Informationstechnische Assistenten **Betriebssysteme Netzwerke**	*Bash-Scripte* *Teil 1* *Meinolf Lutter*

Linux-Kurs: `bash`-Skripte

`bash`-Skripte sind eine schöne Sache um immer wieder kommende Befehle zu automatisieren.
Ein `bash`-Skript ist quasi eine Batch-Datei unter DOS, aber leistungsfähiger ;-)). Unter Windows ist es echomit der PowerShell vergleichbar.

Hallo Tux

- Zuerst schreiben wir mit einem beliebigen Text-Editor die Datei "hallo.sh". Die hat diesen Inhalt:

```
echo "Hallo
Tux!"    echo
```

- Damit das Skript ausführbar wird müssen die Dateirechte geändert werden:

```
[tux]$ chmod u+x ./hallo.sh
```

- Und nun die Datei im aktuellen Verzeichnis ausführen:ech

```
[tux]$ ./hallo.sh
Hallo Tux!
```

Das war es schon. Auf dem Monitor sollte nun "Hallo Tux!" sehen.

Jetzt mit etwas mehr Anspruch:

```
#!/bin/bash
#Programm hallo2.sh!

echo "Wie heißt du ?"
read Name



echo "Hallo $Name"
```

In der ersten Zeile wird der bash mitgeteilt , dass dieses Skript von `/bin/bash` ausgeführt werden soll. Das ist eigentlich nicht umbedingt nötig, aber eleganter und kann Fehler vermeiden helfen.
In der zweiten Zeile steht ein Kommentar, "#", dieses leitet fast immer einen Kommentar ein. Warum fast immer ?
Sehen Sie in die erste Zeile, was es dort im Zusammenhang mit dem "!" heißt ...
In der dritten Zeile geben wie einen einfachen Text aus, und in der vierten Zeile lesen wir mit "read" eine Variable (ein Variable ist eine Art Speicher) ein, hier "Name".
In der letzen Zeile geben wir das Hallo mit dem eben eingegebenen Namen aus. Wird vor einer Variablen ein "$" geschrieben, wird der Inhalt der Variable ausgeben, und das ist der Name den du eben eingegeben hast.

Parameterübergabe

Jetzt schreiben wir ein Programm, das direkt Parameter beim Kommandoaufruf benutzt.
Erst mal das Programm:

```
#!/bin/bash
#Programm hallo3.sh
echo "Hallo $1"
```

Dieses Programm rufst du nun so auf:

```
[tux]$ ./hallo3.sh vorname
Hallo vorname
```

wobei vorname natürlich ihr Vorname ist ...
Darauf sollte ihr Programm "Hallo vorname" ausgeben.

If-Else Anweisung

Eine If-Else Anweisung läßt sich ungefähr mit "Wenn ... dann" übersetzen. Hier sehen sie schon wofür sie diese brauchen:
Sie können einfach bestimmte Dinge abfragen.

Am klarsten wird das an einem Beispiel-Programm:

```
#!/bin/bash
#Programm if.sh

if [ "$1" = "tux" ]; then
   echo "Hallo Pinguin"
   #Wenn der erste Parameter tux ist, schreibe Hallo Pinguin

else
    echo "Hallo $1"
    #Sonst schreibe Hallo Parameter
fi

exit 0
```

Einfach mal ausprobieren mit:

```
[tux]$ ./if.sh tux
Hallo Pinguin
```

und danach:

```
[tux]$ ./if.sh markus
 Hallo markus
```

Damit ist die If-Else Anweisung klar, oder?

Und jetzt mal die Syntax:
Die gesamte If-Else Anweisung beginnt mit dem if und endet mit fi (fi = if umgedreht). Alles dazwischen gehört zur If-Abfrage.
Die wichtigste Zeile:

```
if [ "$1" = "tux" ]; then
```

Hier sagst du:
Wenn (if) der erste Parameter ("$1") gleich (=) Tux ("tux") ist, dann (then)...
Danach schreiben sie was passieren soll wenn die Anweisung zutrifft (echo "Hallo Pinguin").

