Bashing

The Cook Book

Uwe Schimanski

Seab@er Software

Goch, den 13. Dezember 2019

```
132  #=== Function  ================================================================
133  #         NAME:  dist
134  #  DESCRIPTION:  Which distribution
135  # PARAMETER  1:  ---
136  #================================================================================
137
138  dist() {
139   DISTRIBUTION=$(grep -i "id" /etc/os-release | head -n1 | cut -d"=" -f2)
140   insert_data "INFO" "Distribution is $DISTRIBUTION"
141  }
142
143  #=== Function  ================================================================
144  #         NAME:  check_path
145  #  DESCRIPTION:  Check mount path and create path if not exist
146  # PARAMETER  1:  ---
147  #================================================================================
148
149  check_path() {
150  if [ ! -d "/CloudDrive" ]; then
151   if [ "$UID" == "0" ]; then
152    printf "`date +%y` `date +%b` `date +%d` `date +%H:%M:%S` $HOSTNAME INFO: You have root rights.\n"
153    insert_data "INFO" "Root rights found."
154    ROOTID=true
155   else
156    printf "`date +%y` `date +%b` `date +%d` `date +%H:%M:%S` $HOSTNAME INFO: For create path, you need root rigths.\n"
157    insert_data "ERROR" "For create path, you need root rights. Exit Programm."
158    exit 1
159   fi
```

Published by Publisher

seabaer-ag.de

First printing, May 2018

Inhaltsverzeichnis

Vorwort

Ich beschäftige mich schon seit dem Jahr 1993 mit Unix. Unsere CAD Software lief damals auf Unix Maschinen und 1998 habe ich dann Linux auf meinen Rechnern installiert. Damals habe ich SuSE installiert und jetzt läuft OpenSuSE bei mir immer noch auf 2 Laptops. Auf den anderen Rechnern ist nun Manjaro installiert.

Die Shell unter Linux ist ein mächtiges Werkzeug. Daher ist es unerlässlich, das man sich mit Scripten das Leben erleichtert. Damit man ein Nachschlagewerk hat, habe ich mich entschlossen, eine Dokumentation über das Scripten in der Bash zu schreiben. Alle Scripte wurden in der Bash Shell ausprobiert und können auch in einer anderen Shell verwendet werden.

Als OS Systeme wurden von mir OpenSuSE und Manjaro verwendet.

Bei Fragen und Anregungen könnt ihr mir gerne ein Mail schicken.

Kapitel 1

Allgemein

1.1 Variablen

1.1.1 Definition

Jede Variable hat einen Namen und einen Wert. Typen kennt die Bash nur eingeschrängt. Um eine Variable in einem Bash Script zu definieren, kann der Befehl `declare` benutzt werden.

```
uws@tux>wert=3+7
uws@tux>echo $wert
3+7

uws@tux>declare -i wert
uws@tux>echo $wert
3+7

uws@tux>wert=3+7
uws@tux>echo $wert
10

uws@tux>declare -u choise
uws@tux>choise=yes
uws@tux>echo $chaoise
YES

uws@tux>declare -l choise
uws@tux>choise=YES
uws@tux>echo $choise
yes
```

Listing 1.1: Beispiele

Hier eine kurze Liste der Typen.

Attribut	Beschreibung
-p	Zeigt alle Attribute aller Variablen an
-i	Arithmetische Auswertung
-a	Array Index
-A	Array Namen
-u	Der Wert wird in Großs buchstaben umgewandelt
-l	Der Wert wird in Kleinbuchstaben umgewandelt
-r	Nach der Zuweisung kann dieser Wert nicht mehr verändert werden, sie ist dann eine Konstante

Tabelle 1.1: Attribute

Weitere Informationen hierzu gibt es auf der Seite `http://tldp.org/LDP/abs/html/declareref.html`.

```
uws@tux>strMachine="Computer Name: ‘uname -a | cut  -d ’ ’ -f 2‘"
uws@tux>echo $strMachine
Computer Name: tux
```

Listing 1.2: Zuweisung mittels Befehl

Die auszuführenden Befehle stehen zwischen Rückwertigen Hochkommas.
Auch mit `typeset` werden Variablen mit einem Typ definiert. Standardmässig werden auch hier Variablen ohne Angabe des Typs als `string` Variable deklariert. Die Angabe eines Wertes ist optional.
Die Syntax für `typeset` ist:

```
typset [option] [variable] [=wert]
```

```
uws@tux>cat example1.sh
#!/bin/env bash
typeset -i var1=2 # Variable var1 als integer mit einem Wert

typeset +i var1 # Abschalten der definition
```

Listing 1.3: Beispiel typeset

Option	Beschreibung
-a	Array
-i	Integer
-r	Konstante (read only)
-x	Varaible exportieren
-f	Zeigt Funktionen mit ihrer Definition an
-fx	Exportiert eine Funktion
-F	Zeigt Funktionen ohne ihre Definition an

Tabelle 1.2: Typeset Options

1.1.2 Variablen länge

Die Länge eines Strings, die einer Variable zugewiesen worden ist, kann man folgendermaßen ermitteln.

```
uws@tux>cat varlength.sh
#!/bin/bash
A="$1"  X=${#A}
echo -n "*" # Ausgabe ohne newline
printf " Laenge der Variable A: ${X}\n"

uws@tux>./varlength.sh Weihnachten
* Laenge der Variable A: 11
```

Listing 1.4: Variable länge

1.1.3 Variablen default

Falls ein Parameter / Variable nicht gesetzt wird, so kann man ein Default Wert für diese Variable setzen.

```
uws@tux>cat varDefault.sh
#!/bin/bash
MinSize=${1:-50M}
printf "Minimum Size: $MinSize\n"

uws@tux>./varDefault.sh
Minimum Size: 50M
```

```
uws@tux>./varDefault.sh 125M
Minimum Size: 125M
```

Listing 1.5: Variable default

1.1.4 Variablen Warnung

Wenn ein Parameter / Variable nicht definiert wird, so wird dann eine Message ausgegeben. Nicht interktive Shells terminieren anschließend.

```
uws@tux>cat varWarnung.sh
#!/bin/bash
FileName=${1:?Missing filename.}
printf "File Name: $FileName\n"

uws@tux>./varWarnung.sh
varWarnung.sh: Zeile 2: 1: Missing filename.

uws@tux>./varWarnung.sh Mail.log
File Name: Mail.log
```

Listing 1.6: Variable Warnung

1.1.5 Array

Ein Array wird mit `declare -a` oder mit `declare -A` definiert. Bei der ersten Zuweisung werden die Werte mit einer Indexnummer versehen und bei der zweiten Zuweisung kann man einen Bezeichner selbst übergeben. Die Syntax für die Zuweisung von Werten ist folgendermaßen.

```
declare -a <arrayname>
arrayname=(wert1 wert2 wert3)
arrayname[indexnr]=wert
```

Listing 1.7: Syntax Array

Wird keine Index Nummer bei der Zuweisung der Werte angegeben, so startet die Index Nummer bei `0` für den ersten Wert.

```
uws@tux>declare -a MyUser=('Harry' 'Paul' 'Walter')
uws@tux>for ((i=0;i<=2;i++)); do echo User: ${MyUser[$i]}; done
User: Harry
User: Paul
User: Walter

uws@tux>echo ${MyUser[@]}
Harry Paul Walter

uws@tux>echo ${!MyUser[@]}
0 1 2

uws@tux># Anstelle von @ kann auch ein * genommen werden
uws@tux>echo ${#MyUser[*]}
3

uws@tux>for i in ${!MyUser[*]}; do echo User: ${MyUser[$i]}; done
User: Harry
User: Paul
User: Walter

uws@tux>#Wert hinzufuegen zum array
uws@tux>MyUser+=('Andrea' 'Silvia')
uws@tux>echo ${MyUser[@]}
Harry Paul Walter Andrea Silvia

uws@tux>#Loeschen von Werten vom array
uws@tux>unset -v MyUser[2]
uws@tux>echo ${MyUser[@]}
Harry Paul Andrea Silvia

uws@tux>declare -A MyText[Start]='Starts here'
uws@tux>MyText['End Text']='The Last one'
uws@tux>echo ${MyValue['End Text']} - ${MyValue[Start]}
The Last one - Starts here
```

