NT- Kurse der Technikerschule München - Die NT-Skriptsprache

"Stapelprogramme" oder "Batch- Dateien", mit denen man komplizierte Befehlsfolgen automatisieren kann, gab es schon unter DOS. Die Skriptsprache selbst hat zwar gegenüber DOS nur wenige Erweiterungen erfahren, durch die mächtigen NT- Kommandos und Dienstprogramme hat man trotzdem viel mehr Möglichkeiten als in DOS.
Visual- Basic- Skripte werden in diesem Kurs *nicht* behandelt.
Ziel: Sie lernen die NT- Skriptsprache soweit kennen, daß Sie damit Aufgabenstellungen in der NT- Systemverwaltung lösen können.

1. Was ist ein Skript?

Ein NT- Skript ist eine Textdatei mit NT-Kommandozeilen- Befehlen. Sie wird mit der Namenserweiterung **.CMD** - am besten **im Suchpfad** (PATH) - gespeichert und auf der Befehlszeile ("DOS- Fenster") gestartet. Der Befehlsinterpreter **CMD.EXE** arbeitet das Skript dann zeilenweise ab.

Im Registrierungs- Schlüssel **HKEY_CURRENT_USER \Software\Microsoft\Command Processor** muß der Eintrag ***EnableExtensions*** den Wert 1 haben, damit CMD alle Skripterweiterungen gegenüber DOS verarbeitet (Voreinstellung).

Die NT- Kommandos finden Sie in der NT-Systemhilfe im Buch "WINDOWS NT Befehle" gut dokumentiert. Die meisten Kommandos entsprechen in der Funktion den gleichnamigen DOS- Befehlen. Viele sind erweitert worden, andere wurden nur zur Abwärtskompatibilität

beibehalten und arbeiten nur in einem DOS-Teilsystem, zum Beispiel in der AUTOEXEC.NT. Die Unterschiede gegenüber DOS sind in einem eigenen Hilfekapitel aufgeführt.

2. Wozu brauche ich ein Skript?

Gründe, ein Skript einzusetzen gibt es viele:

```
* mehr Komfort:                          komplizie
* Systeminformationen aufbereiten:      Systemsch
* Befehle zeitgesteuert ausführen:      regelmäßi
* maßgeschneiderte Benutzerumgebung:    Loginskri
* maßgeschneiderte Programmumgebung:    Anwenderp
```

3. Info für die Anwender ausgeben

Für eine einfache Textausgabe verwenden Sie das Kommando **ECHO**.
Die Groß- / Kleinschreibung spielt bei den NT-Befehlen übrigens keine Rolle. Großgeschriebene Befehle erleichtern aber die Lesbarkeit.
Bevor Sie Text ausgeben, sollten Sie die Quittungsmeldungen der Befehle ausschalten:

```
@ECHO OFF
REM Dieser Kommentar wird nicht
mehr angezeigt.
```

@ bewirkt, daß auch die Echo-Zeile unsichtbar bleibt.
Festen Text (ein Stringliteral) schreibt man einfach mit einem Leerzeichen oder TAB- Abstand hinter das Echo- Kommando:

```
ECHO Hallo Otto!
```

Leider braucht man für jede Textzeile ein eigenes ECHO- Kommando.
Denken Sie daran, daß nur kurze Textzeilen auch in einem kleinen "DOS-Fenster" lesbar bleiben.

```
ECHO Pfui! Das ist ein
abschreckendes Beispiel für den
```

```
ungewollten Zeilenumbruch in einer
langen Kommandozeile, die in einem
kleinen DOS- Fenster angezeigt
wird.
```

Leerzeilen erzeugt man mit

```
ECHO .
```

Vorsicht mit den Umlauten:

```
ECHO Das waren einmal
ÄÖÜäöü(geschrieben in NOTEPAD).
```

Ausgabe: `Das waren einmal -Í_õ÷³ (geschrieben in NOTEPAD).`

Lösung: Nehmen Sie einen ASCII- Editor, z.B.EDIT, wenn Sie Umlaute ausgeben wollen.

