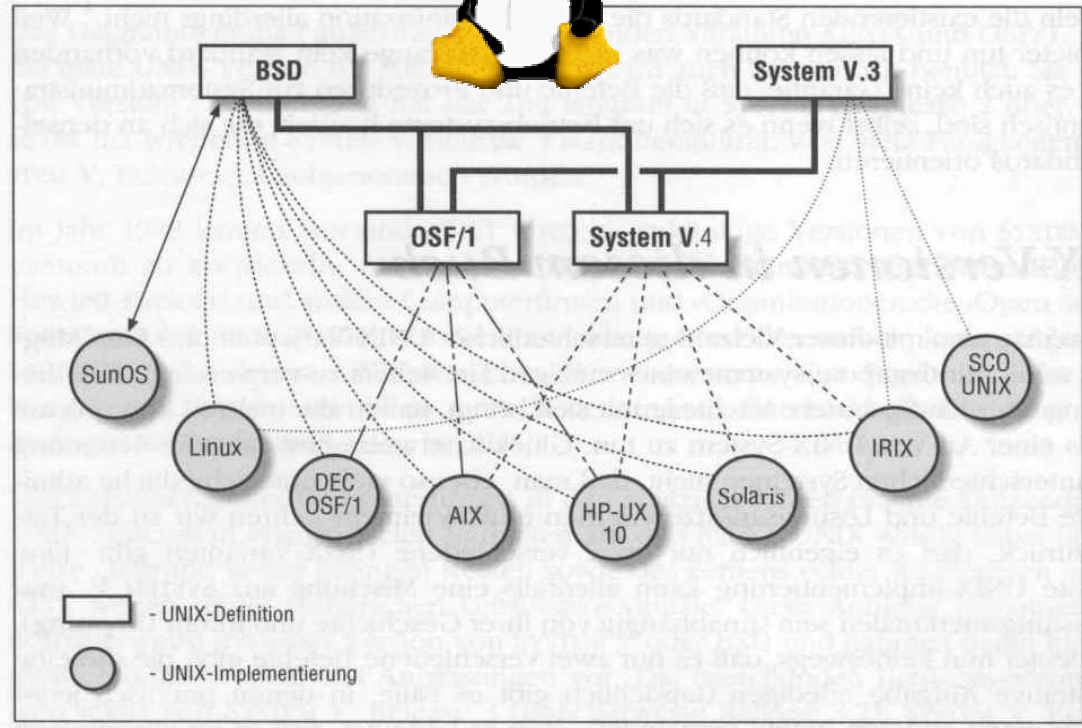


Linux

Linux Geschichte

UNIX-Stammbaum



Linux Bootvorgang

Runlevel

Runlevel kennzeichnen die verschiedenen Zustände des Systems.

Jedes Runlevel hat eine feste Nummer.

Das System der Runlevel ist dadurch entstanden, das Linux ein Multiuser Betriebssystem mit Netzwerkunterstützung ist.

Aus Sicherheitsgründen gibt es deshalb verschiedene Betriebszustände.

Runlevel 0 Systemhalt

Runlevel 1 Multi User ohne Netzwerk

Runlevel 2 Multi User mit Netzwerk

Runlevel 3 Multi User mit Netzwerk und X

Runlevel 4 nicht verwendet

Runlevel 5 SuSE <= 7.0 - nicht verwendet; ab SuSE 7.1 wie Runlevel 3

Runlevel 6 Reboot

Runlevel s Single User Modus

rc

rc ist ein Shell Script welches nach dem Booten von init aufgerufen wird

rc dient zur Ausführung aller Scripte die einem Runlevel zugeordnet sind

rc wird nur bei Multiuser Modus genutzt

Linux

init.d

Das Verzeichnis init.d enthält die für den Bootvorgang nötigen Skripte, Daemonen und Dienste. Jedes Runlevel Skript holt sich diese Dienste, Skripte und Daemonen aus diesem Verzeichnis.

