eingaben bei Ausgabe-Umleitung

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

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) lernt, kommt in der nächsten 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 hängt vom jeweiligen Programm ab.

* Der Umleitungsausdruck, also Pfeil(e) mit Dateiname/Gerät, 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...

* Bei Umleitung mit einfachem Pfeil in einem bedingtem Befehl..

z.B. IF errorlevel 1 echo irgendwas > DATEI

wird die Ausgabedatei neu angelegt, BEVOR die Bedingung geprüft wird. D.h. wenn die Bedingung NICHT WAHR ist, wird trotzdem die Ausgabedatei mit 0 Bytes erzeugt. Falls die Datei bereits vorhanden war, wird sie damit zerstört.

* Die Datei für die Ausgabe-Umleitung darf nicht den Platzhalter einer FOR-Schleife (z.B. %%) enthalten, da die Umleitungsdatei angelegt wird, bevor DOS die FOR-Schleife zur Kenntnis nimmt! Das hier geht nicht:

```
for %%a in (*.TXT) do echo irgendwas >> %%a
```

* 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. COMMAND /?)
```

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

Die Eingabe-Umleitung ist das Gegenstück 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 natürlich, daß die angegebene Datei mit dem erforderlichen Inhalt existiert. Beispiel:

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

Da DOS hier nachfragt "Sollen alle Dateien...gelöscht werden? (J/N)" muß die Datei J.TXT ein "J" und ein CR enthalten, also eine Zeile mit "J" und dem üblichen Zeilenabschluß.

Falls die Datei weitere Daten enthält, werden diese ignoriert, da der DEL-Befehl ja keine weiteren Eingaben erwartet. Grundsätzlich können aber auch mehrfache Eingaben für ein Programm nacheinander aus den einzelnen Zeilen einer Datei entnommen werden. Dies läßt sich z.B. beim Programm DEBUG nutzen, um ein ganzes Skript von Befehlen abzuarbeiten (DEBUG < Datei).

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. Das macht natürlich nur Sinn, wenn die Eingabe aus einer Datei kommt.

Standard Input

Entsprechend der Ausgabe-Umleitung ist auch bei der Eingabe eine Umleitung nur möglich, wenn der Input des Programms über die STANDARD-EINGABE angefordert wird. Da es sich hierbei um die Tastatur handelt, ist klar, aber das betreffende Programm muß die üblichen DOS-Funktionen benutzen. Bei Verwendung von BIOS-Funktionen kann der Input nicht umgeleitet werden.

Quelle für eine Input-Umleitung kann auch ein Gerät sein, z.B. COM1.

Ausgabe direkt in Eingabe leiten (PIPE)

Im obigen DEL-Beispiel wurde angenommen, daß die Datei J.TXT für die Eingabe-Umleitung bereits vorhanden ist. Natürlich 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 umständlich, da auch noch die Hilfsdatei hinterher gelöscht werden muß. Einfacher geht's mit der direkten Methode:

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

Das "|" bewirkt, daß der Output von der linken Seite als Eingabe für die rechte Seite übernommen wird. DOS produziert dazu zwar auch eine Zwischendatei (im temporären Verzeichnis %TEMP%), aber man braucht sich um deren Namen und um das anschließende Löschen nicht zu kümmern.

Solche "PIPES" sind eigentlich dafür 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 (standardmäßig) 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

Ein-/Ausgabeumleitung und mehrere Pipes können in einem Befehl kombiniert werden. Für die richtige Reihenfolge der Angaben ist nur zu beachten, daß mit "|" (Pipe) verbundene Programme immer von links nach rechts abgearbeitet werden. Befehlsparameter und Optionen müssen unmittelbar auf den zugehörigen Befehl folgen.

Umleitungsausdrücke mit Pfeil nach links/rechts sind zwangsläufig dem ersten bzw. letzten Programm (Befehl) zugeordnet, ganz gleich, wo sie platziert werden.

Hinweise

Vorsicht bei der Eingabe-Umleitung für Programme, die ganz bestimmte Eingaben erwarten: Falls die Eingabedatei falsche oder gar keine Angaben enthält, läßt sich auch mit der Tastatur nichts mehr korrigieren, weil die ja ignoriert wird. Mit etwas Glück geht vielleicht noch ein Control-BREAK, sonst ist ein Warmstart fällig.

Und: bitte nicht für Eingabe- und Ausgabeumleitung die selbe Datei verwenden, z.B.:

```
Befehl < DATEI > DATEI
```

Da sich DOS beeilt, die Ausgabedatei neu anzulegen, ist die Eingabedatei hin, bevor sie gelesen werden kann.

Anmerkung: unter DOS 6.xx funktioniert das zwar doch, aber unter DOS 7.x schon wieder nicht mehr....

Sichere Methode:

```
type DATEI | Befehl > DATEI
```

Doppelte Anführungszeichen

Die Umleitungs- und PIPE-Symbole werden von DOS innerhalb von doppelten(!) Anführungszeichen NICHT als solche interpretiert, sondern wie normale Textzeichen behandelt, z.B.:

```
ECHO " irgendwas > datei "
```

Eine brauchbare Verwendung dürfte schwer zu finden sein, aber immerhin funktioniert z.B. ein FIND ">" ohne Probleme.

Achtung: Bug!

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

IF %1==X echo.ldate

MS-DOS reagiert je nach Version und Konfiguration (SHARE, WIN) mit ziemlich unbrauchbaren Fehlermeldungen, oder es passiert einfach gar nichts. Falls nötig, also eine IF...GOTO Konstruktion verwenden!

Ähnliche Probleme gibt es auch bei der FOR-Schleife. Der Grund: MS-DOS legt die Pipes und Umleitungen an, BEVOR die gesamte Zeile interpretiert wird, und nicht erst für die einzelnen Instanzen der FOR-Schleife.

Siehe auch Hinweis zur Ausgabe-Umleitung in Lektion #14.

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

DOS erlaubt es leider nicht, den Errorlevel in Form einer Variablen zu verwenden. Zwar gibt es seit MS-DOS 7.x die undokumentierte COMMAND Option /Z, mit der nach jedem Programm der Errorlevel auf dem Bildschirm ausgegeben wird, aber die Übernahme in eine Variable wäre oft nützlich.

Zur Lösung des Problems gibt es einmal die brutale Methode mit 255 einzelnen Abfragen und SET-Befehlen.

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

Um zu verstehen, wie das funktioniert, soll erst mal der Errorlevel 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 hier
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 die 5 hängen.

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

Errorlevels müssen in FOR-Schleifen immer in aufsteigender Folge

abgefragt werden (auch in Verbindung mit GOTO, s. Lektion 12).

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 anhängen  
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 h hier  
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 angehängten Null - ergibt 00,10,20.... In EL wird aber zunächst nur die Zehner-Ziffer gespeichert.

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

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

Die Hunderter könnte man ja nun auf die gleiche Weise ermitteln. Aber da wird die Sache etwas komplizierter, weil dann zwangsläufig Errorlevels über 255 abgefragt werden - und dabei gibt es Probleme: DOS subtrahiert kommentarlos 256 (verwendet also immer den Rest aus Zahl dividiert durch 256).

Die Abfrage IF Errorlevel 260 bewirkt also: IF Errorlevel 4

Aus diesem Grund muß extra dafür gesorgt werden, daß die Ziffern-Liste nur (0 1 2 3 4 5) enthält, wenn die Schleife über 255 hinausgehen könnte. Dazu werden die Ziffern von 6 bis 9 in einer Variablen (%!) gespeichert, 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 machen, z.B. ELEVEL.BAT. 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. (Natürlich kann ELEVEL.BAT auch auf der Befehlszeile nach irgendeinem Programm benutzt werden.)

Bitte beachten: die Variablen EL und ! sollten in der aufrufenden Batch nicht für andere Zwecke verwendet werden. EL muß später noch gelöscht werden.

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

Mit den berühmten ANSI-Sequenzen lassen sich Farben und Cursor-Steuerungen in BAT-Dateien realisieren. Außerdem können Tasten umbelegt bzw. mit Befehlen belegt werden (nächste Lektion) und der Video-Modus gewählt werden (für Batch nicht zu gebrauchen, Neheres s. DOS-Handbuch).

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

Die dazu erforderlichen Befehle sind irgend wann einmal vom American National Standards Institute ("ANSI"), also einer Art DIN-Behörde der USA, genormt 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 ein kleines Problem: In der BAT-Datei muß ein Byte mit dem Wert 27 (dez) bzw. 1B (hex) gespeichert werden. Wie das eingegeben wird, hängt vom jeweiligen Editor ab. Eines der folgenden Verfahren führt normalerweise zum Erfolg:

* ALT gedrückt 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 zufällig das selbe Symbol ist, das als nächstes 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!
```

Grundstrukturlicher Aufbau einer ANSI-Sequenz

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

Den Abschluß der ANSI-Sequenz bildet ein Buchstabe. Groß-/Kleinschreibung wird hier unterschieden - also aufpassen!

