Inhalt:

Vorbemerkung

Shell-Scripts (Kommandoprozeduren) sind unter Unix das Analogon zu Batch-Dateien (Stapeldateien) von MS-DOS, sie sind jedoch wesentlich leistungsfähiger. Ihre Syntax hängt allerdings von der verwendeten "Shell" ab. In diesem Artikel werden Bourne Shell (sh) Scripts betrachtet. Sie gelten im allgemeinen als zuverlässiger als csh-Scripts.

Allgemeines zu Shell Scripts

Ein Shell-Script ist eine Textdatei, in der Kommandos gespeichert sind. Es stellt selbst ein Kommando dar und kann wie ein Systemkommando auch mit Parametern aufgerufen werden. Shell-Scripts dienen der Arbeitserleichterung und (nach ausreichenden Tests) der Erhöhung der Zuverlässigkeit, da sie gestatten, häufig gebrauchte Sequenzen von Kommandos zusammenzufassen.

Aufruf

Bourne Shell Scripts lassen sich generell wie folgt aufrufen:

sh *myscript*

Im allgemeinen ist es aber praktischer, die Datei als ausführbar anzumelden (execute permission), mittels

chmod +x *myscript*

Das Shell-Script läßt sich dann wie ein "normales" (binäres) Kommando aufrufen:

myscript

Vorausgesetzt ist hierbei allerdings, daß das Betriebssystem das Script mit der richtigen Shell abarbeitet. Moderne Unix-Systeme prüfen zu diesem Zweck die erste Zeile. Für die sh sollte sie folgenden Inhalt haben:

#!/bin/sh

Obwohl das Doppelkreuz normalerweise einen Kommentar einleitet, der vom System nicht weiter beachtet wird, erkennt es hier, daß die "sh" (mit absoluter Pfadangabe: /bin/sh) eingesetzt werden soll.

Einfaches Beispiel

```
#!/bin/sh
# Einfaches Beispiel
```

```
echo Hallo, Welt!
echo Datum, Uhrzeit und Arbeitsverzeichnis:
date
pwd
echo Uebergabe-Parameter: $*
```

Das vorstehende einfache Script enthält im wesentlichen normale Unix-Kommandos. Abgesehen von der ersten Zeile liegt die einzige Besonderheit im Platzhalter "$*", der für alle Kommandozeilen-Parameter steht.

Testen eines Shell-Scripts

sh -n *myscript*
: Syntax-Test (die Kommandos werden gelesen und geprüft, aber nicht ausgeführt)

sh -v *myscript*
: Ausgabe der Shell-Kommandos in der gelesenen Form

sh -x *myscript*
: Ausgabe der Shell-Kommandos nach Durchführung aller Ersetzungen, also in der Form, wie sie ausgeführt werden

Kommandozeilen-Parameter

$0
: Name der Kommandoprozedur, die gerade ausgeführt wird

$#
: Anzahl der Parameter

$1 $2 $3 ...
: erster, zweiter, dritter ... Parameter

$*
: steht für alle Kommandozeilen-Parameter ($1 $2 $3 ...)

$@
: wie $* ($1 $2 $3 ...)

"$@"
: expandiert (im Unterschied zu "$*") zu:
"$1" "$2" "$3" ...

$$
: Prozeßnummer der Shell (nützlich, um eindeutige Namen für temporäre Dateien zu vergeben)

$-
: steht für die aktuellen Shell-Optionen

$?
: gibt den Return-Code des zuletzt ausgeführten Kommandos an (0 bei erfolgreicher Ausführung)

$!
: Prozeßnummer des zuletzt ausgeführten Hintergrund-Prozesses

Beispiel:

```
#!/bin/sh
# Variablen
echo Uebergabeparameter: $*
echo user ist: $USER
echo shell ist: $SHELL
echo Parameter 1 ist: $1
```

```
echo Prozedurname ist: $0
echo Prozessnummer ist: $$
echo Anzahl der Parameter ist: $#
a=17.89       # ohne Luecken am = Zeichen
echo a ist $a
```

Prozeßsteuerung

Bedingte Ausführung: if

if [*bedingung* **]**
  then *kommandos1*
  else *kommandos2*
fi

Anmerkungen: "fi" ist ein rückwärts geschriebenes "if", es bedeutet "end if" (diese Schreibweise ist eine besondere Eigenheit der Bourne Shell). Die "bedingung" entspricht der Syntax von test, siehe auch weiter unten. Im if-Konstrukt kann der "else"-Zweig entfallen, andererseits ist eine Erweiterung durch einen oder mehrere "else if"-Zweige möglich, die hier "elif" heißen:

if [*bedingung1* **]**
   then *kommandos1*
  elif [*bedingung2* **]**
   then *kommandos2*
  else *kommandos3*
fi

Die Formulierung
    if [*bedingung* **]**
ist äquivalent zu
    if test *bedingung*
Alternativ ist es möglich, den Erfolg eines Kommandos zu prüfen:
    if *kommando*
    (beispielsweise liefert das Kommando "true" stets "wahr", das Kommando "false" hingegen "unwahr")

Wichtige Vergleichsoperationen (test)

Hinweis: Es ist unbedingt notwendig, daß alle Operatoren von Leerzeichen umgeben sind, sonst werden sie von der Shell nicht erkannt! (Das gilt auch für die Klammern.)