init

Der Befehl init lässt den Rechner in ein bestimmtes Runlevel fahren.

```
init <Runlevel>  
linux#init 6
```

reboot

Dieser Befehl führt zum Neustart des Rechners, dabei initialisiert der Rechner das Runlevel 6 - entspricht also init 6.

shutdown <option>

Dient zum Herunterfahren des Systems, dabei bietet shutdown noch Optionen für das Herunterfahren des Rechners an, z. B. warnen aller User.

```
linux#shutdown -h +5  
linux#shutdown now
```

Linux Prozesse

Prozesse

Jeder Prozess hat einen PID - Prozess Ident
Prozesse können weitere Prozesse starten
Prozesse können abhängig von anderen Prozessen sein
Zur Steuerung von Prozessen gibt es LINUX-Befehle

ps

Der Befehl ps zeigt dem Nutzer alle Prozesse, die zur Zeit laufen an.

```
linux#ps  
linux#ps -ef  
linux#man ps
```

Standard - Devices

0 – stdin	Tastatur
1- stdout	Terminal
2 – stderr	Terminal

Standard- Devices können umgelenkt werden
linux#shellscript.sh &2>error.log

Linux

kill

Der Befehl Kill erlaubt es dem Benutzer einen Prozess (gewaltsam) zu beenden.

linux#ps

PID z.B. 12345 wird angezeigt

linux#kill -9 12345

nohup

Der Befehl nohup bewirkt, dass ein Prozess nach dem Abmelden nicht abgebrochen wird.

linux#nohup shellsript.sh

Man kann also eine Verbindung trennen und Stunden später den Prozess kontrollieren.

& führt einen Prozess im Hintergrund aus

linux#shellsript.sh &

linux#nohup shellsript.sh &

fg kann den Prozess wieder in den Vordergrund holen

bg kann den Prozess nachträglich in den Background verlegen

Linux Dateisystem

Linux unterstützt zahlreiche Dateisysteme, der Kernel und die Dateien des Lilo-Bootmanagers müssen zwingend auf einem ext2-Dateisystem gespeichert sein.

Das Dateisystem ext2

- ext2 organisiert das Dateisystem in Gruppen
- jede Gruppe beinhaltet
- eine Kopie des Superblocks
- eine Liste aller Gruppendeskriptoren
- die Bitmaps für Inodes und Blöcke
- die Inodes
- Datenblöcke und die Listen derselbigen



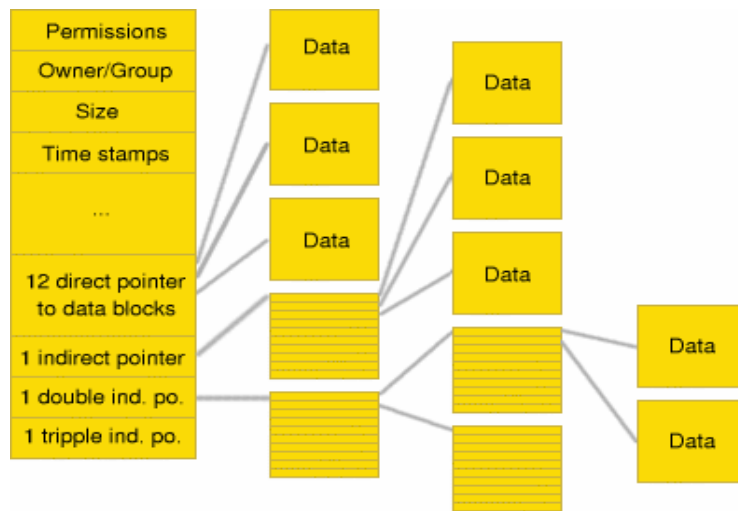
Das Prinzip des Inodes

Verwaltungsinformationen werden von den eigentlichen Daten getrennt gespeichert.

Die Verwaltungsdaten einer Datei enthalten alle Merkmale der Datei.

Diese Daten werden in Inode genannten Speicherbereichen abgelegt.