Listing 1.8: Beispiele Array

1.1.6 Groß- / Kleinbuchstaben

Seit der Bash Version 4 kann man ganz einfach den Wert einer Variable in Groß- oder in Kleinbuchstaben umwandeln. Es können der erste oder alle Buchstaben umgewandelt werden. Mit `-u` wandelt alle Buchstaben in Großbuchstaben um und ein `-l` wandelt alle Buchstaben in Kleinschreibung um.

```
uws@tux>#  Umwandeln in Gro\ss buchstaben
uws@tux>declare -u Wert1
uws@tux>Wert1=klein
uws@tux>echo $Wert1
KLEIN

uws@tux>#  Umwandeln in Kleinbuchstaben
uws@tux>declare -l Wert2
uws@tux>Wert2=GROSS
uws@tux>echo $Wert2
gross
```

Listing 1.9: Groß- / Kleinbuchstaben

Soll der erste Buchstabe umgewandelt werden oder alle Buchstaben, kann man das folgendermaßen machen.

```
uws@tux>an=paulchen
uws@tux>echo ${an^}
Paulchen
uws@tux>echo ${an^^}
PAULCHEN
uws@tux>echo ${an^^c}
paulChen

uws@tux>PAULCHEN
uws@tux>echo ${an,}
pAULCHEN
uws@tux>echo ${an,,}
paulchen
uws@tux>echo ${an,,U}
PAuLCHEN
```

Listing 1.10: Werte umwandeln

1.2 Datum in Bash Scripte

In einem Bash Script kann man sich sein eigenes Datumsformat zusammenstellen.

```
uws@tux>export stamp=$(date +%a)
uws@tux>echo $stamp
Do

uws@tux>export stamp=‘date +%Y_%m_%d‘
uws@tux>echo $stamp
2010_06_17

uws@tux>printf "‘date +%Y_%m_%d‘ - Time\n"
2010_06_17 - Time

uws@tux>cat example.sh
#!/bin/bash
# Example for Date
#
DATUM=‘date +%Y_%m_%d‘
stamp=$(date +%a)
find . -type -f ( -name ’*.jpg’ ) -exec zip ${Datum}_jpg.zip {} \;
mv example.txt example.txt.$stamp
```

Listing 1.11: Beispiele Datumsformat

Eine Auflistung für die Datumformatierung befindet sich in der nachfolgenden Tabelle.

Format	*Beschreibung*
%H	Stunden (00 bis 23)
%I	Stunden (01 bis 12)
%M	Minuten (00 bis 59)
%S	Sekunden (00 bis 23)
%p	Vor- oder Nachmittag (AM oder PM)
%r	Zeitangabe, 12 Stunden (hh:mm:ss AM/PM)
%R	Zeitangabe, 24 Stunden (hh:mm:ss), entspricht damit %H:%M
%s	Sekunden seit dem 1. Januar 1970 (Unix Zeit)
%Z	Aktuelle Zeitzone (CEST, GMT, ...)
%a	Wochentag in Kurzform (Son, Mon, Die, ...)
%A	Wochentag in Langform
%b	Monat in Kurzform (Jan, Feb, ...)
%B	Monat in Langform
%d	Tag in zweistelliger Zahl
%e	Tag (einstellig mit Leerzeichen)
%D	Datum in der Form mm/dd/yy
%j	Zeigt, der wie vielte Tag im angegebenen Jahr ist
%u	Zeigt, welcher Wochentag es ist (1 bis 7)
%U	Zeigt, wie viele Wochen im angegebenen jahr es ist
%m	Monat, zweistellig
%y	Jahr, zweistellig
%Y	Jahr, vierstellig
%%	Ausgabe des Prozentzeichens
%n	Zeilenende
%t	Tabulator

Tabelle 1.3: Formate

1.3 Ein- und Ausgabe auf der Shell

1.3.1 Echo

Möchte man Text auf der Konsole ausgeben, so kann man das mit dem Befehl `echo` erledigen. Den Befehl `echo` kann man nicht nur in einem Script verwenden, sondern auch direkt in einer Shell, wenn man zum Beispiel den Inhalt einer Variable sich anzeigen lassen möchte.

```
uws@tux>echo "Connected user: $USER"
Connected user: uws
```

Listing 1.12: Beispiel echo

Gibt man hinter dem Befehl `echo` die Option `-n` an, so bleibt der Cursour hinter den auszugebenden Text stehen. Die Option `-e` erlaubt die `Backslash Escapes`.

```
uws@tux>cat eingabe.sh
#!/bin/env bash
#
# Einlesen einer Benutzereingabe
#
echo -n -e "\tBitte Mysql Password eingeben: "
read strPasswd
```

Listing 1.13: Beispiel Escape Sequence

1.3.2 Read

Mit dem Befehl `read` kann man die Eingabe in einer Variable speichern. Ohne eine zusätzliche Option bei dem Befehl `read`, wird die Eingabe nach einem Return in der Konsole wiederholt. Soll die Eingabe nicht angezeigt werden, so gibt man hierzu die Option `-s` an. Diese Option unterdrückt das Echo auf der Konsole. Die Option `-v<zahl>` liest die angegebene Anzahl an Zeichen ein.

```
uws@tux>cat eingabe1.sh
#!/bin/env bash
#
# Einlesen einer Benutzereingabe
#
echo -n "Bitte Mysql Password eingeben: "
read -s -n10 strPasswd
```

Listing 1.14: Beispiel read

1.3.3 Printf

In Bash Scripten und auch in der Konsole kann man den Befehl `printf` verwenden. Mit diesem Befehl ist es möglich, Zeilenvorschübe und auch Tabs zu verwenden. Einen Zeilenvorschub wird mit der Option `\n` gemacht und mit `\t` wird ein Tab erzeugt. In der nachfolgenden Tabelle werden die Konfigurationsdateien aufgelistet, die bei einem Login/Aufruf einer Shell in der Reihenfolge verarbeitet werden.

```
uws@tux>cat Ausgabe.sh
#!/bin/env bash
#
# Ausgabe von Text
#
clear
printf "\n\n\t\t ****************************************"
printf "\n\n\t\t  Dieses Programm wir ausgefuehrt"
printf "\n\t\t  mit freundlicher Untestuetzung von"
printf "\n\t\t\       Seab@er Software AG"
printf "\n\n\t\t ****************************************\n"
```

Listing 1.15: Beispiel Printf

Printf wurde von der Programmiersprache C entliehen. Der Aufbau ist "%Format"Daten. In der nachfolgenden Tabelle sind die Formatierungen aufgelistet.

Format	*Beschreinung*
%5.2f	Fließkomma mit fünf Stellen vor dem Komma und 2 Nachkomma Stellen
%.10s	Zeichenkette mit maximal 10 Characters
%X\n	Hexadezimal mit Großbuchstaben
%y\n	Hexadezimal mit Kleinbuchstaben
%#X\n	Hexadezimal mit Großbuchstaben und führenden 0X
%i\n	Ganzzahl
%s	Zeichenkette (string)

Tabelle 1.4: Formate

```
uws@tux>cat format.sh
#!/bin/env bash
a=456.863
b=387,162
c="Ohne_Unterstrich_keine_Ausgabe"
printf "Wert a: %5.2f\n" `echo $a | tr . ,`
printf "Wert b= %5.2f\n" $b
printf "Wert c: %.30s\n Wert d: %.30s" $c "So geht  es ohne _"

uws@tux>./format.sh
Wert a: 456,863
Wert b: 387,162
Wert c: Ohne_Unterstrich_keine_Ausgabe
Wert d: So geht es ohne _
```

