Shellprogrammierung in der Praxis

(C) 2006 Dennis Schulmeister
Schliffkopfweg 12
76189 Karlsruhe
(+49) 721 / 5978883

dennis(at)windows3(punkt)de
http://www.windows3.de

INHALT

1. WAS IST DAS, EINE SHELL?

1.1. DEFINITION: SHELL

Eine Shell ist ein besonderes Programm, welches es dem Anwender ermöglicht, mit dem Betriebssystem, auf dem es läuft, in Verbindung zu treten. Hierzu stellt die Shell verschiedene Möglichkeiten der Datenein- und Ausgabe sowie Befehle zur Dateiverwaltung bereit.

Das Wort "Shell" kommt aus dem Englischen und bedeutet "Muschel". Dies ist als Metapher zu verstehen, da die Shell praktisch den äußersten Teil eines Systems abgrenzt. Sie bildet also die Grenze zwischen dem Betriebssystem und der realen Welt des Benutzers.

Die Herkunft des Wortes "Shell" reicht bis in die Zeit zurück, als es noch keine graphischen Benutzeroberflächen gab. Der Zugang zum Computer erfolgte damals rein textbasiert, häufig durch die Eingabe von Befehlen (Kommandos). Aus diesem Grund werden heute häufig nur Kommandozeileninterpreter oder terminalbasierte Computerzugänge "Shell" genannt. Streng genommen gehören aber auch alle graphsichen Benutzerschnittstellen zu den Shells.

1.2. BEISPIELE VERSCHIEDENER SHELLS

Wie oben erwähnt, gibt es zwei Arten von Shells: Die älteren textbasierten Shells sowie graphische Benutzeroberflächen, welche den Bildschirm in mehrere Sichtfenster einteilen und das Ausführen von Befehlen via Point & Click

ermöglichen.

Die wohl bekannteste, graphische Shell ist der Windows-Explorer. Dieser stellt dem Benutzer den Desktop, das Startmenü sowie Fenster zur Dateiverwaltung zur Verfügung. Am zweitbekanntesten dürfte dementsprechend der MS-DOS Kommando-zeileninterpreter sein. Seine Ahnengallerie reicht bis in die späten 1970er zurück, womit er älter als Windows selbst ist. Entstanden ist er als Teil des Betriebssystems QDOS, welches ein Klon von DR's CP\M war. Aus QDOS ging später MS-DOS hervor, welches zuletzt in Windows NT aufging.

Für MS-DOS und Microsoft Windows werden seit jeher alternative Shells angeboten, welche häufig komfortablere Benutzeroberflächen bieten, als die systemeigenen Programme. Beispiele hierfür sind der Norton Commander oder DR's GEM, welches eine grafische Fensteroberfläche für DOS bietet.

Unter den UNIX-Ablegern existiert ebenfalls eine breit gefächerte Auswahl an den unterschiedlichsten Shells. Der Urgroßvater der UNIX-Shells dürfte wohl sh sein, was knapp für "Shell" steht. Da sh Bestandteil vieler proprietärer UNIXe war und ist, enstand im Rahmen des GNU-Projekts der Free Software Foundation die BASH als vollwertiger Ersatz für sh. Weiterhin sind auch die Korn-Shell oder die C-Shell weit verbreitet.

Grafische UNIX-Shells sind unter Anderem: Fluxbox, XFCE, KDE und GNOME.

1.3. WELCHE SHELL VERWENDEN WIR?

Da es für die verschiedenen Betriebssysteme meist mehrere Shells gibt, ist die Auswahl sehr umfangreich. Auf Grund der großen Verbreitung und der einfachen Erlernbarkeit beschränken wir uns allerdings auf den MS-DOS Kommandozeilen-interpreter, welcher auch heute noch Bestandteil jeder Windows-Distribution ist.

2. WAS IST EIN SKRIPT?

2.1 WAS ZEICHNET EIN SKRIPT AUS?

Ein Skript ist ein Programm, welches nur in Form seines Quelltextes vorliegt und zur Ausführung nicht umgesetzt werden muss. Im Gegensatz zu vollwertigen Programmen entfällt also der Prozess des Compilierens und Linkens. Stattdessen wird der Quelltext von einem Interpreter Zeile für Zeile interpretiert.

Skripts zeichnen sich auch dadurch aus, dass sie vom Anwender selbst für besondere Anwendungsfälle geschrieben werden können. Somit erfüllen sie häufig viele kleine Aufgaben hier und da.