Mit ECHO können Sie bequem Systemvariablen auslesen:

```
ECHO Du bist angemeldet als:
%USERNAME%.
ECHO Dein Homedirectory ist:
%HOMEDIR%.
```

Die Beispiele dieses Kapitels finden Sie zum Ausprobieren in: AUSGABEN1.CMD

Jetzt sind Sie an der Reihe:

Übung 1:

Schreiben Sie die Skripts **SYSINFO.CMD, NETINFO.CMD, USERINFO.CMD**!

SYSINFO soll z.B. die Prozessor- und Betriebssystem- Daten ausgeben (Systemlaufwerk, Systemverzeichnis, Pfad...),

NETINFO soll vom Computernamen bis zur Domäne die Netzwerkinformationen anzeigen,

USERINFO soll Informationen über den angemeldeten Benutzer, sein Homedirectory (auch das Inhaltsverzeichnis) usw. ausgeben.

4. Ausgabe in Dateien umleiten

Für eine Umleitung in Dateien gibt es die Umleitungszeichen > und >>. Das funktioniert mit fast jedem NT- Kommando.

Dieses Beispiel schreibt eine Liste aller DLLs aus dem Systemverzeichnis in eine Datei:

```
DIR %SystemRoot%\*.dll /s &gt;
%TEMP%\DLL_LIST.TXT
```

Das einfache ">" überschreibt eine vorhandene gleichnamige Zieldatei, während das ">>" zum Anhängen an eine bestehende Datei eingesetzt wird.
Auch einen Text können Sie damit in eine Datei umleiten - er darf aber selbst keine Umleitungszeichen enthalten!!

```
ECHO Dieses Umleitungszeichen >
ist falsch plaziert!
```

Der Text wird gar nicht angezeigt, dafür steht anschließend in einer Datei mit dem Namen"ist":
Dieses Umleitungszeichen falsch plaziert!.

Die Beispiele dieses Kapitels finden Sie zum Ausprobieren in: AUSGABEN2.CMD

Jetzt sind Sie wieder dran:

Übung 2:

Schreiben Sie ein Skript **DRIVERSTAT.CMD**!
Es soll Systemdatum und Systemzeit festhalten und anschließend die Pfadnamen aller auf dem Systemlaufwerk gespeicherten Gerätetreiber (Namenserweiterung .SYS) in die Datei DRIVERLOG.TXT speichern. Beim wiederholten Aufuf sollen die neuen Daten angehängt werden.

5. Eingaben

Die Möglichkeiten, per Tastatur Werte an das laufende Skript zu übergeben, sind ziemlich eingeschränkt.
Fangen wir gleich mit dem Notausgang an: Mit der Tastenkombination **STRG C** läßt sich ein Skript abbrechen.
Das DOS- Kommando **CHOICE** gibt es leider nur noch im Resourcekit.
Ergebnis einer Eingabe mit CHOICE ist eine Fehlernummer. Sie entspricht der *Position* in der Liste möglicher Eingaben und wird im weiteren Programmablauf mit IF ERRORLEVEL # ausgewertet:

```
REM Kommentar: A->1, B->2, C->3
CHOICE /C:ABC
IF ERRORLEVEL 3 GOTO C
```

```
IF ERRORLEVEL 2 GOTO B
IF ERRORLEVEL 1 GOTO A
:A
ECHO Du hast a oder A eingegeben.
GOTO ENDE
:B
ECHO Du hast b oder B eingegeben.
GOTO ENDE
:C
ECHO Du hast c oder C eingegeben.
GOTO ENDE
:ENDE
```

Dabei wird geprüft, ob die Fehlernummer >= dem Vergleichswert ist !
Eine Abkürzung des obigen Beispiels auf

```
CHOICE /C:ABC
IF ERRORLEVEL 3 ECHO Du hast c
oder C eingegeben.
IF ERRORLEVEL 2 ECHO Du hast b
oder B eingegeben.
IF ERRORLEVEL 1 ECHO Du hast a
oder A eingegeben.
```

liefert folglich nicht das gewünschte Resultat:

```
[A,B,C]?C
Du hast c oder C eingegeben.
Du hast b oder B eingegeben.
Du hast a oder A eingegeben.
```

