125

Free!

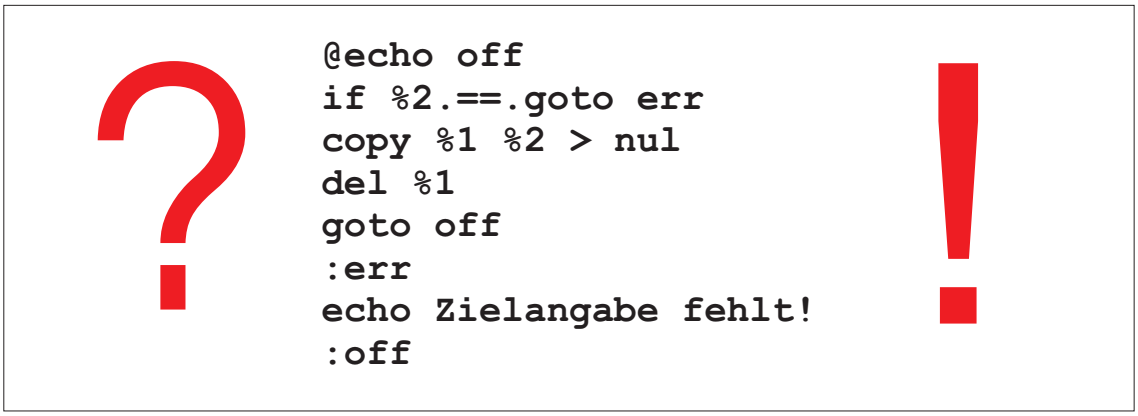
KnowWare

Stapelverarbeitungsdateien

Start mit BATCH Programmierung

Ideen & Beispielsammlung

für DOS - relevant unter Windows 9x



Lars Aschenbach

www.KnowWare.de

2. Ausgabe

Start mit BATCH Programmierung

Lars Aschenbach, lasche68@cs.com
2. Ausgabe, 3. Auflage, 1996-10, 4. Auflage 2001-04
ISBN 3-931666-25-5

www.knowware.de

Vorwort

Batchdateien sind wahre „Helferlein" bei der täglichen PC / EDV-Arbeit. Vielen Programmen sind mit zusätzliche *.BAT vorangestellt um Treiber für Netzwerkkarten, Modem, ISDN etc. zu laden. Daß die tägliche Arbeit mit dem PC mit Batchdateien wesentlich erleichtert werden kann ist, eingefleischten DOS - Fans durchaus bekannt. Nach wie vor laufen reine DOS Programme, meist um Faktor 10 schneller als Programme die eine grafische Benutzeroberfläche benutzen WINDOWS / GEM / MAC etc. Besonders im Netzbetrieb in Verbindung mit Datenbanken ist auf DOS und somit auf Batchdateien und Routinen nicht zu verzichten.

Lars Aschenbach hat sich in den letzten drei Jahren sehr intensiv mit der Batchprogrammierung unter verschiedenen Betriebssystemen auseinandergesetzt und uns eine kleine Auswahl seiner gesammelten Werke, die alle ohne Kopierrecht wiederverwendet werden dürfen, unentgeltlich zur Verfügung gestellt.

Lars ist sehr daran interessiert, weitere BATCHES kennenzulernen. Wer welche hat - bitte per Email schicken.

Wer von euch keine Lust auf die Tipparbeit hat oder mehr Batches kennenlernen möchte, kann
Christian Peter, Im Osterbach 5, 36211 Alheim
einen Scheck über DM 10.- schicken und bekommt eine gute Auswahl mit allen Erklärtexten zugeschickt.
Und nun heißt es: batch as batch can...

Lars Aschenbach

Das kleine DOSsier

Jeder Computer benötigt ein Programm, damit sein Bediener mit ihm kommunizieren kann. Schließlich muß der Computer das über die Tastatur eingegebene analysieren und ausführen, damit der Bediener sein gewünschtes Ziel erreicht.

Dieses Programm nennt man Betriebssystem, da es den gesamten Betrieb im Computer systematisch steuert und die Vorgaben des Bedieners zu erfüllen hat.

Sklave Betriebssystem, denn nur der Bediener befiehlt seine Tätigkeit, ist hauptsächlich dazu da, die Daten des Bedieners zu speichern, auszuführen und zu verwalten. Weil schon alt zu nennende Computer nur über Disketten als Datenträger verfügten, nannte und nennt man das Betriebssystem auch heute noch „Disk Operating System", kurz „DOS".

Auch wenn es heute DOS-Versionen wie MS-DOS, PC-DOS, NWDOS, MSX-DOS, 4DOS, oder DIP OS gibt, so sind doch alle im Kern fast gleich, weshalb ich hier nur die Schreibweise DOS verwenden werde.

Damit das DOS die Daten des Bedieners speichern und verwalten kann, hat es eine Menge Befehle, mit denen der Bediener das DOS befehligen soll und muß.

Es ist deshalb ein Muß, weil das DOS erstens nichts von alleine macht und zweitens keine natürliche Sprache versteht. Wer DOS benutzen will ist gezwungen die Sprache des DOS, also die DOS-Befehle, zu lernen. Ansonsten kommt keine Zusammenarbeit zwischen Bediener und DOS zustande. Man kann das am besten durch einen natürlichen Vergleich verstehen: Wenn ich als Deutscher kein Russisch beherrsche, aber trotzdem einen Russen der kein Deutsch beherrscht, um ein Taschentuch bitte, so kann mich der Russe nicht verstehen und ich warte vergebens auf das Taschentuch.

Um diesen Verständigungsproblemen zu entgehen, haben sich die Entwickler sämtlicher DOS-Versionen auf die Weltsprache Englisch geeinigt, die allerdings nur zu einem kleinen Teil vom DOS verstanden wird. So kann ich zwar das DOS ebenfalls bitten mir ein Taschentuch zu geben, wenn ich 'Give me an handkerchief, please' über die Tastatur des Computers eingebe, aber es wird mich genausowenig wie der Russe zuvor verstehen. Mal abgesehen davon das es meines Wissens nach keinen Computer gibt, der mit Taschentüchern ausgeliefert wird oder wurde!

Auch wenn das DOS von den jeweiligen Herstellern z.T. über die Jahre hinweg weiterentwickelt und bedienerfreundlicher gemacht wurde, soweit ist es heute noch nicht, das es die komplette englische Sprache verstehen würde. Das es überhaupt weiterentwickelt wurde, kann man an mehreren Ziffern sehen, die meistens dem Wort DOS folgen.

Einige Beispiele sind: MS-DOS 6.22, PC-DOS 6.3, MSX-DOS 2.31 und DIP Operating System 2.11. Doch zurück zu den Befehlen, mit denen man das DOS also zur Arbeit antreibt: Bei den ersten DOS-Versionen waren nur solche Befehle erlaubt, die das DOS intern gespeichert hatte. Aus diesem Grunde nennt man solche Befehle auch interne Befehle. Durch die Weiterentwicklung, die z.T. noch anhält, kamen noch externe Befehle dazu, die als Zusatzprogramme zum DOS ausgeliefert wurden.

Interne Befehle kann man dem DOS immer erteilen, aber externe Befehle nur dann, wenn das dazugehörige Programm auch auf dem Computer zur Verfügung steht.

Wir wollen uns hier nur mit den internen Befehlen weiterbeschäftigen, da diese zahlenmäßig unterlegen und bei fast allen DOS-Versionen vorhanden sind. Außerdem reichen diese bereits für die ordentliche Verwaltung der Daten des Bedieners aus. Damit alle stolzen DOS-Besitzer diesen Kurs weiterverfolgen können, setze ich die recht alte DOS-Version 2.11 mit ihrem Befehlssatz voraus. Zwar schauen nun alle MSX-DOS 1 Anwender in die berühmte Röhre, aber diese Version ist schlicht und einfach eine Katastrophe, weil sie nur über dreizehn interne Befehle verfügt und keine geordnete Datenablage ermöglicht.

Es wird zwar nie die perfekte Datenablage geben, dies liegt auch mit am Bediener, aber ab den DOS-Versionen 2.11 wird sie durch ordentliche Planung zumindest möglich.

Hier also endlich die internen Befehle des DOS 2.11 in alphabetischer Reihenfolge:

```
BREAK, CHDIR, CHKDSK, CLS, COPY, DATE,
DEL, DIR, DO, ECHO, ERRORLEVEL, EXIST,
FOR, GOTO, IF, IN, MKDIR, NOT, PAUSE,
```

```
PATH, PROMPT, RMDIR, REM, REN, SET,
TIME, TYPE, VER und VERIFY
```

Was diese Befehle bedeuten und bewirken, erkläre ich in einem späteren Teil des Kursus, da ich vorher noch auf das Thema Dateien und Dateienverwaltung eingehen muß und damit setzen wir in der nächsten Ausgabe fort.

Know How: Batching

Laufe ich auf dem PoFo oder dem PC?

Diese Frage müssen sich Batchdateien stellen und selbst beantworten können, wenn sie auf beiden Computern laufen sollen. Denn es gibt leider ein paar Unterschiede in der Batchsyntax zwischen PoFo und PC!

So kann jede einzelne, der folgenden Batchzeilen, dem PC unter bestimmten Umständen Probleme bereiten:

```
IF %1==L ECHO Der Buchstabe L!
CD..>NUL
MD C:\DOS>NUL
FOR %%A in (C:\DOS\*.*) DO TYPE C:\DOS\%%A
TYPE C:\CONFIG.SYS /P
APP /E
OFF
```

Sie führen zu Fehlermeldungen und bringen damit Daten in Gefahr, da die Batchdatei, die nur eine dieser Zeilen beinhaltet, mit einem falschen Resultat endet. Zumindest tritt das sehr oft beim Einsatz auf dem PC ein. Für den PoFo sind diese Batchzeilen dagegen vollkommen korrekt und werden auch, beim Einsatz in Batchdateien, wohlgemerkt, ordentlich ausgeführt.

Um aber diese Batchzeilen auf dem PC einsetzen zu können, müssen diese so geschrieben werden:

```
IF %1.==L. ECHO Der Buchstabe L!
IF EXIST ..\NUL CD ..
IF NOT EXIST C:\DOS\NUL MD C:\DOS
FOR %%A IN (C:\DOS\*.*) DO TYPE %%A
TYPE C:\CONFIG.SYS | MORE
EDIT - Anm.: MS-DOS als DOS vorausgesetzt
OFF - Anm.: Dazu gibt es keinen vergleichbaren Befehl
  beim PC
```

In dieser Schreibweise sind aber leider nicht alle Zeilen auch für den PoFo gültig und würden bei ihm zu Fehlermeldungen führen.

Eine gemeinsame Schreibweise, die auf beiden Geräten gültig und unproblematisch ist, gibt es immerhin für die ersten drei Batchzeilen:

```
IF %1.==L. ECHO Der Buchstabe L!
IF EXIST ..\NUL.* CD ..
IF NOT EXIST C:\DOS\NUL.* MD C:\DOS
```

Doch bei den restlichen vier Zeilen fand ich keinen gemeinsamen Nenner und mußte daher prüfen, vor dem Einsatz dieser Zeilen, ob die Syntax vom PoFo oder vom PC anzuwenden ist.

Angeregt durch DR-DOS und NovellDOS half ich mir deshalb mit den Variablen OS (Operating System) und VER (Version) weiter, die ich dann auf dem jeweiligen Computer von Hand setzte. So nahm ich Teile der Bildschirmmeldung auf, die sich aus der Anwendung des Befehles VER ergeben, um zwischen PC und PoFo unterscheiden zu können.

```
Beim PoFo:  SET OS=DIP-OS
            SET VER=2.11

Beim PC:    SET OS=MS-DOS
            SET VER=6.22
```

Außerdem fügte ich dann die jeweils zwei gültigen Zeilen noch in die entsprechende AUTOEXEC.BAT meines PCs und des PoFos ein.

Nun kann ich in meinen Batchdateien die Variable OS abfragen und die, für das jeweilige DOS gültige Batchzeile, zur Ausführung kommen lassen:

```
IF %OS%.==DIP-OS. FOR %%A IN (C:\DOS\*.*) DO
  TYPE C:\DOS\%%A
IF NOT %OS%.==DIP-OS. FOR %%A IN (C:\DOS\*.*)
  DO TYPE %%A

IF %OS%.==DIP-OS. TYPE C:\CONFIG.SYS /P
IF NOT %OS%.==DIP-OS. TYPE C:\CONFIG.SYS |
  MORE

IF %OS%.==DIP-OS. APP /E
IF NOT %OS%.==DIP-OS. EDIT

IF %OS%.==DIP-OS. OFF
IF NOT %OS%.==DIP-OS. ECHO Bitte nun den
  Computer ausschalten!
```

Ich teste mit diesen Zeilen nur auf den Variableninhalt DIP-OS ab, da DR-DOS und NovellDOS von Haus aus eine Variable OS

besitzen, die mit DR-DOS, bzw. NWDOS definiert ist. Es reicht ja aus zu prüfen, ob die Variable OS mit DIP-OS definiert wurde, damit die PoFo-Syntax angewendet wird!

Somit kann ich also tatsächlich Batchdateien schreiben, die sowohl auf dem PC, als auch auf dem PoFo lauffähig sind! Doch leider ist damit das Problem noch nicht ganz gelöst:

Keine Batchdatei kann sich mit absoluter Sicherheit darauf verlassen, das die Variable OS vom Bediener angelegt wurde. Fehlt diese Variable, oder ist sie nicht mit DIP-OS definiert, so wird immer die vom PC gültige Syntax verwendet und das ohne Rücksicht auf Verluste und Fehlermeldungen!

Wie eingangs erwähnt, muß sich jede Batchdatei fragen, ob das zugrundeliegende Gerät ein PoFo oder ein PC ist. Diese Frage muß die Batchdatei dann auch noch selbst beantworten können, damit es keine Probleme gibt!

Deshalb habe ich verschiedene Techniken ausprobiert und bin auch zu sehr umständlichen, aber funktionierenden Lösungen gekommen. Weil die nun wirklich nicht akzeptabel waren, habe ich dann für ein halbes Jahr von diesem Thema Abstand genommen.

Vor einigen Tagen dann, ich beschäftigte mich gerade intensiv mit dem Verhalten der Parameter %0 und %1 auf dem PoFo, öffnete mir Kollege Zufall die Augen:

Starte ich eine Batchdatei, z.B. TEST.BAT, auf dem PoFo, indem ich ihren Namen TEST inkl. der Extension .BAT eingebe, so wird der Name im Parameter %0 und die Extension im Parameter %1 gespeichert:

```
PoFo:  %0 = TEST     %1 = .BAT
```

Starte ich auf dieselbe Weise beim PC, so werden der Name TEST und die Extension .BAT zusammen im Parameter %0 gespeichert. Ein Parameter %1 existiert dann nicht:

```
PC  :  %0 = TEST.BAT  %1 =
```

Wer genau hinschaut, wird also deutliche Unterschiede zwischen den Parametern beim PoFo und beim PC feststellen!

Demnach muß es immerhin möglich sein zu prüfen, ob die Batchdatei unter DIP-OS gestartet wurde oder nicht, wenn ich die Batchdatei sich selbst neustarten lasse und den Parameter %1 überprüfe!

Und tatsächlich kann ich das mit der Batchdatei DIPOS.BAT beweisen:

```
01: @echo off
02: rem Ist die Batchdatei unter DIP-OS gestartet worden
    oder nicht?
03: rem
04: rem Selbstaufruf, wegen Trennprüfung beim DIP-OS:
05: rem
06: if not %0==dIpOs.BaT if not %1.==.BaT. dIpOs.BaT
07: rem
08: rem Ergibt folgende Parameter:
09: rem
10: rem bei DIP-OS:   %0 = dIpOs        %1 = .BaT
11: rem bei MS-DOS:   %0 = dIpOs.BaT    %1 =
12: rem
13: if %1.==.BaT. echo Ich laufe unter DIP-OS!
14: if %1.==.BaT. set OS=DIP-OS
15: if %1.==. echo Ich laufe NICHT unter DIP-OS!!!
16: if %1%os%.==. set OS=MS-DOS
```

Die Zeilennummern nebst Doppelpunkt sind nicht abzutippen! Die wechselnde Schreibweise ist zwar lästig, aber absolut notwendig!!!!

DIPOS.BAT sollte durch die Eingabe von "DIPOS" oder "dipos" gestartet werden!

Weil bei diesem Start noch die Extension .BaT fehlt, was ja vollauf beabsichtigt ist, fallen die Prüfungen in Zeile 06: negativ aus und sorgen für den Neustart inklusive der Extension: dIpOs.BaT

Dieser Neustart führt zu unterschiedlichen Reaktionen bei PC und PoFo, weshalb wir diese einmal getrennt betrachten:

Beim PC wird ja der Parameter %0, hier: dIpOs.BaT, unbehelligt gelassen, weshalb die erste IF NOT-Abfrage in Zeile 06: greift und somit kein weiterer Neustart mehr notwendig ist.

Weil kein Parameter %1 existiert der .BaT lauten könnte, kommt die Zeile 15: zur vollen Geltung und gibt kund, das DIPOS.BAT nicht unter DIP-OS (PoFo) gestartet wurde!

Die Zeile 16: hält dann dieses Ergebnis nach Möglichkeit in der Variable OS (für Operating System) fest, sofern bislang keine Variable OS existiert. Damit können nachfolgende Batchdateien das OS prüfen, ohne erst einen Neustart veranlassen zu müssen. Existierte aber bereits eine Variable OS, bei DR-DOS und NovellDOS sollte das ja der Fall sein, so bleibt die bisherige Definition von OS erhalten.

Beim Neustart auf dem PoFo findet dagegen die Aufteilung von dIpOs.BaT in dIpOs und .BaT statt. Somit lautet der Parameter %0 dIpOs und der Parameter %1 .BaT. Diese Aufteilung und Umwandlung findet übrigens noch vor der Ausführung der Zeile 01: statt!

Nun greift die zweite IF NOT-Abfrage der Zeile 06: und verhindert einen weiteren Neustart. Da jetzt der Parameter %1 existiert und zudem noch .BaT lautet, sind die Bedingungen der Zeile 13: erfüllt und es wird gemeldet, das DIPOS.BAT unter DIP-OS (PoFo) gestartet wurde.

Hier hält dann die Zeile 14: das Ergebnis für nachfolgende Batchdateien in der Variablen OS fest. Voila!

Nun kann man entweder DIPOS.BAT vor den eigentlichen Batchdateien starten, damit die Variable OS auch existiert, oder man setzt seine Routine in das folgende Grundgerüst ein:

```
01: @echo off
02: for %%a in (%bdv%. %bdy%.) do if %%a==. goto err
03: if not %0==%BDV%%BDY%bAtCh.BaT if not
   %1.==.BaT. %BDV%%BDY%bAtCh.BaT %1 %2 %3
   %4 %5 %6 %7 %8 %9
04: if %1.==.BaT. set OS=DIP-OS
05: if %1%os%.==. set OS=MS-DOS
06: if %1.==.BaT. shift
07: rem
08: rem Hier muß dann die eigentliche Routine
   eingebunden werden!
09: rem
97: :err
98: for %%a in (%bdv%. %bdy%.) do if %%a==. echo
   Eine der Variablen BDV und BDY existiert nicht!
99: :off
```

In der Zeile 03: muß dann anstelle von bAtCh.BaT und bAtCh.BaT der endgültige Name der Batchdatei eingetragen werden. Bitte achtet dabei auf die wechselnde Schreibweise, da sie sehr, sehr wichtig ist!!!

Neu sind dabei die Variablen BDV und BDY:

Hinter diesen Variablen, die in vielen meiner Batchdateien vorkommen, verbergen sich das BatchDriVe (das Laufwerk AUF DEM sich meine Batchdateien befinden) und das BatchDirectorY (das Verzeichnis IN DEM meine Batchdateien stehen).
So habe ich BDV durch die AUTOEXEC.BAT mit SET BDV=C: und BDY mit SET BDY=\BATCH\ definiert. Damit das DOS meine Batchdateien auch jederzeit wiederfindet, habe ich dann noch das Verzeichnis C:\BATCH mit in die PATH-Anweisung aufgenommen:
PATH=C:\DOS;C:\BATCH

Das ganze hat folgende Vorteile:

Will ich meine Batchdateien veröffentlichen, in ein anderes Laufwerk und Verzeichnis verlagern oder auf einen anderen PC übertragen, so muß weder ich noch ein anderer User, die Laufwerks- und Verzeichnisangaben in den ganzen Batchdateien ändern, damit sie unter den neuen Bedingungen auch funktionieren. Es muß lediglich dafür Sorge getragen werden, daß die Variablen BDV, BDY und PATH angepaßt oder bereitgestellt werden:

Dadurch haben alle User deutlich weniger Tipparbeit und sind sehr flexibel!

Wer dieses Grundgerüst in Zukunft für seine Batchdateien verwendet, kann nun auf zweierlei Wegen abprüfen, ob die Batchdatei auf dem PoFo läuft oder nicht:

Entweder man prüft die Variable OS also auch weiterhin auf den Inhalt DIP-OS ab, oder man prüft den Parameter %0, der dann beim PoFo .BaT heißen muß, wenn die eigentliche Routine zum Zuge kommt. Weil die Variable OS nicht in allen Fällen existiert, oder angelegt werden kann, ist die Prüfung des Parameters %0 auf .BaT besonders zu empfehlen!!!

Warum die Variable OS überhaupt von der Firma Digital Research eingeführt wurde und ich mir diese zu Nutze mache:

So manches Programm will nur partout unter MS-DOS oder PC-DOS funktionieren und fragt daher die Betriebssystemversion ab. Lautet diese z.B. nicht "MS-DOS 4.0", so stellt das Programm seine Arbeit unverzüglich ein. Den Entwicklern von DR-DOS, welches später zu NovellDOS wurde, war dies natürlich ein Dorn im Auge, weshalb sie den Käufern eine einfache Manipulationsmöglichkeit dieser Betriebssystemversion boten:

Mittels SET-Befehl kann man dort die Variablen OS und VER (für Version) auf die gewünschten Werte einstellen, damit das gewünschte Programm überlistet wird und läuft.

Weil das Rad nicht zweimal erfunden werden muß, englischsprachige Variablen bei der internationalen Verbreitung meines Wissens hilfreich sind und ich eine straffe

Organisation bevorzuge, verwende also auch ich die Variable OS!

Die Variable VER spielt zwar derzeit noch keine Rolle, aber ich werde sie dann nutzen, wenn ich auf dem PC die externen Programme des jeweiligen DOS verwenden möchte. Auch für den PoFo ist diese Variable VER noch unbedeutend und muß nicht zwingend angelegt werden. Aber wer weiß???

Laufwerk vorhanden, Diskette eingelegt?

Diese Frage kann per Batchdatei beantwortet werden! Doch zuerst muß einiges Wissen bekannt sein, damit man die Funktionsweise von „LW&DISK.BAT" verstehen kann:
Ist das Laufwerk A: vorhanden und mit einer formatierten Diskette bestückt, so erhält man mit -DIR A:\- etwa folgende Informationen:

```
Datenträger in a: hat keine Bezeichnung
Inhaltsverzeichnis von a:
config.sys  1.01.80 0:00
...
```

Befindet sich aber keine Diskette im real existierenden Laufwerk A:, so sollte man nach -DIR A:\- folgende Meldung erhalten:

```
Nicht bereit, Fehler beim lesen a:
Abbrechen, Wiederholen, Ignorieren?
```

Diese Meldung muß dann noch entsprechend mit <A>, <W>, oder <I> beantwortet werden. Versucht man ein nicht vorhandenes Laufwerk anzusprechen, z.B. mit -DIR Q:\-, so meldet sich DOS mit:

```
Ungültiges Laufwerk
```

zu Wort. Je nach DOS-Version können diese Meldungen aber deutlich abweichend sein! Damit „LW&DISK.BAT" auch unter jedem DOS tadellos funktioniert, muß man diese Fehlermeldungen künstlich erzeugen und „LW&DISK.BAT" entsprechend abändern.

Denn die Routine erstellt selbständig die notwendigen Antwortdateien, die sich meist auf das erste Wort der Meldung beziehen. So wird das Directory in „1.BAT" geschrieben, wenn Laufwerk und formatierte Diskette vorhanden sind.

Anschließend wird „1.BAT" aufgerufen und ruft seinerseits „DATENTR.BAT" auf, da DOS das Wort „Datenträger" als auszuführende Datei definieren kann. Gibt es das gesuchte Laufwerk nicht, so wird die Meldung „Ungültiges Laufwerk" in „1.BAT" geschrieben. Beim folgenden Aufruf von „1.BAT" kürzt DOS das Wort „Ungültiges" auf „UNG" zusammen und ruft die vorhandene Batchdatei „UNG.BAT" auf. Weil die Fehlermeldung „Nicht bereit..." trotz Umleitung nicht in „1.BAT" abgelegt wird, hänge ich per -ECHO-Befehl das Wort „NO" an „1.BAT", damit

„1.BAT" später „NO.BAT" starten kann und mir gemeldet wird, daß das Laufwerk zwar existiert, aber keine, oder eine unformatierte Diskette enthält.

Genug des Geplänkels, hier ist also „LW&DISK.BAT":

```
@echo off
rem falls mal etwas schiefläuft!
if %1.==. if exist c:\temp\a.dat goto off
rem Laufwerksangabe vergessen?
if %1.==. goto err
rem Temporäres Verzeichnis anlegen
md c:\temp> nul
rem Pfad retten und ergänzen
set opath=%path%
set path=%path%;c:\temp
rem Hilfs- & Antwortdateien erzeugen
echo a> c:\temp\a.dat
echo @echo off> c:\temp\datentr.bat
echo echo LW %1 und Diskette vorhanden!>>
   c:\temp\datentr.bat
echo lw&disk>> c:\temp\datentr.bat
echo @echo off> c:\temp\no.bat
echo echo Keine o. unformatierte Diskette im LW %1!>>
   c:\temp\no.bat
echo lw&disk>> c:\temp\no.bat
echo @echo off> c:\temp\ung.bat
echo echo Laufwerk %1 existiert nicht!>> c:\temp\ung.bat
echo lw&disk>> c:\temp\ung.bat
echo @echo off> c:\temp\1.bat
rem Beginn der eigentlichen Prüfung
dir %1\ <c:\temp\a.dat>> c:\temp\1.bat
rem Fehler „Nicht bereit..." wegblenden
cls
rem falls Nicht bereit dann für Start
rem von NO.BAT vorbereiten
echo no>> c:\temp\1.bat
rem 1.BAT starten, 2.Teil d. Prüfung
1
rem wenn Prüfung beendet oder etwas
rem nicht klappte, dann temporäreres
rem Verzeichnis löschen und Ur-Pfad
rem rekonstruieren
:off
for %%a in (c:\temp\*.*) do del c:\temp\%%a
rd c:\temp> nul
if not %opath%.==. set path=%opath%
set opath=
```

```
goto aus
rem falsche Syntax!
:err
echo - Aufruf mit z.B. LW&DISK A:!   -
echo - DOPPELPUNKT ja nicht vergessen! -
:aus
```

Trotz der diesmal vorhandenen Dokumentation muß ich noch etwas näher auf „A.DAT" eingehen:

Diese Datei wird für den Fall bereitgestellt, wenn keine, oder eine unformatierte Diskette, im zu prüfenden Laufwerk ist. Denn die oben bereits erwähnte Fehlermeldung „Nicht bereit..." muß ja noch beantwortet werden. Mittels der Einleitung -<a.dat- wird die evtl. auftretende Fehlermeldung dann auch brav vom DOS aus beantwortet.

Wer diese Batchdatei noch rekursiv ausführbar gestaltet und die Ergebnisse in Variablen wie LW & DISK festhält, kann damit z.B. eine automatische Datensicherung in der Nacht erreichen!!!

KnowHow: DIP-OS

Ausgeklammert?

Das sich hinter der schließenden Klammer) ein weiteres Geheimnis des PoFo verbirgt, hat wohl noch keiner gedacht und entdeckt. Mir ist das auch erst vor kurzem aufgefallen:

Erzeugt doch einfach mal die Dateien J. und J).DAT vom DOS-Prompt aus, mit den Direkteingaben:

```
rem>j.
rem>j).dat
```

Und gebt nun diese Befehlszeile direkt am Prompt ein:

```
for %a in (*.*) do echo %a
```

Bei mir führte das zu dieser Bildschirmausgabe:

```
CONFIG.SYS
TEST.BAT
J
J
```

Die doppelte Anzeige der Datei J. hat mich dann sehr überrascht, hatte ich doch erwartet, auch die Datei J).DAT zu Gesicht zu bekommen. Denn im Normalfall hätte die Anzeige so ausfallen müssen:

```
CONFIG.SYS
TEST.BAT
J
J) .DAT
```

Aber weil im Dateinamen von J).DAT eine schließende Klammer) zu finden ist, macht das DIP-OS aus der FOR-Schleife folgendes, wenn es auf die Datei J).DAT trifft:

```
FOR %A in (J).DAT) DO ECHO %A
```

Zwar wird man diese Zeile selbst bei eingeschaltetem ECHO nie zu Gesicht bekommen, aber so muß das DIP-OS diese Zeile interpretieren, wenn die Datei J).DAT gegriffen wird. Schuld daran ist die schließende Klammer, die die sonst übliche Menge in der FOR-Schleife verfrüht schließt und somit aus der erwarteten Menge (J).DAT) die Menge (J) macht, die dann zur Anzeige kommt.

In der Praxis werden Dateinamen mit Klammern wie (,), { und } nur dann erzeugt, wenn es sich dabei um temporäre, also um kurzweilig existierende Dateien, handeln soll. Meist beschränkt man sich dabei auf die Klammern { und }.

Wer also den Problemen des DIP-OS aus dem Wege gehen will, der sollte es tunlichst vermeiden die schließende Klammer) zu verwenden: egal ob in Dateinamen, Variablen oder Parametern. Wir erinnern uns: Variablen und Parameter sind meist Platzhalter für Dateinamen!

Kann oder möchte man aber nicht auf die schließende Klammer verzichten, als Bestandteil in Dateinamen, so kann das Fehlverhalten des DIP-OS mit diesen Batchzeilen rechtzeitig verhindert werden:

```
if exist )*.* echo Klammer ) in Dateinamen verboten!
if exist )*.* goto :>nul
if exist ?)*.* echo Klammer ) in Dateinamen verboten!
if exist ?)*.* goto :>nul
if exist ??)*.* echo Klammer ) in Dateinamen verboten!
if exist ??)*.* goto :>nul
if exist ???)*.* echo Klammer ) in Dateinamen verboten!
if exist ???)*.* goto :>nul
if exist ????)*.* echo Klammer ) in Dateinamen verboten!
if exist ????)*.* goto :>nul
if exist ?????)*.* echo Klammer ) in Dateinamen verboten!
if exist ?????)*.* goto :>nul
if exist ??????)*.* echo Klammer ) in Dateinamen
   verboten!
if exist ??????)*.* goto :>nul
if exist ???????).* echo Klammer ) in Dateinamen
   verboten!
if exist ???????).* goto :>nul
if exist *.)* echo Klammer ) in Dateinamen verboten!
if exist *.)* goto :>nul
if exist *.?)* echo Klammer ) in Dateinamen verboten!
if exist *.?)* goto :>nul
if exist *.??) echo Klammer ) in Dateinamen verboten!
if exist *.??) goto :>nul
```

Diese Zeilen müssen einmal unmittelbar vor die erste FOR-Schleife eingebaut werden, die in der Menge einen Parameter, eine Variable oder Wildcards enthält, sofern die Menge sich auf Dateinamen bezieht. Ein Beispiel dafür ist meine ADDCOPY.BAT, die Ihr in KH-ADDCP.TXT betrachten könnt. Ich empfehle auch diese Zeilen in jede Batchdatei einzubinden, die mittels einer FOR-Schleife Dateinamen benutzt.

Eine kürzere Schutzvariante ist mir zwar auch eingefallen, aber die erwies sich nicht als sehr sicher.

Auf der anderen Seite könnte man dieses Wissen vielleicht auch dazu nutzen, um eine gewisse Art der Stringverarbeitung zu erreichen. So legt man z.B. die Datei JA).DAT an, um mit der obigen FOR-Schleife daraus ein JA zu machen.

Allerdings ist mein in CUT.BAT gezeigter Tip, der ebenfalls zur Stringverkürzung führt,

besser, weil er nicht auf 11 Zeichen (achtstelliger Dateiname + dreistelliger Extension) beschränkt ist, wie diese Möglichkeit.

Sofern man aber beide Techniken verbindet, so kann das evtl. zu einer sehr gebrauchsfähigen Stringverarbeitung heranwachsen!

Ich habe zwar noch keine vernünftige Verwendung für die hier beschriebene Technik, noch für die Kombination dieser TEST.BAT plus CUT.BAT parat, aber vielleicht hat ja einer von Euch die zündende Idee???!!!

Die Entwicklung

... von Batchdateien ist nicht immer einfach, ganz besonders dann nicht, wenn sie sehr trickreich ausfallen sollen. Durch das berühmte Debuggen, die Fehlersuche, geht gewöhnlich die meiste Zeit verloren. Dabei wird von jedem Lehrbuch empfohlen, das ECHO nicht auszuschalten und an kritischen Stellen PAUSE-Befehle einzubauen, damit die Fehlersuche vereinfacht wird.

Während der Testphase ist dagegen auch nichts einzuwenden, aber was das an Arbeit bedeutet, wenn man diese Zeilen nach den erfolgreichen Tests löschen kann, das verschweigen die Lehrmeister geflissentlich!

Da meine Routinen immer komplexer zu werden scheinen und ich nicht mehr Zeit, als bisher, in die Fehlersuche investieren will, suchte ich eine variable Lösung, damit ich die Batchdateien nicht nachträglich überarbeiten muß. Nach reiflicher Überlegung kam ich auf die Idee, anstelle der Befehle Variablen zu verwenden, weil diese ja beliebig umdefiniert werden können!