Je nach Anwendungsgebiet, können die Skripts dennoch beträchtlichen Umfang annehmen. Ein UNIX ohne Skripte wäre somit undenkbar. Allein das Startup-Skript oder das Skript zum Compilieren einer Anwendung kann eben mal so ein paar Tausend Zeilen enthalten.

Auch die verwendete Skriptsprache übt enormen Einfluss auf das jeweilige Skript aus. Manche Sprachen stellen nur sehr wenige Befehle und Funktionalitäten zur Verfügung (z.B. die Batch-Sprache der MS-DOS-Shell) und manche Sprachen wiederum können als vollwertige Programmiersprachen bezeichnet werden, da sie auch umfassende Kontrollstrukturen (Bedingungsblöcke, Schleifen) oder gar Funktionen bis hin zu Klassen und Objekten anbieten (z.B. Python).

2.2. WELCHE WERKZEUGE BRAUCHT MAN FÜR DIE SKRIPTPROGRAMMIERUNG?

Für die Skriptprogrammierung braucht man nicht viel Handwerkszeug. Ein einfacher Texteditor zum Erzeugen der Skripte sowie ein Skriptinterpreter genügen völlig.

Als Interpreter kommen entweder diverse Shells in Frage (es handelt sich dann um

sogenannte Shellskripte) oder spezielle Interpreter. Diese können entweder Stand-Alone arbeiten (z.B. Perl, Python, PHP, Ruby, Profan, diverse BASIC) oder Bestandteil einer Rahmenanwendung sein. (Häufig Office-Pakete aber auch Grafikanwendungen wie The GIMP oder CorelDraw).

2.3. WELCHE WERKZEUGE VERWENDEN WIR?

Skripte an sich benötigen schon wenig Werkzeuge. Da wir die MS-DOS-Shell verwenden, benötigen wir nicht mehr als einen kleinen Text-Editor. Die Skripte interpertiert die Shell für uns.

3. WIR SCHREIBEN EIN SKRIPT!

3.1. WICHTIGE BEFEHLE UND KONTROLLSTRUKTUREN

Die folgende Liste enthällt die wichtigsten Befehle des MS-DOS Kommandozeilen-interperters. Eine vollständige Liste, welche je nach System variiert, kann mit HELP angezeigt werden.

Bis auf die Befehle IF, GOTO, GOSUB und RETURN bietet die MS-DOS-Shell keinerlei Kontrollstrukturen. Batch-Skripte haben somit sehr imperativen Charakter und sind recht einfach aufgebaut.

Dem Skript können auch Parameter mitgegeben werden, welche einfach durch Leer-zeichen getrennt beim Aufruf hinter den Dateinamen geschrieben werden. Diese befinden sich in den Variablen %1, %2, ...

- :Bezeichner
  Definiert ein Sprunglabel in einem Skript, welches durch GOTO und GOSUB angesprungen werden kann.
- ATTRIB Dateimaske [ATTRIBUTE]
  Zeigt entweder die Dateiattribute der genannten Dateien oder setzt diese.
- CHKDSK
  Überprüft ein Laufwerk auf Fehler. Windows 95 bis ME bieten hierfür auch SCANDISK.
- CLS
  Löscht den Bildschirm.
- DATE
  Befehl zum Anzeigen und Verändern das Datums der Systemuhr.
- DEL Dateimakse
  Löscht eine oder mehrere Dateien.
- DELTREE Dateimaske
  Löscht ein oder mehrere Verzeichnisse sammt Inhalt. Kann auch zum Löschen von Dateien benutzt werden.
- DIR Maske
  Zeigt den Inhalt eines Verzeichnises an. Die Maske kann entweder leer sein oder die Selektion einschränken.
- ECHO Text
  Gibt einen beliebigen Text auf dem Bildschirm aus
- ECHO OFF
  Schaltet das automatische Echo aus. Somit werden die in einem Skript verwendeten Befehle nicht auf dem Bildschirm wiederholt, sofern ihnen ein @ vorgestellt wird.
- ECHO ON
  Schaltet das automatische Echo wieder ein.
- FIND String Dateimaske
  Sucht nach einem Strink in mehreren Datein und zeigt die Fundstellen an.
- FOR %%Variable IN (Datei1, Datei2, ...) DO Befehl ... %%Variable ...
  Führt den Befehl für jede in der Liste gegebene Datei aus. %%Variable kann ein beliebiger Variablenbezeichner sein und wird für den auszuführenden Befehl durch den entsprechenden Dateinamen ersetzt.
- GOSUB Label

Sprint zum angegebenen Label. Der Name muss ohne führenden Doppelpunkt angegeben werden. Durch RETURN kehrt das Skript wieder an den Ausgangspunkt zurück.