Inode



Inodes Dateigröße

- die Inodes erlauben Dateien bis zu einer Größe von 2 GByte
- Anzahl der Dateien je Verzeichnis ist beschränkt
- dies entsteht durch die Handhabung des Kernel, der diese nur mit 32 Bit speichert

Ablauf bei der Erstellung einer neuen Datei

- sobald eine neue Datei erzeugt wird, sucht ext2 einen freien Inode und reserviert ihn für die Datei
- ext2 reserviert dabei gleich eine Reihe von Blocknummern für die Daten
- bevor die Datei nicht geschlossen wurde, ist ihr tatsächlicher Speicherbedarf ungewiss
- ist die Bearbeitung der neuen Datei beendet, gibt ext2 die nicht benötigten Blöcke wieder frei
- um freie Inode oder Block aufzuspüren werden Bitmaps genutzt

Bitmap

Bitmaps sind bitweise kodiert.

Sie enthalten die Statusinformationen (frei oder belegt) über Blöcke bzw. Inodes.

Es existieren jeweils Bitmaps für Blöcke und für Inodes.

Bitmaps dienen zur schnelleren Suche nach freien Datenblöcken und Inodeplätzen.

Andere Dateisysteme unter Linux

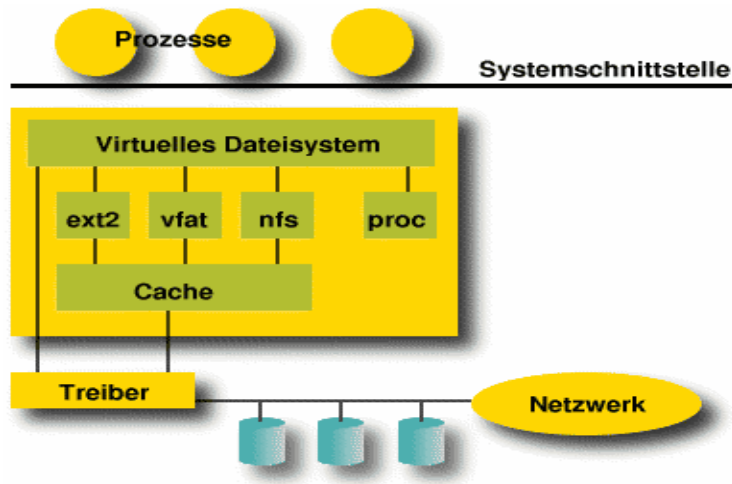
Es gibt viele Betriebssysteme, aber nicht jedes kann den Umgang mit Daten von anderen Dateisystemen.

Linux bietet dem Nutzer ein Virtuelles Dateisystem, das erreicht Linux durch eine Schnittstelle die Betriebssystem und die unterschiedlichen Dateisysteme verbindet.

Das Virtuelle Dateisystem entscheidet selbst, wie es die Systemaufrufe des Kernels in die speziellen Aktionen eines konkreten Dateisystems umsetzt.

Linux

Linux und sein Virtuelles Dateisystem



LINUX-Befehlssatz

- Verzeichnisverwaltung
- Arbeit mit Dateien
- Berechtigungen
- Prozesse

Verzeichnisverwaltung

Hierarchisches Dateisystem

mkdir - make directory
linux#mkdir user
linux#mkdir user/ralf
rmdir - remove directory
linux#rmdir user/ralf
cd - change directory
linux#cd /
linux#cd /etc
linux#cd ..
pwd - print working directory
linux#pwd

Arbeit mit Dateien

ls
alias
cp und mv
ln
rm
cat
more
tail
grep
ls

Linux

Der Befehl **ls** erlaubt ihnen die Anzeige aller Dateien aus einem oder mehreren Verzeichnissen

Ausgabe aller Dateien

```
linux#ls -a
```

Vollständige Anzeige der Dateien

```
linux#ls -lisa
```

cp Kopieren von Dateien und Verzeichnissen

```
linux#cp /etc/passwd /etc/passwd.neu
```

```
linux#mkdir /user; mkdir /user/ralf
```

```
linux#cp /etc/*.sh /user/ralf
```

```
linux#mkdir /etccopy
```

```
linux#cp /etc /etccopy
```

mv Verschieben und/oder Umbenennen von Dateien und ganzen Verzeichnisbäumen

```
linux#mv /etc/passwd.neu /user/ralf
```

ln Der Befehl **ln** ermöglicht das man eine Datei verknüpfen kann.