So ersann ich die Variable O die entweder mit dem Wort ON oder OFF definiert sein soll, damit ich zu Beginn des Batchprogramms das ECHO an- oder abschalte:

```
@ECHO %O%
```

und die Variable & die mit dem Befehl PAUSE oder REM definiert sein soll, damit ich an kritischen Stellen bei Bedarf eine PAUSE einlegen kann oder nicht:

```
%&%
```

Um beide Variablen gleichzeitig auf den jeweiligen Stand zu schalten: testen oder nicht testen, schuf ich dann PHASE.BAT, welche die Variablen & und O mit jedem Aufruf umschaltet:

```
@echo off
if %1.==?. if not %o%.==. goto off
for %%a in (on off) do if not %o%.==%%a. set o=%%a
for %%a in (pause rem) do if not %&%.==%%a. set
  &=%%a
:off
echo Die Testphase ist %o%!
```

Zukünftige Batchdateien müssen also mit @ECHO %O% beginnen und die PAUSE-Zeilen, die man zu Testzwecken einbringt, müssen durch %&% gestaltet werden. Dazu mal eine Testroutine, die Ihr unter dem Namen TEST.BAT abspeichern könnt:

```
@echo %o%
echo Dies ist eine unwichtige Zeile!
%&%
echo *Zeile ohne Wert*
```

Ruft nun abwechselnd PHASE.BAT und TEST.BAT auf, um die Wirkung von PHASE.BAT zu ergründen.

Wenn also demnächst meine Batchdateien mit @ECHO %O% beginnen sollten, so sollte die Testphase möglichst OFF-geschaltet sein.

Den Zustand der Testphase könnt Ihr auch jederzeit durch den Aufruf PHASE? erfragen. Ansonsten werdet Ihr auch nach jedem Aufruf über den aktuellen Stand informiert!

Der gute Ton

... macht erst die wohlklingende Musik, die man immer wieder gerne hört. Auch bei zu programmierenden Batchdateien gibt es ein paar Grundsätze, die man beachten sollte:

- Variablen, die nicht von anderen Routinen mitbenutzt werden, sollten am Ende der jeweiligen Batchdatei wieder gelöscht werden.
- Hilfsdateien sollten ebenfalls nach ihrer Tätigkeit entfernt werden.
- Die Hilfsdateien, die man behalten muß, dürfen nicht sofort vom Bediener gestartet werden können. D.h. ihre Extension sollte nicht BAT lauten.
- Batchdateien sollten ihre Aufrufsyntax dann präsentieren, wenn entweder kein Parameter übergeben wurde, oder der erste Parameter /? lautet.

- Alle Batchdateien sollten in einem Verzeichnis untergebracht werden, das auch in der Pfadangabe gesetzt wurde.
- Nach Möglichkeit sollte jede Batchdatei nicht mehr als zwei Parameter beim Start benötigen, damit der Bediener nicht ellenlange Eingaben erzeugen muß.
- Die Batchdateien sollten so markante Namen haben, das der Bediener sie schnell deuten und lernen kann.
- Keine Batchdatei sollte zu COM- oder EXE-Programmen umgewandelt werden, da diese Compiler entweder mit bestimmten Befehlskombinationen nicht fertig werden, oder die Technik nicht ausführen können.
- Auch die Zusammenlegung von Routinen zu einer riesigen Batchdatei, ist nicht immer förderlich, da die Ausführungsgeschwindigkeit darunter leidet.
- Fortgeschrittene Programmierer sollten lange Befehle in Variablen packen und diese, statt der Befehle, in ihren Routinen verwenden.
- Routinen die Schalter und externe Befehle benutzen, sollten die DOS-Version ermitteln können, auf der sie gerade eingesetzt werden. Gegebenenfalls müssen sie ihre Tätigkeit einstellen, wenn sie nicht unter der laufenden DOS-Version funktionieren können.
- REM-Zeilen sollten dann wieder entfernt werden, wenn die Routine keine Fehler mehr aufweist.
- Sprungmarken funktionieren auch ohne die abschließende :END Markierung, weshalb man auf sie verzichten sollte.
- Hinter dem PAUSE-Befehl und den Sprungmarken sollten auch keine Kommentare angefügt werden, da sie wegen der -@ECHO OFF- Zeile eh nicht angezeigt werden.
- Sollen Treiber, wie ANSI.SYS, Verwendung finden, so sollte deren Vorhandensein vorher ermittelt werden.
- Bevor eine Variable gesetzt wird, sollte möglichst geprüft werden, ob diese nicht bereits existiert. Notfalls muß ihr ursprünglicher Inhalt in einer anderen Variable, oder einer Batchdatei, gesichert werden.

Sicherlich werden gerade die Anfänger große Schwierigkeiten mit der Einhaltung dieser Regeln haben, aber ich habe auch gut zwei Jahre gebraucht, um diese zu etwa 90% einzuhalten. Wenn man seine Routinen nicht weitergeben oder veröffentlichen will, kann man sowieso darauf pfeifen

KILL.BAT - löschen mit Reserve

Der Computerfreak, der noch nie eine Datei versehentlich und voreilig gelöscht hat, dürfte wohl nicht menschlich sein. Wir Menschen lernen ja aus Fehlern und entwickeln unsere Intelligenz gerade dadurch weiter.

Manchmal dauert dieser Lernprozeß aber länger und muß einem gehörig die Finger verbrennen, bis man endlich zur Einsicht gelangt. So sind auch mir schon etliche Daten verlustig gegangen, die ich dann meist mühsam rekonstruierte. Nun bin ich aber des Befehls DEL leid und habe eine Sicherung eingebaut, durch die Dateien wiederhergestellt werden können:

```
@echo off
if %1.==. goto err
md \kill > nul
copy %1 \kill > nul
del %1
goto off
:err
echo Zu löschende Datei fehlt!
:off
```

Beim Aufruf von KILL.BAT, inkl. des Namens der zu löschenden Datei, wird ein Verzeichnis KILL angelegt, in die die zu löschende Datei sicherheitshalber abgelegt wird. Danach wird die ursprüngliche Datei gelöscht. Folglich wurde die Datei nicht richtig vom Datenträger gelöscht, sondern nur ins Verzeichnis KILL transferiert. Von dort kann sie nun jederzeit wiederhergestellt werden, indem man sie ins gewünschte Verzeichnis zurückkopiert.

Sollte bereits ein Verzeichnis namens KILL bestehen, so wird man dank der Nul-Umleitung, von der Fehlermeldung verschont. Auch beim kopieren schweigt sich DOS deshalb aus.

Wem diese Sicherheitslöschung nicht gefällt, bzw. wer ohne Reserve fahren möchte, ist ja nicht an die Benutzung von KILL.BAT gebunden und kann wie bisher mit DEL arbeiten.

Verzeichnisanlage mit folgendem Verzeichniswechsel

Oftmals legt man ein Verzeichnis an und will gleich anschließend in das neue Verzeichnis gelangen, um dort Dateien abzulegen. Sonst hätte ja auch ein neues Verzeichnis kaum Sinn, oder???

Die dafür notwendigen Eingaben: MD <Verzeichnis> und CD <Verzeichnis>, sind wohl mittlerweile jedem DOS-User in Fleisch und Blut übergegangen. Aber findet Ihr diese Vorgehensweise nicht ein wenig lästig oder gar unlogisch? Zumindest ich fragte mich, warum man nicht automatisch in das eben erzeugte Verzeichnis hineinwechselt. Wie dem auch sei, wenn ich also nun ein neues Verzeichnis erzeugen und anschließend hineinwechseln will, benutze ich nun die Batchdatei MDCD.BAT:

```
@echo off
if %1.==. goto err
md %1> nul
cd %1> nul
goto off
:err
echo Aufruf z.B. mit: MDCD TESTVERZ o. mit
echo MDCD \TESTVERZ!!!
:off
```

Gebe ich beim Aufruf einen Verzeichnisnamen als ersten Parameter an, so wird dieses zunächst mit MD %1> NUL erzeugt, bevor der Verzeichniswechsel mit CD %1> NUL vonstatten geht. Solange man sich dabei auf das aktuelle Laufwerk beschränkt, ist der Wechsel sofort sichtbar. Dennoch kann man diese Routine gerne auch auf die anderen Laufwerke anwenden. Ein Laufwerkswechsel findet aber nicht statt!

Eine Zeitschleife im DIP-OS

Schon einmal habe ich darüber berichtet, wie man eine Art Zeitschleife programmieren kann:

```
FOR %%A IN (1 2 3) DO TYPE C:\CONFIG.SYS>NUL
```

Aber weil viele Wege ins ROM führen, kann man auch diese Zeile verwenden:

```
FOR %%A IN (1 2 3 4 5) DO HELP /?>NUL
```

die für MS-DOS und DIP-OS gleichermaßen geeignet ist. Ist nämlich keine CONFIG.SYS in C:\ zu finden, so wird die Wartezeit sehr kurz sein. Beim gegenteiligen Fall kann die Wartezeit sehr lang werden, sofern die CONFIG.SYS sehr groß ist! Die Bildschirmausgabe des Befehles HELP ist dagegen immer und auf jedem PoFo gleich groß und benötigt also eine feste Zeit, bis sie dargestellt wird. Nur wenn der PoFo mit einer höheren Taktfrequenz betrieben wird, an dieser Stelle mal einen Gruß an Stefan Kächele 8-), ist die Wartezeit doch etwas kürzer, als auf den normal getakteten PoFos.

Datei verschieben

Mit der folgenden Routine kann man eine Datei in ein anderes Laufwerk oder Verzeichnis verschieben. Die gewünschte Datei wird dabei nicht einfach kopiert, sondern auch gleich im Ursprungsverzeichnis gelöscht.
Hier kommt MOVEDAT.BAT:

```
@echo off
if %2.==. goto err
copy %1 %2 > nul
del %1
goto off
:err
echo Ziellangabe fehlt!
:off
```

Aufgerufen wird sie z.B. mit „MOVEDAT C:\SYSTEM\PERMDATA.DAT C:\". Wir verschieben in diesem Fall die Datei PERMDATA.DAT ins Hauptverzeichnis des Laufwerkes „C:".
Und so geht's: Die IF-Zeile prüft zunächst ob ein Zielpfad oder –Laufwerk angegeben wurde. Gegebenenfalls wird eine Meldung ausgegeben und abgebrochen. Ist ein Ziel vorgegeben, so wird die im Parameter 1 übergebene Datei ans, im Parameter 2 festgehaltene, Ziel kopiert. Anschließend wird die Datei mit DEL %1 im Ursprungsverzeichnis gelöscht. Fertig ist die Dateiwäscherei!

Der Agentenaustausch

... ist seit jeher ein geheimnisvolles und gefährliches Geschäft, da das kleinste Fehlverhalten in einem Blutbad enden kann! Zum Glück stehen beim REN-Befehl unter DOS und DIP-OS keine Geheimmächte bei Fuß, so daß kein Blutbad zu befürchten ist, oder etwa doch?

Man kann daran zweifeln, wenn man die Namen zweier existierender Dateien austauschen will. Ein schlechter Krimi? Ihr werdet DOS und DIP-OS mit anderen Augen sehen, wenn Ihr meine Vorschläge nachvollzogen habt!

Im Grunde genommen ist es nur eine kleine und einfache Aufgabe für das DOS (und DIP-OS), wenn zwei Dateinamen gegeneinander ausgetauscht werden sollen. Der unverfrorene Tastaturexperte würde diese Aufgabe mit drei Eingaben erledigen, um die Dateinamen T.DAT und L.LAL, zu vertauschen:

```
REN T.DAT %
REN L.LAL T.DAT
REN % L.LAL
```

Und schon heißt T.DAT L.LAL und L.LAL T.DAT. Wobei vorausgesetzt ist, das die Datei % nicht existiert!

Hat man dieses Spielchen öfter vor, so schreibt sich der engagierte Batcher eine Batchdatei namens TAUSCH.BAT, SWAP.BAT oder EXCHANGE.BAT:

```
@echo off
ren %1 %
ren %2 %1
ren % %2
```

Diese wird nun mit den Dateinamen T.DAT und L.LAL, als Parameter %1 und %2, aufgerufen, damit der Namenstausch sauber über die Bühne geht. Sauber??? Weit gefehlt, wie ich es mehr oder weniger leidvoll erfahren mußte!

Denn rufe ich obige Batchdatei auf, die ich als TAUSCH.BAT gespeichert habe, so bekomme ich recht unterschiedliche Reaktionen, die sich auf Grund der Anzahl und der Art der übergebenen Parameter ergeben. Wenn ich mich 1. nicht vertippt habe, 2. nicht eine nicht existierende Datei angebe, 3. wirklich zwei existierende Dateinamen verwende und 4. keine unerlaubten Zeichen in den Dateinamen verwende, so komme ich mit großer Wahrscheinlichkeit zum gewünschten Ergebnis!

Anderenfalls und das ist gar nicht so unwahrscheinlich, erhalte ich die wildesten Fehlermeldungen und Dateinamen, wenn ich die obige TAUSCH.BAT mit folgenden Eingaben starte:

```
TAUSCH T.DAT *.L
TAUSCH T.?AT ?.*
TAUSCH L.LLL T.DAT
TAUSCH L.LAL
TAUSCH
TAUSCH.BAT L.LAL J.LOP
TAUSCH.BAT *.* J.L??
TAUSCH :.DAT *.TXT
TAUSCH C:.DAT L.L
```

Selbst wenn einige der eben gezeigten Dateien wirklich existieren sollten, so sollte TAUSCH.BAT bei den meisten Startvarianten nicht zum gewünschten Erfolg führen! Probiert es doch einfach mal aus, aber wundert Euch nicht, wenn Euch blaue Bohnen um die Ohren fliegen sollten :)

Deswegen habe ich mir eine schußsichere Weste zu stricken versucht, bin aber auch nicht gegen jede Kugel der allgegenwärtigen Fehlermacht DOS gefeiht!

Immerhin ist meine Weste SWAPNAME.BAT sicherer als so manche andere Strickjacke, die nach gleichem Muster unter den Namen SWAP.BAT, TAUSCH.BAT oder FILECHG.BAT angeboten wird. Aber gegen Startvarianten wie:

```
SWAPNAME C:.DAT L.DAT
SWAPNAME L;L.TXT JUHU.DOK
SWAPNAME T.DAT %%%%%%%%
```

ist auch meine Version machtlos:

```
@echo off
if %1%2.==. goto err
if %2.==. goto err
if not exist %1 goto err
for %%a in (%1%2%3) do if not %%a==%1%2%3 goto err
if not exist %2 %0 %1 %1 %2
if not %3.==. shift
ren %1 %%%%%%%%>nul
ren %2 %1>nul
ren %%%%%%%% %2>nul
:err
if %1%2.==. echo Die beiden Dateinamen fehlten!
if not %1%2.==. if %2.==. echo Der zweite Dateiname
  fehlte!
if not %1%2.==. if exist %0%1 %0 %2 %3
if not %1%2.==. if not exist %1 if not %3.==. %0 %2 %3
if not %1%2.==. if not %0==%1 if not exist %1 if exist %2
  echo Die Datei %1 existiert nicht!
if not %1%2.==. if not exist %1 if not exist 2 echo Die
  Dateien %1 & %2 gibt es nicht!
if not %1%2.==. for %%a in (%1%2%3) do if not
  %%a==%1%2 echo Keine Wildcards * ? erlaubt!
```

Tja, wer auch immer den Spruch geprägt hat: „Nobody is perfect!“, hat damit eine wirklich perfekte Feststellung getroffen!

Fehlerhafte Eingaben minimieren

Mal Hand aufs Herz: Wer hat sich von Euch nicht auch schon einmal vertippt??? Also mir passiert das immer wieder, ganz besonders dann, wenn ich es eilig habe. Meistens reagiert das DOS oder das DIP-OS mit der Fehlermeldung „Befehl oder Programm nicht gefunden" anstatt mit dem gewünschten Resultat. Bei den folgenden Eingaben vertippe ich mich z.B. desöfteren:

```
CD system
DIR /p
DIR *.*
DEL datei.ext
COPY *.* a:
TYPE datei.ext /p
```

Und das sieht dann so aus:

```
D system
IR /p
DR *.*
DL datei.ext
COP *.* a:
TYP datei.ext
```

Natürlich sind dem DOS keine Befehle wie D, IR, DR, DL, COP und TYP bekannt, weshalb es mich zu recht mit der o.g. Fehlermeldung verärgert. Weil ich aber nicht auf die Ruhe und Gelassenheit warten will, die das fortschreitende Alter so mit sich bringen soll, behelfe ich mir anders!

Ihr ahnt sicherlich schon, das ich wieder einmal auf Batchdateien zu sprechen komme, weil ich ja meist in der Batchsprache spreche 8-)! Langer Rede kurzer Sinn, hier kommen sie auch schon:

D.BAT:
```
@CD %1
```

IR.BAT:
```
@DIR /p
```

DR.BAT:
```
@DIR %1
```

DL.BAT:
```
@DEL %1
```

COP.BAT:
```
@COPY %1 %2
```

TYP.BAT:
```
@TYPE %1 %2
```

Das Geheimnis im Namen der Batchdateien zu finden ist, mit denen ich also mehr Fehlertoleranz erreiche, sollte Euch jetzt aufgegangen sein.

Allerdings kommt die obige D.BAT in schwere Konflikte mit einer D.BAT, die ich benutze, um einfach auf das Laufwerk D: zu wechseln:

D.BAT:
```
@D:
```

So habe ich ein wenig weiter gedacht und letztlich diese D.BAT geschaffen:

```
@if %1.==. if exist d:\nul.* d:
@if %1.==. if not exist d:\nul.* dir /p
@if not %1.==. if not exist %1\nul.* md %1
@if not %1.==. if exist %1\nul.* cd %1
```

Mit Ihr kann man auf das Laufwerk D: wechseln, sofern es vorhanden ist, oder das Directory des aktuellen Verzeichnisses ansehen, wenn es kein Laufwerk D: gibt.

Ansonsten kann man auch mit Ihr ein neues Verzeichnis erstellen, in das nach der Erstellung automatisch hineingewechselt wird oder einfach nur in ein bestehendes Verzeichnis wechseln.

Für diese nicht unbeachtliche Anzahl von Möglichkeiten sind folgende Startvarianten erlaubt:

D

Ist ein Laufwerk namens D: vorhanden, so wird es das aktuelle Laufwerk. Gibt es D: nicht, so wird das aktuelle Inhaltsverzeichnis seitenweise dargestellt.

D system

Existiert ein Verzeichnis namens SYSTEM, so findet ein Wechsel in dieses statt. Ist das Verzeichnis SYSTEM nicht vorhanden, so wird es zunächst erzeugt und anschließend hineingewechselt.

Damit ist D.BAT nun ein passabler Ersatz für die Befehle D:, DIR /P, MD und CD. Gerne hätte ich auch noch den Befehl RD berücksichtigt, habe aber noch keine Idee, wie ich mit Sicherheit feststellen kann, das der Anwender auch wirklich das Verzeichnis löschen will, wenn er D.BAT aufruft.

KnowHow: DIP-OS

Label löschen

Das Label, das man jedem Datenträger unter DOS aufdrücken kann, soll eigentlich nur nach Rückfrage gelöscht werden. Zumindest möchte das DOS es so. Um es zu löschen, gibt man den Befehl LABEL ein und bekommt etwa diese Reaktion vom DOS:

```
Datenträger in c ist ramdisk
Bezeichnung?

Aktuelle Bezeichnung löschen (J/N)? j
```

Wenn also ein Label vorhanden ist, hier: ramdisk und man auf die Frage nach der Bezeichnung mit der Return-/Enter-Taste reagiert, so muß man beantworten, ob die Bezeichnung (das Label) gelöscht werden soll.

Weil das gerade in Batchdateien nicht unproblematisch ist, man müßte eine Datei bereitstellen, die die entsprechenden Tastendrücke enthält, wird der Befehl Label nur ungern in Batchdateien verwendet.

Vor geraumer Zeit habe ich schon einmal eine Batchdatei namens UNLABEL.BAT vorgestellt, die das löschen des Labels vollautomatisch vornahm. Doch wie ich vor kurzem feststellte, gibt es eine wesentlich kürzere und elegantere Lösung, um das Label zu löschen:

```
LABEL CLOCK$
```

ist alles was Ihr dazu benötigt!

CLOCK$ ist ein nicht dokumentiertes Gerät, welches auch unter MS-DOS vorhanden ist. Ob dort LABEL CLOCK$ denselben Effekt hat, wie hier auf dem PoFo unter DIP-OS, habe ich aber noch nicht ausprobiert!

KnowHow: DIP-OS

Die geheimen Optionen /F und /V

Von MS-DOS kenne ich die Option /V oder /v für den CHKDSK-Befehl. Wird sie mitangegeben, so werden alle Namen der überprüften Verzeichnisse und Dateien angezeigt. Leider bekomme ich auf dem PoFo keine Verzeichnisse und Dateien angezeigt, wenn ich CHKDSK /V eingebe, sondern „Falscher Parameter".

Auch die Option /f oder /F, über die ich schon einmal berichtete, ist erlaubt und soll Fehler im Dateisystem beheben. Sie wird sogar klaglos vom PoFo hingenommen, obwohl ich noch keine Auswirkung gespürt habe. Bei MS-DOS kann es dazu kommen, das die angezeigten Dateien um eine reihe Bytes gekürzt werden, wenn sie schadhaft sind!

Aber es reicht nicht aus, eine der beiden Optionen mit CHKDSK zu verwenden, um eine interessante und nutzbare Fehlfunktion des PoFos zu erzeugen!

Wenn wir uns zunächst auf die Option /F einigen und das Ergebnis von CHKDSK /F in eine Datei namens ABC.DAT umleiten, mit:

```
CHKDSK /F>ABC.DAT
```

so wird die unmittelbar darauffolgende Zeile, die eine Bildschirmausgabe verursachen soll, diese Bildschirmausgabe nach ABC.DAT umleiten und nicht auf den Bildschirm bringen!!!

Ihr könnt das mit der TSTCHK.BAT ausprobieren:

```
@echo off
chkdsk /F>abc.dat
help
```

Wenn ihr das ausprobiert habt und anschließend Euch den Inhalt von ABC.DAT anseht, mittels TYPE-Befehl oder der Textverarbeitung, bekommt Ihr das Ergebnis des HELP-Befehles zu sehen.

Hänge ich aber noch die Zeile:

```
TYPE ABC.DAT
```

an die obige TSTCHK.BAT an, so ist die Umleitung wieder hinfällig! D.h., es landet nur noch das Ergebnis von CHKDSK in ABC.DAT.

Mit der Option /V ist man da ein wenig besser beraten, da es dann nichts mehr ausmacht, ob eine weitere Bildschirmausgabe erfolgt oder nicht! Dafür muß man aber mit der Fehlermeldung „Falscher Parameter" leben, da es mir bislang nicht gelungen ist, diese mittels NUL-Gerät wegzublenden ohne die Umleitung zu annullieren.

Für den Test der Option /V empfehle ich diese TSTCHK.BAT:

```
@echo off
cls
chkdsk /v>abc.dat
help
pause
cls
if exist abc.dat type abc.dat
if exist abc.dat del abc.dat
```

Auf den ersten Blick mögen diese geheimnisvollen Umleitungen, die CHKDSK /F oder CHKDSK /V ermöglichen, nichts besonderes sein, schließlich könnte man das Ergebnis von HELP auch mit:

```
HELP>ABC.DAT
```

nach ABC.DAT umleiten, aber es gibt auch für diese Art der Umleitung einige sinnvolle Anwendungen:

```
if %1.==. chkdsk /v>abc.dat
dir %1
```

wenn man wirklich nur das Verzeichnis anzeigen lassen will, das als %1 übergeben wurde. Sollte nämlich %1 leer sein, so würde DIR %1 zur Anzeige des gerade aktuellen Verzeichnisses führen!
oder wie ist es mit:

```
if not %1.==A. chkdsk /v
```

wäre z.B. ein guter Ersatz für

```
if not %1.==A. echo Falscher Parameter
```

Wie bereits erwähnt, muß hinter der Zeile mit CHKDSK /F>ABC.DAT oder CHKDSK /V>ABC.DAT, eine Zeile stehen, die eine Bildschirmausgabe zur Folge hat, damit diese Bildschirmausgabe in ABC.DAT landet!
Und das kann eine Zeile sein, die einen der folgenden Befehle enthält:
ECHO, HELP, DIR, CD, CHKDSK, FORMAT, TYPE, COPY, PATH, FDISK, LABEL, VOL, DATE, TIME, PAUSE oder VER!
Dasselbe Spielchen geht auch mit den Befehlen
APP, BREAK, CD, CLS, COPY, DIR, FDISK, FORMAT, LABEL, OFF, PAUSE, SHIFT, TYPE, VER, VOL und VERIFY, aber dort haben die Optionen /F und /V ein- und dieselbe Wirkung wie CHKDSK /V>ABC.DAT!

DOS Fehlermeldungen

... kann man sich zu nutze machen und somit einige Bytes sparen, wenn man Batchdateien programmiert!
Ich bin zwar mehr ein Fan von deutlicherer Formulierung, als das DOS bietet, aber es muß ja nicht jeder nach meiner Pfeife tanzen 8-)
Wenn die laufende Batchdatei das Wort USER als Parameter %1 erwartet und es nicht bekommt, so könnte man den Anwender mit dieser Zeile:

```
IF NOT %1.==USER. ECHO Falscher Parameter
  informieren.
```

Etwas kürzer ist dagegen:

```
IF NOT %1.==USER. VER //
```

und gibt auch die Meldung „Falscher Parameter" aus.

Wenn eine gesuchte Datei nicht aufzufinden ist, so kann man dem Anwender mit der Zeile:

```
IF NOT EXIST LOLO.DAT ECHO Datei nicht gefunden
  Bescheid geben. Allerdings ist die Variante:

IF NOT EXIST LOLO.DAT RUN //
```

kürzer und erfüllt den gleichen Zweck.

Auch wenn ein Schreibfehler des Anwenders zu vermuten ist, sollte man darauf hinweisen:

```
IF NOT %1.==DIR. IF %1.==IR. ECHO Syntaxfehler
```

Oder benutzt die Kurzvariante:

```
IF NOT %1.==DIR. IF %1.==IR. FOR
```

Wenn eine Batchdatei unbedingt drei Parameter benötigt, damit sie funktioniert, so ist dieser Check angebracht:

```
IF %1%2%3.==. echo Falsche Parameteranzahl
```

Die gleichsam wirkende Kurzvariante lautet:

```
IF %1%2%3.==. TYPE
```

Soll der Anwender darüber informiert werden, das die gewünschte Datei existiert, so kann man das mit:

```
IF EXIST ABC.DAT ECHO Zieldatei existiert
```

erreichen. Aber auch mit:

```
IF EXIST ABC.DAT REN NUL L
```

ist das möglich!

Exklusives Datei löschen

Ist es Euch nicht auch schon so ergangen, das alle Dateien gelöscht werden sollen, mit Ausnahme einer einzigen?
Im Regelfall erfordert das viel Handarbeit, weil man fast alle anderen Dateien einzelnd, mit dem DEL-Befehl, vernichten muß! Um das gesagte etwas plastischer werden zu

lassen, präsentiere ich nun ein beispielhaftes Verzeichnis:

Datenträger in c hat keine Bezeichnung.
Inhaltsverzeichnis von c:\TEST

```
.           <DIR>                                   4.01.80
22.53
..          <DIR>                                   4.01.80
22.53
@!@-@!@  fdi        0   1.01.80   0.14
permdata dat      730   4.01.80  23.41
undelete dat     1999   4.01.80  23.40
clipbord dat       27   4.01.80  23.26
t        txt        3   4.01.80  23.41
install  txt     2058   1.01.80   1.03
──── ─           0   1.01.80  23.26
install  bat      102   1.01.80  22.41
     10 Dateien 512 Bytes frei
```

Wenn ich, abgesehen von der Datei @!@-@!@.FDI, alle anderen Dateien löschen möchte, so müßte ich nacheinander diese Befehle im DOS eingeben:

```
DEL *.??T
DEL *.─
```

Ein weniger erfahrener DOS-User würde aber eher diese Befehle verwenden:

```
DEL *.DAT
DEL *.TXT
DEL *.BAT
DEL *.─
```

um alle Dateien, mit Ausnahme der @!@-@!@.FDI, zu löschen. Egal welche Reihe von Befehlen man bevorzugt, es ist doch immer mit einer gewissen Tipparbeit verbunden!

Da ich kein Freund von Monotonie und kleinen Tastaturen bin, habe ich nach einer Lösung gesucht, die das vereinfachen könnte. Okay, in der halben Stunde, in der ich die unten befindliche EXDEL.BAT schuf und testete, hätte ich die obigen Befehle bestimmt ein Dutzend mal eingeben können, aber wozu ist man schließlich ein „HighBatcher" 8-)?

```
@echo off
REM EXDEL.BAT - alle Dateien löschen,außer der als
   Parameter %1 übergebenen! BSP.: EXDEL
   autoexec.bat

rem NICHT-ZU-LÖSCHENDE-Datei angegeben ?
if %1.==. goto err
rem Existiert die angegebene Datei überhaupt ?
if not exist %1 goto err
rem Dateinamen in Großbuchstaben umwandeln!
set ß=%1
rem Existiert die Variable ß ?
if %ß%.==. goto err
rem Wildcards im Dateinamen verwendet ?
for %%a in (%1) do if not %1==%%a goto err
rem Alle Dateien löschen, außer der angegebenen!
for %%a in (*.*) do if not %ß%==%%a del %%a
rem Fehlerbehandlung!
:err
if %1.==. echo Die NICHT zu löschende Datei fehlte!
if not %1.==. if not exist %1 echo Die Datei %1 existiert
   nicht!
if not %1.==. if exist %1 if %ß%.==. echo
   Umgebungsspeicher voll!
if not %1.==. for %%a in (%1) do if not %1==%%a echo
   Keine Wildcards * ? erlaubt!
set ß=
```

Die Zeilen, die mit REM beginnen, müssen nicht abgetippt werden!

Wenn ich mich also in dem oben gezeigten Verzeichnis befinde und dort alle Dateien, außer der @!@-@!@.FDI, löschen möchte, so rufe ich nun EXDEL.BAT mit der Eingabe von:

```
EXDEL @!@-@!@.fdi
```

auf. Dabei spült es keine Rolle, ob ich den Namen der nicht zu löschenden Datei in Groß- oder Kleinbuchstaben angebe. Nicht erlaubt wären dagegen Aufrufe wie:

```
EXDEL *.FDI
EXDEL ????????.FDI
EXDEL
```

Ihr könnt es aber trotzdem einmal versuchen! EXDEL.BAT wird darauf mit einer Fehlermeldung antworten und keine einzige Datei löschen.

Gerne hätte ich EXDEL.BAT auch dahingehend dressiert, das man z.B. durch die Eingabe von EXDEL *.DAT alle Dateien löscht, die nicht die Extension .DAT tragen, aber da muß ich noch weiterforschen!

„IF ... ECHO .>Datei“

Eine Kombination dieser Befehle ist mir bislang noch in keiner Batchdatei begegnet, obwohl sie durchaus reizvoll und nutzbar ist. So habe ich diese Technik selbst schon gebrauchen wollen und können, bin aber dabei böse vom DOS überrascht wurden!

So wird immer die Datei angelegt, die auf das Größerzeichen folgt, egal ob die vorausgehende Bedingung zutrifft oder nicht!

Eine beispielhafte Batchzeile dazu:

```
if %1.==. echo j>test.dat
```

War kein Parameter vorhanden, so wird eine Datei TEST.DAT kreiert, die drei Bytes groß ist und das j enthält. War aber der erste Parameter vorhanden, so wird eine Null Byte große Datei namens TEST.DAT erzeugt.

Will man also anschließend prüfen, ob die Datei TEST.DAT existiert oder nicht, so wird man immer vom DOS aufs Glatteis geführt, da die Datei immer existiert, aber nicht immer den gewünschten Inhalt hat.

Das wiegt sehr schwer, wenn diese Datei statt TEST.DAT TEST.BAT heißen soll, also eine später aufrufbare Batchdatei sein soll! Die TEST.BAT wird zwar in jedem Falle funktionieren, aber sie könnte gerade beim PoFo nicht ohne weiteres als rekursive Batchdatei gestaltet werden.

Wie dem auch sei, ich möchte hiermit nur auf das Problem hinweisen, das die Datei in jedem Falle und unabhängig von der vorausgehenden Bedingung erzeugt wird. Dabei kommt es auch nicht darauf an, ob das Vorhandensein oder Nichtvorhandensein eines Parameters, einer Variable oder einer Datei geprüft wird!!!

KnowHow: DIP-OS

In einem Atemzug

... kann man doch das Laufwerk und das Verzeichnis wechseln!

Wahrscheinlich kommen Euch diese Worte zu recht bekannt vor, weil ich mit diesen schon einmal eine Batchdatei CDD.BAT vorstellte, die den gleichzeitigen Wechsel von Laufwerk und Verzeichnis ermöglichte.

Seitdem sind bereits einige Jahre ins Land gegangen, in denen ich mich weiterentwickelt habe und eine Überarbeitung der CDD.BAT möglich wurde. Mit der unten stehenden Version kann man nun auch in das Hauptverzeichnis des jeweiligen Laufwerkes wechseln und auch das als D: anzusprechende DISKFOLIO oder FolioDrive Diskettenlaufwerk ist nun eingebunden. Die Variable LW, aus der ersten Version, ist unnötig geworden und die Fehlerrate deutlich gesunken

Vorhang auf, hier kommt die neue CDD.BAT:

```
@echo off
if %1.==. goto err
if not exist %1\nul.* if not exist %1nul.* goto err
cd %1
if not exist %1nul.* goto dy
rem>%1%1
goto chgdrv
:dy
rem>%1\%1
:chgdrv
if exist %1\d. D:
if exist %1d. D:
if exist %1\c. C:
if exist %1c. C:
if exist %1\b. B:
if exist %1b. B:
if exist %1\a. A:
if exist %1a. A:
if exist d. del d.
if exist c. del c.
if exist b. del b.
if exist a. del a.
:err
if %1.==. echo Ziellaufwerk und -verzeichnis fehlten!
if not exist %1nul.* if not exist %1\nul.* echo Ziellaufwerk
   und -verzeichnis gibt es nicht!
```

Wenn man z.B. vom Hauptverzeichnis des Laufwerkes A: ins Verzeichnis SYSTEM des Laufwerkes C: wechseln will, so gibt man:

```
CDD C:\SYSTEM
```

ein. Möchte man dagegen vom Verzeichnis \SYSTEM auf C: ins Hauptverzeichnis des Laufwerkes A: wechseln, so erreicht man das durch die Eingabe von:

```
CDD  A:\
```

Was bei der Benutzung von CDD.BAT zu beachten ist:

Im gewünschten Laufwerk muß ein Datenträger vorhanden sein, der nicht schreibgeschützt ist und der noch Platz für eine temporäre Datei hat!