Listing 1.16: Beispiel Formatierung

Möchte man den Text in Farbe ausgeben, so wird hierzu das Zeichen ESC (\033) verwendet. Nach dem Escape Zeichen wird die Farbe angegeben. Die Angabe bezieht sich dann nicht nur auf die Zeile, sondern auf alle Ausgaben von Texten. Deshalb sollte man nach dem zu färbenden Text wieder die ursprüngliche Farbe eingestellt werden.

```
uws@tux>cat ColorAusgabe.sh
#!/bin/env bash
#
# Ausgabe von Text in Farbe
#
clear
printf "\033[34m\n\n\t\t ****************************************"
printf "\033[33m\n\n\t\t  Dieses Programm wir ausgefuehrt"
printf "\n\t\t  mit freundlicher Untestuetzung von"
printf "\n\t\t\       Seab@er Software AG"
printf "\n\n\t\t ****************************************\n"
printf "\033[0m" # urspr$\tilde{A}\frac{1}{4}$ngliche Farbe einstellen.
```

Listing 1.17: Beispiel Farbige Ausgabe

Wert	*Farbe*
0m	Reset der Farbe
01m	Fett
04m	Unterstreichen
05m	Blinkend
07m	Vorder- und Hintergrunffarbe vertauscht
22m	Normale Intensität wiederherstellen
30m	Vordergrund schwarz
31m	Vordergrund rot
32m	Vordergrund grün
33m	Vordergrund braun (gelb)
34m	Vordergrund blau
35m	Vordergrund magenta
36m	Vordergrund cyan
37m	Vordergrund weiß
40m	Hintergrund schwarz
41m	Hintergrund rot
42m	Hintergrund grün
43m	Hintergrund braun
44m	Hintergrund blau
45m	Hintergrund magenta
46m	Hintergrund cyan
47m	Hintergrund weiß
49m	Voreingestellter Hintergrund

Tabelle 1.5: Farbwerte

Wird bei `33m` die Farbe braun ausgegeben und man möchte aber die Farbe `Gelb` ausgeben, so kann kann man das mit der zusätzlichen Angabe von `01m` machen.

```
uws@tux>printf "\033[01m\033[33mFarbe Gelb \033[33m und nun Braun."
```

Listing 1.18: Beispiel Gelbe Ausgabe

Eine Farbausgabe des Textes kann auch mit der Sequence `\e[1;32m` gemacht werden. Dieses funktionert auch mit `echo`. Hierzu wird noch die Option `-e` mit angegeben.

```
uws@tux>echo -e " \e[1;32m Dieser Text ist in Gr\"un. \e[1;0m"
```

Listing 1.19: Beispiel Echo

1.3.4 Beispiele

```
uws@tux>cat textbox.sh
#!/bin/env bash
box()
{
  printf '*%.0s' $(seq 1 67) # Print * with length 67
  printf "\n"
}

content()
{
 printf "*"
 if [ $# -eq 0 ]; then
  printf ' %.0s' $(seq 1 65)
 elif [ $# -eq 1 ]; then
  TEXTR="$1"
  printf ' %.0s' $(seq 1 17)
  printf "${TEXTR}"
  printf ' %.0s' $(seq 1 $[48-${#TEXTR}])
 elif [ $# -eq 2 ]; then
  TEXTL="$1"
  TEXTR="$2"
  printf ' %.0s' $(seq 1 3)
  printf "${TEXTL}"
  printf ' %.0s' $(seq 1 $[14-${#TEXTL}])
  printf "${TEXTR}"
  printf ' %.0s' $(seq 1 $[48-${#TEXTR}])
 fi
 printf "*\n"
}

SCRIPTVERSION="13.01.07"

IFS=$'\012'

box
content
content "Script:" "$(basename $0)"
content "Version:" "${SCRIPTVERSION}"
content
content "This script is an example."
content
box

uws@tux>./textbox.sh
**********************************************************************
*                                                                    *
*  Script:       textbox.sh                                          *
*  Version:      13.01.07                                            *
*                                                                    *
*                This script is an example.                          *
*                                                                    *
**********************************************************************
```

Listing 1.20: Beispiel Textbox

```
uws@tux>cat FuncColorText.sh
#!/usr/bin/env bash

err() {
 RESET="\e[1;0m"
 FETT="\e[1;1m"
 ROT="$FETT\e[1;31m"
 local txt=$1; shift
 printf "${ROT}==>${RESET}${FETT} ${txt}${RESET}\n"  "$@" >&2
}

msg() {
 RESET="\e[1;0m"
 FETT="\e[1;1m"
 GRUEN="$FETT\e[1;32m"
 local txt=$1; shift
 printf "${GRUEN}==>${RESET}${FETT} ${txt}${RESET}\ n" "$@" >&2
}

msg "File found in folder."
err "No file found."
```

Listing 1.21: Beispiel Color mit Function

1.4 Set-Befehl im Bash Script

Der Befehl `set -- ...` in einem Bash Script weist alle folgenden Argumenten den Variablen `$1, $2, $3, ...` zu. Den Inhalt der letzten belegten Variable kann man dann mit `$!#` auslesen.

```
uws@tux>cat MeinScript.sh
#!/bin/env bash
# Script fuer den set Befehl
#
SOURCEPATH=/home/uws/Dateien
set -- {SOURCEPATH}/bild_???.jpg # Einlesen der Dateien in $1, $2 usw
lastname=${!#}                   # Letzter Name steht hier.
echo "Lastname: ${lastname}"     # Ausgabe auf dem Schirm
```

Listing 1.22: Beispiel Set

Mit `set -x` kann man in der Shell oder auch in einem Script sich die Verarbeitungsschritte sich anzeigen lassen. Mit `set +x` wird dieses wieder abgeschaltet.

1.5 Programm Parameter auswerten

Möchte man bei einem selbst erstellten Script einen Parameter übergeben, so erfolgt die Auswertung in diesem Script mittels einer `case, if` oder `while` Schleife. Die auszuwertende Variablen fangen immer mit dem `$` an. In der Variable `$0` steht der Name des Scripts, inklusive des Verzeichnisses. Die Variablen `$1 .. $9` enthalten dann die Parameter zum Auswerten. In der Variable `$#` steht die Anzahl der übergebenen Parameter. Die Variablen Nummer können auch mit `{}` kenttlich ge,macht werden, wie zum Beispiel `${1}`. Kommandozeilen-Parameter werden mit `$` abgefragt. In der nachfolgenden Tabelle ist eine Auflistung der wichtigsten Parameter.

Parameter	*Beschreibung*
$0	Name des ausgeführten Shell Scripts, incl. des Pfades
$#	Anzahl der übergebenen Parameter
$1 $2 $3 ...	Erster, zweiter, dritter, ... Parameter
$*	Alle Kommandozeilen-Parameter ($1 $2 $3 ...)
$@	Wie $*
"$@"	Expandiert zu: $1 $2 $3 ...
$$	Prozessnummer der Shell
$-	Ausgabe der aktuellen Shell Optionen
$?	Ausgabe des Return-Codes von dem letzten ausgeführten Befehl
$!	Prozessnummer des zuletzt ausgeführten Hintergrund Prozesses
$HOST	Ausgabe der Umgebungsvariable HOST

Tabelle 1.6: Parameter

1.6 Shell Script Testen

Ein Shell Script kann man vor einer Ausführung mit `sh <option>` testen. In der nachfolgenden Auflistung sind einige Möglichkeiten beschrieben.

```
uws@tux># Syntax Test. Alle Kommandos werden gelesen, aber nicht ausgefuehrt
uws@tux>sh -n <script_name>

uws@tux># Ausgabe der Shell Kommandos in der gelesenen Form
uws@tux>sh -v <script_name>

uws@tux># Ausgabe der Shell Kommandos, nach der Durchfuehrung aller
   Ersetzungen
uws@tux>sh -x <script_name>
```

Listing 1.23: Script Testen

1.7 Tabs setzten

Die Abstände für die Tabs kann man mit `tabs <wert>` setzten. Sie gelten dann für die Console und auch in den Scripten.

