# Makrosprachen für die Computerverwaltung

Die Shell ist ein Interpretor der Kommandos interaktiv oder selbstständig ausführt. Die Shell ist eine Schnittstelle zwischen Benutzer und Betriebssystem.

Es gibt verschiedene Arten der Shell:

|  |  |  |
| --- | --- | --- |
| sh | Bourne Shell | Die Mutter aller Shells |
| csh | C-Shell | Shell mit C-ähnlicher Syntax |
| ksh | Korn Shell | C-orientierte Shell |
| zsh | Z-Shell | Erweiterte Shell, Bash kompatibel |
| bash | Bourne Again SH | Erweiterte Bourne Shell |

# Parameter

Wenn man zu einem Kommando Aufruf noch weitere Parameter mitübergeben möchte, so kann man dies direkt nach dem Aufruf des Kommandos machen.   
z.B.: ./makefiles.sh folders sample

Wenn man nun auf diese Parameter vom Skript aus zugreifen möchte, kann man dies mit $1 bzw. $2 machen.   
$1 beinhaltet in unserem Fall den Wert folders und $2 den Wert sample

Wenn man nun überprüfen möchte, ob der erste Parameter einem bestimmten Wert entspricht, kann man dies mit einem if machen. Ein if Befehl sieht wie folgt aus:

if [ condition1 ]  
then  
 command1  
 command2   
elif [ condition2 ]  
# entspricht else if  
then  
 command3   
else  
 default-command  
fi

# Pipes

Bei einer Pipe wird die Ausgabe eines Kommandos mit dem Eingabewert eines anderen Kommandos zusammengelegt. Beide Kommandos werden als separate Prozesse gleichzeitig gestartet.

# C-ähnliche Befehle mit ()

Um z.B. eine for Schleife ähnlich wie in C zu programmieren, benötigt man die ().

Eine For Schleife, die solange läuft wie i kleiner 5 ist und sich nach jedem Durchlauf um 1 erhöht, sieht dann wie folgt aus:

#!/bin/bash

for ((i = 0 ; i < 5 ; i++)); do

echo "hallo, no.:$i"

done

Im Vergleich dazu, die gleiche For Schleife in der Bash Skript Schreibweise:

#! /bin/sh  
NUMBERS="0 1 2 3 4"  
for i in `echo $NUMBERS`  
do  
 echo "hallo, no.:$i"  
done

# Einlesen von User Eingaben

Mit dem read Befehl in einem Skript wartet man auf eine Benutzereingabe in der Konsole. Nach dem Read Befehl muss man den Namen der Variable angeben, damit man den eingelesenen Wert anschließend später verwenden kann.

z.B.: read jahr

Der Befehl wartet auf eine Benutzereingabe und speichert den eingelesenen Wert nach betätigen der *ENTER* Taste durch den Benutzer in die Variable jahr. Mit $jahr kann man im Programm auf den Wert dann zugreifen.

# Sudo

Für einige Befehle benötigt man root Rechte. Damit man diese Rootrechte zum Ausführen eines Befehles bekommt, muss man den Befehl mit einem sudo Schlüsselwort beginnen. Wenn man dieses Schlüsselwort verwendet, ist es erforderlich das Root-Passwort einzugeben, ansonsten wird der Befehl nicht ausgeführt.

# Besonders wichtige Befehle für die Userverwaltung

adduser … legt Benutzer an

deluser … löscht Benutzer

addgrp … Gruppe erzeugen

passwd .. Kennwort ändern

chmod …ändert Rechte an Ordnern und Dateien

chown … ändert Besitzer einer Datei

chgrp … ändert Gruppenzugehörigkeit

usermod … ändert UID bzw Gruppenzugehörigkeit von Benutzern

befehl - - help gibt meist Hilfe zum Befehl

oder

man befehl liefert Manual zum Befehl

# Aufruf eines Bash Scriptes

Wenn man sich im selben Ordner wie das Skript befindet, kann man es über ./nameDesSkriptes.sh aufrufen und ausführen. Dies setzt allerdings voraus, dass man dem User vorher mit dem Befehl chmod u+x nameDesSkriptes.sh die Rechte zum ausführen der Datei gegeben hat.

U+x bedeutet, dass der Benutzer die Rechte zum executen bekommt.

# .profile

Um ein Script von überall aus aufrufen zu können, muss man in der .profile, welche sich unter /etc/profile befindet, eine neue Funktion hinzufügen.

Eine Function beginnt mit dem Schlüsselwort *function* und den Namen der Funktion.

z.B.: function makefiles()

{

command1

}

Wenn man die .profile Datei nun gespeichert hat, kann man vom gesamten System aus die Funktion makefiles() über die Konsole aufrufen.

Auch scripts, die im Ordner /home/user/bin liegen, können von überall aus ausgeführt werden.

Die Datei .bashrc funktioniert ähnlich wie die .profile, aber nur wen man in der bash Umgebung ist (indem man mit dem Befehl bash in diese Umgebung gewechselt hat)

## Beispiel für ein Bash Makro:

#!/bin/bash

for ((i=1;i<=35;i++)) do

(

username=user$i

echo $username

echo $username >dat.txt

echo $username >>dat.txt

sudo useradd -m $username

sudo passwd $username <dat.txt

sudo usermod -aG www-data $username

) done

erzeugt in der Schleife Zeichenketten in denen user1, user2 usw steht

Schreibt die Zeichenkette zweimal in die Datei dat,txt

legt Benutzer mit user1 user2 usw an und erzeugt home-Verzeichnisse für sie (mit -m)

setzt das Passwort auf user1 user2 usw in dem es das Kennwort und seine Bestätigung nicht von der Tastatur einliest, sondern von der Datei dat.txt wo es vorher 2mal reingeschrieben wurde

wenn man (( )) verwendet, kann man dazwischen c-Code schreiben, zumindest in der bash, deren Verantwortung für die Ausführung oben mit #/bin/bash rzwungen wird

eigentlich ist #.. eine Kommentarzeile, der Interpreter liest sie aber und startet dann das Script in der bash

$ vor der Variable ist ein bisschen wie der Inhaltsoperator bei Pointern in C

usermod fügt den Benutzer zur Gruppe www-data hinzu, was zB bei apache Servern wichtig ist, weil Benutzer, die auf die Seite zugreifen dem System als Benutzer www-data präsentiert werden.