KnowHow: DIP-OS

Stringbearbeitung unter DOS

Unter der Stringbearbeitung versteht man ja gemeinhin das Bearbeiten von Wörtern. So trennt man z.B. das Wort Eis aus Eismeer heraus und arbeitet mit dem so herausgetrennten Eis weiter. Für solche Stringbearbeitung ist das DOS in der Regel nicht geeignet, obwohl diese Möglichkeit schon Sinn macht, wenn man Datei-, Disketten-, Verzeichnisnamen, o.ä., herausfinden will.

Bei den großen PCs unter MS-DOS existiert aber eine unvorhergesehene Möglichkeit, die auf dem Backslash basiert und mit der man das Buchstabieren erreichen kann. Diese funktioniert aber nicht auf dem PoFo!

Durch Zufall bin ich aber auf eine ähnliche Methode gekommen, mit der man von einem Wort zwischen zwei bis vier Buchstaben abschneiden kann. Probieren Sie dazu mal CUT.BAT aus, welche Sie im Hauptverzeichnis der Ramdisk C: abspeichern müssen:

```
@echo off
if %1.==. c:\cut.abc.Portfolio
echo %1
```

Wer aufmerksam ist und sich bereits ein wenig mit Batchdateien auskennt, wird meinen, das CUT.BAT nicht Funktionieren kann, da ein Aufruf der Datei C:\CUT.ABC nicht möglich ist. Schließlich existiert erstens keine solche Datei und zweitens können vom DOS nur die Programme gestartet werden, die mit .EXE, .COM oder .BAT enden - nicht aber mit .ABC!

Und dann löst auch noch der Anhang .Portfolio Verwirrung aus, weil man normalerweise Parameter durch Leerzeichen getrennt übergibt!

Dennoch hat das so seine Richtigkeit und funktioniert, weil der PoFo sich nicht im geringsten um die vorgegebene Endung kümmert und nur so zur Stringbearbeitung gezwungen werden kann. Bisher zumindest.

Starte bitte CUT.BAT ohne Parameter, also durch die Eingabe von CUT und Du sollst folgende Anzeige erhalten:

```
olio
```

Wie unschwer zu erkennen ist, hat der PoFo also von dem Wort Portfolio nur noch die letzten vier Buchstaben behalten. Bitte ändere nun die Zeile zwei von CUT.BAT wie folgt:

```
if %1.==. c:\cut.ab.Portfolio
```

Nach dieser Änderung und dem erneuten Start von CUT.BAT, wie immer ohne Parameter, werden Sie feststellen, das nur noch lio von Portfolio übrigbleibt.

Bitte ändere nun die Zeile zwei von CUT.BAT wie folgt:

```
if %1.==. c:\cut.a.Portfolio
```

Nach dem nunmehr dritten Start bleibt nur noch io vom Portfolio übrig. Das ganze Geheimnis liegt an der Dateiendung und dem darauffolgenden Punkt:

Besteht die Dateiendung aus vier Zeichen, z.B. .abc, so werden die letzten vier Buchstaben abgeschnitten. Besteht sie nur aus drei Zeichen, z.B. .ab, so werden die letzten drei Buchstaben abgeschnitten. Bei nur zwei Zeichen, z.B. .a, werden nur die letzten zwei Buchstaben abgetrennt.

Lassen Sie die Dateiendung, hier .ABC gänzlich weg, so werden generell die letzten vier Buchstaben zum Inhalt des Parameters %1:

```
if %1.==. c:\cut.Portfolio
```

Diese Trennung ist aber in jedem Fall nur möglich, wenn der Punkt vor dem gewünschten Wort, hier Portfolio, nicht fehlt!

Versuchen Sie eine Trennung bei Wörtern durchzuführen, die ASCII-Charaktere oberhalb 128 beinhalten, z.B. ö ä ü ß, so wird die Trennung anders ausfallen. Aber das können Sie ja mal selbst ausprobieren!

CUT.BAT ist nur ein Beispiel um die Stringbearbeitung zu demonstrieren. Wenn Sie also CUT.BAT durch die Eingabe von c:\cut.abc.Portfolio starten, so erhalten Sie als Ergebnis auch wieder den Wortfetzen olio auf dem LCD-Schirm.

Sie könnten also genauso gut die folgende TEST.BAT im Hauptverzeichnis der Ramdisk C: speichern...

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

... und mit der Eingabe \test.123.permdata starten, um das Wort data angezeigt zu bekommen.

Ich bin jetzt auf jeden Fall darauf gespannt, wie Du diesen Trick in Batchdateien verwendest.

Batchdateien in der Literatur

In Zeiten zunehmender Verfensterung ist es natürlich nicht leicht, Bücher zum Thema Batchdateien im Handel zu finden. Aber mit etwas Glück und Geduld kann man gesuchte Bücher von Privat erwerben.

Leider gibt es nur recht wenige Bücher in deutscher Sprache dazu und diese sind meist nur für den wißbegierigen Beginner interessant:

"DOS Batch Programmierung" von Wolfgang Weber. 1992, ISBN 3-8171-1266-1, Verlag Harri Deutsch.

"BATCH-Betrieb und Automation unter DOS und Windows 3.x/95" von Alois Kneisle. 1996 (!), ISBN 3-446-18658-1, Carl Hanser Verlag. Dieses Buch geht sehr in die Tiefe.

„So geht's! Utilities und Batchdateien" von Rudi Kost, Josef Steiner und Robert Valentin. 1993, ISBN 3-87791-467-5, Markt & Technik Verlag.

„Batch-Dateien - Einsteigerseminar" von Dr. Ekkehard Kaier. 1992, ISBN 3-89360-618-1, bhv Verlag.

„Batchdateien für MS-DOS 6.0 entwickeln" von Hellermann und Lemme. 1993, ISBN 3-7723-5023-2, Franzis-Verlag.

„Stapeljobs unter MS-DOS" von Paulissen & Paulissen. 1990, ISBN 3-88963-241-6, Hofacker Verlag.

"BATCH-PROGRAMMIERUNG unter DOS" von Heinz-Peter Glaß. 1991, ISBN 3-88322-332-8, IWT Verlag.

Im Frühjahr '96 soll angeblich noch ein Buch mit dem Titel „Batch mal wieder!" erscheinen. Einer der Autoren soll Werner „Tikki" Küstenmacher sein. Dem Titel nach, könnte es ein Buch für fortgeschrittene Batchprogrammierer werden. Hoffen wir also das Beste!

Weitere Bücher, die meist ein Kapitel zum Thema Batchdateien haben, sind: „MS-DOS 6 - Tips & Tricks" von Helmut und Manfred Tornsdorf. 1993, ISBN 3-8158-1014-0, Data Becker.

„MS-DOS Trainer" von Bach und Backer. 1995, ISBN 3-8043-3019-3, Augustus Verlag.

„Das MS-DOS Kompendium" von Chris DeVoney. 1985, ISBN 3-89090-106-9, Markt & Technik Verlag.

Aus der Rolle fallen die folgenden Bücher: „Data Beckers große MS-DOS Toolbox" von Helmut und Manfred Tornsdorf. 1990, ISBN 3-89011-824-0, Data Becker. Dieses Buch bietet eine ausführliche Einführung in die Batchprogrammierung und zusätzlich viele Hilfsprogramme auf beiliegender Diskette.

„Softwarelexikon MS-DOS" von Thomas Tai und Peter Freese. 1988, ISBN 3-499-18152-5, rororo Verlag. Es ist mehr für den engagierten MS-DOS Anwender gedacht, da es u.a. beschreibt, ab wann welcher Schalter, welche Option und welcher Befehl in MS-DOS eingeführt wurde. Aber auch Begriffe wie Akkustikoppler usw. werden dort erklärt.

An Zeitschriften kann ich empfehlen:

1. Die Sonderhefte des PC-Welt Magazins
2. Die ersten zwei Ausgaben des Magazins „100 Tips & Tools für Windows & MS-DOS" vom Verlag VTP Fürst
3. „Was ist denn DOS?" vom KnowWare Verlag

Ein bis zwei Seiten mit DOS-Tricks und Batchdateien findet man auch monatlich in den Zeitschriften PC-Welt, PC Praxis, PC go!, CHIP, DOS und der Highscreen Highlights.

Ansonsten kann ich den Datendienst CompuServe wärmstens empfehlen, speziell das MSDOS-Forum (GO MSDOS), da man dort die besten Batchprogrammierer antrifft. Selbst wenn eine Frage schon dreißigmal gestellt wurde, wird sie auch beim 31. mal beantwortet!

Über den PC-File-Finder von CompuServe (GO PCFF) kann man auch eine ganze Reihe von Batchdateien finden, wenn man als Schlüsselworte (Keywords) BATCH und DOS, oder als Extension BAT einträgt.

Leerzeilen bei der Bildschirm-ausgabe durch Batchdateien?

Normalerweise kann mit dem -ECHO-Befehl keine Leerzeile auf dem Schirm produziert werden. Bei den großen PC's behilft man sich deshalb mit einem Trick und verlängert den Befehl um beispielsweise einen Punkt: -ECHO.- , doch der PoFo spielt da wieder nicht mit.

Glücklicherweise braucht man aber trotzdem nicht auf die Leerzeile zu verzichten! Mit der folgenden Tastenkombination, die beim schreiben von Batchdateien im PoFo eigenem Texteditor Gültigkeit hat, kann man Leerzeilen für spätere Bildschirmausgaben erzeugen:

- Das Wort -ECHO- eintippen
- Taste <FN> drücken und festhalten
- Taste <N> drücken
- Beide Tasten loslassen, der Cursor sollte nun als Unterstrich dargestellt werden „_"
- Taste <ALT> drücken und festhalten
- Taste <K> drücken, Taste <I> drücken und noch einmal Taste <I> drücken
- Taste <ALT> loslassen
- Taste <FN> drücken und festhalten
- Taste <N> drücken, der Cursor wird wieder zum Vollcursor „¦"
- Die Taste „<+" (Enter o. Return) drücken, fertig!

Mit dieser Tastaturakrobatik hat man dann also endlich eine Leerzeile, für die spätere Bildschirmausgabe, erzeugt. Auch eine zentrierte Textausgabe kann so auf Wunsch kreiert werden! Dazu muß für jedes gewünschte Leerzeichen vor dem Text, die Tastenkombination <Alt> halten, <K> drücken, <I> drücken, <I> drücken, Alt loslassen, durchgeführt werden.

Optisch ansprechende und deutlich erkennbare Bildschirmmasken sind damit garantiert und werden jedem Betrachter ein erstauntes „Oh!" abringen! Das funktioniert auch auf dem PC, auch wenn man dort das ominöse Leerzeichen (ASCII-charachter 255) durch andere Tastendrücke erzeugen muß!

Wildcards * ? - Teil 2

Im ersten Teil zeigte ich Euch, wie man die Wildcards * und? im Parameter %1 erkennen und verbieten kann. Dies ist ja immer dann von Interesse, wenn das Batchprogramm nur mit einer Datei zur Zeit arbeiten soll. Wer die Technik aus dem ersten Teil verwendet, wird dann einen Abbruch des laufenden Batchprogrammes erzwingen, wenn im Parameter %1 zumindest einmal eines der Wildcards vorkommen sollte.

Dieser Abbruch mag aber für manchen von Euch unbefriedigend sein, weil dadurch das gewünschte Ziel nicht erreicht wird. So kann es ja vorkommen, daß das Batchprogramm wirklich mehrere Dateien benutzen oder bearbeiten soll. Darum gilt es nicht abzubrechen, wenn Wildcards verwendet wurden, sondern der Routine mitzuteilen, das mehrere Dateien verwendet werden sollen: Schließlich haben wir ja gerade dafür die Wildcards!!!

Wie man also eine solche intelligente Batchdatei schreibt, die selbständig erkennt, ob eine oder mehrere Dateien verarbeitet werden sollen, will ich nun einmal beispielhaft demonstrieren:

```
@echo off
rem   wildcards checken und erlauben!
for %%a in (%1) do if not %1==%%a goto many
rem  hier die Zeile für eine Datei:
copy %1 b:
goto off
:many
rem  hier die Zeile für mehrere Dateien:
for %%a in (%1) do copy %%a b:
:off
```

Die oberste und erste FOR-Zeile prüft also wieder die Wildcards ab und springt, wenn ein Wildcard im Parameter %1 steckte, zur Sprungmarke :many. Dort wird dann mit der FOR-Zeile der Kopierbefehl für jede vorhandene Datei ausgeführt, auf die die Wildcards passen. War kein Wildcard im Parameter %1 zu finden, so wird nur die Datei kopiert, die als Parameter %1 übergeben wurde:

```
COPY %1 B:
```

Danach wird zur Sprungmarke :off gesprungen, damit die Datei nicht noch einmal kopiert wird.

Leider ist das Beispiel nicht sehr gut, weil man auf die Zeile COPY %1 B: verzichten kann: Denn die FOR-Zeile aus :many funktioniert auch dann, wenn sich kein Wildcard in %1 befindet. D.h., man kann auf die Prüfung der Wildcards verzichten und die Routine wie folgt verkürzen:

```
@echo off
for %%a in (%1) do copy %%a b:
```

Diese Version geht immer, egal ob man Wildcards verwendete oder nicht! Da die

Wildcards ? und * nur mit Dateinamen funktionieren, sollte man immer eine FOR-Schleife verwenden, wenn eine, oder mehrere Dateien bearbeitet werden sollen! Man erreicht durch die FOR-Schleife auf jeden Fall eine höhere Flexibilität.

Sprungmarken - Teil 3

Es soll ja sehr sparsame Leute geben, deren Sparsamkeit sich sogar bei der Programmierung von Batchdateien bemerkbar macht!

So sparen sie sich die üblicherweise einleitende Anfangszeile @ECHO OFF, wenn die gesamte Batchdatei nicht mehr als elf Zeilen lang wird. Dafür beginnen sie dann die zu schreibenden Batchzeilen mit dem Klammeraffen @, der die Bildschirmausgabe unterdrückt:

```
@cls
@dir /p
```

Weniger Geizige würden dieselben Zeilen so schreiben:

```
@echo off
cls
dir /p
```

Wer beide Versionen abtippt und speichert, wird feststellen, das die erste Version um neun Bytes kürzer ist und doch genau wie die zweite Version funktioniert. Warum sollte man also eine Batchdatei mit der Zeile @ECHO OFF beginnen, wenn ein Klammeraffe @, vor jeder Zeile, dasselbe bewirkt??? Weil Sprungmarken da wieder einen Strich durch die Rechnung des Programmierers machen, lautet darauf die Antwort!!! Denn steht vor einer Sprungmarke ein Klammeraffe, so kann diese Sprungmarke nicht vom DOS angesprungen werden. Man erhält dann nämlich die bekannte Fehlermeldung „Sprungmarke nicht gefunden"!

Mit diesen Zeilen könnt ihr das ja einmal ausprobieren:

```
@goto jj
@rem
@:jj
@dir /p
```

Läßt man dagegen den Klammeraffen vor der Sprungmarke :jj wieder weg, so wird die Sprungmarke auch vom DOS gefunden und ausgeführt. Sogar ohne die Anzeige der Zeile auf dem Bildschirm!

Was lernen wir also daraus?

1. Egal ob das ECHO on oder off ist, eine Zeile, die mit einem Doppelpunkt beginnt, wird nie auf dem Bildschirm angezeigt!
2. Möchte man zu Testzwecken die Ausführung einer Sprungmarke verbieten und statt dessen einen Abbruch erzwingen, so fügt man den Klammeraffen @ vor dem Doppelpunkt der Sprungmarke ein! Ein Punkt „." hat übrigens dabei dieselbe Wirkung!

Letzteres ist nur bei der Erprobung und Fehlersuche zu empfehlen. Oder hat jemand eine Idee, wie man das sonst nutzen könnte?

KnowHow: DIP-OS

Bedingtes SHIFTen

Mit dem Befehl SHIFT werden Parameter verschoben, oder Schleifen konstruiert, die in Abhängigkeit von Parametern ausgeführt werden. Wer also diesen Befehl einsetzt, weiß in der Regel genau, was bezweckt werden soll.

Durch ein paar Experimente bin ich auf neue Erkenntnisse zu diesem Befehl gekommen, auch wenn ich noch keine Routine präsentieren kann, die davon profitiert. Angeblich ist der Befehl SHIFT nicht durch Parameter, Variablen oder Optionen ergänzbar. So kann man z.B. keinen Parameter angeben, ab dem geshifted werden soll. Kurioserweise kann ich aber beim PoFo eine beliebige Variable nach -SHIFT- angeben, die damit das Verhalten von -SHIFT- ändern kann:

```
-SHIFT %!%-
```

Wenn keine Variable! existiert, so wird ganz normal geshifted. Enthält aber die Variable! irgendwelchen Inhalt, so bekommt man die Fehlermeldung „Falscher Parameter" und -SHIFT- wird nicht ausgeführt. Auch die Variable bleibt unverändert.

Weil ich die Fehlermeldung unterdrücken wollte, leitete ich sie dann nach NUL um:

```
-SHIFT %!%>nul-
```

Solange die Variable! nicht existiert, wird auch ordnungsgemäß geshifted. Wenn aber! existiert, erscheint nicht nur obige

Fehlermeldung, sondern auch die nächste Zeile kommt nicht zur Ausführung! Sie wird dann einfach übersprungen. Ich kann diese Fehlermeldung also nicht unterdrücken, wohl aber die Ausführung der nächsten Zeile unterbinden.

Wenn Ihr Euch davon selbst überzeugen wollt, so produziert mal die Variable! und modifiziert die „NOSHIFT.BAT“:

```
@echo off
echo %0 %1
shift %!%>nul
echo No SHIFT Test!
echo %0 %1
pause
```

Dank dieser undokumentierten Funktion kann man also Minisprünge machen, durch die eine Passage von Befehlszeilen doppeldeutig verwendbar ist. Es liegt nur an uns, dieses nutzbar zu machen!

Das Gerät NUL

... ist selten dokumentiert und dennoch häufig in Batchdateien zu finden. Was es damit auf sich hat und wie es eingesetzt werden kann, möchte ich hier zu erklären versuchen.

Meistens steht diese Gerätebezeichnung am Ende vieler Batchzeilen und präsentiert sich als Zeichenfolge „> NUL“. Mit dem Zeichen „>“, soviel dürfte bereits bekannt sein, wird eine Aktion, ein Text oder eine Meldung, vom Schirm auf ein anderes Gerät oder in eine Datei umgeleitet. So erzeugt der Befehl -ECHO Hallo User! > USER.TXT- eine Datei namens „USER.TXT“, in der der Text „Hallo User!“ abgelegt wird. Schreiben wir aber nach dem Umleitungszeichen „>“ z.B. „PRN:“, so wird der Text auf dem hoffentlich angeschlossenen Drucker ausgegeben.

Einige andere Aktionen, wie Verzeichnisanlegung, Dateikopierung oder Fehlermeldungen, sollen aber erst gar nicht auf dem Bildschirm erscheinen, da sie evtl. Textmasken zerstören oder wichtige Informationen vom Schirm verdrängen. Weil die Ablage in Dateien unnötige Platzverschwendung wäre, sollten alle Kommentare des Computers, er sabbelt halt gerne, im Nirwana (Nichts, EDV-Loch, Nullspeicher) landen. Und hier tritt das Gerät NUL in Erscheinung, welches alles schluckt, aber nie wieder ausspuckt! Hungrig wie es ist, arbeitet es mit jedem DOS-Befehl zusammen, auch wenn gerade dadurch die gewünschte Aktion vereitelt wird. So findet z.B. kein Verzeichniswechsel statt, wenn man den folgenden Befehl eingibt: -CD \ > NUL-. Die Meldung „X Datei(en) kopiert“, die nach einem erfolgreichen Kopiervorgang ausgegeben wird, kann man aber sehr wohl damit unterbinden: -COPY A:*.* C: > NUL-. Versucht man auf einem Laufwerk ein Verzeichnis zu erzeugen, welches bereits existiert, gibt es immer einen gehässigen Kommentar vom Computer. Hängt man das NUL-Gerät an den Befehl -MD SYSTEM-, der hier als Beispiel zu verstehen ist, übt sich der Computer in Toleranz und schluckt seine Meinung herunter. Die Syntax dazu lautet:

```
-MD SYSTEM > NUL-.
```

Besonders bei den Befehlen -MD-, -COPY- und -REN- kommt das Gerät NUL zur vollen Geltung, läßt sich aber noch bei vielen anderen Befehlen einsetzen. Den Nutzen bemerkt man erst beim ausprobieren!

Etliche Wochen funktionierte unsere spezielle Batchdatei, die vollautomatisch alle Dateien von A: auf C: kopierte, tadellos, aber nun tritt trotz der NUL-Umleitung ein Fehler auf, den der Bediener mit Abbruch, Wiederholen oder Ignorieren beantworten muß. Wozu leiten wir alle Computerkommentare nach NUL um, wenn er doch eine Fehlermeldung ausgibt? Es handelt sich dabei um einen so elementaren Fehler, daß der Computer unsere Umleitung ignorieren muß, damit der Bediener die Chance hat, diesen Fehler zu beseitigen. Kurz: Der Computer kann den Fehler nicht von sich aus beheben, oder gibt es einen der selbständig Disketten wechseln kann?

Guten Batchprogrammierern, und die wollen wir ja alle werden, ist aber selbst diese Unterbrechung ein Dorn im Auge. Sie geben daher die Antwort selbst durch eine bereitgestellte Datei, die die Antwort A, W oder I an den Tastaturpuffer übergibt. Können wir also einen Abbruch verantworten, oder wollen den Fehler auf unsere Art behandeln, so erstellen wir uns doch einmal diese Datei mit einem Texteditor! Editor starten, <A> eintippen, Rettern- bzw. Entertaste drücken und den Text unter dem Namen „A.DAT" abspeichern - fertig. In unserem bisherigen Kopierbefehl -COPY A:*.* C: > NUL- gilt es nun, die Datei „A.DAT" einzubinden.

Weil der geplante Tastendruck <A> aus der Datei kommen soll, muß eine Umleitung aus der Datei erfolgen. Dazu muß das gegenteilige Umleitungszeichen „<„ verwendet werden, inkl. der zu entnehmenden Datei, die in unserem Fall „A.DAT" heißt. Die neue Syntax lautet nun: -COPY A:*.* C: < a.dat > nul-.

Sollte also in Zukunft beim kopieren ein Fehler auftauchen, so wird die Aktion kommentarlos abgebrochen. Diese Technik muß aber jeder Batchprogrammierer auf eigene Gefahr verwenden!!!

Ein DOS vor Version 3.30

... kennt den Klammeraffen @ nicht, mit dem man in der Regel die erste Zeile einer Batchdatei einleitet. Unser PoFo ist da eine rühmliche Ausnahme!

Mit dem Klammeraffen @ erreicht man, und das sollte bereits bekannt sein, die Unterdrückung der Bildschirmausgabe, wenn eine Batchzeile abgearbeitet wird. Läßt man z.B. diese Zeilen:

```
REM TEST.BAT
ECHO Hallo User!
```

als TEST.BAT laufen, so ergeben sich folgende Bildschirmausgaben des DOS:

```
c:\test>REM TEST.BAT

c:\test>ECHO Hallo User!
Hallo User!

c:\test>
```

Stellt man dagegen beiden Zeilen den Klammeraffen @ vor(an):

```
@REM TEST.BAT
@ECHO Hallo User!
```

so sieht die Bildschirmausgabe dann so aus:

```
Hallo User!
c:\test>
```

Um nicht vor jede Zeile den Klammeraffen schreiben zu müssen, gibt es den globalen Befehl:

```
ECHO OFF
```

mit dem das DOS die Anzeige der gerade auszuführenden Batchzeile unterbindet. Weil solche Anzeigen nur stören und viel zu schnell über den Bildschirm huschen, beginnt man also eine Batchdatei mit der Zeile:

```
ECHO OFF
```

Damit auch diese Zeile nicht angezeigt wird, stellt man den Klammeraffen voran:

```
@ECHO OFF
```

Und schon wird nur das vom DOS auf dem Bildschirm ausgegeben, was aus den Befehlen ECHO, DIR, COPY, HELP, CHKDSK, DATE, TIME, FORMAT usw. resultiert. Auch die Fehlermeldungen werden ausgegeben.

Aber was tun, wenn das DOS nicht die Versionsnummer 3.30 trägt und die Batchdatei unter diesem DOS laufen soll? Das DOS würde nämlich die Zeile:

```
@ECHO OFF
```

mit einer Fehlermeldung quittieren: „Befehl oder Programm nicht gefunden", da es den Klammeraffen @ nicht kennt. Das ECHO wäre dann auch weiterhin eingeschaltet!

Aber die Lösung ist ganz einfach: Man lenkt die Fehlermeldung zum Gerät NUL um und wiederholt den Befehl ECHO OFF, aber diesmal ohne @:

```
@ECHO OFF>NUL
ECHO OFF
```

Und schon läuft auch diese Batchdatei unter DOS Versionen vor 3.30! Okay, okay, es kommt nicht mehr allzu häufig vor, das eine DOS Version vor 3.30 noch in Betrieb ist, aber wenn doch, dann wißt Ihr nun jedenfalls, wie Ihr das Problem ganz galant lösen könnt.

Schließlich ist diese Lösung dann unter jedem DOS, ohne Änderung, einsetzbar!

Erste Hilfe!!!

Der erste Start einer fremden Batchdatei ist mit viel Ungewißheit verbunden! So weiß man oft nicht, welche Parameter und wie viele Parameter angegeben werden müssen, welche Variablen vorher schon bereitzustellen sind und was die Batchdatei überhaupt erledigen soll.

Darum sollte man den Anwender nicht mit diesen Fragen im Regen stehen lassen, sondern eine Hilfsfunktion implementieren, beim schreiben des Batchprogrammes, die dem unerfahrenen Anwender etwas unter die Arme greift, bzw. ihn informiert! Ich habe mir dafür die folgenden Zeilen ausgedacht:

```
@echo off
for %%a in (. h. H. help. HELP. -h. -H. -help.
   -HELP.) do if %1.==%%a goto help
if %1.==?. goto help
if %1.==-?. goto help
if %1.==/?. goto help
if %1.==/h. goto help
if %1.==/H. goto help
if %1.==/help. goto help
if %1.==/HELP. goto help
rem
rem Hier wird dann das eigentliche Batchprogramm
   eingebunden!!!
rem
goto off
rem Hier beginnt die erste Hilfe!
:help
echo Hier können beliebig viele Zeilen mit ECHO
   ausgegeben werden, damit
echo man den Anwender über alles informieren kann!!!
:off
```

Wird eine Batchdatei gestartet, die diese Zeilen enthält, so informiert sie den Anwender, wenn dieser eine der folgende Zeichenfolgen, als Parameter %1, übergeben hat:

```
? h H help HELP /? /h /H /help /HELP -? -h -H -help -
   HELP
```

Und wenn kein Parameter beim Start angegeben wurde, so wird der Anwender ebenfalls informiert!

4DOS für den PoFo und andere MS-DOS Pockets

Könnt Ihr Euch noch an diese MS-DOS Erweiterung erinnern? Bis zur MS-DOS Version 3.30 feierte die Erweiterung 4DOS große Erfolge, da sie das MS-DOS um sehr interessante und nützliche Befehle ergänzte:

So kann man mit dem Alias-Befehl eigene Befehle kreieren, mit Gosub und Return in Batchdateien herumspringen, mit Input und Inkey Tastaturabfragen in Variablen packen (kein Errorlevel!), mit Screen Texte an beliebiger Stelle des LCD-Schirms plazieren, mit Call zwischendurch bis zu fünf andere Batchdateien aufrufen, mit Text und Endtext ganze Textblöcke ausgeben, mit Pushdir und Popdir sich Verzeichnisse und Laufwerksbezeichnungen merken u.v.m.

Neulich fand ich also 4DOS inkl. Handbuch auf dem Grabbeltisch eines EDV- Händlers und nahm es auch nach kurzem Studium mit. Problemlos konnte ich 4DOS auf meinem PoFo mit Bios 1.072 installieren. Eigentlich hätte mich das ja schon stutzig machen müssen, nachdem was man immer wieder mit dem PoFo erlebt. Der Reihe nach ging ich also die neuen Befehle durch und probierte sie aus.

Überraschenderweise funktionierte das Gros problemlos, aber die die nicht funktionierten, waren sinnigerweise die wichtigsten!

CLS, LIST, TYPE, HELP, KEYSTACK und DIR wollten nicht wie ich es wollte. Aber weil man sich mit 4DOS eigene Befehle kreieren kann, habe ich diese mit einigen Tricks und Batchdateien selber nachempfunden. Leider ist es dazu unerläßlich 4DOS zu verlassen und ins DIP-OS zurückzukehren. Nur bei den

reparierten Befehlen CLS und KEYSTACK, verbleibt man im 4DOS, sofern man nicht mit KEYSTACK explizit den Ausstieg erzwingt!

Einige bisher vertraute Befehle des DIP-OS sind unter 4DOS leider nicht mehr zugänglich: APP, CHKDSK, FDISK, FORMAT und OFF. Auch diese konnte ich aber mit FAKE, ALIAS und div. Batchdateien dem 4DOS beibringen. Wendet man nun die Befehle APP, CHKDSK und FORMAT wie gewohnt an, so ist die Rückkehr ins DIP-OS vorprogrammiert. Bei der Verwendung von OFF gelangt man aber beim erneuten Einschalten des PoFo, automatisch ins 4DOS. Hat man 4DOS autostartend auf der Memorycard installiert, so gelangt man auch nach dem FDISK-Befehl wieder ins 4DOS.

Die Doskey-Funktion gehört zu den interessantesten Erweiterungen, da man nun alle Eingaben mit den Cursortasten oben/unten durchblättern, modifizieren und reaktivieren kann.

Im großen und ganzen ist 4DOS also eine sehr sinnvolle Ergänzung zum DIP-OS und bringt viele Möglichkeiten zur Batchprogrammierung mit sich. Obwohl ich UPDATE und PORTDIV installiert habe, erschienen zwar so manchmal unsinnige Fehlermeldungen, wie z.B.: 'Unknown command „lias"', aber diese beeinträchtigten die Arbeitsweise des PoFo bislang nicht.

Hier in Deutschland wurde dieses Sharwareprogramm von der Kirschbaum Software GmbH, Kronau 15, 8901 Emmering vertrieben. Ob und zu welchem Preis es noch erhältlich ist, entzieht sich meiner Kenntnis.

Kurios fand ich nur, daß ich in einer Programmliste des Atari-Forums aus dem CompuServe, Stand: Juni '93, diese tolle Erweiterung nicht zu finden war. Dabei stammt 4DOS aus den USA und wurde 1989 programmiert! Vielleicht liegt es aber auch daran, daß man ohne das dazugehörige Handbuch nicht sehr weit kommt, gerade auf dem PoFo, da dort die Help-Funktion nicht ohne weiteres installierbar ist. Die gesamte Erweiterung benötigt schon alleine auf einer Diskette etwa 330 KB.

Zum Abschluß wieder eine Ungereimtheit, die mir in diesem Zusammenhang mit dem PoFo auffiel: Ich habe die 4DOS-Befehle aus dem Handbuch in meinen PC eingetippt und dann auf die Memorycard übertragen. Als ich mir dann diesen Hilfstext auf dem PoFo unter 4DOS anzeigen ließ, funktionierte gleich darauf meine CLS- Reparatur nicht mehr! Auf die Schliche bin ich dem Fehler gekommen, als ich mir diesen Hilfstext mal in der Textverarbeitung des PoFo ansah. Hinter dem Textende prangte doch glatt das Dateienzeichen „^Z". Kaum hatte ich das gelöscht und den PoFo von Grund auf neu gestartet, war dieser Fehler nie mehr gesehen!

ALLREAD.BAT - für Leseratten

Wer eine ganz bestimmte Dateigruppe sichten will, oder muß, ist gezwungen den TYPE-Befehl entsprechend häufig einzugeben. Diese Monotonie macht natürlich krank und schreit geradezu nach einer Batchdatei:

```
@echo off
cls
if %1.==. goto err
for %%d in (%1) do type %%d /p
goto off
:err
echo Dateigruppenangabe fehlt!
:off
```

Ruft man diese Datei mit ALLREAD *.BAT auf, so sollte man alle Batchdateien seitenweise angezeigt bekommen. Doch erste Tests ergaben, das diese Routine nicht unter jeder DOS-Version funktioniert, da es nicht immer die Option /p beim TYPE-Befehl gibt.

Hier die technische Erläuterung:

Zunächst wird nach dem Aufruf geprüft, ob überhaupt eine Dateiengruppe als Parameter %1 angegeben wurde. Entweder springt DOS zur Sprungmarke :ERR und gibt eine Fehlermeldung aus, oder es fährt mit der FOR-Schleife fort. In dieser verweilt das DOS solange, bis die Menge (%1), also alle Dateien die der Maske entsprechen, die als Parameter übergeben wurde, auf dem Schirm ausgedruckt wurde. Danach entfleucht ALLREAD.BAT kommentarlos zur Sprungmarke :off und stellt die Arbeit ein.