Wert	*Beschreibung*
-0	Löscht alle Tabs
-8	Setzt die default Werte
+4	Tabs mit dem Abstand von 4 setzten
+2,6,12	Tabs mit den Abständen 2,6,12 setzten

Tabelle 1.7: Tabs Werte

1.8 Befehle mehrzeilig

Für eine bessere Übersicht in einem Script kann es erforderlich sein, den Befehl in mehrere Zeilen zu schreiben. Dieses kann man mit einem Backslash \ am Ende der Zeile machen.

```
uws@tux>cat example3.sh
#!/bin/env bash
ls \
-lah \
*.txt
```

Listing 1.24: Beispiel Mehrzeilig

1.9 Befehle verketten

Befehle lassen sich mit `&&` in einer Zeile verketten. Diese Verkettung ist eine `und` Verbindung. Eine `oder` Verbindung wird mit zwei Pipe Zeichen `||` gemacht. Anstelle von den Pipe Zeichen kann auch ein Semikolon genommen werden.

```
uws@tux>cat example_command.sh
#!/bin/env bash
ask_fragen() {
 for (( i=1;i<=4;i++ )); do
  echo "$i: $1" && shift
  sleep 2
 done
 sekunden=10
 let timer=10
 while [ "$sekunden" -gt 0]; do
  printf "$timer" && let sekunden-- && let timer--
  sleep 1
 done
 printf "\n"
}
### Main ###
ask_fragen "Was ist die Bash?" "Was ist KVM?" "Was ist LVM?" "???"
```

Listing 1.25: Beispiel Verketten

1.10 XARGS

Das Programm xargs wird mit Pipes benutzt und dabei werden alle Standardeingaben gelesen und als Argumente an ein anderes Programm weitergegeben. Nachfolgend sind ein paar Beispiele, die den Einsatz von xargs zeigen.

```
uws@tux>echo 1 2 3 4 5 6 | xargs
1 2 3 4 5 6

uws@tux>echo 1 2 3 4 5 6 | xargs -n 2
1 2
3 4
5 6

uws@tux>echo 1 2 3 4 5 6 | xargs -p -n 2
echo 1 2 ?... y
1 2
echo 3 4 ?... n
echo 5 6 ?... y
5 6

uws@tux>ls *.sh | xargs -n 1 echo File:
File: Backup.sh
File: Restore.sh
```

Listing 1.26: Beispiele xargs

```
uws@tux>cat inhalt.txt
Dieser Text steht in einer Zeile

uws@tux>xargs --arg-file=inhalt.txt -n 1
Dieser
Text
steht
in
einer
Zeile

uws@tux>cut -d: -f1 /etc/passwd | sort | xargs -n 1 echo User:
User: at
User: avahi
User: bin
```

Listing 1.27: Beispiele xargs

Löschen von Dateien mit einer Abfrage vor dem löschen. Damit auch Dateien gelöscht werden, die ein Leerzeichen enthalten, wrd bei `find` die Option `-print0` und bei `xargs` die Option `-0` gesetzt.

```
uws@tux>find . -name "*.sh" -print0 | xargs -0 -p -n 1 rm -rf
rm -rf ./Backup.sh ?... y
rm -rf ./Restore.sh ?... n
```

Listing 1.28: Beispiele Delete mti xargs

1.11 Signale (Traps)

Signale, wie z.B. ein `Ctrl + c` kann man in einem Programm abfangen. Hierzu wird in dem Programm ein entsprechender `trap` definiert.
Syntax: `trap 'action' signal`
Als Signal kann entweder der Name oder die Nummer angegeben werden. Es können auch mehrere Signale für eine Aktion angegeben werden. Soll ein Signal ignoriert werden, so wird bei `Action` nichts angegeben. Die nachfolgende Liste listet die für ein Programm relevanten Traps auf.
Eine vollständige Liste erhält man, wenn man in der Console ein `kill -l` eingibt.

Signal	*Nr.*	*Beschreibung*
SIGHUB	1	Das Programm hängt
SIGINT	2	Ein Ctrl + C wurde gemacht
SIGQUIT	3	Sendet ein Ctrl + D
SIGFPE	8	Eine illegale mathematische Operation wurde gemacht
SIGKILL	9	Der Prozess wird mit kill beendet
SIGALRM	14	Alarm Clock (für Timer)
SIGTERM	15	Der Prozess wird mit kill beendet

Tabelle 1.8: Auswahl Traps

```
uws@tux>kill -l
 1) SIGHUB       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     16) SIGSTKFLT
17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU
25) SIGXFSZ     26) SIGVTALRM   27) SIGPROF     28) SIGWINCH
29) SIGIO       30) SIGPWR      31) SIGSYS      32) SIGRTMIN
35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3  38) SIGRTMIN+4
39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+4
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+4
47) SIGRTMIN+13 48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14
51) SIGRTMAX-13 52) SIGRTMAX-12 53) SIGRTMAX-11 54) SIGRTMAX-10
55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7  58) SIGRTMAX-6
59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

Listing 1.29: Alle Traps

Einige Beispiele, um Traps in einem Script abzufangen.

```
uws@tux>cat trap1.sh
#!/bin/env bash
# Funktion trap zum Abfangen von Signalen
# Name: trap1.sh
# Signal SIGINT (2) (Strg + C)
trap 'echo SIGINT erhalten' 2

i=0
while [ $1 -lt 5 ]
do
 echo "Habe Zeit, warte ein wenig!"
 sleep 2
 i=`expr $i + 1`
done
```

Listing 1.30: trap1.sh

```
uws@tux>cat trap2.sh
#!/bin/env bash
# Funktion trap zum Abfangen von Signalen
# Hier werden nun mehrere Signale abgefangen
# Name: trap2.sh
# Signal SIGINT (2) (Strg + C)
trap 'echo SIGINT erhalten' 2
# Signal SIGTERM (15) kill -TERM PID_of_trap2
trap 'echo SIGTERM erhalten' 15

i=0
while [ $1 -lt 5 ]
do
 echo "Warte noch immer! ($$)"
 sleep 5
 i=`expr $i + 1`
done
```

Listing 1.31: trap2.sh

```
uws@tux>cat trap3.sh
#!/bin/env bash
# Funktion trap zum Abfangen von Signalen
# Hier werden nun mehrere Signale abgefangen
# Name: trap3.sh
# Signal SIGINT und SIGTERM abfangen
trap 'echo SIGINT / SIGTERM erhalten' 2 15

i=0
while [ $1 -lt 5 ]
do
 echo "Das dauert aber! ($$)"
 sleep 5
 i=`expr $i + 1`
done
```

Listing 1.32: trap3.sh

```
uws@tux>cat trap4.sh
#!/bin/env bash
# Funktion trap zum Abfangen von Signalen
# Eine Funktion (Signalhandler) einrichten
# Name: trap4.sh
sighandler_INT() {
 printf "Habe das Signal SIGINT empfangen.\n"
 printf "Soll das Script beendet werden? (j/n) : "
 read
 if [[ $REPLY = "j" ]]
 then
  echo "Good by!"
  exit 0;
 fi
}

# Signal SIGINT abfangen
trap 'sighandler_INT' 2

i=0
while [ $1 -lt 5 ]
do
 echo "Bitte nicht stÃ¶ren! ($$)"
 sleep 5
 i=`expr $i + 1`
done
```

Listing 1.33: trap4.sh

1.12 Exec

Mit dem Befehl exec kann man Befehle ausführen, ohne eine neue Shell / Process zu starten.

```
uws@tux>bash
uws@tux>pstree
--gnome-terminal
    |-bash--bash--pstree

uws@tux>exec bash
uws@tux>pstree
--gnome-terminal
    |-bash--pstree
```

Listing 1.34: Exec Example

1.12.1 Ein-/Ausgabekanal

In dem nachfolgenden Beispiel werden neue Eingabe- und Ausgabekanäle definiert.