Wenigstens prüft CHOICE die Eingabe und läßt nur die angegebenen Zeichen zu. Auf Wunsch kann es sogar Groß- und Kleinschreibung unterscheiden. Weitere Parameter erlauben die Einstellung eines Timout und eines Default-Werts:

```
CHOICE [/C[:]Zeichen zur Auswahl]
[/N] [/S] [/T[:]c,nn] [text]
/C[:] erlaubte Zeichen selbst
festlegen. Voreingestellt ist JN
bzw. YN.
/N Die Auswahlmöglichkeiten werden
```

```
nicht angezeigt.
/S Unterscheidung von Groß- und
Kleinschreibung.
/T[:]c,nn Nach nn Sekunden wird
automatisch die Auswahl c
verwendet.
text Frei definierbarer
Zusatztext, der ausgegeben wird.
```

Um ein Skript an einer bestimmten Stelle anzuhalten, bis der Anwender eine beliebige Taste betätigt, baut man das Kommando **PAUSE** ein. Wenn ECHO ON aktiv ist, könnte man sogar einen Text mitgeben, aber eine Ausgabe mit ECHO und stummer Pause sieht besser aus:

```
ECHO ON
@ECHO ON
PAUSE Hier sehen Sie die PAUSE
mit.
@ECHO OFF
ECHO So ist es schoener:
PAUSE
```

Tip: Bauen Sie in der Debug- Phase Ihrer Skripts viele PAUSEn und ECHOs ein!
Eingaben lassen sich mit < aus Textdateien beziehen:

Ein "j" in der Datei "JA" bestätigt die Rückfrage des cacls- Kommandos und erspart die Eingabe von Hand. Leider ist das Verhalten vieler NT-Kommandos ist unbestimmt, wenn mehrere verschiedene Eingaben aus einer Datei importiert werden sollen. Manchmal funktionierts mit <<.
Die Beispiele dieses Kapitels finden Sie in:
EINGABEN.CMD

Übung 3:

Schreiben Sie ein Skript **SYSEDIT.CMD**!
Es soll dem Anwender in einem Menü die Bearbeitung der Konfigurationsdateien BOOT.INI, CONFIG.NT und AUTOEXEC.NT in einem geeigneten Editor anbieten.

Selbstverständlich küermt sich das Skript auch um den Schreibschutz dieser Dateien und greift auf die richtigen Pfade zu!

6. Beim Skriptaufruf Werte übergeben

a) Merkwürdige Variablennamen

Soll das Skript mit Parametern gesteuert werden, kommen die Variablen %1 bis %9 zum Einsatz. Die Nummer enspricht dem Platz des Parameters in der Kommandozeile:

```
@ECHO OFF
REM    MCD.CMD legt Verzeichnis an
und wechselt in das Verzeichnis
MD    %1
CD    %1
@ECHO ON
```

Ausführung (gestartet aus dem Verzeichnis d:\bat):

```
D:\BAT>mcd \test
D:\test>
```

Zum Ausprobieren: MCD1.CMD
Das Beispiel nimmt den ersten Übergabeparameter als neuen Verzeichnisnamen, legt das Verzeichnis an und wechselt in dieses Verzeichnis. Das funktioniert aber nur, wenn tatsächlich ein Parameter angegeben wurde und das Verzeichnis noch nicht existiert! Andernfalls gibt es unschöne Fehlermeldungen.