ANSI.SYS-TIP: Escapesequenz

Das man mit dem Bildschirm- und Tastaturtreiber ANSI.SYS viel erreichen kann, ist wohl unumstritten. Oftmals gibt man eine solche Befehlsfolge mit dem PROMPT-Befehl ein, da man sonst die Escapesequenz

(Charakter 27, Darstellung „^[„), nicht unter DOS erzeugen kann.

So kann man seinem Computer ein X für ein U vormachen, wenn man ANSI.SYS installiert hat und den Befehl PROMPT$e[„X";"U"p eingibt. Leider wird aber durch diese Eingabe auch das Prompt gelöscht, welches danach also rekonstruiert werden muß.

Zwar kann man das auch mit der folgenden Batchdatei realisieren, aber man verschenkt doch wertvollen Speicherplatz:

```
@set op=%prompt%
@prompt$e[„X“,“U“p

@prompt=%op%
```

Kürzer und einfacher ist da der Weg über den Befehl ECHO, da erstens ECHO mit @ECHO OFF ausgeschaltet sein darf und zweitens, keine Leerzeile auf den PromptBefehl folgen muß, die die Einstellung vornimmt:

```
@echo off
echo ^[[„X“;“U“p
```

Problematisch an dieser Version ist aber, wie eingangs erwähnt, die Erzeugung der einleitenden Escapesequenz, mit der jede ANSI betreffende Einstellung beginnen muß. Zwar kann man die Zeichenfolge „^[„, die für die Escapesequenz steht, per Tastatur einzeln eingeben, aber sie ist dann nicht gleichbedeutend wie die Escapesequenz „^[„, da sie dann aus zwei Zeichen besteht, während die Escapesequenz tatsächlich nur ein Zeichen beinhaltet, obwohl zwei auf dem Bildschirm ausgegeben werden.

Es bleibt einem also nur der Ausweg über eine „höhere" Programmiersprache, damit der Charakter 27 in Batchdateien genutzt werden kann. Ich habe mir dieses Zeichen mittels GW-BASIC geklaut und in die Batchdatei ANSIECHO.BAT geschrieben:

```
10 open „ansiecho.bat“ for output as #1
20 print #1,“ECHO „;chr$(27);chr$(91);
30 close
```

Möchte ich also nun eine Einstellung mit ANSI.SYS vornehmen, so lade ich ANSIECHO.BAT in meine Textverarbeitung und ergänze den ECHO-Befehl um die gewünschte Syntax, bevor ich den Text unter einem neuen Namen abspeichere.

Weil viele Wege nach ROM führen, kann man sich natürlich die Escapesequenz auch durch andere Programmiersprachen ergaunern!

Portfolio-User haben es dagegen etwas einfacher und behelfen sich mit der eingebauten Tabellenkalkulation. Dort wird einfach in die erste Zelle, des ansonsten leeren Arbeitsblattes, der Wert 27 eingegeben. Nach dem diese Eintragung unter dem Namen UNNAMED.WKS gespeichert wurde, lädt man diese in die Textverarbeitung und löscht alle Zeichen vor und hinter der Zeichenfolge „^[„. Anschließend setzt man vor ihr noch den Befehl ECHO, bevor man den so entstandenen Text als ANSIECHO.BAT abspeichert.

Auch alle anderen Steuerzeichen kann man so einfangen und in Variablen festhalten!

PS: Probiert doch mal in der PoFo-Textverarbeitung die Tastenfolge ALT-Ä ALT-Ü aus. Dieser Weg ist einfacher und erzeugt ebenfalls das ESC-Zeichen.

Batch bearbeitet das Batchverzeichnis

Normalerweise zieht man sich dabei, pardon der Batchdatei, den Boden unter den Bytes weg! Hat nämlich diese Batchdatei ihren Stammplatz in dem Verzeichnis, in dem alle Batchdateien stehen und dieses Verzeichnis muß vorübergehend gelöscht werden, so kann das DOS die entsprechende Batchdatei nicht mehr wiederfinden und bricht etwa mit folgender Meldung ab:

```
Diskette mit Stapeldatei einlegen!
Wenn bereit eine beliebige Taste drücken.
```

Da aber die Batchdatei sich selbst ausradiert hat, kann man so viele Tasten drücken wie man will. Denn das DOS wird diese Batchdatei nicht mehr finden!

Einzig der Abbruch mit <STG>+<C> hilft kurzfristig aus dieser Misere herauszukommen. Die restlichen geplanten Befehle müssen dann von Hand eingegeben werden, damit man die Ordnung wiederherstellen kann.

Ich habe dieses schon mehrmals erlebt und eine Lösung geschaffen, die die Batchdatei aus dem kritischen Verzeichnis verlagert:

```
@echo off
if %1.==. goto err
copy c:\batch\%1.bat c:\>nul
c:\%1 %2 %3 %4 %5 %6 %7 %8 %9
:err
echo Aufruf mit:
echo BATCH Batchdatei Parameter
```

Möchte ich also das Verzeichnis BATCH mit meiner Batchdatei „DEFRAG.BAT" defragmentieren, so starte ich diesen Vorgang mit:

```
-BATCH DEFRAG \BATCH-
```

Die Routine „BATCH.BAT" kopiert daraufhin „DEFRAG.BAT" in das Hauptverzeichnis von C: und startet anschließend „DEFRAG.BAT" vom Hauptverzeichnis von C:. Zwar erfolgt dieser Start nur mit maximal acht Parametern, aber das dürfte in den meisten Fällen auch genügen. Nachdem die eigentliche Defragmentierung abgeschlossen wurde, müßt Ihr die „DEFRAG.BAT" eigenhändig aus dem Hauptverzeichnis von C: entfernen, weil „DEFRAG.BAT" nicht rekursiv programmiert wurde!

```
[01] @echo off
[02] cls
[03] echo ¡————————————————————————+
[04] echo ¦°  Lars Aschenbach's  Batchfiles   °¦
[05] echo ¦              «¡»               ¦
[06] echo ¦ in action: BATHELP .BAT  v02-09-96 ¦
[07] echo ¦°      for DOS 3.30 or higher      °¦
[08] echo +————————————————————————————————¥
[09] rem
[10] ::: User Explanation:
[11] :::
[12] ::: BATCHFILE : BATHELP.BAT
[13] :::
[14] ::: FUNCTION  : show documentation of batch- or
   other files
[15] ::: SYNTAX    : BATHELP filename[.ext]
[16] ::: EXAMPLES  : BATHELP bathelp or: BATHELP
   readme.1st
[17] ::: GET HELP  : BATHELP
[18] ::: HINT      : wildcards * ? aren't allowed in filename
   and extension
[19] ::: HINT      : lines including ::: get interpreted as
   documentation
[20] :::
[21] ::: Technical Explanation:
[22] :::
[23] ::: DOS       : 3.30 or higher
[24] ::: EXTERNAL  : FIND, MORE
[25] ::: PERMVAR   : PATH
[26] ::: OPTIONVAR : BDV, BDY
[27] ::: TEMPVAR   : FL
[28] ::: USEDFILE  : BATHELP.BAT
[29] ::: TEMPFILES : unknown - made by FIND & MORE
[30] ::: MAKEFILE  : none
[31] ::: SCREEN    : 80 columns, 25 rows
[32] ::: VERSION   : 09 Feb 1996
[33] ::: TESTED ON : MS-DOS 6.22
[34] ::: AUTHOR    : Lars Aschenbach, Germany
[35] :::
[36] if %PATH%.==. echo.
[37] if %PATH%.==. echo  Can't work: Variable PATH is
   missing!
[38] if %PATH%.==. goto err
[39] set | find „windir">nul
[40] if not errorlevel 1 if errorlevel 0 echo.
[41] if not errorlevel 1 if errorlevel 0 echo  Can't work:
   Windows is running!
[42] if not errorlevel 1 if errorlevel 0 goto err
[43] if not exist %BDV%%BDY%nul if exist
   %BDV%%BDY%\nul set bdy=%BDY%\
[44] if not %1.==. shift
[45] if exist %BDV%%BDY%%0.bat set
   fl=%BDV%%BDY%%0.bat
[46] if exist %BDV%%BDY%%0.bat if %FL%.==. goto err
[47] if exist %BDV%%BDY%%0 set
   fl=%BDV%%BDY%%0
[48] if exist %BDV%%BDY%%0 if %FL%.==. goto err
[49] if exist %0.bat set fl=%0.bat
[50] if exist %0.bat if %FL%.==. goto err
[51] if exist %0 set fl=%0
[52] if exist %0 if %FL%.==. goto err
[53] if %FL%.==. goto err
[54] find „:::"<%FL% | find „find" /v | more
[55] echo  No (more) lines of documentation in %FL%
   found!
[56] :err
[57] if not %PATH%.==. if %FL%.==. echo.
[58] if not %PATH%.==. if %FL%.==. if %0.==bathelp.
   echo  Try: BATHELP filename[.ext]
[59] if not %PATH%.==.  if not %0.==bathelp. if %FL%.==.
   echo  File not found!
[60] set fl=
[61] echo.
```

Befehle in Variablen

Beim PoFo muß um jedes Byte Speicher gefeilscht werden, da das Originalteil max. 80KB intern für Userdaten bereitstellt. Externe Datenträger bis zu 4MB sind zwar erhältlich, aber unverhältnismäßig teuer. Abhilfe schaffen hier die Packer und Komprimierungsprogramme, die aber nichts mit Batchdateien anfangen können.

Nach dem Motto „Selbst ist der Programmierer", packen wir die Befehle in Variablen, deren Namen kürzer sind, als die der Befehle. Eventuell wird zwar dadurch die Ausführungszeit etwas länger, weil jede Variable wieder in einen Befehl umgewandelt wird, aber man spart doch einige Bytes.

Hier also die Liste der Befehle und der Ersatzvariablen, die ich dafür vorschlage:

BEFEHL:	VARIABLE:
-BREAK ON-	BN
-BREAK OFF-	BF
-CHKDSK -	CD
-COPY -	C
-DATE -	W
-)DO -	D
-ECHO -	E
-FOR %%A IN	FA
-FORMAT -	F
-GOTO -	G
-IF ERRORLEVEL -	IR
-IF EXIST -	IE
-IF NOT EXIST -	INE
-IF NOT %-	INP
-IF %-	IP
-LABEL -	L
-PATH-	A
-PAUSE-	P
-PROMPT-	R
-SET -	S
-SHIFT-	H
-TIME -	Z
-TYPE -	T

Folgende Befehle bringen keine Zinsen ein, wenn man diese in Variablen unterbringt: CLS, CD, DEL, DIR, MD, RD, REM, REN, VER und VOL.

Eine Batchdatei könnte dann z.B. so aussehen:

```
@%E%OFF
%IP%1.==. %G%ERR
%FA%*.*%D%%S%TEST=%%A
%IP%1==%TEST% %E%%1 ist die letzte Datei in diesem
  Verzeichnis
%INP%1==%TEST% %E%%1 ist nicht die letzte Datei in
  diesem Verzeichnis
%G%off
:err
%E%Erster Parameter fehlte!
:off
```

Weil das ganze ziemlich abstrakt ist, hier noch einmal in „Klarschrift":

```
@echo off
if %1.==. goto err
for %%A in (*.*) do set TEST=%%A
if %1==%TEST% echo %1 ist die letzte Datei in diesem
  Verzeichnis
if not %1==%TEST% echo %1 ist nicht die letzte Datei in
  diesem Verzeichnis
goto off
:err
echo Erster Parameter fehlte!
:off
```

Wer sich diese Schreibweise angewöhnen möchte, ich bleibe aus publizistischen Gründen bei der Klarschrift, muß natürlich darauf achten, das er die obigen Variablen nicht für andere Zwecke definiert. Ebenso müssen evtl. die bisherigen Batchdateien entsprechend geändert werden.

Auch wenn sich nur wenige die Arbeit machen sollten, werde ich in Zukunft diese Variablen vermeiden, damit es keine Konflikte gibt!

Übrigens: Die Definition der Variablen, mit Befehlen, sollte in der „AUTOEXEC.BAT" erfolgen, damit man keinen Schiffbruch erleidet. Zu diesem Zwecke spendiere ich Euch die „BEFEHLE.BAT", die Ihr z.B. mit -TYPE \BATCH\BEFEHLE.BAT>> C:\AUTOEXEC.BAT" an die „AUTOEXEC.BAT" hängen könnt:

```
@echo off
set bn=BREAK ON
set bf=BREAK OFF
set cd=CHKDSK
set c=COPY
set w=DATE
set d=)DO
set e=ECHO
set fa=FOR %%A IN (
set f=FORMAT
set g=GOTO
set ir=IF ERRORLEVEL
set ie=IF EXIST
set ine=IF NOT EXIST
set inp=IF NOT %
set ip=IF %
set l=LABEL
```

```
set a=PATH
set p=PAUSE
set r=PROMPT
set s=SET
set h=SHIFT
set z=TIME
set t=TYPE
echo Alle internen Befehle in Variablen echo gepackt. Info
   durch SET Befehl!
```

Printer-Adapter & CE-126P

So heißt zumindest die Druckereinheit, die ich vor kurzem bei Becker & Partner erwarb. Es handelt sich dabei um einen SHARP Thermodrucker, der mittels Interface und Software dem PoFo zugänglich gemacht wird. Dieser wird direkt am Erweiterungsbus des PoFo angeschlossen und durch die CONFIG.SYS aktiviert. Scheinbar gab es aber auch einmal eine Version, die über das interne Memorycardlaufwerk angeschlossen und bedient wurde. Zumindest wird diese Version im mitgelieferten Faltblatt erwähnt. So ist es auch nicht verwunderlich, das die dortige Installationsanweisung nicht bei meinem Gerät funktionierte. Anstelle von DEVICE=A:PFP.SYS mußte ich die Zeile DEVICE=B:PFP.SYS an meine CONFIG.SYS hängen, damit der Drucker ansprechbar war.

Weil der Drucker keine Umlaute beherrscht, sollten an deren Stelle die normalen Buchstaben erscheinen. Erste Testausdrucke brachten aber nicht a für ä usw., sondern unbekannte Grafikzeichen. Mit der Zeile DEVICE=B:PFP/2 in der CONFIG.SYS, sollten die Umlaute durch die normalen Buchstaben plus angehängtem e ersetzt werden. Flugs änderte ich also die CONFIG.SYS entsprechend und führte einen Reset aus. Aber leider brachten die nächsten Tests wieder nur diese unbekannten Grafikzeichen. Daher blieb mir nur ANSI.SYS, mit dem ich die Umlauttasten neu belegte: Aus ä wurde ae, aus Ä wurde Ae, usw. Weil ich diese

Befehlsfolgen nicht immer wieder eintippen will, wenn ich den Printer anschließe, habe ich daraus die Batchdatei PRINTER.BAT generiert:

```
@set op=%prompt%
@prompt $e[„ü“;“ue“p

@prompt $e[„Ü“;“Ue“p

@prompt $e[„ö“;“oe“p

@prompt $e[„Ö“;“Oe“p

@prompt $e[„ä“;“ae“p

@prompt $e[„Ä“;“Ae“p

@prompt=%op%
@cls
```

Sofern ANSI.SYS in der CONFIG.SYS angemeldet wurde, werden die Umlauttasten auch tatsächlich umbelegt, wenn man PRINTER.BAT aufruft. In allen zukünftigen Texten sind dann auch keine Umlaute mehr zu finden.

Was ist aber mit den alten Texten, die ja noch Umlaute enthalten? Diese umzuschreiben ist ein langwieriger Aufwand, der sich auch nicht lohnt. Trotzdem wollte ich nicht mit den Grafikzeichen leben und begann die Treiberdatei PFP.SYS zu untersuchen. Und in der Tat gelang es mir nach einiger Zeit die gesuchten Speicherstellen aufzuspüren und zu ändern. Da ich die PFP.SYS nicht wieder auf B: speichern konnte, B: ist das EPROM im Interface zum Drucker, legte ich die neue Version auf C: ab. Deswegen änderte ich auch die CONFIG.SYS erneut ab und ersetzte den DEVICE=B:PFP.SYS Befehl durch DEVICE=C:\PFP.SYS. Nachdem nun notwendig gewordenem Reset, schließlich mußte der geänderte Treiber aktiviert werden, konnte ich auch die alten Texte wieder ausdrucken. Klar das Worte wie: durfen, loschen usw., beim Leser ein Schmunzeln verursachen, aber sie sind immerhin leichter verständlich, als dieselben Worte, die durch Grafikzeichen verunstaltet werden.

Andererseits sind mir Behelfslösungen auch ein Dorn im Auge, weshalb ich gerade versuche die Firma Becker & Partner zu einem Update zu bewegen.

Und hier noch ein paar technische Details zu diesem transportablen Drucker:

Es handelt sich dabei um einen Thermodrucker, der etwa fünf Zentimeter breites Thermopapier, von der Rolle, verwendet. Auf dieser Breite bringt er immerhin noch 24 Zeichen pro Zeile unter. Seinen Energiebedarf zieht er unterwegs aus vier Mignonzellen und Zuhause per Netzteil aus der Steckdose. Inkl. Batterien und dem

etwa 8,- DM teurem Thermopapier (=5 Rollen der Sorte SHARP EA-1250P) wiegt er ca. 800g. Über die Grafikfähigkeit kann ich noch nichts berichten, da in dem Kaufpreis von 45,- DM auch kein Handbuch inbegriffen war.

Alles in allem bin ich trotz der Probleme und des nicht gerade umwerfenden Schriftbildes, sehr zufrieden mit dem Drucker. Kurznotizen, Rechnungen, Einkaufszettel u.v.m. sind bei entsprechenden Vorarbeiten schnell gemacht.

Umlaute an- oder abschalten!

Mit dem Erwerb des Thermodruckers SHARP CE 126P, der in einer speziellen Version für den PoFo angeboten wurde, kamen Probleme mit den Umlauten „ÄÖÜäöü" auf mich zu. Dieser kleine Drucker beherrscht sie ja leider nicht und gibt statt dessen japanische Hieroglyphen aus.

Weil ich mir dieses Problem auch mit anderen Druckern vorstellen kann, habe ich mir zwei Batchdateien kreiert, mit denen ich die Umlaute entsprechend durch „AeOeUeaeoeue" ersetze oder wieder auf „ÄÖÜäöü" zurückstelle.

Will ich also einen Text mit o.g. Drucker ausdrucken, so rufe ich UMLT-N.BAT auf, bevor ich mit dem schreiben des zu druckenden Textes beginne. Habe ich den Text dann fertig geschrieben und ausgedruckt, so stelle ich die Umlaute wieder mit UMLT-J.BAT her.

Allerdings funktionieren diese beiden Batchdateien nur dann, wenn man den Bildschirmtreiber ANSI.SYS in der CONFIG.SYS installiert hat. Besorgt euch dafür die Datei ANSI.SYS und kopiert sie in das Verzeichnis C:\SYSTEM. Anschließend müßt Ihr noch folgende Zeile an eure CONFIG.SYS hängen:

```
DEVICE=C:\SYSTEM\ANSI.SYS
```

Zu guter letzt muß noch ein Warmstart(Reset) mit den Tasten STRG+ALT+EINF/ENTF erzeugt werden, damit die o.g. Batchdateien auch wirklich tadellos funktionieren.

Hier die Batchdatei UMLT-N.BAT, die die Umlaute auf Ae etc. umstellt:

```
@echo off
cls
echo  [„ü“;“ue“p
cls
echo  [„Ü“;“Ue“p
cls
echo  [„ö“;“oe“p
cls
echo  [„Ö“;“Oe“p
cls
echo  [„ä“;“ae“p
cls
echo  [„Ä“;“Ae“p
cls
echo Umlaute abgeschaltet!
```

Und hier noch UMLT-J.BAT, die die Umlaute Ä etc. wieder herstellt:

```
@echo off
cls
echo  [„ü“;“ü“p
cls
echo  [„Ü“;“Ü“p
cls
echo  [„ö“;“ö“p
cls
echo  [„Ö“;“Ö“p
cls
echo  [„ä“;“ä“p
cls
echo  [„Ä“;“Ä“p
cls
echo Umlaute wieder hergestellt!
```

Wenn ihr diese Batchdateien abtippen wollt, so müßt ihr die Zeichenfolge ^[durch folgende Tastendrücke erzeugen: „ALT+Ä" und danach „ALT+Ü". Erst nachdem ihr dann die ALT-Taste loslaßt, erscheint die Zeichenfolge ^[auf dem Bildschirm, vorausgesetzt, Ihr gebt diese Zeilen in der Textverarbeitung des PoFo ein!

Bereits bestehende Texte, die die Umlaute Ä etc. enthalten, werden allerdings nicht durch die o.g. Batchdateien verändert. Aber unter der Zuhilfenahme von FAKE.COM könnte man auch eine nachträgliche Umstellung erreichen. Momentan sehe ich dafür noch keine Veranlassung, weshalb ich diese Aufgabe gerne zur Disposition stelle.

Resümee: Wer einen Drucker benutzt, der keine Umlaute darstellen kann, kann mit diesen Batchdateien besser lesbare Texte ausdrucken!

Variablen einchecken

Bevor man einfach eine Variable mittels -SET- Befehl anlegt, sollte man prüfen, ob diese Variable nicht bereits existiert. Im Direktmodus, d.h. wenn man das Prompt vor Augen hat, genügt die Eingabe -SET- zur Prüfung. Doch wie will der automatisch ablaufende Batchclan damit fertig werden?

Mit „CHKVAR.BAT" habe ich darum einen ausbaufähigen Ansatz für diese Problematik geschaffen:

```
@echo off
rem CHKVAR - existiert die Variable ?
if %1.==. goto err
if %1==off goto %1
set vt1=%1
copy c:\batch\chva2.dat c:\batch\*.b*>nul
echo %1%>>c:\batch\chva2.bat
echo chkvar off>>c:\batch\chva2.bat
chva2.bat
:off
if not %vt%.==. echo Die Variable „%vt1%“ existiert und
   ist
if not %vt%.==. echo mit „%vt%“ definiert!
if %vt%.==. echo Die Variable „%vt1%“ existiert noch
   nicht!
set vt=
set vt1=
del c:\batch\chva2.bat>nul
goto exit
:err
echo Aufruf mit CHKVAR Variablenname!
:exit
```

Auch CHKVAR benötigt etwas Hilfe und bekommt dazu die „CHVA2.DAT" zur Seite gestellt:

```
@echo off
set vt=%
```

Bei der Kreation von „CHVA2.DAT" wurde nach dem Prozentzeichen nicht die Return-Taste gedrückt!

Ausgetestet werden soll nun die Variable BIOS, die Ihr bitte wie folgt mit -SET BIOS=0815- im Direktmodus erzeugt. Sobald Ihr das erledigt habt, rufen wir „CHKVAR.BAT" mit -CHKVAR BIOS- auf. Da wir einen Parameter übergeben haben und dieser nicht off heißt, werden weder die Sprungmarke (:off) noch (:err) ausgeführt. Dagegen kommt die Zeile -SET VT1=%1- voll zur Geltung und sichert die zu testende Variable in VT1. Jetzt wird „CHVA2.DAT" ausführbar gemacht, indem es mittels -COPY- Befehl zu „CHVA2.BAT" kopiert wird. Es folgt die Ergänzung um unseren Parameter BIOS plus nachgestelltem Prozentzeichen. Die Passage -ECHO %1%- ist dafür zuständig. Mit der nächsten Zeile wird der Rücksprungbefehl -CHKVAR OFF- dem Hilfsbatch „CHVA2.BAT" mitgeteilt.

Nach diesen Zeilen sollte „CHVA2.BAT" also so aussehen:

```
@echo off
set vt=%BIOS%
chkvar off
```

So komplettiert wird sie nun von „CHKVAR.BAT" gestartet. Folglich wird die Variable VT mit dem Inhalt von BIOS definiert. Weil wir uns auf die Definition 0815 für BIOS einigten, enthält VT nun also 0815. „CHVA2.BAT" hat seine Schuldigkeit getan und übergibt das Kommando wieder an „CHKVAR.BAT" zurück, indem es eben genannte Batchdatei mit dem Parameter off startet.

Weiter geht es also bei der Sprungmarke (:OFF), die nun zu prüfen hat, ob VT definiert wurde oder nicht. Ist VT nicht leer, so wird vermeldet, das die gesuchte Variable existiert und womit sie definiert wurde. Aus Gründen der Kompatibilität habe ich diese Meldung auf zwei Zeilen verteilt, damit sie auch auf dem 40-Zeichen Schirm des PoFo lesbar ist.

Beim gegenteiligen Falle, erhaltet Ihr die Rückmeldung, das die Variable nicht existiert. Ordentlich, wie alle meine Routinen sein sollen, werden dann VT, VT1 und „CHVA2.BAT" gelöscht, bevor wir wieder in die Tasten greifen dürfen.

KnowHow: Batching

Tastatureingaben in Batchdateien?

Und es geht doch ohne Zusatzprogramme! Aber man muß erst einmal auf die scheinbar blöde Idee kommen, die Befehle -TYPE- und -CON- zu kombinieren. Gibt man also im Direktmodus diese Kombination -TYPE CON- ein, so kann man solange etwas eingeben, bis man die Return-Taste drückt. Das DOS wartet sogar bis zum Nimmerleinstag, wenn man die Return-Taste nicht drückt. Sobald sie aber gedrückt wurde, wird das gerade eingegebene bestätigend auf dem Bildschirm angezeigt.

Will man aber die Eingabe in eine Datei umleiten, der Befehl dazu könnte -TYPE CON>TEST.TXT- lauten, so werden die gedrückten Tasten nur in "TEST.TXT" umgeleitet und erscheinen daher nicht auf dem Bildschirm.

Aber mit einem kleinen Trick, bei dem der Bediener mitwirken muß, kann man auch die Eingabe vorher sichtbar gestalten.

Folgende Batchdateien sind dafür notwendig:

ANTW.DAT - absichtlich DAT:

```
@echo off
set antw=
```

Hinter -SET ANTW=- darf nicht die Returntaste gedrückt worden sein!!! Die zweite Batchdatei im Bunde ist "INPUT.BAT":

```
01: @echo off
02: copy antw.dat *.b*>nul
03: if %1.==. goto err
04: if %2==w goto watch
05: echo Bitte die Eingabe blind vornehmen
06: echo und mit der Return-Taste beenden.
07: echo _
08: echo _%1
09: type con>>antw.bat
10: echo cls>>antw.bat
11: echo echo Deine Eingabe lautete:
    %%antw%%>>antw.bat
12: antw
13: :watch
14: echo Bitte die Eingabe mit der Return-
15: echo Taste beenden.
16: echo _
17: echo _%1
18: type con
19: echo Drücke nun bitte die Taste F3 und
20: echo abschließend die Return-Taste!
21: type con>>antw.bat
22: echo cls>>antw.bat
23: echo echo Deine Eingabe lautete:
    %%antw%%>>antw.bat
24: antw
25: :err
26: echo Die Eingabeaufforderung fehlte!
```

Zeilennummern nebst Doppelpunkt sind nicht abzutippen!

Nun also die üblichen Statements zum Ablauf:
Da ich diese Eingaberoutine für alle Arten der Fragen und Eingabeaufforderungen flexibel halten wollte, ist der entsprechende Aufforderungstext als Parameter %1 zu übergeben. Weil Leerzeichen vom DOS als Trennung zwischen Parametern gewertet werden, müssen statt Leerzeichen die Unterstriche oder das ASCII-Zeichen mit der Nummer 255, eingesetzt werden.

Sollte beim Aufruf von "INPUT.BAT" kein Parameter übergeben worden sein, so wird die Meldung "Eingabeaufforderung fehlte!" ausgegeben und die Routine abgebrochen.

Übergibt man dagegen noch das kleine "w" als zweiten Parameter, so sieht man auch gleich, was man da überhaupt eintippt. Erfolgt keine Übergabe eines zweiten Parameters, so ist man gezwungen, die Eingabe blind zu erledigen. Gehen wir der ganzen Batchdatei mal auf den Grund:

Nach dem abschalten der Zwischenkommentare des DOS, mittels -@ECHO OFF-, wird "ANTW.DAT" zu "ANTW.BAT" kopiert. Als nächstes wird geprüft, ob überhaupt ein Parameter beim Aufruf mitangegeben wurde. Trifft dieses zu, so wird auch der mögliche Zweitparameter mit dem Zeichen "w" abgeglichen. Auf Grund dieses zweiten Testes wird dann gegebenenfalls zur Sprungmarke (:WATCH) verzweigt.

Zum besseren Verständnis gehen wir aber nun davon aus, das kein "w" als zweiter Parameter angegeben wurde:

Es folgen also die Zeilen, direkt nach der Parameterprüfung und geben einige Hinweise zur Eingabe. Man soll also die Eingabe blind vornehmen und anschließend die Return-Taste drücken. Nach diesem Hinweis wird erst die eigentliche Frage/Eingabeaufforderung angegeben und

zwar durch -ECHO %1-. Die Befehlszeile -TYPE CON>>ANTW.BAT- wartet dann auf die Eingabe und hängt diese, nach dem einmaligen drücken der Return-Taste, an "ANTW.BAT" an. Leider wird aber die Eingabe zweimal hintereinander angehängt, weshalb ich noch mit -ECHO CLS>>ANTW.BAT-, den Befehl zum Bildschirmlöschen anhänge. Dies hat folgenden Sinn: Meistens wird die Eingabe keinem DOS-Befehl oder einem Programmnamen entsprechen, weshalb eine Fehlermeldung auftritt, sobald die Zeile von "ANTW.BAT" abgearbeitet wird, die das Duplikat der Eingabe enthält. Durch das angehängte -CLS- wird die Fehlermeldung zwar nicht unterdrückt, aber sehr schnell weggeblendet, um keine Verwirrung zu stiften.

Nachdem das -CLS- also in "ANTW.BAT" gelandet ist, wird noch die folgende Zeile an "ANTW.BAT" angehängt: -ECHO Deine Eingabe lautete: %antw%-. Ihr merkt schon, heute werden mal wieder sehr viele gehängt! Diese Zeile muß zwar nicht sein, aber ich möchte gerne eine Bestätigung meiner Eingabe bekommen. Wollen wir nicht alle bestätigt werden? Genug des Tumors, über den man trotzdem lacht, jetzt folgt der Aufruf von "ANTW.BAT", durch die Zeile -antw-.

Konstruieren wir also nun eine beispielhafte "ANTW.BAT", in der die Frage nach dem Befinden mit "gut" beantwortet wurde:

```
@echo off
set antw=gut
gut
cls
echo Deine Eingabe lautete: %antw%
```

Nach ihrem Start definiert sie also den Inhalt der Variablen mit dem Wort "gut". Das Duplikat der Eingabe wird daraufhin als Befehl/Programm auszuführen versucht, was zu einer Fehlermeldung führt. Diese wird aber schnell wieder per -CLS-Befehl gelöscht. Zu guter letzt bekommen wir dann unsere Eingabe bestätigt.

Um die Eingabe sichtbar zu machen, ich setze jetzt bei der Sprungmarke (:WATCH) ein, leiten wir diese vorerst nicht in eine Datei um. Anstelle -TYPE CON>>ANTW.BAT- verwenden wir daher den Befehl -TYPE CON-, der die Eingabe sofort sichtbar darstellt. Weil die Funktionstaste 3, kurz: <F3>, immer die letzte Eingabe zugewiesen bekommt, enthält sie auch unsere Eingabe, wenn wir die Return-Taste gedrückt haben. Um unsere Eingabe also in "ANTW.BAT" zu kommen, muß nun doch auf den Befehl -TYPE CON>>ANTW.BAT- zurückgegriffen werden. Weil aber kein Bediener die große Lust verspüren dürfte, seine Eingabe noch einmal blind zu wiederholen und weil das Eingegebene bereits der Taste <F3> zugewiesen wurde, fordern wir also nun den Bediener auf, genau diese Taste <F3> zu drücken. Dadurch wird die letzte Eingabe wiederholt und man muß abschließend die Return-Taste drücken, damit das Getippte wieder in "ANTW.BAT" landet.

Man muß also diese Unbequemlichkeit mit <F3> & <Return> hinnehmen, wenn man seine Eingabe beim tippen beobachten will. Aber ich denke, daß man mit diesem kleinen Haken leben kann. Ansonsten gibt es zum diesem Thema ja auch genügend Zusatzprogramme.

Eine Bemerkung zum Schluß:
Ich habe zwar auch versucht die Eingabe sichtbar zu halten und auf die Tastendrücke <F3> und <Return> zu verzichten, konnte aber das Eingegebene nicht in einer Variablen einfangen, oder als Parameter definieren. Es ist zwar schade, aber nicht machbar, oder???

Unbekannter Befehl in der CONFIG.SYS

Nur selten schreibe ich eine neue CONFIG.SYS auf meinen PoFos, oder ändere die bestehende ab. Doch neulich beim Experimentieren, kam ich wieder in die Verlegenheit eine neue CONFIG.SYS zu schreiben. Also schaute ich wieder einmal ins Handbuch meines PoFo und schrieb die dort abgedruckten Befehle zur CONFIG.SYS ab. Weil mir die Werte für die Befehle FILES und BUFFERS viel zu hoch erschienen, gab ich folglich kleinere Werte an und speicherte die neue CONFIG.SYS auf der Ramdisk C: ab.

Nachdem ich auch alle anderen Vorbereitungen abgeschlossen hatte, startete ich also den PoFo mit dem „Affengriff" (STRG+ALT+EINF/ENTF) neu durch.

Kurz darauf erhielt ich die Fehlermeldung „Unrecognized command in CONFIG.SYS", die ich bis dato noch nie gesehen hatte. Die deutsche Übersetzung dieser Fehlermeldung bedeutet „Unbekannter Befehl in CONFIG.SYS", sofern mich mein Schulenglisch nicht im Stich läßt.