- GOTO Label
  Wie GOSUB nur ohne Wiederkehr durch RETURN. (Vgl. Edsger W. Dijkstra: Go To statement considered harmful. März 1968. z.B. http://www.acm.org/classics/oct95/)
- IF Bedingung Befehl
  Führt den Befehl aus, falls die Bedingung erfüllt ist.
- IF NOT Bedingung Befehl
  Führt den Befehl aus, falls die Bedingung nicht erfüllt ist.
- IF EXIST Datei Befehl
  Führt den Befehl aus, falls die Datei vorhanden ist.
- IF NOT EXIST Datei Befehl
  Führt den Befehl aus, falls die Datei nicht existiert.
- IF EXIST Datei GOTO Label
  Springt zum gegebenen Label, falls die Datei existiert.
- IF NOT EXIST Datei GOTO Label
  Springt zum gegebenen Label, falls die Datei nicht existiert.
- IF "%VARIABLE%" == "Wert" Befehl
  Führt den Befehl aus, falls die Umgebungsvariable den genannten Wert hat.
- IF NOT "%VARIABLE%" == "Wert" Befehl
  Führt den Befehl aus, falls die Variable nicht den genannten Wert enthällt.
- LH Befehl
  Versucht das Programm im hohen Speicherbereich auszuführen. Nur für MS-DOS, LH hat unter Windows keinerlei Wirkung. Der Befehl wird einfach ausgeführt.
- LOADHIGH Befehl
  Langform von LH Befehl.
- MOVE Dateimaske Ziel
  Verschiebt eine oder mehrere Dateien.
- REM
  Leitet einen Kommentar ein. Alles nach REM wird ignoriert.
- REN Von Nach
  Benennt eine oder mehrere Dateien um.
- RENDIR Von Nach
  Benennt ein oder mehrere Verzeichnisse um
- RETURN
  Sprint nach einem GOSUB wieder zum Ausgangspunkt zurück. Somit können einfache Unterpogramme definiert werden.
- SET %Variable%=Wert
  Setzt die genannte Umgebungsvariable auf den gegebenen Wert. Umgebungs-variablen bleiben auch nach der Skriptausführung erhalten. Sie gehen erst verloren, wenn die ausführende Shell beendet wird. (z.B. Terminalfenster) Der Wert darf nicht in Anführungszeichen geschrieben werden.
- TIME
  Befehl zum Anzeigen und Verändern der Systemuhrzeit.
- TYPE Dateimaske
  Zeigt den Inhalt beliebiger Dateien auf dem Bildschirm.

3.2. BEISPIEL 1: >>SWITCH MOUSE<<

Auf einem System sind zwei Maustreiber vorhanden. Einer von DR-DOS, welcher nur sehr wenig Hauptspeicher belegt und mit fast allen Programmen funktioniert, und einer von Logitech, welcher mehr Speicher braucht, dafür aber mit allen Programmen funktioniert.

Da Hauptspeicher unter MS-DOS sehr kostbar ist, soll der Logitech-Treiber nur verwendet werden, wenn es absolut sein muss. Der Bequemlichkeit halber soll darum ein Skript geschrieben werden, welches bei jedem Aufruf den gerade aktuellen Treiber entlädt und an seiner Stelle den anderen Treiber lädt.

Beim Hochfahren wird standardmäßig der DR-DOS Treiber geladen.

Die Kommandos zum Laden der Treiber lauten:
  o DRMOUSE   (für den DR-DOS-Treiber)
  o C:\PROGRAMS\MOUSE.EXE (für den Logitech-Treiber).

Die Kommandos zum entladen lauten:
  o DRMOUSE /U   (für DR-DOS)
  o C:\PROGRMS\MOUSE.EXE OUT (für Logitech).

Die Zahlen in den eckigen Klammern gehören nicht zum Quellcode. Sie dienen nur der späteren Besprechung des Skripts.