b) Aufrufparameter prüfen

Hier die Luxusversion von MCD.CMD:

```
@REM    MCD.CMD Stand 11.11.99
@ECHO OFF
IF %1!==! GOTO Hilfe
IF %1==/? GOTO Hilfe
IF EXIST %1\NUL GOTO Gibts_schon
@MD %1
:Gibts_schon
```

```
CD %1
GOTO Ende
:Hilfe
ECHO        MCD.BAT
ECHO    erzeugt ein
Unterverzeichnis mit dem
ECHO    angegebenen Namen und
wechselt in dieses
ECHO    Verzeichnis.
ECHO.
ECHO    Sie muessen beim Aufruf
von MCD einen neuen
ECHO    Verzeichnisnamen angeben:
ECHO    MCD Verzeichnisname
:Ende
```

Zum Ausprobieren: MCD2.CMD
Das IF Ausdruck1 == Ausdruck2 - Kommando wird nur ausgeführt, wenn beide Ausdrücke übereinstimmen (Groß- / Kleinschreibung ignoriert).
Weil nicht "Nix" mit "Nix" verglichen werden kann, wurde in Zeile 4 ein weiteres Zeichen (!) an beide Operanden angehängt.
Wird beim Aufruf kein Verzeichnisname angegeben oder mit /? Hilfe verlangt, springt das Programm zum Label mit dem Hilfetext.
Mit IF EXIST Dateiname wird geprüft, ob das Verzeichnis überhaupt angelegt werden muß.
Weil nur die Existenz von Dateien, nicht aber von Verzeichnissen überprüft werden kann, wird die Datei NUL angegeben. Den Dummy NUL gibt es auch in einem leeren Verzeichnis.
Nebenbei haben Sie im letzten Beispiel den Sprungbefehl **GOTO** und die Schreibweise von **:LABELs** kennengelernt.
Übung 4: Wenn Sie das Skript MCD.CMD genau verstanden haben, fällt es Ihnen nicht schwer, den Programmablaufplan zu vervollständigen! Speichern Sie dazu die Vorlage und laden Sie diese in ein Zeichenprogramm, mit dem man Text einfügen kann.

c) Aufrufparameter nachrücken

Der Befehl **SHIFT** rückt die Parameterliste eines Skripts um eine Position nach links:

```
@ECHO OFF
REM TESTSHIFT.CMD zeigt die
Verschiebung der Parameterliste
ECHO Vor SHIFT:
ECHO %0 %1 %2 %3 %4
ECHO Einmal nach links:
SHIFT
ECHO %0 %1 %2 %3 %4
ECHO Nochmal nach links:
SHIFT
ECHO %0 %1 %2 %3 %4
ECHO Alle Parameter in der
urspruenglichen Form:
ECHO %*
ECHO Der Dateiname des Skripts:
ECHO %0
```

Die Ausgabe:

```
D:\BAT>testshift Adam Berta Carl
Dora
Vor SHIFT:
testshift Adam Berta Carl Dora
Einmal nach links:
Adam Berta Carl Dora
Nochmal nach links:
Berta Carl Dora
Alle Parameter in der
urspruenglichen Form:
Adam Berta Carl Dora
Der Dateiname des Skripts: Berta
```

Zum Ausprobieren: TESTSHIFT.CMD
Der Parameter %0 enthält den Dateinamen, der allerdings beim ersten SHIFT überschrieben wird. Der Platzhalter %* behält dagegen - bis auf %0 - die ursprüngliche Parameterliste.

Übung 5: Schreiben Sie ein Skript **X.CMD** ! Aufgerufen mit einem NT- Kommando als erstem

Parameter soll dies auf alle nachfolgenden Parameter angewendet werden.
Beispiel: x DEL/S/Q C:\TEMP*.TMP soll alle Dateien mit der Extension .X im Ordner C:\TEMP und dessen Unterverzeichnissen ohne Rückfrage löschen.
Selbstverständlich sollte Ihr Skript dem Anwender auch eine Syntaxhilfe anbieten...

d) Aufrufparameter zu Pfadnamen ergänzen

Dateinamen in den Übergabeparametern lassen sich isolieren oder ergänzen:

```
@ECHO OFF
REM TESTARGS.CMD trennt
Pfadkomponenten im Parameter 1
ECHO Parameter 1:  %1
ECHO Nur das Laufwerk:  %~d1
ECHO Nur das Verzeichnis:  %~p1
ECHO Nur der Dateiname:  %~n1
ECHO Nur die
Namenserweiterung:  %~x1
ECHO Trotzdem richtigrum:  %~xnpd1
```