ln <Option> <Datei> <Linkname>

Bsp. **ln** Datei1 Datei2

ln -s Datei1 Datei2

rm Durch diesen Befehl werden Dateien oder Verzeichnisse gelöscht

rm <Datei/Verzeichnisname> oder

rm <Option> <Datei/Verzeichnisname>

cat <Dateispezifikation>

Der Befehl **cat** gibt die Datei auf stdout (normalerweise Bildschirm) aus, Anzeige aller möglichen Dateisysteme

```
linux#cat /proc/filesystems
```

```
linux#cat /etc/*
```

more <Dateispezifikation>

More gehört zu den sogenannten Pagerprogrammen, diese dienen zur schnellen Ansicht einer Datei ohne sie zu bearbeiten.

mit der Taste q beendet man ein solches Pagerprogramm

```
linux#more /etc/passwd
```

```
linux#more /etc/*.sh
```

Linux

Pipes

- dienen zur Steuerung der Verarbeitungsrichtung mehrerer Linux-Kommandos
- Pipes basieren auf der Tatsache, dass Linux-Befehle und Programme die Standardausgabe lesen und die Standardeingabe schreiben können
- mit Pipes kann die Ausgabe eines Kommandos an die Eingabe eines nachfolgenden Kommandos weiter gegeben werden

Beispiele für Pipes

```
linux#ps -ef | grep mount
linux#cat * | grep root
linux#cat /etc/passwd | more
```

tail

Der Befehl **tail** dient zur Ausgabe des Dateiendes, dabei kann der Nutzer selbst festlegen wie viele Zeilen er vom Ende der Datei ausgeben haben will.

tail - <Anzahl der Zeilen> <Dateiname>

grep

Das Programm **grep** durchsucht eine Datei nach einem vorgegebenen Textmuster.

Grep <Option> Textmuster <Datei>
linux#grep -i root /etc/passwd

Umleitung

- Umleitungen dienen dazu, die Standardeingabe, Standardausgabe und Standardfehlerausgabe auf eine Datei, Terminal etc. umzuleiten
- Das erreicht man mit den Umlenkoperationen >, >> und <

z.B.:

Find / '*.doc' > /worddokumente.txt
dieser befehl hängt

man

Der Befehl **man** steht für Manual Pages, was soviel heißt wie Hilfsseiten
man < Befehl der Erklärung bedarf>
nach Eingabe des Befehl zeigt „man“ alle wichtigen Informationen zu diesem Thema an.

Mount

Der befehl **mount** dient dazu bestimmte Laufwerke mit in das Linux system einzubinden.

mount <Option> <Dateisystemtyp> <Laufwerkname> <Einbindungsverzeichnis>
Bsp. Mount -t msdos /dev/fd0/ /floppy/

Linux

fstab / mtab

Diese Dateien dienen zum automatischen mounten eines Laufwerkes, sie enthalten bereits Information zu den Festplatten.

Alias

Der Alias-Mechanismus erlaubt es eigene Abkürzungen oder Namen für Kommandos zu definieren.

alias <alias Name> <Kommando>

Beispiel :alias ll ls -l

Erzeugt eine alias mit Namen ll für das Kommando ls -l
Nutzerverwaltung

su

passwd

chmod

chown

chgrp

datei passwd

su

Dieser Befehl erlaubt es ihnen sich als anderer Nutzer anzumelden, ohne aber dabei ihren alten Nutzer auszuloggen.

su <Username>

zum Abmelden einfach Exit eingeben.