[DATEI: SWMOUSE.BAT]
```
[1]  @ECHO OFF
[2]  @IF "%MOUSE%" == "L" GOTO DRM
[3]  @IF NOT "%MOUSE%" == "L" GOTO LOGI
[4]
[5]  :LOGI
[6]  @DRMOUSE /U
[7]  @LH C:\PROGRAMS\MOUSE.EXE
[8]  @SET MOUSE=L
[9]  @GOTO ENDE
[10]
[11] :DRM
[12] @LH C:\PROGRAMS\MOUSE.EXE OUT
[13] @DRMOUSE
[14] @SET MOUSE=D
[15]
[16] :ENDE
```

Zeile [1] schaltet in Verbindung mit dem @ vor jedem Befehl das automatische Echo ab. Somit wird der aktuelle Befehl nicht nochmal vor seiner Ausführung ausgegeben.

Zeile [2] überprüft ob die Umgebungsveriable MOUSE gleich L ist. Falls ja, geht das Programm in Zeile [12] (nach dem Label DRM in Zeile [11]) weiter. Falls MOUSE ungleich L ist, bewirkt Zeile [3], dass es in Zeile [5] weiter geht.

Zeile [5] und [11] definieren zwei Label, welche die zwei möglichen Programm-abläufe markieren.

Falls MOUSE gleich "L" ist, geht es im Skript nach Zeile [11] weiter. Die Zeilen [12] und [13] entladen dann den Logitech-Treiber aus dem Speicher und laden stattdessen den DR-DOS Treiber. In Zeile [14] wird die Variable MOUSE darum auf "D" gesetzt. Da keine weiteren Befehle mehr folgen, bricht das Skript an dieser Stelle ab.

Wird das Skript nun wieder aufgerufen, steht die Variable MOUSE auf "D". Somit geht das Programm in Zeile [6] weiter, wo der DR-DOS-Treiber entladen wird. An seiner Stelle wird dann in Zeile [7] der Logitech-Treiber geladen. In Zeile [8] wird schließlich die Variable MOUSE darum auf "L" gesetzt. Zeile [9] sorgt dafür, dass das Skript jetzt nicht einfach mit den folgenden Zeilen weitermacht. Stattdessen wird in Zeile [16] gesprungen, nach der das Skript endet.

3.3. BEISPIEL 2: >>ABSTURZKONTROLLE<<

Vor Windows 95 gab es noch keine Absturzkontrolle, welche nach einem System-absturz die Festplatte nach Fehlern untersucht. Wir wollen uns so eine Funktion darum selbst schreiben.

Wir erzeugen uns dazu ein Skript namens WIN.BAT, welches bei jedem Aufruf

zunächst überprüft, ob eine Datei namens C:\ABSTURZ vorhanden ist. Ist dies der Fall, deutet dies darauf hin, dass das System zuletzt nicht ordnungsgemäß beendet wurde und die Festplatte mit CHKDSK überprüft werden muss.

Ist die Datei C:\ABSTURZ nicht vorhanden, soll sie einfach erzeugt werden.

Auf jeden Fall soll anschließend Windows gestartet werden. Nach der Rückkehr von Windows soll die Datei C:\ABSTURZ wieder gelöscht werden.

```
[DATEI: WIN.BAT]
[1]  @ECHO OFF
[2]  @IF EXIST C:\ABSTURZ. GOTO Absturz
[3]  @IF NOT EXIST C:\ABSTURZ. GOTO KeinAbsturz
[4]
[5]  :Absturz
[6]  @ECHO Windows wurde nicht odrnungsgemäß beendet.
[7]  @ECHO Die Festplatte wird nun auf Fehler untersucht.
[8]  @CHKDSK /F /B
[9]  @GOTO Windows
[10]
[11] :KeinAbsturz
[12] @ECHO Dennis Absturzkontrolle > C:\ABSTURZ.
[13] @ATTRIB C:\ABSTURZ. +H
[14]
[15] :Windows
[16] @C:\WINDOWS\WIN.COM
[17] @DEL C:\ABSTURZ.
```

Zeile [1] unterdrückt zusammen mit den @-Zeichen wieder das automatische Echo.

Wird in Zeile [2] die Datei C:\ABSTURZ gefunden, wurde Windows zuvor nicht richtig beendet und das Programm fährt mit Zeile [6] fort. In dieser Zeile sowie in Zeile [7] wird erstmal eine kleine Meldung ausgegeben, bevor in Zeile [8] die Festplatte kontrolliert wird. Zeile [9] überspringt das Anlegen der ABSTURZ-Datei, da diese bereits existiert.

