Vorlesung 4: Shell-Scripting (Teil 1)

Shell-Beispiele:

Bash	Weitestverbreitete Shell bei Linux-Systemen
Dash	Standard-Shell unter Ubuntu
Fish	Shell mit der expliziten Zielsetzung Benutzerfreundlichkeit
Z-Shell (zsh)	Mächtige Shell mit einem sehr ausgereiften Command-Editor
Korn-Shell (ksh)	Von David Korn an den AT&T Bell Laboratories entwickelter Kommandozeileninterpreter wie auch die Beschreibung der Scriptsprache, welche durch diesen Interpreter implementiert wird. Die Sprachbeschreibung selbst ist gemeinfrei, nicht jedoch jede Implementierung. Der originale Quellcode der ksh93 ist seit 2000 ebenfalls gemeinfrei. Die KornShell (in der Version von 1988) bildete die Grundlage für die POSIX-Shell-Spezifikation und erfüllt den POSIX2-Standard. Sie ist vollständig abwärtskompatibel mit der Bourne Shell (sh bzw. bsh) und übernimmt viele Neürungen der C Shell (csh)
Mksh	Direkte Nachfolger der Korn Shell /bin/ksh, ist aber nicht verbunden mit der AT&T Korn Shell und deren Programmierern; vielmehr ersetzt es die Public Domain Korn Shell pdksh, und implementiert gleichzeitig die Korn

	Shell programming language, wie sie auch als Untermenge in der Bash vorhanden ist. Gleichzeitig enthält seit Ubuntu 12.10 das mksh-Paket die Legacy Korn Shell lksh; lksh ist ein schlanker Kommandointerpreter, mit der Intention zum Ablauf von Shell-Skripten nach dem ksh-Standard. lksh und mksh unterliegen in keinem Fall der ShellShock-Problematik. Die mksh ist an Stelle der Bash im Android-System vertreten, und nach Aussagen der Firma Google damit häufiger in UNIX-artigen Systemen vertreten als die Bash

Bei vielen Linux-Distributionen ist die Standard-Shell die *bash*.

Um ein Script script.sh zu starten muss man es mit *chmod +x script.sh* ausführbar machen und *./script* in das Terminal eingeben.
Um ein Script script.sh mit einer bestimmten Shell zu starten muss man <shell> *script.sh* eingeben.
Beispiel: *bash script.sh*
Aber am besten ist es am Anfang den Pfad zu der Shell zu schreiben. Das sagt dem System mit welcher Shell das Script ausgefürt wird. Aber davor muss man #! schreiben.
Beispiel: *#!/bin/bash #Das Script wird mit bash gestartet*

Kommentare:

#<text>
Beispiel: *#Das ist ein Script*
Anmerkung: *#!* ist kein Kommentar

Ein- und Ausgabe:

Ausgabe:
echo <parameter>
Beispiel: *echo "Hello World" #Gibt "Hello World" aus*

Der Zeilenumbruch erfolgt automatisch
Die Ausgabe kann auch farbig gemacht werden:
echo -e <Steuerzeichen><parameter><Steuerzeichen>
Beispiel:
echo -e "\033[32mHello World\033[0m" #Gibt "Hello World" in grün aus

Farben:

\033[0m	alle Attribute zurücksetzen
\033[1m	**Fettschrift**
\033[4m	Unterstreichen
\033[5m	Blinken
\033[7m	inverse Darstellung
\033[30m	Schriftfarbe schwarz
\033[31m	Schriftfarbe rot
\033[32m	Schriftfarbe grün
\033[33m	Schriftfarbe gelb
\033[34m	Schriftfarbe blau
\033[35m	Schriftfarbe magenta
\033[36m	Schriftfarbe türkis
\033[37m	Schriftfarbe weiß
\033[40m	Hintergrund **schwarz**
\033[41m	Hintergrund **rot**
\033[42m	Hintergrund **grün**
\033[43m	Hintergrund **gelb**
\033[44m	Hintergrund **blau**
\033[45m	Hintergrund **magenta**
\033[46m	Hintergrund **türkis**
\033[47m	Hintergrund weiß

Man kann etwas in eine Datei ausgeben/schreiben:
echo <parameter> > <datei>
Beispiel: *echo "Hello World" > hello.txt #Übersreibt alles in Datei hello.txt mit "Hello World"*

echo <parameter> >> <datei>
Beispiel: *echo "My name is Alex" >> hello.txt #Fügt "My name is Alex" am Ende der Datei hinzu*
Wenn man eine Variable ausgeben will, dann muss man ein '$'-Zeichen davor setzen
echo <text>*$*<var><text>
Beispiel: *echo "Wert von i ist $i" #Gibt den text "Wert von i ist " und den Inhalt der Variable i aus*
Anmerkung: Wenn man ein '$'-Zeichen ausgeben möchte muss man \$ schreiben.
Es gibt folgende Ausdrücke:

\a	Gibt einen Ton aus
\n	Neü Zeile
\\	Gibt \ aus
\"	Gibt " aus
\'	Gibt ' aus
\	Gibt ein Leerzeichen aus
\c	Verhindert den Zeilenumbruch
\$	Gibt $ aus

Anmerkung: Bei \n, \a und \c muss man *echo -e* schreiben
Den Zeilenumbruch kann man auch mit -n verhindern. Zum Beispiel:
echo -n "Hello World"
Als Ausgabe kann man auch *printf* nutzen:
printf "<text>*%*<datentypzeichen><text>*" $*<variablenname>
Beispiel:
i=5
printf "Wert von i ist %d\n" $i #Gibt "Wert von i ist 5" aus
Anmerkung: Die Zeichen *\n*, *\a* und andere. werden bei *printf* ganz normal akzeptiert. Aber es gibt weder *\c* oder *\n*, da der Zeichenumbruch nicht automatisch erfolgt.
Wichtige Datentypenzeichen:

%d	Integer (Ganze Zahl)
%f	Float (Gleitkommazahle)
%c	Char (Zeichen)
%s	String (Zeichenkette)

Eingabe:
read <variable>
Beispiel: *read var #Eingabe von Variable var*
Man kann die Eingabe mit einem Text zusammensetzen:
read -p <text> <variable>
Beispiel: *read -p "Eingabe: " i #Gibt "Eingabe:" aus und lässt die Variable i eingeben*

Variablen:

Um eine Variable zu deklarieren muss man gleichzeitig auch einen Wert zuweisen. Dass nennt sich Initialisierung
Beispiel: *i=0 #Variable i wird deklariert und mit 0 initialisiert*
Um aud den Wert von der Variable zugreifen zu können muss man ein '$'-Zeichen davor schreiben.
Beispiel: *$i #Wert von Variable i*
Um arithmetische Zuweisungen auszuführen muss man *let* davor schreiben
Beispiel: *let i=$i+1 #Variable i wird um 1 erhöht*
Mann kann auch inkrementieren oder dekrementieren, aber dann muss der Ausdruck in doppelte Klammern gesetzt werden.
Beispiel: *((i++))*
Um 2 Zeilen zu konkatenieren muss man die Variablen hintereinanderschreiben
Beispiel:
i="Hallo " #in der Zeile ist ein Leerzeicehn enthalten
k="Welt"
i=ik
echo "$i" #Gibt "Hallo Welt" aus
Beispiel:
i="Hallo" #in der Zeile ist kein Leerzeichen enthalten
k="Welt"
i=$i\ $k #es wird ein Leerzeichen zwischen den Variablen gesetzt
echo "$i" #Gibt "Hallo Welt" aus

Arrays:
Ein Array ist eine endliche Folge von Werten, die in eine Variable gespeichert werden. So sieht ein Array aus:
<array>=(<variable 1> <variable 2> ... <variable n>)
Beispiel: *arr=(one two three four five) #Array arr hat 5 Elemente*
Jedes Element hat ein Index, d.h. die Nummer der Reihenfolge.

Das erste Element hat den Index 0, das zweite den Index 1 usw.
Um das ganze Array auszugeben benutzt man *${arr[*]}*
* ist der Allfilter, so werden alle Elemente ausgegeben.

Man kann ein bestimmtes Element verändern, indem man darauf durch den Index zugreift.

Beispiel:
arr=(1 2 3 5 5 6)
echo ${arr[*]}
arr[3]=4
echo ${arr[*]}
#Die Ausgabe ist folgendes:
#1 2 3 5 5 6
#1 2 3 4 5 6
Um alle Indices des Arrays auszugeben benutzt man *${!arr[*]}*
Beispiel:
arr=(1 2 3 4 5)
echo ${!arr[*]}
#Ausgabe: 0 1 2 3 4
Um die Länge des Array auszugeben, benutzt man *${#arr[*]}*
Beispiel:
arr=(1 2 3 4 5)
echo ${#arr[*]}
#Ausgabe: 5
Um die Länge eines Elements auszugeben, benutzt man
${#arr[<Index>]}
Beispiel:
arr=(Hallo ich bin Alex)
echo ${#arr[1]}
#Ausgabe: 3

if-else-Anweisungen:

Die if-else-Anweisungen sind Fallunterscheidungen. Es wird überprüft ob eine Bedingung stimmt und dann die dazugehörigen Schritte ausgeführt.
Die if-else-Anweisung sieht wie folgt aus:

if [<Bedingung>]
then
    <Code>
else
    <Code>
fi

Zuerst müssen wir noch die Vergleichsoperatoren anschauen:

-eq	gleich (equal)
-ne	ungleich (not equal)
-gt	größer (greater than)
-lt	kleiner (lesser than)
-ge	größer oder gleich (greater or equal)
-le	kleiner oder gleich (lesser or equal)
-a	und (and)
-o	oder (or)
!	nicht (not)

Mann kann auch normale Vergleichsoperatoren benutzen, aber dann muss man den Ausdruck in Doppelklammer setzen.
Beispiel: *(($i>0))*

Bei den Zeichenketten sind die Vergleichsoperatoren anders:

<String 1> = <String 2>	String 1 gleich String 2
<String 1> != <String 2>	String 1 ungleich String 2
<String>	Die Länge ungleich 0
-n <String>	Die Länge ungleich 0
-z <String>	Die Länge gleich 0

Beispiel:

```
#!/bin/bash
read -p "Eingabe: " i                          #Eingabe von i
if (($i<=0))                                   #Wenn i größer/gleich 0 ist
then                                           #dann
        echo "$i ist größer oder gleich 0"     #Ausgabe: "i ist größer oder
                                               #gleich 0"
else                                           #Anderenfalls
        echo "$i ist kleiner 0"                #Ausgabe: "i ist kleiner 0"
fi                                             #if-else-Ende
```

Man muss nicht unbedingt *else* benutzen, wenn man das nicht braucht. Aber, wenn man mehrere Fälle haben will, kann man den Befehl *elif* benutzen.

Beispiel:

```
#!/bin/bash
read -p "Eingabe: " i                          #Eingabe von i
if [ $i -gt 0 ]                                #Wenn i größer/gleich 0 ist
then                                           #dann
        echo "$i ist größer 0"                 #Ausgabe: "i ist grösser 0"
elif [ $i -eq 0 ]                              #Wenn i gleich 0 ist
then                                           #dann
        echo "$i ist gleich 0"                 #Ausgabe: "i ist gleich 0"
else                                           #Anderenfalls
        echo "$i ist kleiner 0"                #Ausgabe: "i ist kleiner 0"
fi                                             #if-else-Ende
```

Mann koennte auch so was wie *if (($i>0))* schreiben. Und das hätte funktioniert.

Beispiel:

```
#!/bin/bash
read -p "Eingabe von i: " i          #Eingabe von i
read -p "Eingabe von k: " k          #Eingabe von k
if [ $i -eq 0 -a $k -eq 0 ]          #Wenn i=0 und k=0
then                                 #dann
        echo "i und k sind 0"        #Ausgabe: "i und k sind 0"
elif [ ! $i -eq 0 -a ! $k -eq 0 ]    #Wenn nicht i=0 und nicht k=0
then                                 #dann
        echo "i und k sind nicht 0" #Ausgabe: "i und k sind nicht 0"
elif [ ! $i -eq 0 -a $k -eq 0 ]      #Wenn nicht i=0 und k=0
then                                 #dann
        echo "k ist 0"               #Ausgabe: "k ist 0"
else                                 #Anderenfalls
        echo "i ist 0"               #Ausgabe: "i ist 0"
fi                                   #if-else-Ende
```

Beispiel:

```
#!/bin/bash
read -p "Eingabe von str: " str      #Eingabe von str
if [ $str ]                          #Wenn die Länge von str ungleich 0
then                                 #dann
        echo $str                    #Ausgabe von str
else                                 #Anderenfalls
        echo 0                       #Ausgabe: 0
fi                                   #if-else-Ende
```

Beispiel:

```
#!/bin/bash
read -p "Eingabe von str1: " str1            #Eingabe von str1
read -p "Eingabe von str2: " str2            #Eingabe von str2
if [ $str1 = $str2 ]                         #Wenn str1 gleich str2
then                                         #dann
        echo "str1 und str2 sind gleich"     #Ausgabe
else                                         #Anderenfalls
        echo "str1 und str2 sind ungleich"   #Ausgabe
fi                                           #if-else-Ende
```

Schleifen:

Eine Schleife ist eine Befehlsfolge, die mehrmals ausgeführt wird.
Ein Schleifendurchlauf nennt man Iteration.
In bash gibt es 2 Schleifenarten: for und while. Leider gibt es kein do-while.

for-Schleife:

for-Schleife sieht wie folgt aus:

```
for ((<var>=<var/wert>; <var><vergleich><var/wert>; <varveränderung>))
do
        <Code>
done
```

Beispiel:

```
#!/bin/bash
for ((i=1; $i <= 5; i++))   #Schleifenbedingung
do                          #Scheifenanfang
        echo "Iteration $i" #Ausgabe
done                        #Schleifenende


#Am Anfang ist i=1, bei jedem Schleifendurchlauf wird i um 1 erhöht
#(i++) = i=$i+1. Die Schleife laüft bis i<=5 ist. Dann höer sie auf.
#Ausgabe:
# Iteration 1
# Iteration 2
# Iteration 3
# Iteration 4
# Iteration 5
```

Anmerkung: Die üblichen Vergleichsoperatoren, Inkrementierung (++) und Dekrementierung (--) sind in der bash nur in for-Schleifen erlaubt.

Die for-Schleifen können auch gut mit Arrays arbeiten

```
for <var> in <array>
do
        <Code>
done
```

Beispiel:

```
#!/bin/bash
array=(one two three four five)        #Array aus 5 Elementen

echo "Array size: ${#array[*]}"        #Ausgabe von Arraylänge
echo "Array items:"                    #Ausgabe
for item in ${array[*]}                #für jede Variable "item" in Array
do
   printf "%s\n" $item                 #Ausgabe von Element (item)
done
echo "Array indexes:"                  #Ausgabe
for index in ${!array[*]}              #für jedes "index" in Arrayindicies
do
   printf "%d\n" $index                #Ausgabe von dem Index
done
echo "Array items and indexes:"        #Ausgabe
for index in ${!array[*]}              #für jedes "index"
do
   printf "%d: %s\n" $index ${array[$index]}
                                       #Ausgabe von Indicies und Elementen
done
```

while-Schleife:

Die while-Schleife sieht wie folgt aus:

```
while [ Bedingung ]
do
        <Code>
done
```

Beispiel:

```
#!/bin/bash
i=0                    #i=0
while [ $i -lt 5 ]     #Wird wiederholt, wenn i<5 ist
do                     #Schleifenanfang
        echo $i        #Ausgabe von i
        let i=$i+1     #i wird um 1 erhöht
done                   #Schleifenende
```

Funktionen:

Funktion ist eine Sammlung von Befehlen, die ein Ergebniss zurückliefern kann.
Die Funktionen in bash haben folgende Form:

```
function <funktionsname>(<parameter 1>, ... , <parameter n>)
{
        <Code>
}
```

Beispiel:

```
#!/bin/bash
function add()              #Funktion add
{
        let sum=$1+$2       #sum=parameter1+parameter2 (a+b)
        echo $sum           #Ausgabe von sum
}
read -p "Wert 1: " a        #Eingabe von a
read -p "Wert 2: " b        #Eingabe von b
add a b                     #Funktionsaufruf mit parameter a und b
```

Mann kann auch ein Wert zurückgeben, aber der kann 256 nicht überschreiten
Beispiel:

```
#!/bin/bash
function add()              #Funktion add
{
        let sum=$1+$2       #sum=parameter1+parameter2 (a+b)
        return $sum         #Gib Variable sum zurück
}
read -p "Wert 1: " a        #Eingabe von a
read -p "Wert 2: " b        #Eingabe von b
add a b                     #Funktionsaufruf mit parameter a und b
echo $?                     #Ausgabe von letzter return Anweisung
```

Wenn man größere Werte eingibt, kommen komische Werte raus. *return* ist besser für exit-code zu verwenden (Also geklappt oder nicht):

Beispiel:

```
function mastermind()           #Funktion mastermind
{
        if [ $1 -eq 5 ]         #Wenn parameter1=5 (number=5)
        then                    #dann
                return 1        #Gebe 1 zurück
        else                    #Anderenfalls
                return 0        #Gebe 0 zurück
        fi                      #if-else-Ende
}
read -p "Eingabe: " number      #Eingabe von number
mastermind $number              #Funktionsaufruf mit parameter number
if [ $? -eq 1 ]                 #Wenn return Wert ist 1
then                            #dann
        echo "Gewonnen"         #Ausgabe "Gewonnen"
else                            #Anderenfalls
        echo "Verloren"         #Ausgabe "Verloren"
fi                              #if-else-Ende
```