```
uws@tux>cat DemoExec.sh
#!/bin/env bash
#-------------------------------------------------------------------------
#
# Example to create File-Descriptors
#
#-------------------------------------------------------------------------
#
# Help
printf "\nDemo create File-Descriptor.\n"
printf "============================\n"
printf "exec Ziffer>Ziel\t\tAusgabekanal anlegen\n"
printf "exec Ziffer<Quele\t\tEingabekanal anlegen\n "
printf "exec Ziffer<>Datei\tEin/Ausgabekanal anlegen\n"
printf "Befehl >&Nr\t\t\t\tAusgabe auf Kanal Nr.\n"
printf "exec Ziffer>&-\t\t\tAusgabkanal loeschen\n"
printf "exec Ziffer<&-\t\t\tEingabekanal loeschen\n"
printf "\nJeder angelegter Kanal muss separat geloescht werden.\n"
printf "==================================================\n\n"

# Create test data
printf "First string\nSecond string\nThird string\n " > data1.txt
printf "one\ntwo\nthree\n" > data2.txt

# Create Ausgabekanal
printf " Anlegen Ausgabekanal 3.\n"
exec 3>data3.txt
printf " Anlegen Eingabekanal 4.\n"
exec 4<data1.txt
printf " Anlegen Ein-/Ausgabekanal 5.\n"
exec 5<>data5.txt

# Setzen Umleitung stdout und stderr
exec >data6.txt 2>data.err
echo "Das kommt in der Datei."

# Ausgabe in Kanal 3
echo "Das kommt in den Kanal 3." >&3
echo " Die einzelne Anweisung zur Umleitung ist nicht noetig."

# Erzeugen einer Fehlermeldung
```

```
ls new.txt

# Auslesen Kanal 4 und schreiben auf Kanal 3-
read A <&4
echo $A >&3

# Benutze den Ein-/Ausgabekanal 5.
tail -1 <&5
echo "Neue Zeile" >&5

# Schliessen der Kanaele
exec 3>&-
exec 4<&-
exec 5>&-
exec 5<&-

exit 0
```

Listing 1.35: DemoExec.sh

Kapitel 2

Schleifen / Bedingungen

2.1 Vergleichsparameter

Zahlen werden nicht mit =, >=, <= verglichen, sondern mit -eq, -ge, -gt, -ne, -lt, -le. Nachfolgend sind ein paar mögliche Vergleiche ausgelistet.

Ausdruck	*Beispiel*	*Erklärung*
-a Datei	`[ -a uschi.txt ]`	Wahr, wenn die Datei vorhanden ist
-d Verzeichnis	`[ -d /home ]`	Wahr, wenn das Verzeichnis vorhanden ist
-e Datei	`[ -e uschi.txt ]`	Wahr, wenn die Datei vorhanden ist
-f Datei	`[ -f uschi.txt ]`	Wahr, wenn die Datei vorhanden ist
-G Datei	`[ -G uschi.txt ]`	Wahr, wenn die Datei vorhanden ist und eff. Gruppen-ID gehört
-g Datei	`[ -g uschi.txt ]`	Wahr, wenn die Datei vorhanden ist und set-group-id gesetzt ist
-k Datei	`[ -k uschi.txt ]`	Wahr, wenn das Sticky Bit gesetzt ist
-L Datei	`[ -L uschi.sh ]`	Wahr, wenn es sich um ein Symbolischen Link handelt
-O Datei	`[ -O uschi.sh ]`	Wahr, wenn Datei vorhanden ist und eff User-ID gehört
-r Datei	`[ -r uschi.sh ]`	Wahr, wenn die Datei lesbar ist
-S Datei	`[ -S uschi.sh ]`	Wahr, wenn die Datei ein Socket ist
-s Datei	`[ -s uschi.txt ]`	Wahr, wenn die Datei grösser 0 ist
-u Datei	`[ -u uschi.sh ]`	Wahr, wenn das Bit set-user-id gesetzt ist
-w Datei	`[ -w uschi.txt ]`	Wahr, wenn die Datei existiert und Schreibzugriffe erlaubt sind
-x Datei	`[ -x uschi.sh ]`	Wahr, wenn die Datei existiert und die Ausführung erlaubt ist
-z str1	`[ -z "\$1" ]`	Wahr, wenn der Wert leer ist
-n String	`[ -n "\$name" ]`	Wahr, wenn die Variable nicht leer ist
str1 = str2	`[ "\$1" = "uwe" ]`	Wahr, wenn beide Werte identisch sind
str1 != str2	`[ "\$1" != "uwe" ]`	Wahr, wenn die Werte ungleich sind
z1 -eq z2	`[ 1 -eq \$summe ]`	Wahr, wenn beide Zahlen gleich groß sind
z1 -lt z2	`[ 10 -lt \$1 ]`	Wahr, wenn die erste Zahl kleiner ist als die zweite Zahl
z1 -gt z2	`[ 20 -gt \$1 ]`	Wahr, wenn die erste Zahl größer ist als die zweite Zahl
z1 -ne z2	`[ \$1 -ne 10 ]`	Wahr, wenn beide Zahlen ungleich sind
!ausdruck	`[ ! 1 -eq \$1 ]`	Wahr, wenn der Ausdruck falsch ist

Tabelle 2.1: Vergleiche

2.2 Case Anweisung

Mir der Anweisung case können die Kommandozeilen Parameter ausgewertet werden. Das Ende einer Auswertung wird mit einem doppelten Semikolon abgeschlossen, da ein Semikolon als Trennzeichen für Kommandos gilt. Eine case Anweisung wird mit einem esac (end case) abgeschlossen.

```
#!/bin/env bash
#Als erstes wird der Parameter uebergeben
action=$1

#Hier faengt die Auswertung and
case "$action" in
  -a)
     <Befehle>
     ;;
  -v)
     <Befehle>
     ;;

#Mehrfachauswertung
  -j*|-J*|-y*|-Y*)
     <Befehle> ;;
  *)
     echo " Usage: uschi.sh [-a] [-v]"
esac
exit 0
```

Listing 2.1: Beispiel case

2.3 If Anweisung

Möchte man eine Bedingung abfragen, so nutzt man hierzu eine `if` Abfrage. Die allgemeine Form der `if` Abfrage ist:

```
if Bedingung
 then Anweisung
 else Anweisung
fi
```

Listing 2.2: Syntax

Außerdem können mehrere `else if` Abfragen erstellt werden. Der Befehl hierzu ist `elif`.

```
if Bedingung1
 then Anweisung
 elif Bedingung2
  then Anweisung
 else Anweisung
fi
```

Listing 2.3: Syntax elif

Um abzufragen, ob eine Datei vorhanden ist, so nutzt man hierzu die Option `test -r`.

```
uws@tux>cat exist.sh
#!/bin/env bash
if test -r /home/uws/scripts/sicherung.sh
then
  bash /home/uws/scripts/startmount.sh
fi
```

Listing 2.4: Beispiel Datei existiert

Handelt es sich nicht um eine Datei, sondern um ein Verzeichnis, so wird hierzu die Option `-d` verwendet. Die Option `-L` prüft, ob es sich um ein symbolischen Link handelt. Mit der Option `-f` wird überprüft, ob es sich um eine gewöhnliche Datei handelt. Man kann mit `test` auch mehrere Überprüfungen ausführen. Dafür wird die Option `-a` (für und) und die Option `-o` (für oder) benutzt.

```
uws@tux>cat exist1.sh
#!/bin/env bash
if test -d /home/uws/scripts -a -f /home/uws/scripts/MyBackup.conf
then
  bash /home/uws/scripts/startmount.sh
fi
```

Listing 2.5: Beispiel Pfad check

Anstelle von `test` kann die Überprüfung mittels eckigen Klammern erfolgen.

```
uws@tux>cat exist2.sh
#!/bin/env bash
if [ -d /home/uws/scripts -a -f /home/uws/scripts/MyBackup.conf ]
then
  bash /home/uws/scripts/startmount.sh
fi
```

Listing 2.6: Beispiel Pfad Check

```
uws@tux>cat exist3.sh
#!/bin/env bash
if [ "$1" = "tux" ]; then
 echo "Hallo Pinguin."
else
 echo "Hallo $1, es wurde die \$1 Variable ausgelesen."
fi
exit 0
```

Listing 2.7: Beispiel Vergleich

Überprüfen, ob eine Datei existiert.

```
uws@tux>cat exist4.sh
#!/bin/env bash
strUser=$1
if [ ! -r ~/.dbuser/$strUser ]; then
  echo "File not exist"
else
  echo "File exist"
fi
```

Listing 2.8: Beispiel Datei exist

Existiert das Verzeichnis nicht, wird es angelegt.