Altenativ kann die Datei C:\ABSTURZ nicht vorliegen (Zeile [3]). In diesem Fall wird in Zeile [12] gesprungen, wo sie erzeugt wird, indem einfach ein kleiner Text in die Datei geschrieben wird. (Beachte den Umlenkoperator > nach dem ECHO-Befehl). Zeile [13] versteckt die Datei, damit sie im Verzeichnislistung nicht auftaucht.

In beiden Fällen fährt das Skript schließlich in Zeile [16] fort, indem Windows gestartet wird. Nach der Rückkehr aus Windows geht es in Zeile [17] weiter, wo die ABSTURZ-Datei wieder gelöscht wird, da das System ordnungsgemäß beendet wurde.

Dieses Beispiel benutzt eine Datei im Hauptverzeichnis um zu erkennen, ob Windows gestartet und wieder richtig beendet wurde. Umgebungsvariablen können hier nicht eingesetzt werden, da diese nach dem Neustart des Rechners (bei einem Absturz) verloren gehen.

4. AUFGABEN

4.1. WAS IST EIN SKRIPT? WAS BRAUCHT MAN DAZU?

s. Kapitel 2!

4.2. WO WERDEN SKRIPTS NOCH VERWENDET?

Skripts nehmen dem Anwender an vielen Stellen viel Arbeit ab. Die mit Abstand am häufigsten verwendeten Skripts sind sicher Startup- und Shutdown-Skripts,

welche das kontrolleirte Hochfahren und Herunterfahren des Betriebssystems steuern. (z.B. CONFIG.SYS und AUTOEXEC.BAT unter MS-DOS, die init.d-Skripts unter UNIXen mit System-V init ...)

Des Weiteren lassen sich viele Anwendungen mit Hilfe von Skripts durch den Benutzer um neue Funktionalitäten erweitern. z.B. Microsoft Office durch Visual Basic for Applications. OpenOffice durch StarBasic, The GIMP durch Python oder Script-Fu. Diese Liste lässt sich beinahe beliebig fortführen. Auch bietet das GNU-Projekt einen eigenen LISP-Interpreter namels GUILE an, welcher es auf einfache Art und Weise ermöglicht, eigene Programme um Skriptingfähigkeiten zu erweitern.

Skripts sind fester Bestandteil des Computeralltags eines jeden erfahrenen Anwenders.

4.3. SCHREIBE EIN EINFACHES MAKE-SKRIPT
Schreibe ein einfaches make-Skript, welches in Abhängigkeit eines Kommando-zeilenparameters ein bestimmtes Programm entweder compiliert, installiert oder editiert. (Parameter compile, install, edit).

Zunächst soll in das Programmverzeichnis C:\PROGRAMS\EIGENE\KDBB gesprungen werden.

Lautet der Befehl "compilieren", so muss das folgende Kommando ausgeführt werden:
TCC KDBBTEST.C

Lautet der Befehl "installieren", so soll erst geprüft werden, ob das Programm bereits compiliert wurde. Dies ist der Fall, wenn die Datei KDBBTEST.EXE existiert. Ist sie nicht vorhanden, muss eine Fehlermeldung erfolgen. Andernfalls kann das Verzeichnis C:\PROGRAMS\KDBBTEST erzeugt und die Datei KDBBTEST.EXE dort hineinkopiert werden.

Lautet der Befehl "editieren", muss nur das Kommando TC ausgeführt werden.

Lösung:
[DATEI: MAKE.BAT]
```
[1]  @ECHO OFF
[2]  @CD C:\PROGRAMS\EIGENE\KDBB
[3]
[4]  @IF "%1" == "compile" GOTO Comp
[5]  @IF "%1" == "install" GOTO Inst
[6]  @IF "%1" == "edit"    GOTO Edit
[7]  @TYPE Unbekannter Befehl! Benutzen Sie compile, install oder edit.
[8]  @GOTO Ende
[9]
[10] :Comp
[11] @TCC KDBBTEST.C
[12] @GOTO Ende
[13]
[14] :Inst
[15] @IF EXIST KDBBTEST.EXE GOTO Inst1
[16] @TYPE Sie müssen das Programm erst kompilieren.
[17] @GOTO Ende
[18]
[19] :Inst1
[20] @MKDIR C:\PROGRAMS\KDBBTEST
[21] @COPY KDBBTEST.EXE C:\PROGRAMS\KDBBTEST
[22] @GOTO Ende
[23]
[24] :Edit
[25] @TC
```

```
[26] @GOTO Ende
[27]
[28] :Ende
```