Ab der Zeile mit `else` steht was passieren soll, wenn `if` nicht paßt (echo "Hallo $1").
In der letzen Zeile steht ein `exit 0`. Damit übergiben sie dem nachfolgenden Programm einen sogenannten Rückgabe-Parameter. Damit können sie dem nachfolgenden Programm anzeigen, ob das Skript erfolgreich beendet wurde oder ein Fehler aufgetreten ist. Übrigens, viele Unix/Linux-Programme nutzen diese Option.

Achtung!! Die Syntax der If-Else Anweisung muß genauso geschrieben werden (die Leerzeichen sind EXTREM wichtig !), sonst geht deine If-Abfrage nicht.

Noch ein Tipp: Wollen sie Zahlen vergleichen mit == , >= , < ; müssen sie -eq (== equal), -ge (>= greater equal), lt (< less than) nehmen.

Case Anweisung

If-Else ist ja ganz nett, aber wenn sie mehrere verschieden Parameter abfragen wollen, wird es schon etwas komplexer. Hier hilft die Case Anweisung.
Man kann sie so beschreiben: Wenn 1 dann das, Wenn 2 dann dieses, Wenn 3 dann dasda, Wenn nichts davon dann...

Auch hier wieder ein erklärendes Beispiel:

```
#!/bin/bash
#Programm case.sh

case "$1" in
     tux)
            echo "Hallo Pinguin"
                 #Wenn Parameter 1 = tux
            ;;
     markus)
            echo "Hallo, alter Sack ;-)"
                 #Wenn Parameter 1 = markus
            ;;
     *)
             echo "Hast du keinen richtigen Namen;-)"
                  #Wenn vorher nichts passte
             ;;
esac

exit 0
```

Einfach mal ausprobieren mit:

```
[tux]$ ./case.sh tux
Hallo Pinguin
```

und dann:

```
[tux]$ ./case.sh markus
Hallo, alter Sack ;-)
```

und danach:

```
[tux]$ ./case.sh dave
Hast du keinen richtigen Namen ? ;-)
```

Eigentlich ist die damit Syntaxschon fast klar:
Zuerst sagen sie worauf die Case-Anweisung reagieren soll: Hier auf Parameter 1 also `case "$1" in`. Danach wird der Reihe nach abgeklappert ob eine Bedingung paßt. Unsere erste Anweisung beschreibe ich mal genauer: Zuerst steht in der Anweisung : `tux)`, und das heißt: wenn der erste Parameter "tux" ist, dann schreibe "Hallo Pinguin" (echo "Hallo Pinguin").
Wichtig ist die Anweisung mit " ;; " abzuschließen !!
Mit der letzen Anweisung (*)) wird alles abgefangen, was noch übrig ist, quasi eine default-Anwendung.

---

while-Schleifen

While-Schleifen laufen gewisse Anweisung solange durch, solange die Bedingung erfüllt ist.
Das kann man gebrauchen, wenn man die Zahlen 1-10 ausgeben will, die Schleife muß solange laufen, bis der Zähler nicht kleinergleich als 10 ist (immer dran denken: solange die Bedingung erfüllt

ist).

Auch hier wieder ein Beispiel:

```
#!/bin/bash
#Programm while.sh

count=0           #Zähler auf Null setzen

#So lange durchlaufen bis count nicht mehr unter 10 ist
while [ $count -le 10 ]
     do

        echo $count        #Zähler ausgeben
        count=$[$count+1]  #Zähler um eins erhöhen
      done

exit 0
```

Und einfach ausführen: Jetzt werden die Zahlen von 0 bis 10 hochgezählt:

```
[tux]$ ./while.sh
0
1
2
3
4
5
6
7
8
9
10
```

Leider ist die Syntax hier etwas schwieriger:
Zuerst setzen wir unseren Zähler count auf 0: `count=0`
Jetzt kommt die while-Bedingung: `while [ $count -le 10 ]` ; das dürfte dir zum Teil schon von der If-Else Anweisung bekannt vorkommen. Also, wir setzen hier als Bedingung count kleinergeich 10 ($count -le 10).
In der `do`-Anweisung steht drin, was ausgeführt wenn die Bedingung erfüllt ist. Wir geben hier den Wert des Zähler aus (echo $count) und erhöhen den Zähler um eins (count=$[$count+1]).

for-Schleifen

Die for-Schleife verhält sich unter der `bash` anders als unter C, und deswegen spreche ich sie nur ganz kurz hier an.