Die Ausgabe:

```
D:\BAT>Testargs TESTARGS.CMD
Parameter 1:    TESTARGS.CMD
Nur das Laufwerk:    D:
Nur das Verzeichnis:  \BAT\
Nur der Dateiname:  TESTARGS
Nur die Namenserweiterung:  .CMD
Trotzdem
richtigrum:  D:\BAT\TESTARGS.CMD
```

Dieses Beispiel finden Sie in: TESTARGS.CMD

7. Variable

Die vordefinierten System- oder Environmentvariablen kennen Sie bereits. Auch die Variable **ERRORLEVEL** zählt dazu. Ihren Inhalt kann man unmittelbar nach einem

Kommando auswerten.
Mit **SET** können Sie im Skript neue Variablen setzen und löschen.
Im Beispiel TESTSHIFT.CMD könnte man den Dateinamen z.B. aus dem Parameter %0 in eine Variable retten:
SET FileName=%0
SHIFT darf nun %0 überschreiben, der Skriptname bleibt in FileName gesichert. Die Variable bleibt solange verfügbar, wie der Prozeß läuft, der das Skript gestartet hat oder bis man sie durch eine leere Zuweisung wieder löscht:
SET FileName=
Die Anweisungen **SETLOCAL** und **ENDLOCAL** beschränken Sie die Gültigkeit von Variablen auf einen Block. Damit vermeidet man z.B., daß bereits bestehende Sysstemvariablen versehentlich überschrieben werden.

```
@ECHO OFF
REM LocalVars.CMD zeigt die
Verwendung von SETLOCAL
SET Bubi=Osterhasi
CLS ECHO Bubi sagt: %Bubi%
REM Blockanfang
SETLOCAL
ECHO Bubi wird im Block
ueberschrieben:
SET Bubi=Nikolausi
ECHO Bubi sagt: %Bubi%
ENDLOCAL
REM Blockende
ECHO Wir sind wieder ausserhalb
des Blocks.
ECHO Bubi sagt: %Bubi%
ECHO Bubi verschwindet im Sack...
SET Bubi=
ECHO %Bubi%?
```

Die Ausgabe:

```
Bubi sagt: Osterhasi
Bubi wird im Block ueberschrieben:
Bubi sagt: Nikolausi
```

```
Wir sind wieder ausserhalb des
Blocks.
Bubi sagt: Osterhasi
Bubi verschwindet im Sack...
?
```

Zum Ausprobieren: LOCALVARS.CMD
Leider ist es in der einfachen NT- Skriptsprache nicht möglich, Systemvariablen und Benutzervariablen dauerhaft zu verändern. Dazu braucht man einen leistungsfähigeren Skript-Interpreter wie z.B.KiXtart95, der auch in die Registry schreiben darf.

8. Schleifen

Die **FOR**- Schleife gibt es wohl in jeder Programmiersprache. Anstelle der unverständlichen Syntaxbeschreibung lieber gleich ein Beispiel:

```
@ECHO OFF
REM KUSS.CMD ECHO Uschi
FOR %%A IN
(Anton,Berta,Georg,Susi) DO ECHO
küßt %%A, %%A
ECHO riecht aber nach Knoblauch.
```

Zum Ausprobieren: KUSS.CMD
Die Ausgabe:

```
Uschi
kuesst Anton, Anton
kuesst Berta, Berta
kuesst Georg, Georg
kuesst Susi, Susi
riecht aber nach Knoblauch.
```

Für die Schleifenvariable %%A (es darf nur ein Zeichen nach den %% stehen) wird in jedem Durchlauf ein Wert aus der geklammerten Liste eingesetzt.
Der nachfolgende Befehl (z.B. ECHO) kann auf den aktuellen Wert zugreifen.
Eine weitere Variante ähnelt der FOR- Schleife in