passwd

passwd dient zum ändern das eigenen Passwortes

nur Eingabe von passwd im Terminal Fenster

chmod

chmod dient zum ändern der Rechte einer Datei

chmod <Benutzerklasse> <Operation> <Zugriffsrechte> <Datei/Verzeichnis>

Beispiel:

chmod a=rwx datei1

chmod Zeichen Erklärung

Benutzer : u = User , r = Root,

g = Gruppe und a = alle

Operation : + Rechte werden hinzugefügt

- Rechte wegnehmen

= Rechte so zu Ordnen

Rechte : r = Lese- , w = Schreib- und

x = Ausführrechte

Linux

chown und chgrp

Mit **chown** legt man den Besitzer einer Datei fest.

chown <neuer Eigentümer> <Datei>

Der Befehl **chgrp** legt die Gruppe fest welche auf die Datei zugreifen kann.

chgrp <neue Gruppe> <Datei>

datei passwd

Die Datei passwd enthält alle relevanten Benutzerdaten.

Linux Editor vi

vi ist der klassische Unix-Editor und einer der ältesten, zwei Betriebsmodi

Befehlsmodus

Einfügemodus

Arbeiten mit dem vi Editor

Start des vi durch Eingabe vi

dann den Modi Wechsel durch i

Eingabe des Textes oder Quellcodes

danach Esc Drücken um in Befehlsmodi zu kommen

Eingabe von :w <Dateiname>

Beendigung des Vi durch :q

vi Befehle

:x - vi Beenden

:wq - Puffer sichern und vi beenden

:w - Puffer sichern

:q - Bearbeiten des aktuellen Puffers (Datei) beenden

:e! - Zustand nach letzter Speicherung restaurieren

:vi - Rücker zum vi nach einem Q-Befehl

Linux Netzwerk

ping

Der Befehl ping sendet ein Signal zu dem Angeben Host und wartet dann auf eine Rückantwort.

ping <Option><Host>

Route

Der Befehl Route zeigt uns Informationen zur Netzkonfiguration des Rechners an.

Im Terminalfenster nur Route eingeben und bestätigen

Shell-Skripte

Was sind Shell-Skripte?

- Shell-Skripte sind Programme
- Sie bestehen aus zusammengeführten Kommandofolgen
- Sie fügen häufig auftretende Anweisungsfolgen zu einem Kommando zusammen

Wozu dienen Shell-Skripte

- Shell-Skripte ersetzen lange Kommandofolgen
- Shell-Skripte können vorhandene Kommandos benutzerfreundlich und benutzungssicher machen (DAU)
- Shell-Skripte erlauben es, den Programmablauf besser zu verstehen

Schrittfolge für das Erstellen eines Shell-Skripts

- Schreiben der Kommandofolgen in eine Datei z.B. mit vi
- Shell-Skript ausführbar machen
- Ausführen des Skripts
- Wenn nötig Fehlersuche und Korrektur

Entwurf eines Shell-Skripts

- Auswahl der für den Shell-Skript nötigen Kommandos oder Programme
- Ablaufplan für einen Shell-Skript erstellen

Schreiben eines Shell-Skripts

Start des vi Editors

Linux#vi Script1.sh

Eingabe der folgenden Zeilen

echo "Mein erstes Shell-Skript"

echo "Ausgabe des Datums"

date

echo "Am System arbeiteten"

who

Speichern der Datei und beenden des vi mit :x

Ausführbar machen des Skripts

Um das Shell-Skript nun zum Laufen zu bringen, müssen wir es noch als ausführbar (executable) markieren. Das erreicht man indem man die Rechte für die Datei ändert.

Der Befehl dazu sieht wie folgt aus

Linux#chmod u+x Script1.sh

Linux

Start des Shell-Scripts

Um einen Script zu starten muss man lediglich seinen Namen eingeben.

```
Linux#script1.sh
```

Wenn diese Möglichkeit nicht funktioniert hat man zwei Möglichkeiten das Script trotzdem zu starten.

```
Linux# sh script1.sh (muss nicht ausführbar sein)
```

```
Linux# ./script1.sh
```

Fehlersuche in Shell-Scripts

Um die Fehler in einem Shell-Script zu finden, lässt man den Shell-Script einfach von einer anderen Shell bearbeiten.

```
Linux# sh -v shell_script
```

Dieser Befehl zeigt jede Kommandozeile vor deren Ausführung nochmals an.

