Siehe hierzu auch "Linux basic - die shell"!
(Variablen, Pipes, Verkettung von Befehlen....)

Variablen	
$* oder "$@"	Alle übergebenen Variablen als Liste
$* oder $@	Alle übergebenen Variablen als ein Wort
$#	Anzahl der übergebenen Werte
$?	Rückgabewert des zuletzt ausgeführten Kommandos
$$	Prozess-ID der aktuellen Shell
$!	Prozess-ID des zuletzt gestarteten Subprozesses
$0	Der Programmname
$1 - $9	Die ersten 9 Übergabeparameter
shift verschiebt die Übergabeparameter um eins nach links. Der linkste Wert fällt nach "Lemmingmethode einfach über die Klippen". D.h. der ehemalige Wert von $2 ist nun der Wert von $1. Auf den Wert von $1 können Sie zu diesem Zeitpunkt nicht mehr zugreifen!	
[VARIABLE]="[WERT]"	Eine Variable VARIABLE auf einen WERT setzen
$[VARIABLE]	Zugriff auf den Wert der Variable. z.B. echo $VARIABLE.

Rechenoperationen mit dem Befehl expr	
Addition	ERGEBNIS=`expr 1 + 2`
Subtraktion	ERGEBNIS=`expr 1 - 2`
Multiplikation	ERGEBNIS=`expr 2 * 2`
Ganzzahlige Division	ERGEBNIS=`expr 8 / 2`
Rest einer Division	ERGEBNIS=`expr 7 % 2`
Binäres Oder	ERGEBNIS=`expr 1 | 0`
Binäres Und	ERGEBNIS=`expr 1 & 1`
Vergleichsoperationen	< <= = == != >= >
Die Beispiele zeigen auch, wie man eine Variable mit dem Wert des Ergebnisses eines Befehls setzt. Auf diesem Weg können Sie Ergebnisse anderer Befehle, wie z.B. "date", "ifconfig" etc. in einer Variable auffangen. Das Ergebnis ist dann der Wert der Variable, auf den Sie mit einem vorangestellten $ zugriff haben. Beispiel: echo $ERGEBNIS - gibt das Ergebnis auf den Bildschirn aus	
Ergebnisse von Vergleichsoperationen sind 1=wahr 0=unwahr	

Umsetzungen für "Standard-For-Schleifen"

```
for ZAEHLER in `seq 1 100`
do
echo "Durchlauf $ZAEHLER"
done
```

`[BEFEHL]` beeinhaltet den Rückgabewert des Kommandos!!!

```
ZAEHLER=1
while [ $ZAEHLER -ne 101 ]
do
ZAEHLER=`expr $ZAEHLER + 1`
echo "Durchlauf: $ZAEHLER"
done
```

for-Schleife

Die For-Schleife ist nicht klassisch als "Zählwerkzeug" belegt, wie in anderen Programmiersprachen, sondern durchläuft eine Liste. Wenn Sie eine Zählschleife wünschen, müssen Sie mit einer while-Schleife und einer arithmetischen Operation (expr) arbeiten.

```
for [VARIABLE] in [LISTE]
do
....
done
```

while-Schleife

Eine While-Schleife wird nur durchlaufen, wenn die Bedingung wahr ist

```
while [Bedingung]
do
....
done
```

until-Schleife

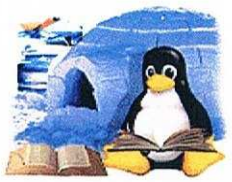
Eine until-Schleife wird mind. einmal durchlaufen.
Mehrmals, solange die Bedingung wahr ist

```
until [Bedingung]
do
....
done
```

Startskripten (Ziel des Programmierexkurses)

```
#! /bin/sh
# 2004-12-20-bs-0001 bs@muekno.de
start ()
{
Datum=`date`
echo "Starte den Dienst um $Datum"
}
stop ()
{
Datum=`date`
echo "Stoppe den Dienst um $Datum"
}
case $1 in
        status) PIDS=`ps aux | grep sdjadhagh | grep -v grep | cut -b9-14`
            if [ "$PIDS" = "" ]
            then echo "Is not running!"
            else echo "Is running with PID(s): $PIDS"
            fi
            }
        start)  start ;;
        stop) stop ;;
        restart) stop; start ;;
        *) echo "Usage $0 [start|stop|restart|status]
esac
```

Startskripten müssen mindestens auf die Übergabewerte ($1) start und stop reagieren. Sie liegen in der Regel in /etc/init.d. Starten (Stoppen = K99startskript) in Runlevel 3:
ln -s /etc/init.d/startskript.sh /etc/rc.d/rc3.d/S99startskript



Linux basic

programmierung

Version 2005-11-29

Dipl.Inf. (FH) Bernd Schwägerl
Dipl.Inf. (FH) Stefan Edenhofer

Abfragen

if-Statement:

```
if [Bedingung]
then
 echo „Bedingung war wahr=0"
else
 echo „Bedingung war unwahr=1"
fi
```

case-Statement:

```
case [VARIABLENWERT] in
                    1) echo „war der Wert 1" ;;
                    2) echo „war der Wert 2" ;;
                    *) echo „war irgendwas anderes ;;
esac
```

Unterrroutinen (Funktionen)

Funktionen definieren:

```
[funktionsname] ()
{
# Ausgabe des Übergabeparameters
echo $1
}
```

Funktionen aufrufen:

```
[funktionsname] UEBERGABEPARAMETER
```

Teststatements (Gebräuchlichste Art der Bedingungen)	
[$A = $B]	$A gleich $B (Chars, Strings)
[$A != $B]	$A nicht $B (Chars, Strings)
[$A -eq $B]	$A = $B (nummerisch)
[$A -ne $B]	$A nicht $B (nummerisch)
[$A -lt $B]	$A < $B (nummerisch)
[$A -gt $B]	$A > $B (nummerisch)
[$A -le $B]	$A <= $B (nummerisch)
[$A -ge $B]	$A=> $B (nummerisch)
[-r file]	file existiert und ist lesbar
[-w file]	file existiert und ist beschreibbar
[-x file]	file existiert und ist ausführbar
[-f file]	file ist ein FILE und existiert
[-d file]	file ist ein DIR und existiert
Verknüpfen von Teststatements (AND und OR):	
[$A > $B -o $C < $B]	OR-Verknüpfung
[$A > $B -a $C < $B]	AND-Verknüpfung

Da jeder Shellskriptaufruf einen Aufruf einer eigenen Shell darstellt, sind Variablen in oder aus "Subskripten" nicht verfügbar. (Abhilfe: ". /skript")