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

Die ANSI-Sequenzen können an beliebiger Stelle in Textausgaben eingesetzt werden, normalerweise im ECHO-Befehl, oder in Dateien, die z.B. per TYPE ausgegeben werden. Grundsätzlich können 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 aufwärts
```

#[3B 3 Zeilen abwärts
#[25C 25 Spalten nach rechts
#[12D 12 Spalten nach links

Bei zu großen Werten wird der Cursor an die äußerste mögliche Position gesetzt (!). Bei fehlenden Werten gilt 1.

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

* Cursor absolut setzen (Zeile;Spalte).

#[1;1H Zeile 1, Spalte 1 (linke obere Ecke)
#[12;33H Zeile 12, Spalte 33

Bei ungenügenden Angaben wird der Befehl ignoriert.
Fehlende Angaben werden durch 1 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 nächsten Zeile. Soll ein Text an eine bestimmte Stelle positioniert werden, muß dieser unmittelbar auf den Cursor-Befehl folgen, z.B.:

ECHO #[1;30Hhier ist Position 30 in der ersten Zeile

* Position speichern/wiederherstellen

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

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

Funktionen: Löschen

#[2J Bildschirm löschen und Cursor Home
(die "2" ist eine Konstante!)
#[K Zeile ab aktueller Position löschen
(großes "K")

Funktion: Farben

Für 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 grün | 5 blinkende Zeichen

33 43 braun/gelb | 7 schwarz auf weiß
34 44 blau | 8 schwarz auf schwarz
35 45 magenta (violett) |
36 46 zyan (türkis) |
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 können auch einzeln angegeben werden, wenn die übrigen Werte unverändert 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 zurück, sondern auch die Farben (weiß auf schwarz).

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

```
#[0;36;40m normalisieren, türkis auf schwarz
```

* Attribut 7 soll offiziell "invertieren", macht aber immer schwarz auf weiß, egal was vorher war. Die gesetzten Werte für 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 für alle folgenden Zeichen-Ausgaben sowie für Löschbefehle und neue Zeilen beim Scrollen. (Mit dem Farbbefehl selbst wird also zunächst auf dem Bildschirm nichts verändert). Beispiel:

```
#[37;44;1m#[2J hiermit wird der Bildschirm für hell weiße
Zeichen auf blauem Grund vorbereitet und dann
gelesen.
```

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

Für 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 übersichtlicher und geht außerdem schneller. Für ganz Eilige gibt es auch TYPE-Ersatzprogramme, die noch um ein Vielfaches schneller sind als DOS.

brigens: ANSI-Grafiken und spezielle Software gibt's in Mailboxen.

Konsol-Treiber

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

Prompt

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

Zum Schluß noch ein Goody:

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

===== 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-BildschirmAusgabe an den Konsoltreiber geschickt.

Abschluß-Buchstabe der Sequenz für 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
" "-Zeichen (Code 156) ersetzt. Anstelle der dezimalen Zahlen können
hier aber auch die Zeichen selbst verwendet werden, und zwar in
(einfachen oder doppelten) Anführungsstrichen:

```
#[ "$";' '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 abwärts (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 gedrückten 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 Anführungszeichen (auch gemischt).

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

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

Umbelegung rückgängig 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 standardmäßig mit bestimmten Editier-Funktionen belegt, z.B. liefert die Taste F1 (ohne DOSKEY) schrittweise die Zeichen des letzten eingegebenen Befehls.

Diese Standard-Funktionen werden natürlich durch Umbelegung per ANSI-Sequenz aufgehoben, können aber wieder aktiviert werden, indem der Taste der eigene 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 zur Abfrage der Tastatur, und diese gehen NICHT über ANSI.SYS. Die Umbelegungen sind wohl in erster Linie auf der Befehlsebene brauchbar.

Noch ein Hinweis

Jede fremde ASCII-Datei kann über eine solche ANSI-Sequenz sogar die ENTER-Tasten mit einem gefälligen Befehl belegen ("ANSI-Bomben")
Wer sich schützen will, verwendet einen ANSI-Treiber, der Tasten-Umbelegungen erst gar nicht (oder nur bedingt) zulässt. Eine andere Methode: den Buchstaben "p" in der ANSI.SYS in einen "privaten" Code abändern (DEBUG Adresse: hex 161).

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

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

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

Syntax -----

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

/C[:]Tasten Angabe der zulässigen Tasten. Standard ist JN.
/N Keine Anzeige der zulässigen Tasten am Ende der