Hier ein Beispielprogramm:

```
#!/bin/bash
#Programm for.sh

for i in "Hallo du da am Monitor”
    do
      echo $i
    done

exit 0
```

Einfach ausführen und das Ergebnis bitte erst mal selber auswerten:

```
[tux]$ ./for.sh
Hallo du
da
am Moni
t
or
```

For-Schleifen arbeiten der Reihe nach die angegebene Liste ab. Für Skripte ist das kaum zu gebrauchen aber für Dateioperationen ganz nützlich :

```
[tux]$ for i in *.sh; do cp $i $i.bak; done
```

.
Damit wird von alle Dateien im aktuellen Verzeichnis die auf .sh enden eine Kopie erstellt. Diese Kopie bekommt am Ende ein .bak angehängt.

Um diese Übung durchzuführen muss ein Programm nachinstalliert werden.
Dieses funktioniert mit :
sudo apt install dialog

Dialog

Zum Ende kommen wir zur grafischen Programmierung der `bash` (naja, es ist bunt und hat Felder).
Das kann die `bash` nicht selber dafür brauchen wir einen kleinen Helfer. Dieser heißt "dialog".
Hier wieder mal ein Beipiel (ist das letztnane auf dieser Seite):

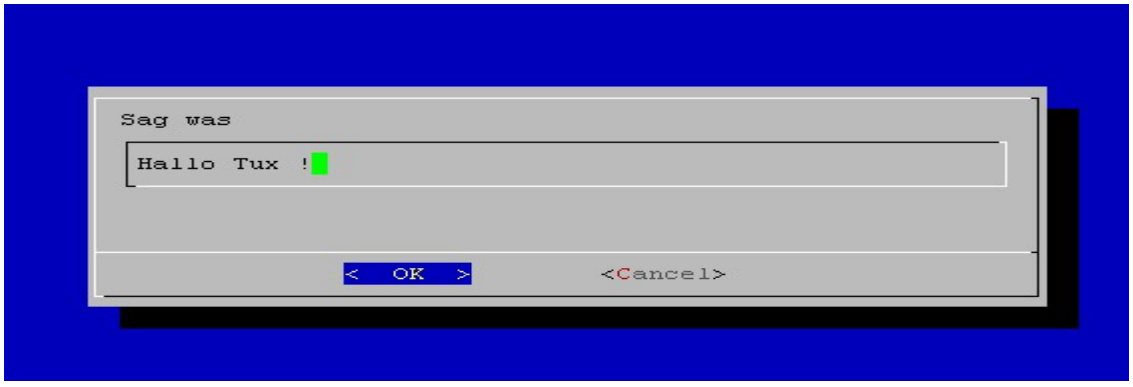
```
#!/bin/bash
#Programm dialog.sh

dialog --clear --inputbox  "Sag was" 10 60 2> eingabe.tmp

clear
echo  ""
printf "Du sagtest: "
cat eingabe.tmp
echo  ""
exit 0
```

Und ausprobieren ... :

```
[tux]$ ./dialog.sh
```



```
Sie sagten: Hallo Tux !
[tux]$
```

Das Wichtigste ist, dass du den eingegeben Text von dialog nicht direkt nutzen kannst, du mußt ihn in eine Datei schreiben (2> eingabe.tmp).
Dann lesen wir diese Datei aus (`cat eingabe.tmp`) und geben den Text aus. Hier sind die Anführungszeichen sehr wichtig (es sind die mit Shift+TasteNebenBackspace).