Parameter in Shell-Scripts

Parameter werden bei Shell-Scripts zur Übergabe von bestimmten Informationen mit bei der Kommandoeingabe genutzt.

Es können bis zu 9 Parameter übergeben die mit \$0-\$9 angesprochen werden.

```
Linux#Script1 test    Variable
$0                   $1    $2
```

Beispielscript 2: Parameter

```
echo "Name des Shell-Script $0"
echo -n "Der Benutzer $1 ist so oft angemeldet: "
who | grep $1 | wc -l
```

Systemvariablen in Shell-Scripts

Shell-Skripte haben neben den Standardparametern auch Zugriff auf Systemvariablen. Diese enthalten zusätzliche Informationen über das Script.

\$# enthält Anzahl der übergeben Parameter

\$\$ enthält die Prozessnummer des Scripts

\$* liefert alle Parameter die an das Script übergeben wurden

Beispielscript 3: Systemvariablen

```
echo "Die Prozessnummer des Scripts ist $$"
echo "Es wurden $# Parameter übergeben"
echo "Alle Parameter: $*"
```

Benutzerdefinierte Variablen

Anzahl beliebig

Linux

Sie werden wie im Beispiel dargestellt erstellt:

```
Datum='date'
```

Verwendung mit Präfix \$

```
echo "$Datum"
```

Eingabe in Shell-Scripts

Um eine Information während des laufenden Shell-Scripts einzugeben nutzt man den **read** Befehl. Er verlangt von dem Benutzer eine Eingabe und speichert diese in einer Variable

```
read <Variablenname>
```

Beispielscript 4: read

```
echo -n "Name der Datei"
```

```
read name
```

```
find / -name "$name"|more
```

Verzweigungen

Shell-Skripte erlauben es ihnen wie in der Programmierung, durch eine Bedingungsprüfung, bestimmten Aufgabe auszuführen.

Dabei ist der Quelltext wie folgt aufgebaut:

```
im Bedingung
```

```
then Kommandofolge 1
```

```
else Kommandofolge 2
```

```
fi
```

Bedingung test

Der Befehl **test** wird in der Shell- Programmierung zum Auswerten von Ausdrücken verwendet. In Kontrollstrukturen können sie mit test – Anweisungen, Dateien , Zeichenketten und Zahlenwerte auswerten.

Die Optionen von Test

Kommando	zur Bewertung der Frage
test datei	existiert die Datei?
test -r datei	existiert die Datei und ist lesbar?
test -w datei	existiert die Datei und ist schreibbar?
test -x datei	existiert die Datei und ist ausführbar?
test -s datei	existiert die Datei und enthält (mehr als 0) Zeichen?
test -d datei	existiert die Datei und ist ein Verzeichnis?
test -f datei	existiert die Datei und ist eine „normale“ Datei?
test zeichen	ist die Zeichenkette nicht leer (nicht ein „null string“)?
test -n zeichen	ist Länge der Zeichenkette ungleich null?
test -z zeichen	ist die Länge der Zeichenkette gleich null?
test "zk1" = "zk2"	sind erste und zweite Zeichenkette gleich?
test zahl1 -eq zahl2	sind erste und zweite Zahl algebraisch gleich?
test zahl1 -ne zahl2	sind erste und zweite Zahl algebraisch ungleich?
test zahl1 -lt zahl2	ist die erste kleiner als die zweite Zahl?
test zahl1 -le zahl2	ist die erste kleiner gleich der zweiten Zahl?
test zahl1 -gt zahl2	ist die erste größer als die zweite Zahl?
test zahl1 -ge zahl2	ist die erste größer gleich der zweiten Zahl?