Zwar ergab sich dann später noch ein nicht nennenswerter Fehler, aber weil der auf meiner Vergeßlichkeit beruhte, konnte ich Schussel den leicht beheben. Deshalb lud ich die CONFIG.SYS zunächst in die interne Textverarbeitung und kontrollierte die folgenden Zeilen, die ich ja vorher selbst geschrieben hatte:

```
FILES=6
BUFFERS=10
DEVICE=C:\ANSI.SYS
```

Obwohl der DEVICE-Befehl nicht im PoFo-Handbuch erwähnt ist, so ist er doch ein erlaubter Befehl und war nicht die Ursache für die Fehlermeldung. An den Befehlen FILES und BUFFERS kann es auch nicht gelegen haben und andere, unbekannte, Befehle enthält diese CONFIG.SYS ja nicht. Weil ich mir den Fehler partout nicht erklären konnte, setzte ich nun die Werte für FILES und BUFFERS auf die vom PoFo-Handbuch empfohlene Werte, bevor ich die geänderte CONFIG.SYS abspeicherte und einen erneuten Reset ausführte.

Diesmal blieb ich von der Fehlermeldung verschont, was mich natürlich stutzig machte. Schließlich hatte ich ja nur die Werte von FILES und BUFFERS geändert, anstatt einen Befehl zu löschen. Ich startete die Textverarbeitung erneut und änderte nun den Wert von FILES wieder auf sechs, speicherte erneut ab und startete zum dritten mal den PoFo. Jetzt bekam ich wieder die oben genannte Fehlermeldung zu Gesicht und war noch mehr verwundert als vorher. Sie, die Fehlermeldung, mußte folglich durch den Wert von FILES entstanden sein, obwohl sie eigentlich eine andere Fehlerquelle angibt. Schließlich ist ein falscher Wert nicht mit einem unbekannten Befehl einfach gleichzusetzen!

Aber tatsächlich, nachdem ich den Wert von Files von mal zu mal um eins erhöhte, bekam ich diese Fehlermeldung nicht mehr, als der Wert von FILES acht, oder mehr betrug!

Sollte diese Fehlermeldung einmal bei Euch auftauchen, so habt Ihr hoffentlich nun gelernt, das der Wert von FILES nicht unter acht sein darf, damit Ihr Euch nicht bei der Fehlersuche im Kreis verlauft!!!

Kontrolliert also bei dieser Fehlermeldung die Zeile mit FILES und setzt den Wert mindestens auf acht, bevor Ihr nach anderen Fehlerquellen sucht. Wie gesagt, die Fehlermeldung ist irritierend, da sie nicht den wirklichen Fehler beschreibt! Es kann natürlich auch einmal vorkommen, das Ihr ein unerlaubtes Kommando in die CONFIG.SYS schreibt, oder ein erlaubtes falsch eintippt, aber das passiert in der Regel viel seltener!

Die versteckte Datei des PoFo

So manchmal, wenn ich eine Memorycard (MMC) oder die Ramdisk mit dem Befehl CHKDSK prüfe, wird auch von einer versteckten Datei berichtet, die aber ohne Inhalt ist. Weder mit dem PoFo, noch mit dem PC, konnte ich bislang diese ominöse Datei sichtbar machen oder deren Namen erfahren.

Als ich neulich in der Buchhandlung mehrere neue DOS-Bücher nach Anregungen für Batchdateien durchblätterte, stieß ich auf die Herkunft dieser versteckten Datei:

„Wird beim formatieren oder nachträglich mit dem Befehl LABEL ein Datenträger auf einen Namen getauft, so wird dieser als eine Art Datei auf dem Datenträger abgelegt.

Diese Datei ist allerdings ohne Inhalt und kann nicht sichtbar gemacht werden."

Flugs zückte ich meinen PoFo und probierte dieses gleich aus, indem ich mit der bloßen Eingabe von LABEL und zweimaligem drücken der Return-Taste, gefolgt vom bestätigen per „J", die Bezeichnung meiner Ramdisk löschte. Und siehe da, beim nächsten Report von CHKDSK war keine versteckte Datei gemeldet worden! Kaum hatte ich die Bezeichnung mit LABEL RAMDISK wiederhergestellt, tauchte die versteckte Datei wieder im Statusbericht von CHKDSK auf.

Dieses Spielchen bleibt übrigens nicht nur auf dem PoFo beschränkt, sondern ist auch unter vielen DOS-Versionen wiederholbar!

Verzeichnis kopieren

Auch das kopieren eines Verzeichnisses wird nicht vom DOS vereinfacht, sondern verlangt viel Handarbeit. Ich habe mir das wieder mit einer Batchdatei erleichtert, die ich Euch hier vorstellen will.

Mit COPYDIR.BAT kann ich eine Kopie eines Verzeichnisses erstellen, die auf Wunsch in einem anderen Laufwerk oder einem Verzeichnis, angefertigt wird.

Will ich z.B. das Verzeichnis C:\SYSTEM auf das Laufwerk A: kopieren, so lautet die Syntax „COPYDIR \SYSTEM A:". Soll das Verzeichnis C:\SYSTEM dagegen nur unter dem Namen C:\DOS dupliziert werden, so gebe ich „COPYDIR \SYSTEM \DOS" ein. Voraussetzung ist in beiden Fällen, das man tatsächlich gerade auf dem Laufwerk C: arbeitet, dieses also als aktiv angemeldet ist. Notfalls muß man durch die Eingabe von „C:" das Laufwerk aktivieren.

Hier die besagte Batchdatei COPYDIR.BAT:

```
@echo off
if %2==a: goto lw
if %2==b: goto lw
if %2==c: goto lw
if %2==A: goto lw
if %2==B: goto lw
if %2==C: goto lw
if %2.==. goto err
goto vz
:lw
md %2%1 > nul
copy %1\*.* %2%1 > nul
goto off
:vz
md %2 > nul
copy %1\*.* %2 > nul
goto off
:err
echo Ziellaufwerk oder -verzeichnis
echo fehlt!
:off
```

Und so funktioniert das ganze:

Weil der zweite Parameter entweder ein Laufwerk oder ein Verzeichnis sein kann, muß zunächst durch die IF-Zeilen geprüft werden, ob ein Laufwerk als Ziel angegeben wurde. Ist dies zutreffend, so wird die Routine LW ausgeführt. Anderenfalls handelt es sich um ein Verzeichnis, so daß die Routine VZ ausgeführt wird.

Da beide fast nach demselben Prinzip arbeiten, werde ich hier die Routine LW erläutern:

Auf dem Laufwerk, welches als Parameter 2 angegeben wurde, wird durch „MD %2%1 > NUL" ein gleichnamiges Verzeichnis erstellt, sofern es noch nicht existiert. Mit „COPY %1*.* %2%1 > NUL" werden dann alle Dateien aus dem Quellverzeichnis ins gleichnamige Verzeichnis des gewünschten Laufwerkes kopiert.

Bei der Routine VZ, und dies ist der eigentliche Unterschied, wird der Parameter 2 als Verzeichnis definiert und somit alle Dateien des Quellverzeichnisses ins neu geschaffene Verzeichnis kopiert.

Sollte das zu kopierende Verzeichnis noch weitere Verzeichnisse enthalten, so bleiben diese unberücksichtigt. Aber durch entsprechende Eingaben und mehrfach Aufrufe, sind auch diese kopierbar.

Nun mach aber mal einen Punkt!

Ein Fehler im DOS des PoFo erlaubt die Konstruktion einer Batchdatei namens„ . .BAT". Dazu erstellt man die Batchdatei „PUNKT.BAT" und benennt sie mit dem Befehl -REN PUNKT.BAT .BAT- in „. .BAT" um.

Diese Batchdatei ist insofern interessant, weil sie durch Dutzende von einzelnen ASCII-Zeichen aufgerufen werden kann:

. , ; + [ ] „ / =, durch die ASCII-Zeichen 0-2, 4-7, 10-12, 14-18, 20-26, 28-31 und alle ab 127 aufwärts

In der Regel gibt man diese einzelnen Zeichen nicht als Aufruf ein, da meist keine Programme solchen Namens existieren. Aber wenn man diese Batchdatei mit entsprech-

endem Inhalt füllt, kann sie z.B. Fehlermeldungen abfangen, ohne das ein Automatikprozess unterbrochen wird:

```
@echo off
if exist %1 %1>nul
hauptprg
```

Existiert das als Parameter %1 übergebene Programm nicht, oder ist nicht auffindbar, so wird das beispielhafte Hauptprogramm „HAUPTPRG.BAT" gestartet. Ansonsten wird das gewünschte Programm gestartet. Ein anderes Einsatzgebiet ist das Ausspionieren der Unterverzeichnisse innerhalb von Unterverzeichnissen. Dazu schreibt man das entsprechende Directory in eine Batchdatei, die anschließend gestartet wird. Wenn die Zeile „. <DIR>" zur Ausführung kommt, ruft sie damit die Batchdatei „. .BAT" auf, die dann die restlichen übergebenen Parameter, ebenfalls Directoryeinträge, auswerten könnte. Zwar ist damit noch keine echte Nachbildung des XCOPY-Befehls, als Batchdatei erreicht, aber wieder ein Stück näher gerückt!

Countdown - der Computer lernt zählen

Zugegeben, ganz ohne Unterstützung ist diese Eigenschaft nicht zu erreichen, aber auch wir Menschen bedienen uns der Mentoren. So habe ich eine Batchdatei kreiert, mit deren Hilfe der Computer die Zahlen von 40 bis 0 herunterzählen kann.

Was zunächst als bloße Spielerei aussieht, läßt im nachhinein phantastische Möglichkeiten bei der Batchprogrammierung zu. Unter der Zuhilfenahme von COUNTDWN.BAT kann man nun andere Batchprogramme so oft ausführen lassen, wie es COUNTDWN.BAT eben zuläßt.

Aber auch zur Zeitverzögerung ist diese Datei gut zu gebrauchen. Während des optisch ansprechenden Countdowns, der bei jeder Zahl zwischen 10 und 1 beginnen kann, bleibt dem Betrachter entsprechend Zeit, um seine Entscheidung zu fällen.

Weil COUNTDWN.BAT recht umfangreich ist, wollen wir uns hier nur einmal die wichtigsten Partien zu Gemüte führen:

```
@echo off
rem Zahl definieren
set oz=%z%
set z=
if %1.==. set z=%oz%
if %z%.==. set z=%1
if %2.==. goto %z%
if not %2.==. goto %z%
rem Zahl verringern
:40
set z=39
goto 0
:39
set z=38
goto 0

.....

:11
set z=10
if %2.==. goto 0
cls
copy 10.txt con > nul
:10
set z=9
if %2.==. goto 0
cls
copy 9.txt con > nul
:9
set z=8
if %2.==. goto 0
cls
copy 8.txt con > nul
:8
set z=7
if %2.==. goto 0
cls
copy 7.txt con > nul
:7
set z=6
if %2.==. goto 0
cls
copy 6.txt con > nul
:6
set z=5
if %2.==. goto 0
cls
copy 5.txt con > nul
:5
set z=4
if %2.==. goto 0
cls
copy 4.txt con > nul
:4
set z=3
if %2.==. goto 0
cls
copy 3.txt con > nul
:3
set z=2
if %2.==. goto 0
cls
copy 2.txt con > nul
:2
set z=1
```

```
if %2.==. goto 0
cls
copy 1.txt con > nul
:1
set z=0
if %2.==. goto 0
cls
copy 0.txt con > nul
:0
if %2.==. echo %z%
```

Die Variable z spielt in dieser Routine die tragende Rolle und wird deshalb zu Beginn in der Variable oz gesichert. Sollte COUNTDWN.BAT ohne Parameter aufgerufen worden sein, so geht die Routine davon aus, das der derzeitige Wert von z um eins verringert werden soll und führt dieses auch aus. Übergibt man aber beim Aufruf einen Wert, zwischen 40 und 1, als ersten Parameter, so wird der übergebene Wert um eins verringert und in der Variable z gespeichert.

Den echten Countdown löst man aus, indem man eine Zahl zwischen 11 und 1 als ersten und ein x-beliebiges Zeichen als zweiten Parameter übergibt. Durch die Eingabe von COUNTDWN 8 m wird also ein Countdown gestartet, der bei 7 beginnt und mit 0 endet. Die angezeigten Ziffern, die ich als Grafiken in der Textverarbeitung erzeugt habe, werden dabei mit dem COPY-Befehl auf den Schirm gebracht.

Wie eingangs erwähnt, ist dieser Countdown nur ein nettes Abfallprodukt, da es hauptsächlich darum geht, eigene Batchroutinen so oft zu wiederholen, wie man es sich gewünscht hat. Natürlich ist dazu ein gewisses Maß an Programmierfertigkeit vonnöten, aber wer diese Routine einmal begriffen hat, sollte damit keine Schwierigkeiten haben.

Viele Dateien - ein Datum

Auch das Datum in den Directoryeinträgen kann ein Kriterium zur Unterscheidung sein. Deshalb möchte man desöfteren das Dateidatum auf einen gewissen Tag festlegen. Weil bisher bekannte Lösungen nicht auf dem PoFo funktionierten, habe ich eine Lösung mit „DDA.BAT" herbeigeführt:

```
@echo off
rem Datei-Datum aktualisieren
if %3.==. goto err
date %2
md c:\dda>nul
for %%a in (%1) do type %%a>c:\dda\%%a
for %%a in (%1) do del %%a
for %%a in (c:\dda\*.*) do copy c:\dda\%%a>nul
for %%a in (c:\dda\*.*) do del c:\dda\%%a
rd c:\dda
date %3
echo Das Datei-Datum wurde aktualisiert!
goto off
:err
echo Aufruf mit:
echo DDA Datei.Ext gew.Datum akt.Datum
:off
```

Nachdem das auserwählte Datum eingestellt wurde, wird die Datei, oder die Dateigruppe, per -TYPE- in das temporäre Verzeichnis DDA kopiert. Weil beim kopieren, mittels -COPY-, das Dateidatum beibehalten wird, weiche ich auf den Befehl -TYPE- aus, der sich am aktuellen Datum orientiert. Nun wird die ursprüngliche Datei, oder Dateigruppe, gelöscht. Als nächstes wird die aktualisierte Version zu ihrem Stammplatz transferiert und das temporäre Verzeichnis DDA wieder entfernt. Kurz vor Schluß wird dann das aktuelle Datum eingestellt und der Erfolg vermeldet.

Dateinamen in Variablen

Die unten stehende Batchdatei legt alle Dateinamen des Hauptverzeichnisses in Variablen ab. Leider muß ich Eure Freude empfindlich dämpfen, da die Dateinamen Variable und Variableninhalt zugleich sind. Alle Versuche, diese in ausgesuchte Variablen zu packen, schlugen bisher fehl.

Eventuell kann man das aber doch noch mit FAKE und ANSI in den Griff bekommen. Weil ich mit beiden noch kaum gearbeitet habe und möglichst alles mit Batchdateien erreichen will, stelle ich diese Routine zur Disposition:

```
@echo off
for %%a in (\*.*) do set set %%a=%%a
```

Entgegen meiner Gewohnheit erkläre ich hier nichts dazu und fordere zum experimentieren auf!!!

Ich wollte ursprünglich damit den Befehl XCOPY nachempfinden, doch bisher blieb ich erfolglos. Wer weiß weiter?

Aufräumarbeiten

Um so öfter man seine Meisterwerke überarbeitet und ergänzt, ich meine die Dateien die dabei herauskommen, desto mehr verteilen sie sich zerstückelt auf den Datenträgern. DOS hat damit zwar keine Probleme, weil es diese Fragmente zusammensucht, braucht aber mehr Zeit dafür.

Die Zeit ist ein wichtiger Faktor, wenn man wie ich mit einem 4,9152 MHz getakteten PoFo arbeitet. Aus diesem Grunde ist „DEFRAG.BAT" entstanden, welches der Zerstückelung Einhalt gebietet:

```
@echo off
rem Verzeichnis defragmentieren
if %1.==. goto err
echo Bitte etwas Geduld ...
md \defrag>nul
copy %1\*.* \defrag>nul
for %%a in (%1\*.*) do del %1\%%a
rd %1>nul
md %1>nul
copy \defrag\*.* %1>nul
for %%a in (\defrag\*.*) do del \defrag\%%a
rd \defrag>nul
echo Defragmentierung abgeschlossen!
goto off
:err
echo Aufruf mit DEFRAG \Verzeichnis!
:off
```

Diese Routine legt das Verzeichnis DEFRAG an und kopiert zunächst alle Dateien des gewünschten Verzeichnisses dort hinein. Dadurch finden die Dateifetzen wieder zueinander und werden alsbald in ihr ursprüngliches Verzeichnis rückverlegt. Vorher wurde natürlich das Verzeichnis komplett geleert, gelöscht und neu kreiert. Sobald das erledigt ist, wird das nun nicht mehr benötigte Verzeichnis DEFRAG getilgt und der gesamte Vorgang für abgeschlossen erklärt.

Verzeichnis löschen

Auch diese Eingaben waren mir zu umständlich, weshalb ich dafür die Batchdatei DELDIR.BAT ersann, die ein Verzeichnis automatisch löscht:

```
@echo off
if %1.==. goto err
for %%a in (%1\*.*) do del %1\%%a
rd %1
goto off
:err
echo Verzeichnis fehlt o. nicht
echo existend!
:off
```

Mal angenommen, das Verzeichnis SYSTEM im Laufwerk C: soll gelöscht werden, dann geben wir dafür folgendes ein:

```
DELDIR C:\SYSTEM
```

Zum Programmablauf:
Ist kein Verzeichnis angegeben, so wird wieder darauf hingewiesen und die Routine beendet. Beim gegenteiligen Fall werden zunächst alle Dateien im gewünschten Verzeichnis gelöscht. Als nächstes wird dann das Verzeichnis selbst gelöscht, sofern es keine Unterverzeichnisse enthält.

Logischerweise sollte man sich nicht gerade in dem zu löschenden Verzeichnis befinden, da es sonst nicht gelöscht wird.

Variablen tilgen

Im Direktmodus gibt man -SET Variablenname= - ein, wenn man die gewünschte Variable löschen möchte. Weil ich recht tippfaul bin und alles automatisieren will, habe ich mir die „DV.BAT" geschrieben, mit der ich nun überflüssige Variablen entferne:

```
@echo off
rem DV - Del Variable
if %1.==. goto err
set ß=%1
set %1=
echo Die Variable %ß% wurde gelöscht!
set ß=
goto off
:err
echo Aufruf mit: DV Variablenname!
:off
```

Wird beim Aufruf von „DV.BAT“ eine Variable als Parameter %1 mitübergeben, so wird diese durch die Zeile -SET %1= - gelöscht. Anschließend erhält man noch eine Bestätigung. Alle anderen Zeilen entsprechen nur einer gewissen Kosmetik und könnten weggelassen werden, sofern man diese Routine nicht auf dem PoFo einsetzt.

Übergeordnetes Verzeichnis löschen

Es soll ja mit dem Befehl -DEL ..- funktionieren, was ich selbst noch nicht ausprobiert habe, aber meiner Erfahrung nach geht es nicht auf dem PoFo!So schuf ich mir die „DVV.BAT“, die dieses Manko beseitigt:

```
@echo off
rem übergeordnetes Verzeichnis löschen
if %1==off goto %1
copy c:\batch\dvv2.dat c:\batch\*.b*>nul
cd>>c:\batch\dvv2.bat
echo dvv off>>c:\batch\dvv2.bat
cd ..
for %%a in (*.*) do del %%a
c:\batch\dvv2
:off
del c:\batch\dvv2.bat
echo Das übergeordnete Vezeichnis wurde
echo komplett gelöscht!
```

Behilflich ist dabei auch „DVV2.DAT“:

```
@echo off
cd
```

Hinter dem Befehl -CD-, Ihr ahnt es vielleicht schon, steht noch ein Leerzeichen ohne krönendes Return.

Stellen wir uns vor, das wir gerade im Verzeichnis \TEXT\PRIVAT\FREUNDE stehen und das Verzeichnis \TEXT\PRIVAT komplett löschen wollen. Also rufen wir einfach „DVV.BAT“ durch die Eingabe DVV auf.

-IF %1==off GOTO %1- muß fehlschlagen, da wir keinen Parameter übergeben haben. Statt dessen wird „DVV2.DAT“ als „DVV2.BAT“ dupliziert und per -CD>>DVV2.BAT- mit dem aktuellen Verzeichnis versehen. Um den Absprung nicht zu verpassen, hängen wir noch die Zeile -DVV off- an.

So versichert wechseln wir ins übergeordnete Verzeichnis \TEXT\PRIVAT\ und löschen dort alle Dateien per -FOR-Schleife. Nachdem der eigentliche Part erledigt ist, rufen wir „DVV2.BAT“ auf, damit wir wieder in unser Verzeichnis \TEXT\PRIVAT\FREUNDE gelangen:

```
@echo off
cd \TEXT\PRIVAT\FREUNDE
dvv off
```

Sobald wir also im ursprünglichen Verzeichnis stehen, sorgt „DVV.BAT“ für die Aufräumarbeiten und entläßt uns dem Prompt.

Vermeidung von endlosen Eingaben

Windows ist nicht nur wegen seiner hübschen, bunten Bildchen beliebt, sondern weil man dort kaum die Tastatur benötigt. Als DOS-Fanatiker und Maushasser braucht man aber auch nicht auf Bequemlichkeit verzichten, sofern man sich Zeit nimmt und gewisse Überlegungen anstellt.

Ich habe mir bereits einige Gedanken dazu gemacht und möchte Euch diese hier einmal präsentieren!

Angeregt wurde ich dazu von Jens Sewitz, der mir bei einem Treffen seine Batchdateien DP.BAT und DW.BAT präsentierte. Jens war es nämlich leid immer DIR /W oder DIR /P zu schreiben, weshalb er diese Befehle durch die o.g. Batchdateien ersetzte, damit er in Zukunft z.B. mit DP das Inhaltsverzeichnis seitenweise angezeigt bekommt. Er, wie ich nun auch, müssen nun nur noch DP eingeben anstelle des längeren Befehls DIR /P. Zum besseren Verständnis hier einmal die Batchdatei DP.BAT:

```
@echo off
dir /p
```

Erstellt man ähnliche Batchdateien für die Befehle COPY, REN, DEL, MD, RD, CD, DATE, TIME, VER usw., so kann man mit der Zeit sehr schreibfaul werden ohne das das DOS nur noch Bahnhof-Bratkartoffeln versteht. Im Endeffekt tippt man letztendlich nur noch unverständliche Kurzbefehle ein, zumindest dem Zuschauer erscheint es so, aber das DOS spielt da problemlos mit.

Oder wie wäre es, wenn man alle Befehle in Batchdateien packt, die die Namen von Verwandten trügen ??? Auch die Eingabe von „Theo liebt Gaby", mit der die Datei „LIEBT." in „GABY." umbenannt wird (REN-Befehl steht in THEO.BAT), würde bei jedem zuschauenden DOS-User Verwirrung stiften!!!

FASTDEL.BAT - löschen ohne Rückfrage

Will man alle Dateien in einem Verzeichnis löschen, mit DEL *.*, so fragt DOS zur Sicherheit noch einmal nach: Sind Sie sicher (J/N)?

Generell ist man sich seiner Sache schon sicher und gibt das J ein. Wozu dann also diese nervige Rückfrage?

Gute Programmierer unterbinden deshalb diese Rückfrage, indem sie das J in die Datei J.DAT schreiben und den Löschbefehl mit folgender Syntax anwenden: DEL *.* < J.DAT > NUL

Damit werden die Abfrage und die entsprechenden DOS-Kommentare komplett unterbunden. Ziel erreicht, so könnte man jetzt meinen, übersieht aber die Hilfsdatei, die ja, je nach Rechner und Datenträger, so zwischen 64 und 2048 Bytes verbraucht, auch wenn sie nur drei Bytes groß ist.

Dieser Verschwendung kann man entgegenwirken, wenn man jede Datei einzelnd löscht: DEL BEISPIEL.PRG. Nebenbei wird auch keine Sicherheitsabfrage stattfinden, weil DOS es nicht für nötig hält. Auf diese Weise eine ganze Reihe von Dateien zu löschen, ist wahrlich sehr umständlich und kann nicht automatisiert werden.

Verwendet man aber eine FOR-Schleife dazu, so kann man alle gewünschten Dateien in einem Verzeichnis löschen, ohne eine Hilfsdatei zu benutzen und die Rückfrage von DOS vermeiden. Ich nenne die dazu notwendige Batchdatei FASTDEL.BAT, die wie folgt geschrieben werden muß:

```
@echo off
for %%a in (*.*) del %%a
```

Die Liste (*.*) definiert %%a bei jedem Durchlauf als eine andere Datei, aus der Menge der vorhandenen Dateien (*.*) und löscht explizit nur die in %%a gespeicherte Datei. Weil eben bei jedem Durchlauf, der automatisch vom DOS so oft wiederholt wird, wie noch Dateien vorhanden sind, nur eine Datei gelöscht wird, entfällt die Rückfrage. Somit ist auch keine Hilfsdatei J.DAT mehr notwendig, was wiederum Speicherplatz spart.

Aufgerufen wird FASTDEL.BAT nur durch die Eingabe von FASTDEL, wenn man sich in dem Verzeichnis befindet, in welchem die Dateien gelöscht werden sollen.

FASTPRNT.BAT - Massendrucksache

Auch das Ausdrucken von Texten ist recht lästig, wenn man mehrere Artikel auf Fehler kontrollieren muß. Ist der Drucker aber mit Endlospapier bestückt, oder verfügt über einen automatischen Einzelblatteinzug, so kann man die Arbeit sehr gut automatisieren und die Wartezeit für andere Aufgaben nutzen.

```
@echo off
if %1.==. goto err
echo  > prn
echo Es folgen die Dateien: > prn
echo  > prn
for %%a in (%1) do echo %%a > prn
echo  > prn
for %%a in (%1) do copy %%a prn >
nul
echo  > prn
goto off
:err
echo Zu druckende Dateigruppe fehlt!
:off
```

Gestartet wird diese Batchdatei mit z.B. FASTPRNT *.TXT. Sollte die Dateigruppe fehlen, sie wird ja wieder als Parameter %1 übergeben, erhält man eine Rückmeldung. Ansonsten wird zunächst mit ECHO > PRN eine Leerzeile auf dem Drucker erzeugt. Hinter dem ECHO stehen aber keine zwei Leerzeichen, sondern ein Leerzeichen und der Charakter 255.

Würde ich letzteren weglassen, den ich mit der ALT-Taste erzeugte, so würde statt einer Leerzeile der Text ECHO ist off gedruckt. Die erste FOR-Schleife druckt dann alle gefundenen Dateinamen aus, bevor die zweite Schleife die eigentlichen Dateien ausdruckt.

Den Dateinamen vor dem Dateiinhalt auszudrucken wäre auch mir lieber gewesen, aber ich habe dies bisher noch nicht weiter versucht. Gewöhnt man es sich an, am Ende eines jeden Textes eine Leerzeile zu erzeugen, so ist auch eine saubere Trennung zwischen den gedruckten Texten zu sehen.

Fiese Verzeichnisse

... bieten zwar auch keinen absoluten Schutz, verwehren aber unerfahrenen Bedienern die Einsicht.

Das Dateieinträge im Directory aus dem, bis zu achtstelligen, Dateinamen und der, bis zur dreistelligen, Extension bestehen, sollte mittlerweile hinlänglich bekannt sein.

Directoryeinträge bestehen zwar meist nur aus dem, bis zu achtstelligen, Directorynamen, können aber ebenso mit einer, bis zur dreistelligen, Extension versehen werden. So wird z.B. mit „MD 12345678.ABC" tatsächlich ein gültiges Verzeichnis erzeugt.

Bis hierhin war ja noch alles human, aber was haltet Ihr von den Verzeichnissen:

```
Datenträger in c ist ramdisk
Inhaltsverzeichnis von c:\

system                                  <DIR>
1-01-80   9:53a
+──────+       <DIR>       1-01-80   9:55a
¦hallo!¦                                <DIR>
1-01-80   9:57a
+──────+       <DIR>       1-01-80   9:58a
ÛÛÛÛÛÛÛ ÛÛÛ <DIR>       1-01-80   9:59a
                                        <DIR>
1-01-80  10:02a
  6 Dateien 44032 Bytes frei
```

Überrascht von diesen einfachen Beispielen? Mit mehr Phantasie kann man glatt Portraits oder kleine Zeichnungen herstellen!

Das ganze Geheimnis beruht auf den Grafikzeichen des Zeichensatzes, die man mit der <Alt>-Taste erzeugt. So ist der Directoryeintrag „Kastenoberteil" durch folgende Tastaturakrobatik, auf dem Portfolio, entstanden:

```
<M>, <D>, <Leer>, <Fn>+<N>,
<Alt>+<K>+<M>+<J>, <Alt>+<K>+<M>+<I> (noch
fünf mal wiederholen), <Alt>+<J>+<8>+<7>,
<Fn>+<N>, <+
```

Zum Verständnis: Jeder Text in spitzen Klammern soll eine Taste sein. Ein plus bedeutet, daß die erste Taste festgehalten werden muß und die anderen bei gehaltener Taste gedrückt werden sollen. Die Kommatas weisen nur darauf hin, das jetzt eine neue Sequenz kommt und <+ steht für die Enter-Taste.

Groß-PC-Besitzer schalten bitte NumLock ein und tippen die Zahlenwerte über denZehnerblock ein:

<M>, <D>, <Leer>,
<Alt>+<2>+<0>+<1>,

<Alt>+<2>+<0>+<5> (noch fünf mal wiederholen), <Alt>+<1>+<8>+<7>, <+

Minimal sind also in diesem Beispiel 36 Tasten zu drücken, bevor man in dieses Verzeichnis gelangt. Weil das selbst für DOS-Puristen abschreckend wirkt, sind solche kunstvollen Verzeichnisse ein guter Platz, für möglichst wichtige Dateien. Mit dem ATTRIB-Befehl kann man dann die Dateien ja auch noch unsichtbar machen. Damit aber für uns die Verzeichniswechsel nicht zur Tortur werden, sollten solche Verzeichnisnamen in Variablen eingespeichert werden: „SET VZ1=+———+".

Per „CD %VZ1%" kann man nun problemlos in solche Verzeichnisse gelangen.

Allerdings funktioniert der eben genannte Befehl nicht im Direktmodus, sondern muß per Batchdatei ausgeführt werden!

CTRL.BAT - Steuerzeichen in Variablen

Als Steuerzeichen werden die Charakter von Null bis Einunddreißig bezeichnet, die man im Zeichensatz wiederfindet. Mit ihnen wird z.B. der Bildschirm gelöscht, ein Signalton ausgegeben, eine Datei geschlossen, Textausgaben kurzfristig angehalten und noch vieles mehr. Auch Ihr habt sicherlich schon mit ihnen Bekanntschaft geschlossen, sei es nur die STRG+C-Funktion, mit der Ihr eine Programmausführung vorzeitig beendet habt.

Ich habe diese Steuerzeichen, soweit als möglich, mit einer Programmiersprache eingefangen und zu einer Batchdatei verarbeitet:

```
@echo off
rem Steuerzeichen in Variablen
set c1=^A
set c2=^B
....
set c30=^^
set c31=^_
```

Welchen Sinn das ganze haben kann, will ich natürlich nicht verschweigen:

1. Batchdateien generierende Batchdateien, die z.B. Warntöne ausgeben:
   ```
   ECHO %c7%
   ```

2. Ausführbare Maschinenspacheprogramme unter DOS schreiben:
   ```
   ECHO %c4%1%c27%L8% > TEST.COM
   ```

Während die erste Idee noch für jeden Batchprogrammierer verständlich werden kann, so ist doch die zweite eher für die Assemblerfreaks einleuchtend. So werden mit dem ECHO-Befehl die gewünschten Opcodes in ein ausführbares MC-Programm namens TEST.COM geschrieben. Das ich selbst kein Assmblerfreak bin, dürfte auch an den obigen Opcodes erkennbar sein. Dennoch kann man damit schnell kleine COM-Programme schreiben, wenn man z.B. auf den Opcode 26 verzichten kann. Diesen konnte ich beim besten Willen nicht zufassen bekommen!

Verzeichnis verschieben

Manchmal möchte man ein Verzeichnis auf ein anderes Laufwerk auslagern, damit man wieder etwas Platz für andere Daten hat. Weil mir die dafür notwendigen fünf Eingaben zu umständlich sind, habe ich sie in der Batchdatei MOVEDIR.BAT zusammengefaßt:

```
@echo off
if %2.==. goto err
md %2%1 > nul
copy %1\*.* %2%1 > nul
for %%a in (%1\*.*) do del %1\%%a
rd %1
goto off
:err
echo Ziellaufwerk fehlt!
:off
```

Will ich z.B. das Verzeichnis SYSTEM vom Laufwerk C: auf das Laufwerk A: verschieben, so muß ich jetzt nur noch folgendes eingeben:

```
MOVEDIR C:\SYSTEM A:
```

Zum Programmablauf:

Ist der zweite Parameter, der beim Aufruf das Ziellaufwerk enthalten sollte, leer, so wird man entsprechend darauf hingewiesen. Ansonsten wird auf dem Ziellaufwerk das Verzeichnis erzeugt, welches mit dem Parameter 1 angegeben wurde. Die nächste Zeile kopiert dann alle Dateien aus dem Verzeichnis in das gleichnamige Verzeichnis des Ziellaufwerkes. Anschließend werden die Daten im Quellverzeichnis gelöscht, bevor es dann selbst gelöscht wird. Damit ist die Verschiebung komplett und erledigt.

Sollten sich aber noch Unterverzeichnisse im zu verschiebenden Verzeichnis befinden, so werden diese nicht mit verschoben und das Verzeichnis wird nicht gelöscht. Diese Routine funktioniert nur dann fehlerfrei, wenn man sich nicht im zu verschiebenden Verzeichnis befindet!

Multiprocessing mit dem PoFo

Nicht schlecht gestaunt habe ich neulich, als ich wieder mit GW-BASIC arbeitete und die interne Textverarbeitung des PoFos aufrufen konnte!

Das laufende Programm wurde zwar angehalten, solange ich im Texteditor meinen Text schrieb, ist aber nach dem Rücksprung ins GW-BASIC weiter abgearbeitet worden. Allerdings funktioniert der Wechsel nur dann, wenn das laufende Basicprogramm einen Tastendruck erkennt und zuläßt. Weil es sich bei der Atari- Taste um einen Hotkey handelt, muß das Basicprogramm nicht einmal entsprechend programmiert werden, damit man in eine der fünf integrierten Applikationen gelangt.