C, allerdings sind Reinitialisierung und Wiedereintrittsbedingung vertauscht:

```
@REM WUFFI.CMD
@ECHO OFF
ECHO Wuffi bellt
@FOR /L %%A IN (1,5,15) DO ECHO
%%A mal,
ECHO dann fliegt ein Schlappschuh.
```

DieAusgabe:

```
Wuffi bellt
1 mal,
6 mal,
11 mal,
dann fliegt ein Schlappschuh.
```

Die FOR- Schleife wurde in NT noch erweitert. Sehen Sie sich die Syntaxbeschreibung an und entscheiden Sie selbst, ob Sie damit klar kommen. Es soll auch Holzfäller geben, die Bäume mit dem Kartoffelmesser umschneiden, statt gleich eine richtige Kettensäge zu nehmen.
Zum Ausprobieren: WUFFI.CMD

9. Bedingte Ausführung

Die meisten Skriptkommandos zur bedingten Ausführung haben Sie bereits "on the fly" kennengelernt.
Hier nochmal eine Zusammenfassung:

```
IF ERRORLEVEL # Befehl
IF EXIST Datei Befehl
IF Zeichenkette_1 = =
Zeichenkette_2 Befehl
```

Außer dem Gleichheitszeichen sind noch weitere Operanden erlaubt:

NEQ ungleich
LSS Zeichenkette_1 alphabetisch kleiner als Zeichenkette_2
LEQ Zeichenkette_1 alphabetisch kleiner oder gleich Zeichenkette_2
GTR Zeichenkette_1 alphabetisch

größer als Zeichenkette_2
GEQ Zeichenkette_1 alphabetisch größer oder gleich Zeichenkette_2

Ob eine Variable schon existiert, prüft man mit
IF defined Variable Befehl
In allen Varianten kann mit **IF NOT** die Auswertelogik invertiert werden.
Wenn es sein muß können Sie mehrere Befehle in einer Zeile mit **&** verketten und nach einem ^ sogar in der nächsten Zeile fortsetzen. Wie Sie im Beispiel WENNDANN.CMD sehen können, entspricht das Ergebnis nicht immer von einzelnen Kommandozeilen!
Ein && führt den nächsten Befehl nur dann aus, wenn der erste erfolgreich war. Das Skript wird dadurch aber nicht besser lesbar, bleiben Sie lieber bei IF ERRORLEVEL...

10. War das schon alles?

Sicher kennen Sie jetzt noch nicht jedes Zängelchen aus dem NT- Skript- Werkzeugkasten, aber viel mehr wird tatsächlich nicht geboten. Gut, daß es Erweiterungen gibt.

3.1 KiXStart 95

wird in einer älteren Version auch im Resourcekit mitgeliefert. Auf dem Übungsserver finden Sie es im Verzeichnis \Programme\NTReskit\Kix95.
Einige Features:
* bequeme Ein- und Ausgaberoutinen
* Stringbearbeitung
* einfaches Variablenhandling
* Zugriff auf viele Netz- und Systemvariablen
* Lesen und Schreiben in der Registry
Die Anleitung steckt im Word- Dokument KIX32.DOC. Sie enthält eine brauchbare Beschreibung aller Features und gute Beispiele.
Im WWW war das Archiv am 06.12.99 unter
http://netnet.net/~swilson/kix/nfKiX.htm
erreichbar. Dort können Sie auch eine aktuelle

Version herunterladen und ausgewählte Beispiele studieren.

3.2 PERL für NT

Wenn Sie schon Erfahrungen mit PERL gesammelt haben, sollten Sie sich mal die Perl-Module für NT genauer ansehen. Sie sind ebenfalls im Resourcekit enthalten und auf dem Übungsserver ereichbar. Die Skriptsprache Perl ist noch wesentlich leistungsfähiger als Kixtart 95 und ihr Einsatz ist nicht auf WINDOWS beschränkt.

---

adaptiert von A. Allmann