Tabelle 10.1: Befehlsformate des test Kommandos

Linux

Beispielscript 5:Verzweigungen

```
echo -n "Startverzeichnis angeben: "  
read verz  
if test -d $verz  
    then cd $verz  
        ls -l | more  
    else echo "Fehler: $verz existiert nicht"  
fi
```

Mehrfach Verzweigungen

Um eine Mehrfachverzweigung in einem Shell-Script zu erstellen, wird der Befehl **case** verwendet.

```
case Zeichenkette in  
Muster 1) Kommandozeile 1  
Muster 2) Kommandozeile 2  
Muster 3) Kommandozeile 3  
Muster n) Kommandozeile n  
*) Kommandozeile else  
esac
```

Beispiel Script ist menue1

Wiederholungsstrukturen

Es gibt drei mögliche Wiederholungsstrukturen

Wiederhole bis Bedingung erfüllt ist.
Wiederhole bis die Bedingung nicht mehr erfüllt ist.
Wiederhole so lange bis bestimmter Wert erreicht ist.

until Schleife
while Schleife
Zählschleife

until Schleife

Until Schleifen wiederholen einen Vorgang so lange bis die Bedingung nicht mehr erfüllt ist.

until Bedingung

```
do  
Kommandofolge  
done
```

Beispielscript 6: until

Programm gibt solange Hallo aus bis der Benutzer abbricht.

```
until test "$auswahl" = "q"  
do  
    echo "Hallo lieber Nutzer um die Schleife zu beenden „q“ eingeben"  
    read auswahl  
done
```

Linux

while Schleife

Die While Schleife ist das Gegenstück zur Until Schleife, es wird solange wiederholt bis die Bedingung nicht mehr erfüllt ist.

```
while Bedingung
do
Kommandofolge
done
```

Beispielscript 7: while

Programm wird solange wiederholt wie eine Benutzereingabe erfolgt.

```
Eingabe=`1`
while test $Eingabe
do
    echo "Um die Schleife zu verlassen nichts Eingeben und bestätigen."
    read Eingabe
done
```

Die Zählschleife

Sie wiederholt eine Kommandofolge solange bis ein bestimmter Wert erreicht ist.

```
for Variable
do
Kommandofolge
done
```

Beispielscript 8: for

Dieser Script gibt die Parameter, die übergeben wurden wieder aus.

```
for argu
do
    echo $argu
done
```

break

Der Befehl **break** erlaubt es, eine Schleife ohne das eine Abbruchbedingung erfüllt wurde, zu beenden.

Dabei wird die zur Zeit innerste Schleife verlassen und ein Sprung in die nächste Kommandozeile nach der Schleife ausgeführt.

continue

Der Befehl **continue** dient ebenfalls zur Schleifensteuerung

Anders als bei **break** wird die Schleife nicht verlassen, sondern zum Anfang des nächsten Durchlaufs gesprungen.

Linux

Ablaufpause sleep

Dieser Befehl erlaubt es ihnen ein Shell-Script für eine gewisse Zeit in einen Wartezustand zu versetzen.

sleep <Sekunden>

exit

Der Befehl **exit** erlaubt es ihnen ein Script vorzeitig zu beenden.

Beispielscripte

Quadervolumen

```
if test $# -eq 3
then
    if test $1 -gt 0 -a $2 -gt 0 -a $3 -gt 0
    then
        VOLUMEN=$(( $1*$2*$3 ))
        echo "Das Quadervolumen beträgt " $VOLUMEN
    else
        echo "Mindestens ein Parameter ist Null -> Volumen gleich 0"
    fi
else
    if test $# -lt 3
    then
        echo "Fehlender Parameter"
    else
        echo "Zu viele Parameter"
    fi
    echo "usage quadervolumen.sh <hoehe> <breite> <tiefe>"
fi
```

Summe

```
ZAHL1=1
ZAHL2=2
echo '$ZAHL1+$ZAHL2'
echo $ZAHL1+$ZAHL2
echo [$ZAHL1+$ZAHL2]
ZAHL3=$(( $ZAHL1+$ZAHL2 ))
echo $ZAHL3
```