Zeichenketten

"s1" = "s2"
    wahr, wenn die Zeichenketten gleich sind
"s1" != "s2"
    wahr, wenn die Zeichenketten ungleich sind
-z "s1"
    wahr, wenn die Zeichenkette leer ist (Länge gleich Null)
-n "s1"
    wahr, wenn die Zeichenkette nicht leer ist (Länge größer als Null)

(Ganze) Zahlen

n1 -eq n2

wahr, wenn die Zahlen gleich sind

n1 -ne n2

wahr, wenn die Zahlen ungleich sind

n1 -gt n2

wahr, wenn die Zahl n1 größer ist als n2

n1 -ge n2

wahr, wenn die Zahl n1 größer oder gleich n2 ist

n1 -lt n2

wahr, wenn die Zahl n1 kleiner ist als n2

n1 -le n2

wahr, wenn die Zahl n1 kleiner oder gleich n2 ist

Sonstiges

!

Negation

-a

logisches "und"

-o

logisches "oder" (nichtexklusiv; -a hat eine höhere Priorität)

\(... \)

Runde Klammern dienen zur Gruppierung. Man beachte, daß sie durch einen vorangestellten Backslash, \, geschützt werden müssen.

-f *filename*

wahr, wenn die Datei existiert. (Weitere Optionen findet man in der man page zu test)

Beispiel:

```
#!/bin/sh
# Interaktive Eingabe, if-Abfrage
echo Hallo, user, alles in Ordnung?
echo Ihre Antwort, n/j:
read answer
echo Ihre Antwort war: $answer
# if [ "$answer" = "j" ]
if [ "$answer" != "n" ]
  then echo ja
  else echo nein
fi
```

Mehrfachentscheidung: case

case *var* **in**
  muster1) *kommandos1* **;;**
  muster2) *kommandos2* **;;**
  *) *default-kommandos* **;;**
esac

Anmerkungen: "esac" ist ein rückwärts geschriebenes "case", es bedeutet "end case". Die einzelnen Fälle werden durch die Angabe eines Musters festgelegt, "muster)", und durch ein doppeltes Semikolon abgeschlossen. (Ein einfaches Semikolon dient als Trennzeichen für Kommandos, die auf derselben Zeile stehen.) Das Muster "*)" wirkt als "default", es deckt alle verbleibenden Fälle ab; die Verwendung des default-Zweiges ist optional.

Beispiel:
```
#!/bin/sh
# Interaktive Eingabe, Mehrfachentscheidung (case)
echo Alles in Ordnung?
echo Ihre Antwort:
read answer
echo Ihre Antwort war: $answer
case $answer in
  j*|J*|y*|Y*) echo jawohl ;;
  n*|N*) echo nein, ueberhaupt nicht! ;;
  *) echo das war wohl nichts ;;
esac
```

Schleife: for

for *i* **in** *par1 par2 par3 ...*
  do *kommandos*
done

Anmerkungen: Die for-Schleife in der Bourne Shell unterscheidet sich von der for-Schleife in üblichen Programmiersprachen dadurch, daß nicht automatisch eine Laufzahl erzeugt wird. Der Schleifenvariablen werden sukzessive die Parameter zugewiesen, die hinter "in" stehen. (Die Angabe "for i in $*" kann durch "for i" abgekürzt werden.)

Beispiel:
```
#!/bin/sh
# Schleifen: for
echo Uebergabeparameter: $*
# for i
for i in $*
  do echo Hier steht: $i
done
```

Schleife: while und until

while [*bedingung* **]**
  do *kommandos*
done
until [*bedingung* **]**
  do *kommandos*
done

Anmerkung: Bei "while" erfolgt die Prüfung der Bedingung vor der Abarbeitung der Schleife, bei "until" erst danach. (Anstelle von "[*bedingung*]" oder "test *bedingung*" kann allgemein ein Kommando stehen, dessen Return-Code geprüft wird; vgl. die Ausführungen zu "if".)

Beispiel:
```
#!/bin/sh
# Schleifen: while
# mit Erzeugung einer Laufzahl
i=1
while [ $i -le 5 ]
do
  echo $i
  i=`expr $i + 1`
done
```