```
if [ ! -d ~/.dbuser ]; then
  mkdir ~/.dbuser
fi
```

Listing 2.9: Beispiel Create Path

Ist die Variable leer? Bei `-n` => wahr, wenn gefüllt, `-z` => wahr, wenn leer.

```
if [ -z "${strV1}" ]; then
  printf "\nVariable is empty.\n"
fi
```

Listing 2.10: Beispiel Variable leer

```
#!/bin/env bash
Var1=""
[ -z "${Var1}" ] && echo "Variable leer: Ja" || echo "Variable leer: Nein"
```

Listing 2.11: Beispiel Variable leer

2.4 While / Until Schleife

Die Prüfung der Bedingung erfolgt bei `while` vor der Abarbeitung der Anweisung und bei `until` erst nach der Abarbeitung.

```
#!/bin/env bash
count=0
while [ $count -le 10 ]
do
  echo $count
  count=$[count+1] # Zaehler un 1 hochzaehlen
done
exit 0
```

Listing 2.12: Beispiel While

```
#!/bin/env bash
PING_HOST=$1
printf "\nWaiting for network "
while [ $(ping -w1 -c1 $PING_HOST | grep -c "0 received") - eq 1 ]; do
  printf "."
done

printf "\n\nNetwork is now up\n"
for i in $(grep noauto /etc/fstab | grep -o "[^]*"); do
  mount $i
done
```

Listing 2.13: Beispiel Network

```
#!/bin/env bash
while [ "$1" != "" ]
do
  [ "$1" == "-b" ] && BACKUP=true && printf "Backup now!\n" && shift
  [ "$1" == "-r" ] && RRSTORE=true && printf "Restore or what else.\n" &&
     shift
  [ "$1" == "-h" ] && HELPNEED=true && printf "Help needed?\n" && shift
done
```

Listing 2.14: Beispiel

```
#!/bin/env bash
i=1
until [ $1 -gt 6 ]
do
  printf "Tick Tack $i\n"
  i=$(( i+1 ))
done
```

Listing 2.15: Beispiel until

2.5 For Schleife

Die `for` Schleife erhält eine Liste von Werten und führt die Kommandos mit jedem Wert aus. Möchte man eine Schleife abbrechen, so wird hierzu der Befehl `break [n]` benutzt. Um wieder am Anfang zu springen, benutz man den Befehl `continue [n]`.

```
uws@tux>cat example.sh
#!/bin/env bash
for a in $1
do
  printf "\n\tUebergabe Wert: ${a}\n"
done
```

Listing 2.16: Beispiel For Schleife

Hier ein Beispiel für `break / continue`.

```
uws@tux>cat example1.sh
#!/bin/env bash
for a in 1 2 3 4 5
do
  if [ ${a} -eq 2 ]
  then
    continue
  fi
  if [ ${a} -eq 4 ]
  then
    break
  fi
  printf "\n\tWerte: ${a}\n"
done
```

Listing 2.17: Beispiel Break / Continue

```
uws@tux>cat example2.sh
#!/bin/env bash
i=1
wochentage="Montag Dienstag Mittwoch Donnerstag Freitag"
fo tage in ${wochentage}
do
  printf "\n\tWochentag $((i++)) : ${tage}\n"
done

uws@tux>./example2.sh
Wochentag 1 : Montag
Wochentag 2 : Dienstag
Wochentag 3 : Mittwoch
Wochentag 4 : Donnerstag
Wochentag 5 : Freitag
```

Listing 2.18: Example2.sh

```
uws@tux>cat example3.sh
#!/bin/env bash
i=1
cd ~/bin
fo item in *
do
  printf "\n\tItem $((i++)) : ${item}\n"
done

uws@tux>./example3.sh
Item 1 : example.sh
Item 2 : example1.sh
Item 3 : example2.sh
Item 4 : example3.sh
```

Listing 2.19: Example3.sh

```
uws@tux>cat example4.sh
#!/bin/env bash
i=1
fo file in /etc/[abcd]*.conf
do
  printf "\n\tFile $((i++)) : ${file}\n"
done

uws@tux>./example4.sh
File 1 : /etc/alsa-pulse.conf
File 2 : /etc/asound-pulse.conf
File 3 : /etc/blkid.conf
File 4 : /etc/dhcpclient.conf
```

Listing 2.20: Example4.sh

```
uws@tux>cat example5.sh
#!/bin/env bash
i=1
fo file in $(ls ~/bin/*.sh)
do
  printf "\n\tFile $((i++)) : ${file}\n"
done

uws@tux>./example5.sh
File 1 : /home/uws/bin/example.sh
File 2 : /home/uws/bin/example1.sh
File 3 : /home/uws/bin/example2.sh
File 4 : /home/uws/bin/example3.sh
File 5 : /home/uws/bin/example4.sh
File 6 : /home/uws/bin/example5.sh
```

Listing 2.21: Example5.sh

Beispiel einer `for` Schleife in einer Zeile.

```
uws@tux>for a in 01 02 03; do ls bild${a}.jpg; done
bild01.jpg
bild02.jpg
bild03.jpg
```

Listing 2.22: Eine Zeile

In dem nächsten Beispiel wird automatisch hochgezählt und in Schritten.

```
uws@tux>cat example6.sh
#!/bin/env bash
for ((i=1;i<=5;i++)); do
  echo "$i: "
done

#von 1 bis 10
for i in {1..10}
do
  printf "Number: $1\n"
done

#von 1 bis 10, aber in 2er Schritten
for i in {1..10..2}
do
  printf "Number: $i\n"
done
```

Listing 2.23: example6.sh

Nun wird eine Datei eingelesen und nur einzelne Spalten werden ausgegeben.

```
uws@tux>cat example7.sh
for SHARE in $(grep -i "nfs" /etc/fstab | cut -d' ' -f2)
do
  printf "\nShare to mount: $SHARE"
done
```

Listing 2.24: example7.sh

Die Ausgabe erfolgt mittels einer Array-Variable.

```
uws@tux>declare -a MyUser=('uws' 'paul' 'Hans')

uws@tux>for ((i=0;i<=2;i++));do echo User: ${MyUser[$i]};done
User: uws
User: paul
User: Hans

uws@tux>for i in ${!MyUser[*]};do printf "User: ${MyUser[$i]};done
User: uws
User: paul
User: Hans
```

Listing 2.25: example8

2.6 Functions

In einem Bash Script müssen die `functions` an Anfang definiert werden, da sie sonst nit gültig sind. `Functions` können auch in einer separaten Datei stehen, die dann mit per `./<pfad>/<datei>` geladen werden. Mit `source <pfad/datei>` kann man sie Datei auch laden. Die Angabe von `function` vor dem Namen ist nicht zwingend, man kann sie auch weglassen.