Von meinem Ramspeicher mit 384 KB, konnte ich maximal 208 KB für die Ramdisk reservieren, damit dieses Multiprocessing funktioniert. Leider funktionierte es nicht auf einem 128 KB PoFo, was aber sicherlich an der Größe vom GW-BASIC liegt (80 KB). Da bleibt eben nicht genügend Speicher frei, um die Textverarbeitung aufzurufen.

Bei geschickter Programmierung kann man also, dank des Multiprocessings, Texte unter GW-BASIC modifizieren, bevor man diese dann andersweitig einsetzt. Es muß aber nicht nur bei GW-BASIC bleiben, sondern kann eventuell auch mit anderen Programmen funktionieren.

Kurzfristige Pfadänderung

Die mit PATH gesetzte Pfadangabe ist der Wegweiser für den Computer, damit er die gewünschten Programme findet und startet. Sobald ein neues Programm installiert werden soll, kann es notwendig sein, die Pfadangabe kurzfristig zu verlängern. Die folgende Batchdatei soll uns dabei behilflich sein:

```
@echo off
path
set opath=%path%
set path=%path%;C:\NEU
rem Hier fügen Sie Ihre Routinen
rem ein!
path
set path=%opath%
set opath=
path
```

Die drei path-Zeilen dienen uns dabei nur als Kontrolle und sollten später gelöscht werden. Schauen wir uns aber die anderen Zeilen an, die das ganze Geheimnis bargen:

Weil die Pfadangabe später wieder in den Ursprungszustand zurückversetzt werden soll, muß sie mit „SET OPATH=%PATH%" in die Variable OPATH gerettet werden. Nun wird die Pfadangabe mit „SET PATH=%path%;C:\NEU" um das Verzeichnis „C:\NEU" verlängert. Anstelle der REM-Zeilen fügen Sie also Ihre gewünschten Routinen ein. Sind diese abgearbeitet, wird die alte Pfadangabe durch „SET PATH=%OPATH%" wiederhergestellt. Ist auch dies erledigt, so kann die Variable OPATH gelöscht werden, was durch „SET OPATH=" geschieht.

Soll die Pfadangabe dauerhaft verlängert werden, so muß die Zeile „SET PATH=%PATH%;C:\" durch die folgende Batchdatei an die AUTOEXEC.BAT gehängt werden:

```
@echo off
echo set path=%path%;c:\neu >> autoexec.bat
```

Damit hätten wir also auch wieder ein Problem weniger, obwohl dieses noch von sehr leichter Natur war - zumindest für mich.

Eine zweite Ramdisk namens D:

Voller Überraschung stellte ich neulich fest, das der Blockgerätetreiber RAMDRIVE.SYS aus diversen MS-DOS Versionen, auch auf dem PoFo funktioniert.

Mit diesem Treiber kann man nun also eine zweite Ramdisk auf dem PoFo einrichten, die dann den Namen D: erhält. Dazu muß die folgende Zeile in die Datei CONFIG.SYS aufgenommen werden:

```
DEVICE=RAMDRIVE.SYS 64
```

Nach dem ich dann den PoFo mit STRG+ALT+EINF/ENTF neustartete, wurde eine zweite Ramdisk mit 64KB Größe eingerichtet. Auf diese konnte ich genauso zurückgreifen, wie auf die erste Ramdisk C:.

Dabei habe ich auch festgestellt, das diese zweite Ramdisk nur dann auf einem PoFo mit 128KB Speicher funktioniert, wenn die erste Ramdisk nicht größer als 32KB ist und man RAMDRIVE.SYS aus der MS-DOS Version 3.30 verwendet.

Ein zweiter Test, diesmal mit RAMDRIVE.SYS aus MS-DOS 6.22, ergab eine maximale Größe der ersten Ramdisk mit 24KB. Haben Eure PoFos aber mehr Speicher als 128KB, so könnt Ihr den Wert 64, in obiger Zeile, auch gerne Schritt für Schritt um 64 erhöhen: 128,192,256,...

Die Verwendung der zweiten Ramdisk ist aber nicht ganz risikolos: Sobald der PoFo neugestartet wird, egal ob Warm- oder Kaltstart, geht die zweite Ramdisk und mit Ihr alle Daten, verloren. Die Ramdisk C: verliert Ihre Daten zumindest bei einem Warmstart (STRG+ALT+EINF/ENTF) nicht!

Die Größe der Ramdisk C: verändert man ja bekanntlich mit dem Befehl FDISK: FDISK 32 und FDISK 24 sind für diese Tests die richtigen und vollständigen Befehle. Dringend dabei zu beachten ist aber, daß man alle Daten und Programme auf C: verliert, wenn man den FDISK Befehl startet!

„Wozu soll eine zweite Ramdisk gut sein?"

Man kann z.B. die Dateiprobleme des PoFo auch dann in den Griff bekommen, wenn man keine Speicherkarte besitzt. Oder man legt die Daten auf D: ab, die nur sehr kurzfristig benötigt werden und deren Verlust man leicht verkraften kann. Auch eine bessere und geordneter Dateiablage kann man mit der Verwendung der zweiten Ramdisk erreichen.

Diese Erfahrungen beruhen auf PoFos mit den Bios Versionen 1.072 und 1.130.

Rekursive Batchdateien

... sind besonders interessant, weil man durch sie große Aktionen mit kleinem Aufwand erreichen kann. Gerade für tagtäglich wiederkehrende Aufgaben, haben wir entsprechend große Routinen parat, deren Inhalte sich oftmals gleichen. Diese Speicherplatzverschwendung kann leicht verhindert werden, wenn man die Kernroutinen isoliert, in eigenen Batchdateien, und für Rücksprünge sorgt.

Dafür werden in den Batchdateien, die jeweils eine Kenroutine enthalten, ich nenne sie mal Slave-Batchdateien, Variablen gesetzt, die von der großen Batchdatei, die ich Master-Batchdatei nennen möchte, überprüft werden. Durch die Parameter %0 und %1 werden die rekursiven Aufrufe getätigt.

Hier die beispielhaften Batchdateien:

MB.BAT:
```
@echo off
if %rek%==1 goto zwei
if %rek%==2 goto drei
echo Masterbatch zum ersten .
sb1 %0
:zwei
echo    Masterbatch zum zweiten ..
sb2 %0
:drei
echo   Masterbatch zum letzten mal!
set rek=
```

SB1.BAT:
```
@echo  off
echo Slavebatch eins
set rek=1
%1
```

SB2.BAT:
```
@echo off
echo Slavebatch zwei
set rek=2
%1
```

Und so funktioniert das ganze:

Die Master-Batchdatei, im folgenden nur noch MB.BAT genannt, prüft bei jedem Aufruf zunächst den Inhalt von REK. Weil REK beim ersten mal nicht definiert ist, werden einige Befehlszeilen abgearbeitet. Irgendwann kommt dann der Aufruf der Slave-Batchdatei Nr.1, kurz: SB1.BAT, die nun die Kontrolle übernimmt. Sie wird mit dem Parameter %0 aufgerufen, der immer den Namen der Batchdatei enthält, welche gerade abgearbeitet wird (hier: MB). Am Ende definiert sie REK=1 und übergibt die Kontrolle wieder an die MB.BAT, indem sie die MB.BAT durch den Parameter %1 aufruft, welcher ja hier den Inhalt MB aufweist. MB.BAT prüft wieder die REK-Variable und stellt fest, das sie mit 1 definiert ist. Daher überspringt MB.BAT die nächsten Zeilen und fährt bei der Routine :zwei fort. Kurz darauf wechselt nun die Kontrolle zu SB2.BAT, welches im späteren Verlauf REK auf den Wert 2 setzt, bevor es die Kontrolle wieder an MB.BAT zurückgibt. MB.BAT prüft REK erneut ab und springt zur Routine :drei, in deren Verlauf es REK wieder löscht und seine Tätigkeit beendet.

Würde REK nicht gelöscht, so würde bei späteren Aufrufen eine falsche Reihenfolge eingehalten werden, durch die evtl. Dateien irreparabel beschädigt oder vernichtet werden. Durch die Parameterübergabe, beim Aufruf der Slave-Batchdateien, werden diese so dressiert, das sie sich jeder Master-Batchdatei unterordnen und immer zur richtigen zurückfinden.

Mit dieser Technik kann man z.B. gezielte Datensicherungen, Dateimanipulationen oder Datenauswertung betreiben!

Verzeichnis umbenennen

Für die Umbenennung von Dateien existiert ja der RENAME-Befehl, aber für die Verzeichnisse? Ja, da versagt das DOS ganz kläglich. Trotzdem braucht man aber nicht darauf verzichten, sondern muß einen kleinen Umweg gehen.

Man kreiert einfach ein Verzeichnis mit dem gewünschten Namen, kopiert alle Dateien ins neue Verzeichnis und löscht zuletzt das alte Verzeichnis inkl. der Dateien. Kurzweilig existiert also eine Doublette des Verzeichnisses, die auch dementsprechend Platz benötigt. Daran muß natürlich gedacht werden, bevor man also diese automatisierende Routine, namens RENDIR.BAT, einsetzt:

```
@echo off
if %2.==. goto err
md %2 > nul
copy %1\*.* %2\ > nul
for %%a in (%1\*.*) do del %1\%%a
rd %1
goto off
:err
echo   Neuer Verzeichnisname fehlt!
:off
```

Aufgerufen wird sie z.B. mit: „RENDIR C:\SYSTEM C:\DOS“, wenn das Verzeichnis SYSTEM in DOS umbenannt werden soll. Aber auch hier darf man sich nicht in dem Verzeichnis befinden, welches umbenannt werden soll. Befinden sich noch weitere Unterverzeichnisse in dem umzubenennenden Verzeichnis, so wird es nicht gelöscht!

Variablen umbenennen & kopieren

Ob man diese Routinen tatsächlich benötigt, sei einmal dahingestellt. Immerhin kann man damit seine Variablen vor überschreiben schützen, wenn einem bewußt ist, das eine andere Routine dieselbe Variable benötigt.

Schauen wir uns „RV.BAT“ doch einmal an:

```
@echo off
rem RV - Rename Variable
if %1==off goto %1
if %2.==. goto err
copy c:\batch\rv2.dat c:\batch\*.b*>nul
echo %1%%>>c:\batch\rv2.bat
echo set %1=>>c:\batch\rv2.bat
echo echo Variable %1 in %2
  umbenannt!>>c:\batch\rv2.bat
echo rv off>>c:\batch\rv2.bat
rv2 %2
:err
echo   Aufruf mit:
echo    RV alter Variablenname neuer Variablenname
:off
del  c:\batch\rv2.bat>nul
```

Benötigt wird außerdem noch „RV2.DAT“:

```
@echo off
set %1=%
```

Und so soll das ganze funktionieren:

Weil „RV.BAT“ später von „RV2.BAT“ rekursiv aufgerufen wird, mit dem Wort off als ersten Parameter, darf zum einen keine Variable off umbenannt werden, und zum anderen sollte diese Zeile beim ersten Aufruf übergangen werden. Ist die korrekte Syntax beim Aufruf eingehalten worden, so wird „RV2.DAT“ zu „RV2.BAT“ kopiert. Nun wird der Name der alten Variable an „RV2.BAT“ gehängt, wodurch die Zeile -SET %1=%- um diesen Variablenamen und das Prozentzeichen vervollständigt wird. Weil die alte Variable ja auch gelöscht werden soll, gelangt als nächstes die Zeile -SET %1=- per -ECHO-Befehl in „RV2.BAT“. %1 wird aber dabei schon durch den alten Variablennamen ersetzt! Jetzt wird die Rückmeldung für den Betrachter in „RV2.BAT“ transferiert, wie auch der Rekursivaufruf von „RV.BAT“, der da lautet -RV off-.

Erfolgte also der Start durch RV test new, so sieht die „RV2.BAT“ wie folgt aus:

```
@echo off
set %1=%test%
set test=
echo Variable test in new umbenannt!
rv off
```

Die Vorkehrungen sind damit abgeschlossen, so daß der Aufruf mit -RV2 new- erfolgt. Mit dem ersten -SET- Befehl wird die neue Variable new mit dem Inhalt von test definiert. Der zweite -SET- Befehl löscht nun die Variable test, damit diese für andere Zwecke verwendet werden kann. Im nächsten Moment erhaltet Ihr die Bestätigung für den korrekten Verlauf, bevor „RV.BAT" erneut durch -RV off- aufgerufen wird.

Weil jetzt der erste Parameter off lautet, trifft die erste -IF- Zeile zu und bedingt den sprung zu (:off). Diese Sprungmarke löscht „RV2.BAT" und beendet die ganze Routine.

Die „CV.BAT", die dieselbe Technik verwendet, um eine Variable zu kopieren, unterscheidet sich nur in zwei Zeilen von „RV.BAT". Ich gehe daher nicht weiter auf „CV.BAT" und „CV2.DAT" ein.

Schalt-Technik

Einige Batchdateien werfen geradezu mit Bildschirmtexten um sich, die keinen Benutzer mehr interessieren, weil er sie bereits auswendig kennt. Gerade die als die Nonplusultra gepriesenen Menüsysteme, kennen keinen Pardon!

„Wie schön wäre es doch, wenn man solche Schirmmasken einfach abschalten könnte.", höre ich Euch seufzen. Die Rettung naht bereits, zumindest wenn diese Menüschirme durch Batchdateien kreiert werden.

Handelt es sich bei dem Menüschirm nur um eine Textdatei, die per Batchdatei auf den Schirm kopiert wird, so kann die Befehlszeile COPY „MENÜ.TXT" CON einfach aus der Batchdatei gelöscht werden. Wird die Maske durch mehrere ECHO-Befehle generiert, so können auch diese gelöscht werden.

Was werden aber Eure Bekannten sagen, wenn Ihr ihnen das Menüsystem kopiert? „Ich komm damit nicht klar, weil nirgendwo zu sehen ist, was ich einzugeben habe! Hilf mir doch mal weiter."

Die goldene Lösung ist also eine Bildschirmmaske, die bei Bedarf an- oder abgeschaltet werden kann.

Und wer ist flexibler als unsere Helfer, die Parameter? Natürlich die Variablen! Aber diesmal teilen sich beide die Arbeit und schalten mit der folgenden Batchdatei, die integrierte Bildschirmmaske ON oder OFF:

```
@echo off
if %1.==. goto prüf
set sw=%1
:prüf
if %sw%==off goto off
if %sw%==OFF goto off
cls
echo ¦+ (T)exteditor  (S)ystem (.)..... +
echo ¦¦ (A)dressverw. (P)basic (.)..... ¦
echo ¦¦ (Z)eitplaner  (D)fü   (C)hdir  » ¦
echo ¦¦ (K)alkulation (F)ilem. (H)ilfe  » ¦
echo ¦¦ (R)echner     (U)til's (O)ff     ¦
echo ¦+—— Anfangsbuchstabe + RETURN ——+
:off
```

Zuerst wird geprüft, ob eventuell ein Parameter beim Aufruf der Batchdatei mitgegeben wurde. Ist dies nicht der Fall, wird durch die Routine :prüf abgecheckt, ob die Variable SW als OFF oder off definiert ist. Wenn auch dieser Check negativ ausfällt, so soll das Menübild sichtbar sein und wird deswegen auch aufgebaut. Enthält der o.g. Parameter die Definition OFF oder off, so wird kein Menübild gezeigt. Jede andere Parameterübergabe wird als ON interpretiert und blendet die Maske wieder ein. Der jeweilige Schaltzustand bleibt solange bestehen, bis durch ein neuer Aufruf, inkl. Parameter, eine Umschaltung erfolgt.

Ur-Startlaufwerk ermitteln!

Neulich bekam ich ein PD-Programm, welches sich nur halb installierte und dann startete. Das Programm suchte dann auch bald seine Dateien im Laufwerk C:, weil es ja von dort gestartet wurde. Leider schlummerten aber die fehlenden Dateien noch im Laufwerk A:, von dem ich bestimmungsgemäß die Installationsroutine gestartet hatte. Irgendwann sah auch das Programm es ein, das hier etwas nicht stimmen konnte und überliess mich dem Prompt! Gottlob konnte ich die Installationsroutine, die aus einer AUTOEXEC.BAT inkl. CONFIG.SYS-Erweiterung bestand, reparieren. Diese Routine fand einfach nicht mehr zu dem Laufwerk zurück, von dem sie gestartet wurde!!!

Aber überlassen wir den Programmierer seiner Schande und versuchen selbst das Ur-Startlaufwerk festzustellen und ähnlich „doofen" Routinen an die Hand zu geben.

Beim Start der AUTOEXEC.BAT meines Portfolios, wird automatisch der Parameter

%0 mit dem folgenden Inhalt gefüttert: „A:\AUTOEXEC". Vorausgesetzt, die AUTOEXEC.BAT befindet sich im A:-Laufwerk - und das setze ich hier einmal voraus!

Festhalten wollen wir also nur die Zeichenfolge „A:" und das möglichst zur weiteren Nutzung in einer Variablen. Alle Versuche, das überflüssige Wort „AUTOEXEC" abzuschneiden, waren zum Scheitern verurteilt. Weil aber der Inhalt des Parameters %0 bekannt ist, konnte ich diesen mit allen Variationsmöglichkeiten vergleichen und bei passendem Vergleich, die Variable LW, die ich zur Aufnahme auserkoren hatte, dementsprechend selbst mit dem Laufwerksnamen definieren! Die folgende AUTOEXEC.BAT erledigt das nun für mich:

```
@echo off
set lw=%0
if %lw%==A:\AUTOEXEC goto a
if %lw%==B:\AUTOEXEC goto b
if %lw%==C:\AUTOEXEC goto c
goto fehler
:a
set lw=A:
goto message
:b
set lw=B:
goto message
:c
set lw=C:
:message
echo Das Startlaufwerk ist %lw%!
goto ende
:fehler
echo Das Startlaufwerk ist zumindest nicht
echo A:, B: oder C:! Näheres konnte leider
echo nicht ermittelt werden, sorry!!!
:ende
```

Diese Routine funktioniert wirklich nur unmittelbar nach dem Einschalten des Rechners, oder nach einem Reset. Wird sie nach längerer Computerbenutzung nochmals aufgerufen, so geht damit der Inhalt der Variablen LW unwiederbringlich verloren!

Mit der folgenden Befehlszeile kopiere ich alle Dateien vom Ur-Startlaufwerk nach „C:", wenn ich sie in einer Batchdatei einsetze:

```
copy %lw%*.* c:\
```

Im Direktmodus, so ist es zumindest beim Portfolio (=PoFo), kann ich diesen Befehl nicht verwerten.

Nun ist der PoFo auch sonst ein sehr eigenwilliger Compi, weshalb der Inhalt des Parameters %0, nämlich „A:\AUTOEXEC", nicht dem Inhalt desgleichen Parameters, auf „großen" PC's, gleichen muß!!!

Mit der folgenden AUTOEXEC.BAT kann man aber den Inhalt von %0 überprüfen:

```
@echo off
echo %0 %1 %2 %3
```

Sollte also das Ergebnis anders ausfallen, so ist die obige Batchdatei entsprechend abzuändern.

Eine unlöschbare Variable simulieren!

Das geht nur direkt am DOS-Prompt indem man eingibt:

```
SET VAR=^Z
```

Wobei ^Z für die Tastenkombination STRG+Z steht!
Im Gegensatz zu den echt unlöschbaren Variablen, die auftreten wenn das Environment voll ist, kann diese Variable wieder durch die Eingabe von:

```
SET VAR=
```

gelöscht werden. Sie kann aber auch durch eine Batchdatei gelöscht werden! Eine vermeintliche Leerzeile im Environment erreicht man durch diese Direkteingabe:

```
SET ^Z=^Z
```

Auch diese Leerzeile ist eine vermeintlich unlöschbare Variable! Diese kann nur durch die Direkteingabe von:

```
SET ^Z=
```

gelöscht werden. Eine Batchdatei kann diese Variable nicht löschen!

KnowHow: DIP-OS

SYSTEMFEHLER -Verzeihung!

... bittet meist die Textverarbeitung, wenn sie einen Text nicht mehr laden kann. Dies liegt meistens an einer Leerzeichen/Return-Kombination, die am Ende des zuletzt gespeicherten Textes zu finden ist. Sie zwingt damit zunächst den PoFo zu einem Neustart, setzt aber auch auf Dauer die Textverarbeitung schachmatt. Leider wird ja immer der letzte Text automatisch geladen, der in diesem Fall fehlerhaft ist und somit eine Kettenreaktion auslöst. Erfahrene und langjährige User benutzen deshalb das Programm PORTDIV.COM, welches diesen Systemfehler verhindert. Aber was tun wenn man dieses Programm nicht hat?

Damit man die Textverarbeitung überhaupt wieder nutzen kann, kann man die Datei PERMDATA.DAT im Verzeichnis C:\SYSTEM löschen. Mit „DEL C:\SYSTEM\ PERMDATA.DAT" wird das erledigt. Weil damit der Text nach wie vor nicht repariert wird, ist die folgende Lösung besser und erfolgreicher: Gebt einfach am Prompt den Befehl „ECHO M>>BEISPIEL.TXT" ein!

Damit wird der Text, hier BEISPIEL.TXT, um den Buchstaben M verlängert und überlistet somit die Textverarbeitung. Problemlos kann man nun den Text wieder editieren und korrigieren!

Als ich diese Lösung dem Jens Sewitz vom Atari Computer Team e.V. Bremen mitteilte, wir lernten uns auf der 10. Bremer Computer- & Videobörse kennen, brachte er mich auf die Idee, diese Lösung in eine Batchdatei zu packen.

Und hier ist also die NOERROR.BAT:

```
@echo off
if %1.==. goto err
echo m>>%1.txt
app/e
goto off
:err
echo Textdatei ohne .TXT angeben!
:off
```

Wenn ich also unseren Text BEISPIEL.TXT korrigieren muß, lautet die Eingabe:"NOERROR BEISPIEL". NOERROR hängt dann automatisch ein kleines „m" an BEISPIEL.TXT und ruft anschließend die Textverarbeitung auf. In dieser sind nun die letzten drei Zeichen zu löschen, bevor man den Text neu abspeichert.

Theoretisch kann man dieses Löschen auch mit FAKE.COM automatisieren, aber ich muß ja nicht alle Lösungen präsentieren. Schließlich sollt Ihr ja auch Nachdenken und Erfolgserlebnisse haben!!!

Vergaloppiert & -kopiert?

Stellt Euch einmal folgende Situation vor:

Larry L. benutzt gerade sein DOS und steckt im Verzeichnis SEX auf dem Laufwerk A:. Die dort befindlichen Dateien will er nun ins Verzeichnis BILDER auf Laufwerk C: kopieren. Unaufmerksam, wie er nun einmal ist, gibt er den Kopierbefehl wie folgt ein: COPY *.* C:. Flugs werden alle Dateien von A:\SEX nach C:\TEXTE kopiert, da unser Larry vergessen hat, zuvor auf Laufwerk C: ins Verzeichnis \BILDER zu wechseln! Da stecken nun also Texte und Bilder doch unter einer Decke, Pardon: in einem Verzeichnis, obwohl sie fein säuberlich getrennt sein sollten.

Larry macht sich also nun mühsam daran alle fehlplazierten Dateien von Hand zu löschen, damit auch ja kein wertvoller Liebesbrief verloren geht.

Armer Larry! Hättest du diese Batchdatei UNCOPY.BAT, so wäre dir viel Handarbeit erspart geblieben:

```
@echo off
if %1.==. goto err
for %%a in (*.*) do if exist %1\%%a del %1\%%a
goto off
:err
echo   Laufwerk und Verzeichnis fehlen!
:off
```

Du mußt sie mit dem Laufwerks- und Verzeichnisnamen aufrufen, in denen die fehlplazierten Dateien liegen. In deinem Fall also mit UNCOPY C:\BILDER . Denn die FOR-IN-DO-Schleife prüft zunächst nach, welche Dateien (*.*) des aktuellen Verzeichnisses, sich ebenfalls im, als Parameter %1 übergebenen, Zielverzeichnis befinden. Sind dort gleichnamige Dateien zu finden, so werden sie dank DEL %1\%%A auch gleich gelöscht. Somit verlierst Du mit Sicherheit keinen deiner Liebesbriefe und vergißt auch keine Datei zu löschen!

Larrylob sind wir, verehrte Leser, nicht so larry drauf, weshalb UNCOPY.BAT nur sehr selten bei uns zum Einsatz kommt, oder???

UPDATE.COM & PORTDIV.COM

Weil mich mindestens einmal pro Monat ein Hilferuf zu diesen Dateien erreicht, möchte ich mit diesem Text, der frei kopiert und weitergegeben werden darf, die Verwendung und die Installation dieser Programme erklären. Allerdings sind gewisse Vorkenntnisse übers DOS und die Textverarbeitung des PoFos vonnöten, denn alle möglichen Varianten sind mir nicht bekannt und von Fall zu Fall verschieden. Diese Anleitung beschreibt den generellen Fall.

Wer also nicht weiß, was Hauptverzeichnis, Unterverzeichnis, AUTOEXEC.BAT und CONFIG.SYS sind, der sollte sich schleunigst ein Buch zu MS-DOS kaufen, indem auch die im PoFo vorhandenen Befehle stehen und erklären, wie man Dateien ablegt, ordnet und verändert.

1. Übertragen dieser Programme auf die Ramdisk C:
1.1 von der Memorycard im Kartenschacht A: des PoFo mit:

```
COPY A:UPDATE.COM  C:\
COPY A:PORTDIV.COM C:\
```

1.2 vom PC via Serialinterface:
1.2.1 Eingabe am PoFo:

```
COPY AUX C:\UPADATE.COM
```

1.2.2 Eingabe am PC :

```
COPY UPDATE.COM AUX
```

1.2.3 Eingabe am PoFo:

```
COPY AUX C:\PORTDIV.COM
```

1.2.4 Eingabe am PC :

```
COPY PORTDIV.COM AUX
```

Andere Übertragungsarten sind auch möglich, habe ich aber noch nicht ausprobiert.

2. Einbinden der Programme:
2.1 Starten Sie die interne Textverarbeitung des PoFo:

```
APP /E
```

2.2 Laden Sie die AUTOEXEC.BAT ein, in dem Sie nach einander die folgenden Tasten drücken: ATARI-Taste (ganz links unten, rotes Atari-Logo), D-Taste, L-Taste folgenden Text eingeben: C:\AUTOEXEC.BAT und die Winkelpfeil-Taste drücken

2.3 Erscheint die Fehlermeldung „Datei nicht gefunden", so drücken Sie zweimal die ESC-Taste (ganz links oben) und drücken dann die N-Taste.
2.4 Gehen Sie mit dem Cursor zum Anfang der Datei und fügen Sie dort die folgenden drei Zeilen vor:

```
@echo off
update
portdiv
```

2.5 Ist die AUTOEXEC.BAT nicht vorhanden, so geben Sie bitte die folgenden Zeilen ein:

```
@echo off
update
portdiv
prompt$p$g
path=c:\;c:\system
```

2.6 Speichern Sie die AUTOEXEC.BAT wie folgt ab: Atari-Taste, D-Taste, S-Taste, folgenden Text eingeben: C:\AUTOEXEC.BAT und die Winkelpfeil-Taste drücken
2.7 Verlassen Sie die Textverarbeitung mit einmaligem Druck auf die ESC-Taste.
3. Start der Programme:
3.1 Führen Sie einen Warmstart aus, indem Sie die Tasten STRG, ALT und EINF/ENTF zugleich drücken.

- Alle Angaben ohne Gewähr -

Wo bin Ich?

Ob sich das DOS an diesem beliebten deutschen Ratespiel beteiligt, ist leider nicht überliefert. Anscheinend hat es die Laufwerksverwaltung so gut im Griff, das diese Frage nicht aufkommen kann. Für automatische Abläufe kann es aber sehr hilfreich sein zu wissen, welches Laufwerk gerade aktuell ist. Dem soll meine Batchdatei „WOBINICH.BAT" Rechnung tragen:

```
@echo off
rem Wo bin ich momentan ? - aktuelles Laufwerk
rem ermitteln!
if %1==. goto .
copy c:\batch\wbi*.dat c:\batch\*.b*>nul
cd>>c:\batch\wbi2.bat
cd\
cd>>c:\batch\wbi3.bat
echo echo Das Laufwerk %%alw%% ist
  aktuell!>>c:\batch\wbi3.bat
type c:\batch\wbi2.bat>>c:\batch\wbi3.bat
echo wobinich .>>c:\batch\wbi3.bat
wbi3
```

```
:.
del c:\batch\wbi*.bat>nul
```

„WBI2.DAT“:

```
cd
```

„WBI3.DAT“:

```
@echo off
set alw=
```

Besprechen wir zunächst die Gedanken, die dieser Routine vorausgegangen sind:
Weder Laufwerk, noch Verzeichnis, sind beim Start bekannt. Helfen kann da nur der Befehl -CD-, dessen Reaktion in Dateien umgeleitet wird. Unter alten DOS-Versionen ist ja keine Trennung zwischen der Laufwerkskennung und dem Verzeichnisnamen möglich, weshalb ein wechsel in ein bekanntes Verzeichnis stattfinden muß. Bestens geeignet ist dafür das Hauptverzeichnis, da es erstens auf jedem Datenträger vorhanden ist und zweitens, einen markanten Namen besitzt, der leicht verglichen werden kann. Bevor aber dieser Wechsel ausgeführt wird, sollte noch das aktuelle Verzeichnis gerettet werden, damit es später wieder aktualisiert wird.
Soweit die Vorgeschichte, jetzt geht es an des Routinens Kern:
Beim ersten Start d.h., wenn Ihr „WOBINICH.BAT“ durch die Eingabe WOBINICH aufruft, werden als erstes die Hilfsdateien kopiert. Es folgt die Sicherung des aktuellen Verzeichnisnamens in die Datei „WBI2.BAT“, der Wechsel ins Hauptverzeichnis und die Sicherung des Hauptverzeichnisnamens in „WBI3.BAT“. Jetzt wird die Rückmeldung in „WBI3.BAT“ plaziert, bevor sie durch „WBI2.BAT“, mittels -TYPE- Befehl, verlängert wird. Die „WBI2.BAT“ enthält ja den Verzeichnisnamen, der beim Start von „WOBINICH.BAT aktuell war. Per -ECHO- wird nun der Befehl zum Rücksprung -WOBINICH .-, der „WBI3.BAT“ mitgeteilt, bevor sie dann gestartet wird.
Angenommen, mein PoFo steckt gerade in dem Verzeichnis SYSTEM des Laufwerkes C:, dann sähe die „WBI3.BAT“ schließlich so aus:

```
@echo off
set alw=c:\
echo Das Laufwerk %alw% ist aktuell!
cd c:\system
wobinich .
```

Sie setzt die Variable ALW und definiert sie mit dem Laufwerksnamen und dem Backslash, der die Kennzeichnung für das Hauptverzeichnis ist. Eigentlich müßte dieser Backslash noch weg, da wir nur die Laufwerkskennnung C: suchten, aber die Routine wäre dadurch noch unverhältnismäßig länger geworden. Nun bekommt Ihr die Rückmeldung auf den Schirm und es folgt der Rückwechsel ins Verzeichnis SYSTEM, von wo die ganze Routine gestartet wurde. Mit dem erneuten Start von „WOBINICH.BAT“, sorgt „WBI3.BAT“ für die Wiederherstellung der vorherigen Ordnung. Die Variable ALW wird allerdings nicht gelöscht, damit andere Routinen auf sie zurückgreifen können!

KnowHow: DIP-OS

Auch Kleines wird einmal Groß

Mit der folgenden Routine kann man Parameter in Großschrift umwandeln. Somit kann man viele Überprüfungszeilen vermeiden, oder Worte besonders hervorheben. Doch leider funktioniert die Technik nur auf dem PoFo:

```
set ß=hallo
echo %ß%
```

Werden diese Zeilen in eine Batchdatei gepackt, so wird nach dem Aufruf derselbigen, das Wort hallo in Großbuchstaben auf dem LCD-Schirm erscheinen. Definiert man nämlich eine Variable, in deren Name ein ASCII-Charakter oberhalb von 126 steckt, so wird der Inhalt automatisch vom PoFo in Großschrift umgewandelt. Weil lange Variablennamen nur Speicherplatz verschwenden und das „ß“ direkt neben dem „=“ auf der Tastatur liegt, habe ich mir diesen Variablennamen ausgesucht. Ärger, Müll, Öl, Übel oder alle Konstruktionen aus Grafikzeichen vermögen dasselbe zu leisten.
Diese Erfahrung brachte mich auf die Idee, eine Batchdatei zu schreiben, mit der bestehende Variablen in Großschrift umgewandelt werden können. Und weil mir das ganze so gut von der Hand ging, kann man damit auch noch nicht existierende Variablen gleich in Großschrift definieren.

GV.BAT:

```
@echo off
rem GV - Großgeschriebene Variableninhalte
if not %ü%.==. if %ß%.==. goto err2
if exist c:\batch\gv2.bat goto msg
if %1.==. goto err
if %2.==. goto old
:new
set ü=%1
set ß=%2
set %1=%ß%
:msg
echo Die Variable %ü% enthält nun %ß%
set ß=
set ü=
goto off
:old
set ü=%1
copy c:\batch\gv2.dat c:\batch\*.b*>nul
echo %1%%>>c:\batch\gv2.bat
echo set %%1=%%ß%%>>c:\batch\gv2.bat
echo gv>>c:\batch\gv2.bat
gv2 %ü%
:err
cls
echo Aufruf bei noch nicht existierender
echo Variable:
echo  GV Variablenname Variableninhalt
echo
echo Aufruf bei existierender Variable:
echo  GV Variablenname
goto off
:err2
echo Die Variable %ü% existiert nicht!
set ü=
:off
del c:\batch\gv2.bat>nul
```

GV2.DAT:

```
@echo off
set ß=%
```

Gehen wir nun einmal in der Annahme, daß die Variable TEST existiert und mit „ja" definiert ist. Wollen wir dieses „ja" in Großschrift verwandeln, so rufen wir „GV.BAT" mit -GV Test- auf. Da die Variablen ü und ß sowie „GV2.BAT" noch undefiniert, bzw. nicht existieren, werden weder die Sprungmarke „err2" noch „msg" ausgeführt. Als nächstes wird das Vorhandensein des Parameters %1 geprüft. Wurde kein Parameter angegeben, so wird man auf den fehlerhaften Start hingewiesen und bekommt die mögliche Syntax angezeigt. Weil bei unserem Testlauf eine bereits vorhandene Variable umgewandelt werden soll, trifft es zu, das kein zweiter Parameter angegeben wurde. Somit erfolgt ein Sprung zu (:old). Dort wird aus optischen Gründen zunächst der erste Parameter, der ja einem Variablennamen gleicht selbst in Großschrift umgewandelt: -set ü=%1-.