Menü 1

```
clear
echo "-----Menueauswahl-----"
echo
echo " 1 - Anzeige des Aktuellen Datums"
echo " 2 - Wer ist angemeldet"
echo " 3 - Zeigt Verzeichnis Inhalt"
echo " 4 - Suchen einer Datei"
echo " 5 - Beenden des Menüs"
echo
```

```
echo "-----"
echo -n "bitte Eingabe : "
read menue
case $menue in
    1 ) echo "Datum und Uhrzeit: `date`";;
    2 ) echo -n "Angemeldet ist: "
        who
        ;;
    3 ) echo -n "Bitte Verzeichnisnamen eingeben: "
        read verz
        if test -d $verz
        then
            ls -l $verz | more
        else
            echo "fehler"
        fi;;
    4 ) echo "Bitte geben sie den Namen der Datei ein."
        read datei
        find / -name "$datei"|more ;;
    8 | q ) exit ;;
    * ) echo "falsche Eingabe";;
esac
```

Menü 2

```
while true
do
clear
echo "-----Menueauswahl-----"
echo
echo " 1 - Anzeige des Aktuellen Datums"
echo " 2 - Texteditor"
echo " 3 - Anzeige von Dateiinhalten"
echo " 4 - Auflisten von Verzeichnisseintraegen"
echo " 5 - Suchen einer Datei"
echo " 6 - Eine Datei ausführbar machen"
echo " 7 - Ausführung eines Shell Scripts"
echo " 8 - Beenden des Menüs"
echo
echo "-----"
echo -n "bitte Eingabe : "
read menue
uverz=`pwd`
case $menue in
    1 ) echo "Datum und Uhrzeit: `date`";;
    2 ) echo -n "Bitte Dateiname eingeben: "
        read datei
        vi $datei;;
    3 ) echo -n "Bitte Dateinamen eingeben: "
        read datei
        if test -f $datei
        then
            more $datei
        else
            echo "Fehler"
        fi;;
    4 ) echo -n "Bitte Verzeichnisnamen eingeben: "
        read verz
```



```
        if test -d $verz
        then
            ls -l $verz | more
        else
            echo "fehler"
        fi;;
5 ) echo "Bitte geben sie den Namen der Datei ein."
    read datei
    find -name "$datei"|more ;;
6 ) echo -n "Bitte Verzeichnisnamen eingeben: "
    read verz
    if test -d $verz
    then cd $verz
        echo "Bitte geben sie den Namen der Datei ein."
        read datei
        chmod a+x $datei
    else
        echo "fehler"
    fi;;
7 ) echo -n "Bitte Verzeichnisnamen eingeben: "
    read verz
    if test $verz
    then
        if test -d $verz
        then cd $verz
            echo "Bitte geben sie den Namen des Scripts ein."
            read datei
            ./$datei
        else
            echo "fehler"
        fi
    else cd $uverz
        echo "Bitte geben sie den Namen des Scripts ein."
        read datei
        ./$datei
    fi;;
8 | q ) exit ;;
* ) echo "falsche Eingabe";;
esac
cd $uverz
read
done
```

Script 1

```
echo "Mein erstes Shell Script"
echo "Ausgabe des Datums"
date
echo "Am System arbeiten:"
who
```

Script 2

```
echo "Name der Shell" $0
echo -n "Der Benutzer $1 ist so oft angemeldet:"
who | grep $1| wc -l
```

Linux

Script 3

```
echo " Die Prozessnummer des Shell Scripts ist $$. "  
echo " Es wurden $# Parameter übergeben."  
echo " Alle Parameter : " $*
```

Script 4

```
echo -n "Name der Datei: "  
read name  
find / -name "$name"|more
```

Script 5

```
echo -n "Startverzeichnis angeben : "  
read verz  
if test -d $verz  
    then cd $verz  
         ls -l | more  
    else echo "Fehler : $verz ist nicht Vorhanden."  
fi
```

Script 6

```
until test "$name" = "q"  
do  
echo -n "Hallo lieber User um das Programm zu beenden, einfach q eingeben.: "  
read name  
done
```

Script 7

```
eingabe='1'  
while test $eingabe  
do  
    echo "Um die Schleife zu verlassen einfach nichts Eingeben und  
bestätigen."  
    read eingabe  
done
```

Script 8

```
for argu  
do  
    echo $argu  
done
```