```
uws@tux>cat ExampleFunc.sh
#!/bin/env bash
function <name>()
{
  <Befehl>
  <Befehl>
}
```

Listing 2.26: ExampleFunc.sh

Kapitel 3

GUI

3.1 Dialog Aufruf

In der Shell kann man grafische Dialoge aufrufen, um in eine Interaktion mit dem Anwender zu treten. Die grafischen Dialoge können mit dem Befehl `dialog` auch in einer `SSH Sitzung` aufgerufen werden, da der Befehl keinen X-Server benötigt. Die Syntax lautet: `dialog [optionen] [dialog_aufruf] "Text"[breite] [höhe]`. Steht ein X-Server zu Verfügung, so kann man den Befehl `xdialog` nehmen. Damit man `xdialog` verwenden kann, muss die Software von http://xdialog.dyns.net heruntergeladen und installiert werden.
In einer KDE Umgebung kann man die Dialoge mit dem Befehl `kdialog` aufrufen. Eine Übersicht der Dialoge erhält man, wenn man `kdialog` ohne Parameter in der Shell aufruft.
Folgende Dialog Aufrufe gibt es:

Dialog	Beschreibung
--buildlist	Liste zum zusammen stellen
--calendar	Kalender
--checklist	Auswahlliste
--dselect	Auswahl Verzeichnis
--editbox	Editor
--form	Formbox
--fselect	Verzeichnis und Dateiauswahl
--gauge	Fortschrittsanzeige
--infobox	Informationsausgabe
--inputbox	Eingabe Dialog
--inputmenu	Menu
--menu	Menu
--mixedform	Mixed Form
--mixedgauge	Mixed Fortschrittsanzeige
--msgbox	Informationsausgabe
--passwordbox	Wie inputbox, die Eingabe wird mit Sternechen angezeigt
--passwordform	Wie passwordbox.
--pause	Pause mit Fortschrittsbalken
--prgbox	Programm Box
--programbox	Programm Box
--progressbox	Fortschritts Box
--radiolist	Wie checklist, aber hier nur eine Auswahl möglich
--rangebox	Range Box
--tailbox	Wie tail -f
--tailboxbg	Wie tail -f &
--textbox	Viewer für Ascii Dateien mit der Möglichkeit zum blättern
--timebox	Ausgabe der Uhrzeit
--treeview	Liste als Tree

Continued on next page

continued from previous page.

Dialog	Beschreibung
--yesno	Ja / Nein Dialog

Tabelle 3.1: Liste Dialog

Optionen	*Beschreibung*
–title	Überschrift für die Dialog Box
–backtitle	Überschrift für den Hintergrund der Shell
–begin	Startposition y x
–cancel-label	Überschreibt das Label des Cancel Buttons
–clear	Der Inhalt des Bildschirms wird nach dem beenden gelöscht
–exit-label	Überschreibt das Label des Exit Buttons
–defaultno	Setzt den Fokus auf den Button No
–extra-button	Zeigt zwischen Ok und Cancel einen extra Button an
–extra-label	Label Name des Extra Buttons
–print-maxsize	Ausgabe der Maximum Größe der Dialoge, wird auch mit dem Befehl `resize` angezeigt

Tabelle 3.2: Optionen für Dialog

Alle Optionen kann man sich in der Manpage anzeigen lassen. Mit `dialog -help` wird in der Shell die Hilfe angezeigt. Wird bei der Angabe der Höhe und Breite eine `0` angegeben, so werden Standardwerte hierfür genommen. Den Return Code, ob `Ok` oder `Abbrechen` gedrückt wurde, kann man mit der Variable `$?` auslesen. Der Return Code bei `Ok` ist eine `0` und bei `Abbrechen` eine `1`.

3.1.1 buildlist

Syntax: `dialog --buildlist <höhe> <breite> <höhe auswahl> <tag1> <item1> <status1> ...`

```
uws@tux>dialog --buildlist "Your choise:" 0 0 0 01 "Wert 1" off 02 "Wert 2"
    on 03 "Wert3" off
```

Listing 3.1: Buildlist

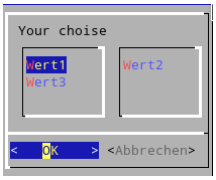

Abbildung 3.1: Dialog buildlist

3.1.2 Calendar

Syntax: `dialog --calendar "Text" <höhe> <breite> <tag> <monat> <jahr>`

```
uws@tux>dialog --calendar "Datum:" 0 0 17 08 2010
```

Listing 3.2: Calendar

Abbildung 3.2: Dialog calendar

3.1.3 Checklist

Syntax: `dialog --checklist "Text" <höhe> <breite> <zeilen> <einträge> <liste...>`

```
uws@tux>dialog --checklist "Bitte ausw"ahlen::" 20 30 3 \
01 "Erste Auswahl" on\
02 "Zweite Auswahl" off\
03 "Dritte Auswahl" off
```

Listing 3.3: Checklist

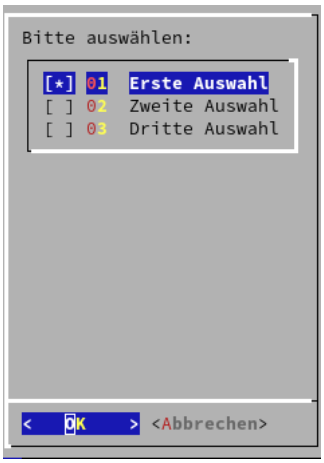

Abbildung 3.3: Dialog checklist

3.1.4 dselect

Syntax: `dialog --dselect <directory> <höhe> <breite>`

```
uws@tux>dialog --dselect /var/log 10 20
```

Listing 3.4: Dselect

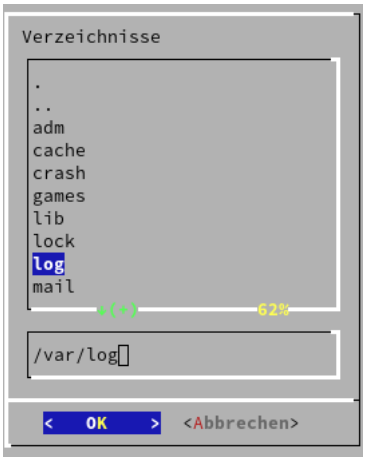

Abbildung 3.4: Dialog dselect

3.1.5 editbox

Syntax: `dialog --editbox <file> <höhe> <breite>`

```
uws@tux>dialog --dselect Bashing.out 0 0
```

Listing 3.5: Editbox

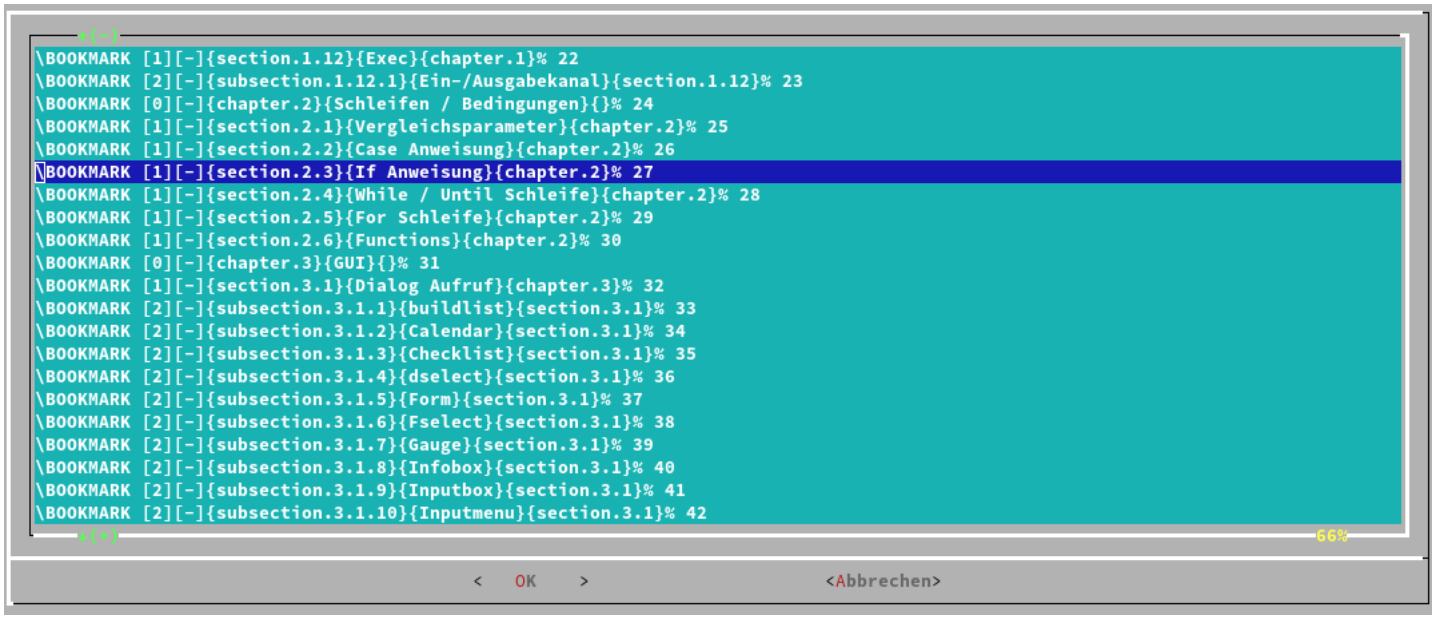

Abbildung 3.5: Dialog editbox

3.1.6 Form

Syntax: `dialog --form "Text" <höhe> <breite> <form_höhe> <text> <y> <x> <text> <y> <x> <flen> <ilen>`