Nun tritt aber das Problem auf, das man nicht an den Inhalt der Variablen kommt, die als Parameter übergeben wurde. -SET ß=%%1%- wäre zwar die richtige Lösung für dieses Problem, funktioniert aber leider nicht. Deswegen nehmen wir „GV2.DAT" zu Hilfe, welche zu „GV2.BAT" kopiert wird. Im nächsten Moment wird die unvollständige Zeile -set ß=%- von „GV2.BAT" mit „test%" komplettiert, weil %1 in unserem Falle ja dem Wort „test" entspricht. Somit kommen wir doch an den Inhalt der Variablen heran, die als Parameter übergeben wurde. Gleichzeitig wird dadurch der dem Wort „ja" entsprechende Inhalt in Großschrift verwandelt, sofern „GV2.BAT" zur Ausführung kommt. Damit wir aber später über die geglückte Umwandlung informiert werden, muß „GV2.BAT" noch angewiesen werden, zur „GV.BAT" zurückzukehren. Dies geschieht mittels des Befehles -echo gv>>c:\batch\gv2.bat-. Endlich sind alle Vorkehrungen getroffen und „GV2.BAT" wird durch die Zeile -gv2-gestartet.

Kaum das Ruder in die Hand genommen, wird nun die Variable ß mit „JA" definiert, bevor die Kontrolle wieder an „GV.BAT" zurückgeht.

„GV.BAT" stellt nun die Existenz von „GV2.BAT" fest, geht daher von der korrekten Umwandlung aus und springt zu (:msg). Es folgt die Ausgabe der Bestätigung, bei der Variablenname und der Variableninhalt in Großschrift erfolgt. Der guten Ordnung halber werden dann noch die Hilfsvariablen ß und ü, sowie die „GV2.BAT" gelöscht.

Wer schreibfaul ist oder nicht die SHIFT-Taste findet, kann die Variable AUTO mit NEIN definieren, wenn er „GV.BAT" durch die Eingabe von „GV auto nein" startet. In diesem, unserem Falle muß die Sprungmarke (:new) ausgeführt werden.

Durch die drei dort befindlichen -SET-Zeilen wird die Umwandlung vollzogen, was Ihr ebenfalls leicht nachvollziehen solltet.

Nahtlos geht es mit (:msg) weiter, die den korrekten Verlauf bestätigen soll und für die Beseitigung der Hilfsvariablen zuständig ist.

Obwohl wir bei diesem zweiten Testlauf „GV2.BAT" nicht angelegt hatten, war ja nicht notwendig, wird diese Datei versucht zu löschen. Dank der Umleitung nach NUL bekommen wir aber keine Fehlermeldung mitgeteilt.

Natürlich kann man die Variable AUTO auch im Direktmodus mit NEIN definieren wenn man -SET AUTO=NEIN- eingibt. Dieser Weg ist zwar einfacher, aber bei automatisch ablaufenden Batchdateien ist meist keine userbedingte Eingabe möglich oder gar unerwünscht. Deshalb hat die zweite Möglichkeit der Umwandlung auch Sinn!

GW-Basic auf dem PoFo - Teil 3

Ich habe mich noch einmal hingesetzt und die Startroutine zum GW-Basic überarbeitet, weil meine alte Version, aus dem Jahre 12 nach DOS (1993), nicht mehr zeitgemäß und viel zu umständlich ist!

Herausgekommen ist eine neue AUTOEXEC.BAT, die ins Hauptverzeichnis einer 128KB Memorycard gehört:

```
@echo off
if not exist a:\gwbasic.exe goto err
if not exist c:\system\*.* goto next
if exist c:\system\@!@-@!@.gwi goto next
cls
echo +-----------------------------------+
echo ¦!ATTENTION! ATTENZIONE! ACHTUNG! ¦
echo +-----------------------------------+
echo
echo Try formatting C: with 200K!
fdisk 200
echo Try formatting C: with 8K!
fdisk 8
:next
md c:\system>nul
rem>c:\system\@!@-@!@.gwi
prompt $p$g
path=a:\;c:\
a:\portdiv>nul
cls
echo  +-----------------------------------+
echo  ¦ GW-BASIC 3.22  - Startroutine      ¦
echo  ¦developed 1996 by Lasse Assebasse¦
echo  +-----------------------------------+
a:\gwbasic
:err
if not exist a:\gwbasic.exe echo File GWBASIC.EXE is
   missing!
```

Ersatzlos gelöscht werden können die Dateien AUTO2.BAT, AUTO3.BAT, AUTO4.BAT, **AUTOEXEC.ORI und J.DAT.**

Diese Version formatiert die Ramdisk nicht mehr automatisch, weil man den PoFo ja auf drei verschiedene Sprachen (Englisch/Francais/Deutsch) einstellen kann und dieses zu Problemen führt:

Bei der alten Version bin ich von „auf Deutsch eingestellten" PoFos ausgegangen und habe die automatische Formatierung durch:

```
FDISK 8 < j.dat > nul
```

erreicht. Bei auf Deutsch eingestellten Geräten gibt es damit auch keine Probleme, aber bei denen, die auf Englisch oder Französisch eingestellt sind!

Denn dort muß man die Rückfrage vom DIP-OS mit der Taste Y für YES oder O für OUI beantworten!!! Und somit kann die nicht-deutsche-Rückfrage nichts mit der Taste J anfangen, die ich in die Datei J.DAT, zwecks Umleitung an FDISK, gestellt habe!

Meine Versuche alle möglichen und erlaubten Tasten (YOJ) in YOJ.DAT zu stellen, quittierte der PoFo durch totale Arbeitsverweigerung!

Wer es probieren möchte, sollte so

```
YOJ.DAT mit der Textverarbeitung schreiben:
Y
O
J
```

und dann den Befehl: FDISK 8 < YOJ.DAT im DIP-OS (DOS) eingeben.

Zu gerne hätte ich dieses Problem aus der Welt geschafft und EINE Lösung für alle Sprachen geschaffen, aber der PoFo spielt da (noch) nicht mit!

Und was lernen wir daraus:

Wenn wir Batchdateien schreiben, die Tastendrücke aus Dateien benötigen, so müssen wir daran denken, das es nicht nur die deutsche Sprache gibt!!!

Aber allem Anschein nach sind die CompuServe-User echte Freaks und haben meine alte Lösung, ohne zu murren!, Ihren Sprachen gemäß verändert. Ich hoffe nur, das Ihr nicht so tolerant seit, weil solche Fehler nicht gerade jahrelang unentdeckt bleiben sollten!!!

Noch ein paar Worte zur Programmtechnik:

Wie allseits bekannt sein dürfte, formatiert man mit dem Befehl FDISK die Ramdisk C: des PoFo. Weil dadurch ALLE Dateien auf C: verloren gehen, auch CONFIG.SYS und

AUTOEXEC.BAT, muß der PoFo danach neu gestartet werden. Denn ein evtl. vorher vorhandener Gerätetreiber ist nun nicht mehr da und kann auch nicht mehr genutzt werden! Deshalb nimmt der frisch formatierte PoFo den Neustart automatisch vor.

Nun bedeutet ein Neustart auch die wiederholte Suche nach einer ausführbaren CONFIG.SYS und AUTOEXEC.BAT.

Da auf C:\ keine AUTOEXEC.BAT mehr existieren kann und eine auf A:\ zu finden ist, nämlich die obige, wird die von A:\ gestartet. Selbst wenn eine AUTOEXEC.BAT auf B:\ zu finden ist, so kommt diese nicht zur Ausführung, weil die auf A:\ Vorrang hat!

Rein theoretisch müßte die obige AUTOEXEC.BAT also wieder die Ramdisk C: formatieren und damit wieder einen Neustart auslösen. Und wieder, und wieder und wieder, ...: Eine Schlange die sich selbst in den Schwanz beißt!!!

Wenn Ihr aber diese AUTOEXEC.BAT ausprobiert, und das hoffe ich doch sehr, so werdet Ihr feststellen, daß dies aber nicht der Fall ist!

Schuld daran ist die Zeile:

```
if not exist c:\system\*.* goto next
```

... die prüft, ob sich keine Datei im Verzeichnis C:\SYSTEM befindet, was nur unmittelbar nach einer Formatierung der Ramdisk C: so sein sollte. Wenn diese Prüfung positiv ausfällt, so wurde also die Ramdisk erfolgreich formatiert und das GW-Basic kann gestartet werden, was die Sprungmarke :next erledigt.

Und damit nicht bei jedem Warmstart des PoFos die Ramdisk C: formatiert wird, lasse ich von :next eine leere Datei @!@-@!@.gwi im Verzeichnis C:\SYSTEM anlegen:

```
rem>@!@-@!@.gwi
```

und diese mit der Zeile:

```
if exist c:\system\@!@-@!@.gwi goto next
```

prüfen. Nur wenn jemand von Euch so perfide Gedanken hegt, die Datei C:\SYSTEM\@!@-@!@.GWI mutwillig zu löschen und eine x-beliebige Datei nach C:\SYSTEM zu kopieren, wird mit dem nächsten Warmstart eine Formatierung der Ramdisk C: fällig!!!

Spielereien mit dem PROMPT

Als Prompt bezeichnet man die Eingabeaufforderung, die uns das DOS zeigt, wenn es für weitere Aktionen bereit ist. Oftmals hat es die folgenden, beispielsweisen Erscheinungsformen: „a>" oder „a:\>".

Im Handbuch stehen auch einige Beispiele, wie man diese Formen ändern kann. Doch leider werden dort zumeist nur die Standardvariationen wiedergegeben. Dabei kann man das Prompt ganz nach seinen Wünschen gestalten!

So wird das Prompt auf das blinkende Quadrat reduziert, wenn man den Befehl „PROMPThh" eingibt. Wer auf die sonst übliche Laufwerks- und Pfadangabe „A:\" verzichten kann, hat auch die Möglichkeit, daß Prompt mit einem individuellen Text zu belegen. Die Eingabe von „PROMPT PoFo" erzeugt so immer die Meldung „PoFo" als Prompt. Grafische Prompts kann man mittels der Alt- oder der Strg- Taste erreichen. Ändert man das Prompt durch „PROMPT Wie lautet die richtige Antwort?", so kann man damit dem Bediener eine Tastatureingabe vorgaukeln.

Weil in Batchdateien keine echten Tastatureingaben möglich sind, muß eine Batchdatei existieren, die als Dateinamen die Antwort enthält und die die Auswertung vornimmt. Für jede mögliche Antwort muß also eine eigene Batchdatei produziert werden.

Sobald die Frage beantwortet und ausgewertet wurde, sollte das Prompt natürlich seine vorherige Form wieder annehmen. Deshalb wird es zuerst in einer Variablen gerettet, dann wie gewünscht geändert und abschließend von der Antwortdatei wiederhergestellt. In der Batch-datei, in der also die Frage gestellt wird, müssen die folgenden Zeilen zu Beginn plaziert werden:

```
SET PRZ=%PROMPT%
PROMPT=Wie lautet die korrekte Antwort ?
```

Jede Batchdatei, die jeweils eine mögliche Antwort auswertet, ist um die folgende Zeile zu ergänzen:

```
PROMPT=%PRZ%
```

Weil viele Antwortmöglichkeiten auch dementsprechend viele Batchdateien zur Auswertung verlangen, sollte man die Fragen möglichst so gestalten, das mit J für JA oder N für NEIN geantwortet werden kann. Es werden dann also nur die Batchdateien J.BAT und N.BAT benötigt.

Gesetzt der Fall, es sollen mehrere Fragen vom Bediener per J/N beantwortet werden, so können J.BAT und N.BAT die Antworten fehlinterpretieren, da sie ursprünglich nur für eine Frage konzipiert wurden. Man stelle sich einmal vor, daß der Computer fragt, ob wir heute schönes Wetter hätten. Durch die Antwort mit J erhielten wir dann den Kommentar vom Computer, daß man doch besser im Garten arbeiten solle, als auf ihm, den Computer, rumzuhacken. Im nächsten Moment soll dann die Frage, ob wir mit dem Computer zufrieden sind, beantwortet werden. Die meisten von uns werden auch diese Frage mit J beantworten und bekommen erneut den Kommentar geliefert, daß man doch besser im Garten arbeiten solle. Eine Reaktion wie „Vielen Dank, das ehrt mich sehr!" wäre hier nicht nur passender, sondern auch logischer.

Aber auch diese Unlogik ist mit ein paar Zeilen behebbar! So definiert man in jeder Frage-Batchdatei eine Variable und baut in J.BAT und N.BAT entsprechende Prüfroutinen ein, mit denen dann die richtigen Kommentare geliefert werden. Die definierte Variable sollte immer denselben Namen haben und nur von jeder Frage- Batchdatei mit neuem Inhalt definiert werden. Nennen wir diese Variable FRG für Frage und geben wir ihr den Inhalt „1", wenn die Frage nach dem Wetter beantwortet werden soll. Die Frage-Batchdatei, die ich WETTER.BAT nennen möchte, sähe dann so aus:

```
@echo off
set frg=1
set prz=%prompt%
prompt=Ist heute schönes Wetter (J/N) ?
```

Die Frage-Batchdatei nach der Zufriedenheit, ich nenne sie hier COMPUTER.BAT, definiert die Variable FRG mit „2" und sieht dann etwa wie folgt aus:

```
@echo off
set frg=2
set prz=%prompt%
prompt=Bist Du mit mir, dem Computer, zufrieden (J/N) ?
```

Und so sieht dann beispielsweise die Antwort-Batchdatei J.BAT aus:

```
@echo off
if %frg%==1 goto eins
if %frg%==2 goto zwei
goto end
:eins
echo Du solltest besser im Garten arbeiten!
goto end
:zwei
echo Vielen Dank, das ehrt mich sehr!
:end
prompt=%prz%
```

Durch die IF-Zeilen wird die Variable FRG überprüft und zu der Routine weitergeleitet, die dann den richtigen Kommentar ausgibt. Bei entsprechender Weiterprogrammierung kann man damit aus seinem Computer glatt den privaten Psychiater oder Lebensberater machen!!!

Echten Sinn macht diese Frage-Antwort-Technik aber erst in Installationsroutinen oder Menüsystemen. Die hier vorgestellte Technik ist ausführlich auf dem Atari Portfolio getestet worden und kann sich natürlich durch neuere DOS-Versionen erübrigen.

Die automatische Sicherung des Environments

Ein volles Environment, zu altdeutsch: Umgebungsspeicher, macht sich ja immer durch die Meldung „Umgebungsspeicher voll!" bemerkbar und ist sehr ärgerlich, weil keine Variablen mehr gesetzt werden können. Startet man nach dieser Meldung noch weitere Batchdateien, die Variablen anlegen wollen, so werden diese Batchdateien nicht mehr korrekt funktionieren und womöglich wichtige Dateien löschen. Also heißt es entweder Variablen von Hand löschen oder DELENV.BAT einsetzen, damit der Ablauf von Batchdateien nicht gefährdet ist. Doch was ist, wenn man Variablen löscht, die später wieder benötigt werden? Um später nicht wie der Ochs vorm Berg zu stehen, sollte man vor dem löschen der Variablen diese lieber sichern, damit sie später wiederhergestellt werden können. Einen automatisierten Lösungsansatz dafür stellt SAVENV.BAT dar:

```
@echo off
if %1.==. c:\batch\savenv.bat §§
if not %§§%.==. set §§=
if not %1.==§§. goto nx
echo @c:\batch\savenv.bat>c:\ batch\savenv2.bat
echo @echo off>c:\batch\rstenv.bat
echo PATH=%path%>>c:\batch\rstenv.bat
path=-
goto sv
:nx
if not %1.==. shift
if not %1.==. path=%1
if not %2.==. path=%path% %2
if not %3.==. path=%path% %3
if not %4.==. path=%path% %4
if not %5.==. path=%path% %5
if not %6.==. path=%path% %6
if not %7.==. path=%path% %7
if not %8.==. path=%path% %8
if not %9.==. path=%path% %9
if %0==PATH goto ex
if %1.==. goto ex
echo set %0=%path%>>c:\batch\rstenv.bat
set %0=
:sv
if exist c:\batch\savenv2.dat copy c:\batch\savenv2.dat
   c:\batch\*.b*>nul
if exist c:\batch\savenv2.bat set>>c:\batch\savenv2.bat
if not %0.==§§. if exist
   c:\batch\savenv2.batc:\batch\savenv2.bat
:ex
echo Ist RSTENV.BAT so okay?
echo.
if exist c:\batch\rstenv.bat type c:\batch\rstenv.bat
if exist c:\batch\savenv2.bat del c:\batch\savenv2.bat
if exist c:\batch\rstenv.bat c:\batch\rstenv.bat
```

Diese Batchdatei sichert nach Möglichkeit das gesamte Environment in einer ausführbaren Batchdatei namens RSTENV.BAT. Dazu notwendig ist allerdings auch die Datei SAVENV2.DAT die durch folgende Eingaben zu erstellen ist:

```
copy con c:\batch\savenv2.dat<ENTER>
@echo off<ENTER>
c:\batch\savenv.bat<SPACE><STRG>+<Z><ENTER>
```

Dabei steht <ENTER> für die Enter- bzw. Return-Taste, <SPACE> für ein Leerzeichen und <STRG>+<Z> soll das Dateiendezeichen sein welches man erhält, wenn man die Taste STRG drückt, festhält und dann die Taste Z drückt. Anstelle von <STRG>+<Z> reicht aber auch ein Druck auf <F6> aus! Die Laufwerks- und Verzeichnisangaben müssen Sie gegebenenfalls Ihrem System entsprechend anpassen. Aus folgenden Gründen ist SAVENV.BAT nur bedingt einsetzbar:

1. Es kann das Environment nur dann gesichert werden, wenn die Variablennamen aus einem Wort bestehen. Variablen wie TEST;VAR1=X oder TEST,VAR1=H führen zu Endlosläufen, weil Semikolon, Kommata, Gleichheits- und Leerzeichen im DOS als Trennungszeichen fungieren. Diese Trennungszeichen dürfen also nicht in Variablennamen vorkommen, wenn SAVENV.BAT korrekt sichern soll.
2. In den Variableninhalten dürfen maximal neun Worte durch Trennungszeichen getrennt, enthalten sein. Bei der Variable TEST=1 2 3 4 5 6 7 8 9 0 würde die Null nicht mehr mitgesichert.
3. Ist eine Variable ohne Definition im Environment, d.h. ohne das übliche Gleichheitszeichen und erkennbaren Inhalt, so wird nur bis zu dieser Variable gesichert. Sie und die nachfolgenden Variablen müssen aus technischen Gründen unberücksichtigt bleiben.

Ansonsten sollte SAVENV.BAT korrekt funktionieren und das Environment in RSTENV.BAT startfähig ablegen. Weil die Sicherung über die Variable PATH erfolgt, werden die Variableninhalte automatisch in Großbuchstaben umgesetzt, sofern es sich dabei um Buchstaben handelt und die Inhalte nicht bereits in Großbuchstaben geschrieben wurden. Dies gilt allerdings auch nur bis zum Auftritt einer unlöschbaren Variable, wie im dritten Ausnahmegrund beschrieben.

Weil während der Sicherung die bestehenden Variablen nach Möglichkeit gelöscht werden müssen, damit die automatische Sicherung funktioniert, wird RSTENV.BAT abschließend gestartet, um das Environment wiederherzustellen. Folglich werden dann alle, bzw. die meisten, Variableninhalte bereits in Großbuchstaben umgewandelt sein. Sollten sich in den Variableninhalten andere Trennungszeichen als das Leerzeichen befinden, so werden sie durch das Leerzeichen ersetzt. Die Beibehaltung der anderen Trennungszeichen ist aus DOS-internen Gründen leider nicht möglich.

Zum Schluß wird dann der Inhalt von RSTENV.BAT zur Sicherheit angezeigt, damit

man diesen Inhalt mit dem noch aktuellen Environment vergleichen kann. Der SET Befehl wirkt da ja bekanntlich Wunder.

Eine halbautomatische Lösung, bei der die aus einem Wort bestehenden Variablennamen als Parameter an die Batchdatei übergeben werden, ist zwar sicherer, aber leider auch nicht gerade sehr komfortabel.

Aber wie gesagt: Dies ist nur ein Lösungsansatz, der in Zukunft weiterverfolgt und verbessert werden soll!

Umgebungsspeicher löschen

Als Umgebungsspeicher, zu neudeutsch: Environment, bezeichnet man den Speicherbereich vom DOS, der die Variablen enthält. Diesen Speicherbereich kann man ja bekanntlich mit dem Befehl SET verändern oder anzeigen lassen:

```
COMSPEC=C:\COMMAND.COM
PROMPT=$p$g
PATH=C:\DOS;C:\BATCH;C:\WINDOWS
TEMP=C:\temp
OS=MS-DOS
VER=6.22
BDV=C:
BDY=\BATCH\
EV1=123456789012345678901
EV2=1234567890123456789012
EV3
```

Bei vollautomatisch ablaufenden Batchdateien kann das Löschen des Umgebungsspeichers sehr nützlich sein, damit wieder Platz für neue Variablen geschaffen wird. Und genau dieses Löschen bewerkstelligt DELENV.BAT bis zu einem gewissen Grad, sogar ohne die Variablennamen zu kennen:

```
01: @echo off
02: if %0==delenv if %1.==.:.. %0
03: if %2.==. if exist c:\batch\delenv2.bat
   c:\batch\delenv.bat %1 .:.
04: if %1.==. c:\batch\delenv.bat . .
05: set path=%PATH%
06: if %PATH%.==. set path=---
07: set prompt=%PROMPT%
08: set comspec=%COMSPEC%
09: set temp=%TEMP%
10: if %.:.%.==.:.. set .:.=
11: if not %1.==PATH. set %1=
12: echo @c:\batch\delenv.bat .:. .:.>c:\batch\delenv2.bat
13: if exist c:\batch\delenv2.dat copy c:\batch\delenv2.dat
   c:\batch\*.b*>nul
14: if exist c:\batch\delenv2.bat set>>c:\batch\delenv2.bat
15: if not %1.==PATH. if not %2.==.:..
   c:\batch\delenv2.bat
16: if not %1.==.:.. if %2.==.:.. echo Ich konnte die
   Variable %1 und die ihr    folgenden Variablen nicht
   löschen!
17: if not exist c:\batch\delenv2.bat echo Ich konnte den
   Umgebungsspeicher    nicht löschen, weil
   DELENV2.BAT fehlt!
18: if %1.==.:.. if %2.==.:.. echo Kein löschen möglich, da
   DELENV2.DAT nicht    oder fehlerhaft erstellt wurde!
19: if %PATH%==--- echo Die Pfadangabe muß neu
   eingegeben werden!
20: if %PATH%==--- set path=
21: if exist c:\batch\delenv2.bat del c:\batch\delenv2.bat
```

Die Laufwerks- und Verzeichnisnamen müssen sowohl in DELENV.BAT, als auch in der selbst zu erstellenden DELENV2.DAT den eigenen Verhältnissen entsprechend angepaßt werden. Und mit diesen Eingaben erstellt man DELENV2.DAT:

```
copy con c:\batch\delenv2.dat<ENTER>
@echo off<ENTER>
c:\batch\delenv.bat<SPACE><STRG>+<Z><ENTER>
```

Dabei steht <ENTER> für die Enter- bzw. Return-Taste, <SPACE> für ein Leerzeichen und <STRG>+<Z> soll das Dateiendezeichen sein welches man erhält, wenn man die Taste STRG drückt, festhält und dann die Taste Z drückt. Anstelle von <STRG>+<Z> reicht aber auch ein Druck auf <F6> aus!

Das Leerzeichen hinter c:\batch\delenv.bat ist ungemein wichtig und darf auf gar keinen Fall vergessen werden!!! Wird dieses Leerzeichen nämlich vergessen, so muß DELENV.BAT vorzeitig abbrechen, ohne eine Variable gelöscht zu haben!

Hier die Funktionsweise von DELENV.BAT in groben Zügen:

Weil DELENV.BAT die einzelnen Variablen aus dem Environment nicht kennt und auch nicht kennen soll, müssen die einzelnen Variablen als Parameter, beim Start von DELENV.BAT angegeben werden. Aus diesem Grunde wird beim Start von DELENV.BAT die Datei DELENV2.BAT gefertigt, die durch Umleitung des Befehles SET, ein Abbild des Environments aufnehmen muß:

```
@echo off
c:\batch\delenv.bat OS=MS-DOS
VER=6.22
```
...usw.

Sie startet nach dieser Aufnahme DELENV.BAT mit den Angaben OS und MS-DOS als ersten und zweiten Parameter neu. Zeile elf von DELENV.BAT kommt nun zum Zuge und löscht die als Parameter %1 übergebene Variable OS. Zwischenzeitlich wurde vom nun mehr verkleinerten Environment ein neues Abbild in die erneut gefertigte DELENV2.BAT gepackt:

```
@echo off
c:\batch\delenv.bat VER=6.22
BDV=C:
...usw.
```

Diese neue DELENV2.BAT startet also wieder DELENV.BAT und sorgt damit für die Löschung der Variable VER. Theoretisch würde sich dieses rotierende System so lange fortsetzen, bis keine Variable mehr existiert!

Aber dem aufmerksamen Beobachter wird nicht entgangen sein, das die erste zu löschende Variable eigentlich COMSPEC hätte sein müssen, wenn man das Abbild des Environments betrachtet, welches eingangs gezeigt wurde.

Weil es natürlich nicht sehr klug ist das gesamte Environment zu löschen, schließlich sind Variablen wie PATH, PROMPT, COMSPEC und TEMP für DOS und uns nützlich oder gar notwendig, werden diese Variablen vorher "gesichert". Sie werden durch die Zeilen fünf bis neun mit Ihrem eigenen Inhalt definiert und rutschen dadurch ans Ende des Umgebungsspeichers:

```
OS=MS-DOS
VER=6.22
BDV=C:
BDY=\BATCH\
EV1=123456789012345678901
EV2=1234567890123456789012
EV3
PATH=C:\DOS;C:\BATCH;C:\WINDOWS
PROMPT=$p$g
COMSPEC=C:\COMMAND.COM
TEMP=C:\temp
```

Weil sie aber selbst dort nicht sicher sind, das rotierende System löscht fast alles, fungiert die Variable PATH als Grenze zwischen den zu löschenden und den zu erhaltenen Variablen. Sobald also DELENV.BAT die Variable PATH als ersten Parameter geliefert bekommt, bricht sie das Löschen ab. Somit bleiben bei diesem Beispiel die Variablen PATH, PROMPT, COMSPEC und TEMP erhalten.

Für mich, bzw. meine Batchdateien, sind die Variablen BDV und BDY ebenfalls sehr wichtig, weshalb ich die folgenden Zeilen noch zwischen die Zeilen neun und zehn einfüge, um auch BDV und BDY zu behalten:

```
set bdv=%BDV%
set bdy=%BDY%
```

Nach diesem Schema können Sie also alle wichtigen Variablen bewahren, wenn Sie diese nach Zeile neun von DELENV.BAT einfügen! Auf jeden Fall sollte keine Variable vor der Zeile fünf eingefügt werden, weil alle Variablen, die vor PATH stehen, gelöscht werden!
In bislang nur einem bekannten Fall muß das Löschen vorzeitig abgebrochen werden, da DELENV.BAT die Variable nicht löschen kann. Die Rede ist von EV3, die man im oben dargestellten Environment entdecken kann.

Sie ist wie alle Variablen durch den SET Befehl entstanden, konnte aber nicht korrekt angelegt werden, weil der Umgebungsspeicher während der Anlegung voll wurde. Trotz intensivster Bemühungen ist es mir bislang nicht gelungen, diese Variable wieder per SET Befehl zu löschen. Und genau deswegen kann sie anscheinend auch nicht per Batchdatei gelöscht werden: Sie bleibt solange im Environment, bis der Rechner neu gestartet wird!!!

Es spielt keine Rolle wie die Variable bei Ihnen heißt oder wie lang ihr Name ist, sobald sich eine Variable ohne Gleichheitszeichen im Environment befindet, kann diese nach derzeitigem Kenntnisstand nicht vom DOS gelöscht werden.
Somit muß das rotierende System von DELENV.BAT fehlschlagen, wenn die fehlerhafte Variable an die erste Stelle des Environments hochgerückt ist. In Mitleidenschaft sind all die Variablen gezogen, die sich dann zwischen der fehlerhaften und der PATH Variable befinden. Diese können dann auch nicht mehr gelöscht werden, weil das Environment immer wieder mit der störenden Variable beginnt.

Immerhin habe ich DELENV.BAT so dressiert, daß es dieses Problem erkennt und die Arbeit dann einstellt. Auch wenn die

Variable PATH, die Datei DELENV2.DAT, die Datei DELENV2.BAT nicht existiert, wird das bemerkt und entsprechend reagiert. Selbst einen unsinnigen Start von DELENV.BAT durch die Eingabe DELENV .:. .:. verzeiht sie großzügig!

Selbst wenn eine brandgefährliche Variable im Environment vorkommen sollte, ich denke da z.B. an FDISK, so wird dieser Variablenname nicht zum Start des gleichnamigen Programmes führen, welches ansonsten die gesamte Festplatte formatieren könnte!!!!

Weil Kontrolle seit jeher besser als Vertrauen ist, kann man ja dieses Risiko herausfordern und eine Variable namens FORMAT definieren, die als Inhalt mit den Werten gesetzt wird, die ansonsten zur Formatierung einer 1.44MB Diskette im Laufwerk A: nötig sind:

```
set format=a: /f:1440
```

Nun sollte man eine Diskette ins Laufwerk A: einlegen, DELENV.BAT starten und abwarten.

Sollte sich aber in Ihrem Environment eine Variable befinden, in deren Name sich ein Leerzeichen, oder ein ähnliches Trennungszeichen, befindet, so wird DELENV.BAT diese Variable nicht löschen können und das führt zu einer Endlosschleife. D.h. DELENV.BAT wird nicht mehr beendet!

Wer DELENV.BAT auch tatsächlich in einen vollautomatisch ablaufenden Batchdateienverbund nutzen möchte, kann dies erreichen, wenn er DELENV.BAT um die 22. Zeile ergänzt. Weil auch ich noch keine Patentlösung dafür habe, muß sich jeder Leser selbst so seine Gedanken dazu machen.

GW-Basic auf dem PoFo!!!

Nach dem also das GW-BASIC 3.22 auf meinem PoFo lief, störte ich mich noch an dem Bildschirmformat von 80 Zeichen und 25 Zeilen. Der PoFo hat ja nur ein Bildschirmformat von 40 Zeichen und 8 Zeilen, weshalb etliche Basicprogramme unübersichtlich wirken.

Versuchsweise begann ich also das Programm GWBASIC.EXE zu patchen, hörte aber alsbald wieder damit auf, weil die Werte 80 und 25, die es ja in 40 und 8 abzuändern galt, etwa 3000 mal vorkamen. Das war mir dann doch zuviel Arbeit, so daß ich mich eines einfacheren Lösungsweges besann und der Firma Microsoft mein Problem schilderte. Kurz darauf erhielt ich von Microsoft, es war nicht Bill Gates persönlich, den gewünschten Rückruf: „Weil die Entwicklung schon solange zurück liegt, ist es leider nicht möglich, die gewünschte Arbeit vorzunehmen."

Also blieb es doch wieder an mir haften und ich machte mich an diese langwierige Prozedur. Mit mehreren Basicprogrammen und Batchdateien, die sich gegenseitig aufriefen und die Ergebnisse festhielten, ist es mir dann doch schon nach einer Stunde gelungen, auf einem 386er PC mit 25 MHz wohlgemerkt, die Zeilenzahl von 25 auf 8 abzuändern. Die Zeichenzahl von 80 mußte ich kurioserweise nicht mehr ändern, da das GWBASIC 3.22 dann problemlos auf dem PoFo funktionierte. Dazu muß der Bildschirmmodus „MODUS" im Systemmenü, auf Normal eingestellt werden. Die Funktionstasten werden dann allerdings nicht mehr komplett angezeigt, sondern nur die ersten fünf. Um die restlichen fünf sichtbar zu machen, muß man den Befehl KEY LIST verwenden.

Zwar ist die unterste, also die achte Zeile, nicht im Eingabemodus zu erreichen, aber sie kann von den Programmen problemlos mitbenutzt werden. Bevor Ihr nun beginnt Euer GWBASIC 3.22 mit dem unten stehenden Basicprogramm zu patchen, solltet Ihr sicherheitshalber eine Kopie der Datei GWBASIC.EXE anfertigen.

Sollte nämlich die gepatchte Version nicht korrekt auf dem PoFo funktionieren, so habt Ihr immer noch die Originalversion für den PC!!!

```
 10 OPEN“gwbasic.exe“ AS #1 LEN=1
 20 FIELD #1,1 AS B$
 30 RESTORE 260
 40 FOR A=1 TO 5
 50 READ B,C
 60 LSET B$=CHR$(C)
 70 PUT #1,B
 80 NEXT
 90 FOR A=73923 TO 73985
100 READ A$
110 LSET B$=CHR$(VAL(„&H“+A$))
120 PUT #1,A
130 NEXT
140 FOR A=22652 TO 22655
150 READ A$
160 LSET B$=CHR$(VAL(„&H“+A$))
170 PUT #1,A
180 NEXT
190 RESTORE 320
200 FOR A=77473 TO 77632
210 READ A$
220 LSET B$=CHR$(VAL(„&H“+A$))
230 PUT #1,A
240 NEXT
250 CLOSE
260 DATA 21942,0,21943,30,39445,0,39446,7,39471,8
270 DATA 47,57,2D,42,41,53,49,43,20,33,2E,32,32,
   20,2D,20
280 DATA 50,6F,46,6F,20,56,65,
   72,73,69,6F,6E,0D,0A,28,43
290 DATA 29,20,43,6F,70,79,72,69,
   67,68,74,20,4D,69,63,72
300 DATA 6F,73,6F,66,74,20,31,39
   ,38,33,2D,31,39,39,33,0D
310 DATA 0A,7F,7F,7F,7F,80,00,00,C9,00,28,00,
   00,C9,00,77
320 DATA 43,4C,53,27,0D,00,00,00,00,00,00,
   00,00,00,00,00
330 DATA 41,55,54,4F,20,00,00,00,00,00,
   00,00,00,00,00,00
340 DATA 47,4F,54,4F,20,00,00,00,00,00,00,00 ,00,00
   ,00,00
350 DATA 4C,49,53,54,20,00,00,00,00,00,
   00,00,00,00,00,00
360 DATA 52,55,4E,0D,00,00,00,00,00,
   00,00,00,00,00,00,00
370 DATA 46,49,4C,45,53,00,00,00,00,00
   ,00,00,00,00,00,00
380 DATA 4C,4F,41,44,22,00,00,00,00
   ,00,00,00,00,00,00,00
390 DATA 53,41,56,45,22,00,00,00,
   00,00,00,00,00,00,00,00
400 DATA 4F,50,45,4E,22,00,00,00,00,00,
   00,00,00,00,00,00
410 DATA 53,59,53,54,45,4D,0D,00,
   00,00,00,00,00,00,00
```

Dieses Programm habe ich mehrfach auf meinem PC angewandt und immer eine funktionsfähige Basicversion für den PoFo bekommen. Sollte es also nicht bei Euch funktionieren, so kann ich Euch leider nicht mit der angepaßten Version aushelfen, da sie dem Copyright von Microsoft unterliegt.

Weitere Erkenntnisse zum GWBASIC 3.22:

Es ist nicht alles Gold was glänzt und daher gibt es einige Befehle des GWBASIC, die nicht auf dem PoFo funktionieren:

Circle, Draw, Line, Get (x1,y1)-(x2,y2), Put (x1,y1)-(x2,y2), Color, Pset, Preset, Paint - und alle weiteren Grafikbefehle Enter, On Timer Gosub, On Key Gosub, On String(x) Gosub - und ähnliche Interruptgesteuerte Befehle

Die Notbremse STRG+PAUSE, mit denen man gewaltsam Basicprogramme auf dem PC abbrechen kann, ist auf dem PoFo dank der fehlenden Pause-Taste, nicht möglich.

Darum ist es notwendig, in jedem Basicprogramm für den PoFo, eine Abbruchfunktion selbst zu programmieren. Ich erledige das über die folgende Befehlszeile:

```
xxx IF INKEY$=CHR$(27) THEN CLOSE:CLS:END
```

Diese Befehlszeile muß immer in dem Hauptteil des Programmes landen, damit man das Programm dann auch mit der ESC-Taste abbrechen kann. Setzt man diese Zeile in ein Unterprogramm, die drei xxx stehen für die Zeilennummer, so kann das Programm nicht jederzeit abgebrochen werden, was ja nicht gerade wünschenswert ist. Ich habe bewußt die ESC-Taste ausgewählt, weil ja damit auch jede der eingebauten Applikationen, verlassen wird. Außerdem wird sie so gut wie nie von anderen Basicprogrammen benötigt!

Relative Dateien stellen auch ein Problem dar: Sie dürfen eine Maximalgröße nicht überschreiten, die sich auf die augenblickliche Sektorengröße bezieht. Entweder ist also nach 128, nach 256 oder nach 512 Bytes Schluß. Sequentielle Dateien sind davon nicht betroffen. Allerdings gilt diese Einschränkung nur für die PoFos mit den BIOS-Versionen 1.030, 1.052 und 1.072!

Wer weitere Probleme mit GWBASIC entdeckt oder mit dieser Ausführung nicht zurecht kommt, sollte darüber berichten, damit ich darauf eingehen kann.

"GROß und artig"

Flexible Batchdateien sind oftmals auf Parameter (%1 usw.) angewiesen, die der

User beim Start der Batchdatei mitangeben muß. Durch diese Parameter, das können Zahlen, Zeichen, Wörter oder gar ganze Sätze sein, wird dann der Ablauf der gestarteten Batchdatei beeinflußt. Diese Parameter auf Richtigkeit zu überprüfen, ist eine nicht leichte Aufgabe für jede Batchdatei, ganz besonders wenn es sich um Wörter handelt.

Denn es ist schon ein großer Unterschied, ob ich das Wort TEST in Großbuchstaben, als test in Kleinbuchstaben oder gemischt als Test schreibe.

Um dieses Problem der Groß-, Kleinschreibung und auch der gemischten Schreibweise zu beheben, behalf ich mir zuerst mit z.B. folgenden Batchzeilen:

```
if %1==test echo Parameter korrekt
if %1==Test echo Parameter korrekt
if %1==TEST echo Parameter korrekt
```

um die häufigsten Schreibweisen des benötigten Wortes TEST zu prüfen. Als ich dann mit der FOR-IN-DO Schleife vertraut war, konnte ich diese Prüfung wie folgt verkürzen:

```
for %%a in (test Test TEST) do if %%a==%1 echo
  Parameter korrekt
```

Doch leider werden die beiden Verfahren nicht mit den Schreibweisen TeST, TEst, teST, TesT usw. fertig, die durch überhastetes Eintippen entstehen können. Bei kurzen Worten wie TEST kann man zwar alle Möglichkeiten im voraus einprogrammieren, aber bei Worten wie DISKETTE oder FESTPLATTE wäre diese intelligente Programmierung zu umfangreich und ineffektiv.

So kam ich nach eingehenden Beobachtungen und Versuchen zu dieser Technik:

```
set oldpath=%PATH%
path=%1
set xyz=%PATH%
path=%OLDPATH%
if %XYZ%==TEST echo Parameter korrekt
```

Ihr ganzes Geheimnis liegt in der Zeile PATH=%1, die für die Umsetzung des Parameters in Großbuchstaben zuständig ist.

Damit der PC auch später noch die Programme wiederfindet, wird der ursprüngliche Pfad vorher gesichert und nachher wieder restauriert. Zum Schluß findet man den Parameter %1 komplett in Großbuchstaben geschrieben in der Variable XYZ und braucht zum prüfen nur noch eine Abfrage.

Bis ich zum ersten Mal die Fehlermeldung "Umgebungsspeicher voll" vom DOS zu Gesicht bekam, konnte ich mich auch auf diese Technik verlassen. Weil diese Fehlermeldung besagt, daß man keine weiteren Variablen anlegen kann, bevor man nicht einige andere Variablen gelöscht hat, kann es passieren das die Variablen OLDPATH und XYZ nicht oder fehlerhaft angelegt werden.

Und wenn das passiert, dann kann die Pfadangabe auch nicht mehr korrekt restauriert werden, was zu einer ganzen Menge Ärger führt: Schließlich wird der PC einen Großteil der gespeicherten Programme nicht mehr starten können!

Als Alheilmittel wird dagegen der Aufruf eines zweiten Kommandointerpreters empfohlen, mit gleichzeitigem Neustart der o.g. Technik in Form einer Batchdatei. Sobald aber der zweite Kommandointerpreter beendet wird, ist auch die Variable XYZ und ihre Definition wieder verschwunden. Ein nachfolgender Batch wird, der auf XYZ angewiesen ist, die Variable vergebens suchen und sehr wahrscheinlich fehlerhaft enden. Das kann vom Abbruch mit Fehlermeldung über den endlosen Ablauf der Routine bis zum Verlust von Dateien reichen! So kam ich nach umfangreichem Brainstorming und etlicher Tests auf eine sicherere Technik der Umwandlung, die hauptsächlich durch Parameter getragen wird. Nur die Variable PATH wird auch weiterhin verändert, da ohne sie bisher keine Umwandlung von Klein- in Großbuchstaben möglich zu sein scheint.

UPCASE.BAT:
```
@echo off
if not %0==%BDV%%BDY%upcase.bat goto off
if %1.==. goto off
if not %1.==.. if %2.==. goto off
if not exist %2.bat if not exist %BDV%%BDY%nul goto off
echo @echo off>%BDV%%BDY%pfad.bat
if not exist %BDV%%BDY%pfad.bat goto off
echo path=%PATH%>>%BDV%%BDY%pfad.bat
path=%1
shift
if %1.==.. shift
if exist %1.bat call %1 %PATH% %2 %3 %4 %5 %6 %7
   %8 %9
set path=
:loop1
shift
if %1.==. goto off
if %1.==:. goto loop2
if not %1.==:. goto loop1
:loop2
shift
if not %1.==. if not exist %1\nul goto loop2
if not %1.==. if %PATH%.==. set path=%1
if not %1.==. if not %PATH%.==. if not %PATH%.==%1.
   set path=%PATH%;%1
if not %1.==. goto loop2
:off
if not exist %BDV%%BDY%pfad.bat if not exist %3.bat
   echo Die Umwandlung ist unmöglich, weil der PFAD
   nicht rekonstruiert werden kann!
if %PATH%.==. if exist %BDV%%BDY%pfad.bat call
   %BDV%%BDY%pfad.bat
if %PATH%.==. echo ACHTUNG: Der PFAD wurde
   gelöscht! Bitte neu eingeben.
```

Sie wird von der Batchdatei gestartet, die den Parameter %1 in Großschrift umgewandelt haben will. Ein erstes Beispiel dafür ist UPTEST1.BAT:

UPTEST1.BAT:
```
@echo off
if not %0==%BDV%%BDY%uptest1
   %BDV%%BDY%upcase.bat %1 . %BDV%%BDY%%0 :
   %PATH%
echo.
if %1.==BATCHING. echo %1 macht Spaß!!!
if not %1.==BATCHING. goto err
goto off
:err
echo Probieren Sie doch einmal das Wort BaTcHiNg
   aus...
:off
echo.
```

Um dem Beispiel gerecht zu werden, muß UPTEST1.BAT mit dem Wort Batching als ersten Parameter gestartet werden. Und das können z.B. folgende Eingaben sein:

```
UPTEST1 BATCHing
UPTEST1 BATchING
UPTEST1 BaTcHiNg
```

Ein Beispiel für die Parameter beim Start von UPCASE.BAT, durch Zeile zwei der UPTEST1.BAT veranlaßt, könnte sein:

```
BaTcHiNg . c:\batch\uptest1 : C:\DOS
C:\BATCH C:\WINDOWS
```

Und das macht UPCASE.BAT mit diesen Parametern:

```
BaTcHiNg - umwandeln in BATCHING
   - löschen durch SHIFT-Befehl
c:\batch\uptest1 - Batchdatei starten, da sie das Wort
   BATCHING haben will:
   - löschen durch SHIFT-Befehl
C:\DOS - Pfadangabe wiederherstellen: PATH=C:\DOS;
C:\BATCH - Pfadangabe wiederherstellen:
   PATH=C:\DOS;C:\BATCH;
C:\WINDOWS - Pfadangabe wiederherstellen:
   PATH=C:\DOS;C:\BATCH;C:\WINDOWS
```

Bevor UPCASE.BAT aber die Pfadangabe restauriert, ruft es die ursprüngliche Batchdatei, hier: UPTEST1.BAT, inklusive neun Parameter auf. UPTEST1.BAT kann nun den Parameter %1, das Wort BATCHING, prüfen und entsprechend reagieren. Am Ende kommt wieder UPCASE.BAT zum Zuge, damit die Pfadangabe restauriert wird.

Zwar werden beim besagten Restart von UPTEST1.BAT, in Zeile zwölf der UPCASE.BAT, noch die Parameter %2 bis %9 mitübergeben, aber da UPTEST1.BAT sie nicht benötigt, bleiben sie unbeachtet.

Um die Übergabe und Verwendung dieser acht Parameter zu demonstrieren, habe ich die Batchdatei ZU.BAT aus der Ausgabe 12/95, Seite 212-214, umgeschrieben und für UPCASE.BAT aufbereitet:

ZU.BAT:
```
 1: @echo off
 2: if not %0==%BDV%%BDY%zu
   %BDV%%BDY%upcase.bat %1 . %BDV%%BDY%%0
   %2 : %PATH%
 3: if not %2.==. if exist %2\nul echo if not %%1.==. if
   %%1.==%1. cd %2>>%BDV%%BDY%zu.bat
 4: rem Hier landen die gewünschten und geprüften
   Verzeichnisse:
```

Die Syntax von ZU.BAT lautet auch weiterhin: ZU Kurzname Laufwerk+Verzeichnis. So wird durch den einmaligen Start mit der Eingabe "ZU ute

C:\TEXTE\PRIVAT\UTE" ein schneller Verzeichniswechsel in C:\TEXTE\PRIVAT\UTE ermöglicht, wenn man bei späteren Starts nur "ZU ute" oder auch "ZU UTE" eingibt.

Beim ersten Start sind also zwei Parameter, %1 und %2, nötig, damit man später ZU.BAT nur noch mit einem Parameter, %1, aufzurufen braucht und in den Vorteil des einfachen Verzeichniswechsels kommt.

Um die Funktionsweise dieser Parameterübergabe zu verstehen, stellen wir einmal die Zeilen mit der Nummer zwei aus UPTEST1.BAT und ZU.BAT gegenüber:

```
if not %0==%BDV%%BDY%uptest1
   %BDV%%BDY%upcase.bat %1 . %BDV%%BDY%%0
   : %PATH%
if not %0==%BDV%%BDY%zu
   %BDV%%BDY%upcase.bat %1 . %BDV%%BDY%%0
   %2 : %PATH%
```

Ins Auge springen sollte die Parameterangabe %2 in der Zeile aus ZU.BAT, da sie in der Zeile von UPTEST1.BAT nicht zu finden ist. Demnach wird also C:\TEXTE\PRIVAT\UTE aus unserem Beispiel zu ZU.BAT, als weiterer Parameter an UPCASE.BAT gegeben, wenn dieser als %2 hinter %BDV%%BDY%%0 eingefügt wird. Für UPCASE.BAT ist dieser Parameter zwar nicht wichtig, aber sie gibt ihn doch beim Restart von ZU.BAT zurück, weil dieser Parameter ansonsten verloren wäre.
Und so kann man alle neun Parameter aus der eigenen Batchdatei zur Sicherung an UPCASE.BAT geben:

```
if not %0==%BDV%%BDY%batchdateiname
   %BDV%%BDY%upcase.bat %1 . %BDV%%BDY%%0
   %2 %3 %4 %5 %6 %7 %8 %9 : %PATH%
```

Welche Parameter man dabei verwendet oder in welcher Reihenfolge man diese schreibt, das bleibt einem selbst überlassen. Hauptsache ist, daß der Parameter, der in Großbuchstaben umgewandelt werden soll, hinter %BDV%%BDY%upcase.bat eingetragen wird!

Der Doppelpunkt vor %PATH% soll durch seine Anwesenheit der UPCASE.BAT klar machen, das die auf ihn folgenden Parameter den ursprünglichen Inhalt der Pfadangabe darstellen. Wenn die also auf den Doppelpunkt folgenden Parameter auch tatsächlich Verzeichnisse darstellen, UPCASE.BAT prüft auch das, so landen diese Parameter wieder in der Variable PATH.

Wird beim Aufruf der Batchdatei, die die Umwandlung eines Parameters in Großbuchstaben wünscht, vergessen ein Parameter anzugeben, so schützt der Punkt vor %BDV%%BDY%%0 vor Fehlfunktion von UPCASE.BAT. Vorausgesetzt, man hat diesen Punkt selber nicht beim schreiben der Zeile zwei vergessen!

Wer UPCASE.BAT nutzen möchte, muß mit diesen Zeilen seine Batchdatei beginnen, damit UPCASE.BAT benutzt werden kann:

```
@echo off
if not %0==%BDV%%BDY%batchdateiname
   %BDV%%BDY%upcase.bat %1 . %BDV%%BDY%%0
   %2 %3 %4 %5 %6 %7 %8 %9 : %PATH%
```

Anstelle von Batchdateiname muß man den bis zu achtstelligen Namen dieser Batchdatei eintragen. Wichtig dabei ist, das alle Dateinamen in Kleinbuchstaben geschrieben werden! Diese Zeile zwei ist in Sachen Parameterreihenfolge und -anzahl als Standardbeispiel zu verstehen und den eigenen Wünschen gemäß anzupassen. Wer aber meine Erklärungen dazu nicht verstanden haben sollte, sollte dieses Beispiel in allen Batchdateien verwenden, die UPCASE.BAT benutzen sollen!

Wer es mag, kann noch zwischen den beiden Zeilen eine Zeile einfügen, die das Vorhandensein des ersten Parameters prüft und im negativen Falle, z.B. zu einer Sprungmarke führt, die den Bediener auf den fehlenden Parameter hinweist und die Batchdatei anschließend beendet.

Ausgefuchste Freaks, d.h. sehr gute Batchprogrammierer, können UPCASE.BAT auch durch Ergänzung um eine 29. Zeile, so gestalten, daß UPCASE.BAT vollautomatisch, in einem Verbund von Batchdateien, des Nachts arbeitet!

Es ist nicht notwendig UPCASE.BAT zu verstehen, aber man muß wissen das UPCASE.BAT immer nur einen Parameter und auch nur den ersten übergebenen Parameter, pro Durchlauf, in Großbuchstaben umwandelt!

Für den Fall der Fälle, Verlust der Variable PATH, hat UPCASE.BAT auch einige Sicherheitsmaßnahmen eingebaut und bricht die Umwandlung möglichst vor dem Verlust

ab. Sollte dennoch einmal eine so unglückliche Konstellation zum Verlust der Variable PATH führen, so wird UPCASE.BAT dieses wenigstens melden und seine Tätigkeit einstellen.

Ungefährliche Testläufe kann man auch mit UPTEST2.BAT fahren und lernt durch aufmerksame Beobachtung, wie UPCASE.BAT arbeitet:

UPTEST2.BAT:

```
@echo off
if not %0==%BDV%%BDY%uptest2
   %BDV%%BDY%upcase.bat %1 . %BDV%%BDY%%0
   %2 %3 %4 %5 %6 %7 %8 %9 : %PATH%
cls
echo.
echo Die Parameternamen stehen nicht immer genau
   über dem Parameterinhalt!
echo.
echo %%1 %%2 %%3 %%4 %%5 %%6 %%7 %%8 %%9
echo %1 %2 %3 %4 %5 %6 %7 %8 %9
echo.
```

Den ersten Test sollte man durch die Eingabe von:
UPTEST2 Hallo! a b c d e f g h
starten und bei jedem weiteren Test einen Buchstaben weglassen.

KnowHow: Batching

Das kleine Einmaleins für DOS

Ein Computer der nicht rechnen kann, dürfte erst gar nicht Computer genannt werden. Dennoch schimpft man die PC's so, obwohl ihr Betriebssystem nicht in der Lage ist, Berechnungen durchzuführen. Dummesdos, oder wie hieß der Haushaltsreiniger noch einmal?

Aber DOS ist ja auch sehr flexibel und lernt immer wieder dazu. So konnte ich es per Batchdateien testweise zu kleinen Berechnungen überreden. Weil ich die Ergebnisse später in anderen Batchdateien verwenden will und DOS auch nur ganze und positive Zahlen verarbeiten kann, werden Negativergebnisse zu 0 und krumme Werte einfach abgerundet.

Den Beginn macht die Batchdatei RECHNE.BAT, mit der die Berechnung gestartet wird. Folgende Aufrufe sind z.B. denkbar:

```
RECHNE 0 + 8
RECHNE 7 - 9
RECHNE 5 * 2
RECHNE 1 : 4
RECHNE + 3
RECHNE 6 -
```

Anhand des Rechenzeichens wird dann die Batchdatei PLUS.BAT, MINUS.BAT, MAL.BAT oder GETEILT.BAT gestartet. Weil das allgemeinübliche Zeichen "/", um Zahlen zu teilen, unter DOS eine bestimmte Funktion inne hat, mußte ich auf das Zeichen ":" ausweichen. Kommen wir aber nun zu den nötigen Batchdateien, hier RECHNE.BAT:

```
@echo off
cls
set z1=%1
set op=%2
set z2=%3
set g=goto
rem Volle Syntax angegeben?
if not %3.==. %g% run
rem Syntax unvollständig, mit E soll
rem also weitergerechnet werden!
rem Ist E leer, so wird E zur 0:
if %e%.==. set e=0
rem Entspricht E einem Wert von 0 bis
rem 10 ? Wenn ja, so kann mit E
rem gerechnet werden.
if %e%==0 %g% w
if %e%==1 %g% w
if %e%==2 %g% w
if %e%==3 %g% w
if %e%==4 %g% w
if %e%==5 %g% w
if %e%==6 %g% w
if %e%==7 %g% w
if %e%==8 %g% w
if %e%==9 %g% w
if %e%==10 %g% w
rem E war kein Wert zwischen 0 und 10!
echo Mit dem Ergebnis %e% kann nicht
echo weitergerechnet werden!!!
rem Abbruch durch Sprung zu :off
%g% off
rem E liegt zwischen 0 und 10
:w
rem Ist %1 ein Rechenzeichen, so soll
rem E als erste Zahl gelten.
if %1==+ %g% 1
if %1==- %g% 1
if %1==* %g% 1
if %1==: %g% 1
rem %1 war kein Rechenzeichen, folg-
rem lich soll E als z2 gelten.
set z2=%e%
%g% run
:1
rem %1 war ein Rechenzeichen, deshalb
rem müssen die Werte neu definiert
rem werden.
```

```
set z2=%2
set op=%1
set z1=%e%
:run
rem Volle Syntax vorhanden, bzw.
rem hergestellt, die Berechnung kann
rem beginnen.
if %op%==+ plus %z1% %z2%
if %op%==- minus %z1% %z2%
if %op%==* mal %z1% %z2%
if %op%==: geteilt %z1% %z2%
:err
rem Syntax unbekannt oder fehlend:
echo Aufruf z.B. mit:
echo RECHNE 5 + 7, o. RECHNE 1 - 8, o.
echo RECHNE 2 * 0, o. RECHNE 9 : 3, o.
echo RECHNE 4 +  , o. RECHNE 6 -  usw.
echo Zahlen von 0 bis 10 sind erlaubt!
:off
```

Weil die eigentlichen Batchdateien, die die Berechnungen durchführen zu umfangreich sind, hier ein Ausschnitt aus PLUS.BAT:

```
@echo off
%g% %1%2
:00
set e=0
%g% o
:01
:10
set e=1
%g% o

....

:o
echo %e%
```

Beschäftigen wir uns nun zunächst mit RECHNE.BAT:

Die übergebenen Parameter werden in Variablen gespeichert, wie auch der Befehl GOTO, der somit in allen Batchdateien durch %g% ausgeführt wird.

Wird die volle Syntax eingehalten, also beide Zahlen und die Rechenart als Parameter übergeben, so ist %3 als Zahl definiert. DOS springt deshalb zur Marke RUN und fährt dort fort. Ist aber %3 nicht definiert, so geht DOS davon aus, das das in E gespeicherte Ergebnis, zur Neuberechnung herangezogen werden soll und arbeitet die nächste Zeile ab, in der die Variable E mit 0 definiert wird, sofern E nicht definiert war. Dieser Fall kann theoretisch nur dann auftreten, wenn man zum ersten mal etwas berechnen läßt und beim Aufruf eine Zahl vergißt, oder absichtlich wegläßt.

Wenn E zur Berechnung herangezogen werden soll, so muß DOS jetzt prüfen, ob E überhaupt einem der erlaubten Werte 0 bis 10 entspricht. Im positiven Falle, geht DOS zur Sprungmarke :w. Im negativen Falle wird eine entsprechende Meldung auf den Schirm gebracht und anschließend komplett abgebrochen.

Gelangt DOS zur Sprungmarke :w, so muß es nun abklären, ob das letzte Ergebnis als erste oder zweite Zahl in die Berechnung einfließen soll. Dafür prüft es den Parameter %1 ab, ob dieser als Rechenzeichen definiert wurde. Tritt dieser Fall ein, so verzweigt DOS nun zu der Sprungmarke :1, wo es die Variablen umdefiniert, das Ergebnis aus E einbindet und damit wieder die volle und korrekte Syntax erreicht. Soll aber das Ergebnis als zweite Zahl gelten, so muß nur die Variable Z2 mit dem Inhalt von E definiert werden, damit die gestellte Aufgabe der korrekten Syntax entspricht.

Letztendlich landet man aber bei der Sprungmarke :RUN, die die Variable OP prüft und die entsprechende Batchdatei aufruft, die die Berechnung durchführen soll. Ist OP z.B. als + gesetzt, so wird PLUS.BAT inkl. beider Zahlen aufgerufen.

Besprechen wir also nun die Datei PLUS.BAT, die beispielsweise die Zahlen 1 und 0 addieren soll:

Durch die Zeile %g% %1%2 wird aus den beiden Zahlen 1 und 0 eine Sprungmarke gebildet, nämlich :10, die durch den Befehl GOTO, der in der Variable %g% gespeichert ist, angesprungen wird. Die Variable E wird daraufhin mit dem Ergebnis 1 definiert. Anschließend wird mit %g% o zur Marke :o gesprungen. Sie gibt nun das Ergebnis mittels ECHO %E% auf dem Schirm raus. Theoretisch ist damit die Berechnung beendet, aber praktisch gesehen muß ich noch ein paar Erklärungen abgeben:

Bei etlichen Berechnungen entsteht oftmals dasselbe Ergebnis. Daher wird zwar jede Sprungmarke angesprungen, die sich aus den übergebenen Zahlen konstruiert, aber erst etliche Zeilen später, wird das Ergebnis durch SET E "errechnet". In Wirklichkeit wird ja nur das logische Ergebnis, welches ich vorprogrammiert habe, ermittelt und ausgegeben.

Hätte ich jeder Sprungmarke gleich die entsprechende SET E-Anweisung nachgestellt, so wäre der Speicherbedarf um ein vielfaches angestiegen! Zu dieser Erkenntnis, mit den oftmals gleichen Ergebnissen, kam ich aber auch erst relativ spät. Es bedurfte erst drei vorhergehenden Versionen, die von anfangs 30KB Programmlänge auf nun 5KB zusammenschrumpften!

Wie und wofür man die Ergebnisse weiterverarbeitet, überlasse ich nun Euch. Auf jeden Fall kann man damit viele Routinen gestalten, wenn man diese Dateien rekursiv nutzbar macht.

KnowHow: MS-DOS

Fehlverhalten des Befehles SET unter MS-DOS!

Unter MS-DOS gibt es manchmal ein Problem, wenn man eine Variable mit SET setzen mochte. So erhält man desofteren die Fehlermeldung das der "Umgebungsspeicher voll!" ist. Da eine Batchdatei, die eine Variable setzt, meist auf diese Variable angewiesen ist, kann das katastrophale Auswirkungen haben, wenn diese Variable eben nicht existiert (Datenverlust o.a. je nach Batchdatei)!!!

Normalerweise kann man dieses Problem wie folgt beheben, muß aber mit der o.g. Fehlermeldung leben, wenn das Environment (der Umgebungsspeicher) voll ist:

```
SET O=P
IF %O%.==. GOTO OFF
rem
rem Rest der Routine
rem
:off
```

Eine Batchdatei mit diesen Zeilen bewahrt den Anwender vor dem Datenverlust, weil sie bei nicht vorhandener Variable O rechtzeitig aussteigt!

Jede Batchdatei, die Variablen erzeugt, sollte also immer prüfen, ob die gerade gesetzte Variable auch existiert und gegebenenfalls rechtzeitig aussteigen.

Aber leider kann sich dieses Problem gerade unter MS-DOS noch schlimmer auswirken:

Bei mir ist es schon desofteren vorgekommen, das der Platz im Environment nicht mehr fur die ganze Variable ausreichte. So wollte ich z.B. eine Variable EV3 setzen, die mit einer Zahlenkolonne wie: 01234567890123456789012345 67 definiert werden sollte. Wenn ich Gluck hatte, dann wurde diese Variable wenigstens noch so gesetzt:

```
EV3=0123456788901234
```

Dann kann ich wenigstens noch die EV3 wieder löschen, da sie nicht vollstandig gesetzt wurde:

```
SET EV3=01234567890123456789012345 67
IF NOT %EV3%.==01234567890123456789012345 67.
  SET EV3=
IF %EV3%.==. GOTO OFF
```

Aber im krassesten Fall hatte ich dann das folgende Abbild vom Environment:

```
COMSPEC=C:\COMMAND.COM
PATH=C:\DOS;C:\BATCH
PROMPT=$P$G
TEMP=F:\
OS=MS-DOS
VER=6.22
BDV=C:
BDY=\BATCH\
LG=049
MSG=2
EV3
```

Im Environment ist zwar die Zeichenfolge EV3 zu sehen, aber kein Gleichheitszeichen und keine Definition mehr! Was ich auch versuchte, ich konnte diese Zeichenfolge EV3 nicht mehr mit dem SET-Befehl löschen!!! Nur über DEBUG gelang mir das löschen per Hand!

Die meisten Batchprogrammierer wurden nun mit den Achseln zucken und sagen:"Na und? Mit dieser Variablen-Leiche kann ich leben!".

Aber ich kann es nicht, da ich auch Batchdateien habe, die das Environment manipulieren (z.B. vollautomatisch die Reihenfolge der Variablen umstellen können)!!!!

Sobald also so eine Variablen-Leiche im Environment steckt, müssen meine Routinen die Waffen strecken und vorzeitig abbrechen - ich habe sie auch darauf getrimmt.

Ich habe wegen dieses, fur mich sehr relevanten Problems, bereits mit Microsoft und anderen Batchexperten kommuniziert - ohne Erfolg! MS denkt z.B. auch nicht daran, die fehlerhafte COMMAND.COM zu überarbeiten weil DOS fur sie (MS) tot ist!!!

Ob dieses Problem unter PC-DOS, NovellDOS oder auch PTS-DOS existiert, ist mir nicht bekannt! Auf jeden Fall gibt es unter 4DOS und auf meinem PoFo keine Variablenleichen, wie ich durch intensive Tests feststellte.

Ich habe dennoch nicht die Flinte ins Korn geworfen und versucht dieses Problem in den Griff zu bekommen - und tatsächlich gibt es eine Losung, die sogar die Fehlermeldung "Umgebungsspeicher voll!" nicht erscheinen laßt:

```
@echo off
rem
rem How to solve the problem with environmententries
   without equal-sign or
rem with a full environment! (knowned at MS-DOS)
rem
rem C:\BATCH\TRYTOSET.BAT - this file
rem C:\BATCH\TTS2.BAT     - made by
   C:\BATCH\TRYTOSET.BAT
rem
rem SYNTAX: TRYTOSET VARIABLE=DEFINITION
rem SYNTAX: CALL C:\BATCH\TRYTOSET.BAT
   VARIABLE=DEFINITION
rem
rem written by: Lars Aschenbach, Germany
rem
if %1%2.==. goto err
if %2.==. goto err
if %3.==@. goto next
if %COMSPEC%.==. goto err
echo @set %1=%2>c:\batch\tts2.bat
if not exist c:\batch\tts2.bat goto err
echo @if %%%1%%.==. c:\batch\trytoset.bat %1 %2
   @>>c:\batch\tts2.bat
if exist c:\batch\tts2.bat %COMSPEC% /c
   c:\batch\tts2.bat>nul
:next
if %0.==trytoset. if %3.==@. echo Can't set the variable
   %1!
if %0.==TRYTOSET. if %3.==@. echo Can't set the
   variable %1!
if %3.==. set %1=%2
:err
if %1%2.==. echo The name and the definition of the
   variable is missing!
if %2.==. echo Definition of the variable is missing!
if %COMSPEC%.==. echo The variable COMSPEC is
   missing!
if not %COMSPEC%.==. if not exist c:\batch\tts2.bat echo
   Disk is full!
if exist c:\batch\tts2.bat del c:\batch\tts2.bat
```

Weil das Problem international bekannt und reproduzierbar ist, habe ich die Meldungen in Englisch gehalten und will diese Losung auch noch an einige bekannte Batchexperten schicken!

Will man nun z.B. im Direktmodus die Variable O anlegen und mit P definieren, so ruft man diese Batchdatei mit der Eingabe: TRYTOSET O=P auf.

In einer Batchdatei konnte TRYTOSET.BAT so eingebunden werden, wenn nicht ein Parameter als Variable oder Definition in Frage käme (CLTST1.BAT):

```
@echo off
call c:\batch\trytoset.bat o=p
if not %o%.==p. echo The environment must be full!
if %o%.==p. echo Well done!
```

Und so lautet die Einbindung von TRYTOSET.BAT, wenn der Parameter %1 als Variablenname und der Parameter %2 als Definition genutzt werden soll (CLTST2.BAT):

```
@echo off
call c:\batch\trytoset.bat %1 %2
set | find /i "%1=%2">nul
if errorlevel 1 echo The Environment must be full!
if not errorlevel 1 if errorlevel 0 echo Well done!
```

Aber selbst bei diesen Losungen tauchen neue Probleme auf: Wird die Hilfsdatei TTS2.BAT ein Null-Langen-File, also die Diskette muß voll sein, so kann nicht versucht werden die gewünschte Variable zu setzen!
Ist das Programm FIND unauffindbar, weil PATH leer oder inkorrekt, so wird CLTST2.BAT fehlschlagen!
Dennoch ist TRYTOSET.BAT weitaus sicherer als der SET-Befehl und wird daher demnächst verstärkt in meinen Batchdateien auftauchen. Die Verzeichnisse werden dann auch durch die Variablen BDV und BDY ersetzt worden sein.