```
uws@tux>dialog --form "Info" 0 0 0 "User: " 1 1 "uws" 1 10 10 0 \
"Shell: " 2 1 "Bash" 2 10 15 0
```

Listing 3.6: Form

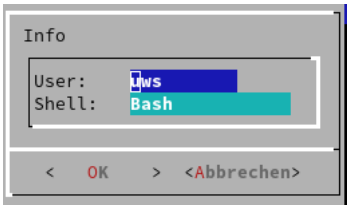

Abbildung 3.6: Dialog form

3.1.7 Fselect

Syntax: `dialog --fselect <directory> <höhe> <breite>`

```
uws@tux>dialog --fselect /var/log 10 20
```

Listing 3.7: Fselect

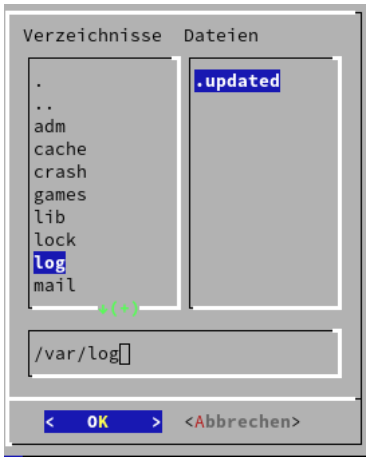

Abbildung 3.7: Dialog fselect

3.1.8 Gauge

Syntax: `dialog --gauge "Text" <höhe> <breite> <fertigstellung`

```
uws@tux>dialog --gauge "Fertig zu:" 7 30 50
```

Listing 3.8: Gauge

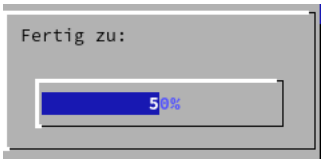

Abbildung 3.8: Dialog gauge

Beispiel eines laufenden Balkens.

```
for I in $(seq 1 100); do
  echo $I | dialog --backtitle "Progress Status" --gauge "Fertig zu:" 8 50 0
  sleep 0.01
done
```

Listing 3.9: Praxis Beispiel

3.1.9 Infobox

Syntax: `dialog --infobox "Text" <höhe> <breite>`

```
uws@tux>dialog --infobox "Es ist bald Mittag." 5 30
```

Listing 3.10: Infobox

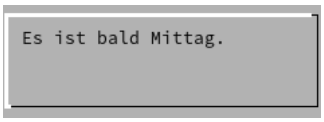

Abbildung 3.9: Dialog infobox

3.1.10 Inputbox

Syntax: `dialog --inputbox "Text" <höhe> <breite> <?>`

```
uws@tux>cat demo_inputbox.sh
#!/bin/env bash
dialog --inputbox "Dein Name bitte:" 10 60 2> eingabe.tmp
clear
echo "dein Name lautet: ‘cat eingabe.tmp‘"
exit 0
```

Listing 3.11: Inputbox

Abbildung 3.10: Dialog inputbox

Anstelle von `cat` kann man die Datei auch mit < einlesen. Für eine Passwort Eingabe gibt es die `passwordbox`.

3.1.11 Inputmenu

Syntax: `dialog --inputmenu "Text" <höhe> <breite> <menu_höhe> <tag1> <item1> ...`

```
uws@tux>dialog --inputmenu "Menu:" 30 50 10 "01" "Montag" "02" "Dienstag"
```

Listing 3.12: Inputmenu

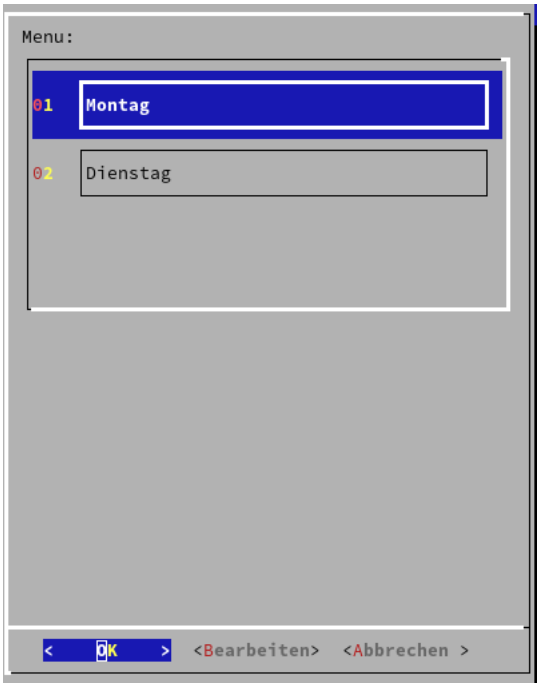

Abbildung 3.11: Dialog inputmenu

3.1.12 Menu

Syntax: `dialog --menu "Text" <höhe> <breite> <menu_höhe> <tag1> <item1> ...`

```
uws@tux>dialog --menu "Menu:" 30 50 10 "A" "Taschenrechner"
```

Listing 3.13: Menu

Abbildung 3.12: Dialog menu

3.1.13 Mixedform

Syntax: `dialog --mixedform "Text" <höhe> <breite> <form height> <label1> <l_y1> <l_x1> <item1> <i_y1> <i_x1>`

```
uws@tux>dialog --mixedform "Mixed Form:" 0 0 0 "A" 1 1 "B" 2 2 "C" "D" "E"
```

Listing 3.14: Mixedform

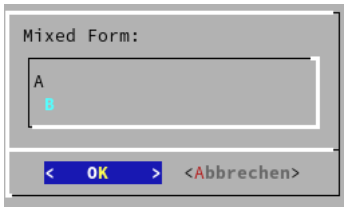

Abbildung 3.13: Dialog mixedform

3.1.14 Mixedgauge

Syntax: `dialog --mixedgauge "Text" <höhe> <breite> <percent> [tag1 item1]`

```
uws@tux>dialog --mixedgauge "Mixed Gauge" 0 0 30 "A" "text1"
```

Listing 3.15: Mixedgauge

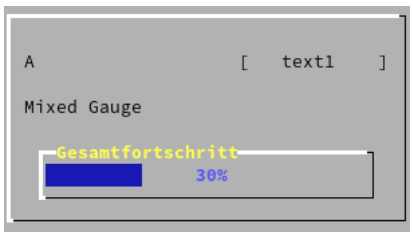

Abbildung 3.14: Dialog mixedgauge

3.1.15 Msgbox

Syntax: `dialog --menu "Text" <höhe> <breite>`

```
uws@tux>dialog --msgbox "Hier koennte ihr Text stehen." 8 30
```

Listing 3.16: Msgbox

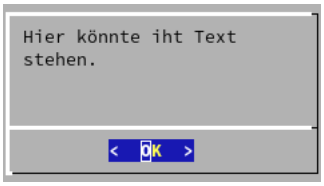

Abbildung 3.15: Dialog msgbox

3.1.16 Passwordbox

Aufruf und auch die Syntax ist die gleiche wie bei der `inputbox`.

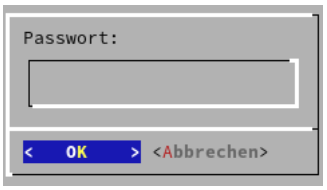

Abbildung 3.16: Dialog passwordbox

3.1.17 Passwordform

Syntax: `dialog --passwordform "Text" <höhe> <breite> <form_höhe> <label1> <l_y1> <l_x1> <item1> <i_y1> <i_x1> <flen1> <ilen1>`

```
uws@tux>dialog --passwordform "Passwort:" 0 0 0 "Type" 1 1 "Item1" 1 1 5 5
```

Listing 3.17: Passwordform

Abbildung 3.17: Dialog passwordform

3.1.18 Pause

Syntax: `dialog --pause "Text" <höhe> <breite> <sekunden>`

```
uws@tux>dialog --pause "Wait for " 8 0 20
```

Listing 3.18: Pause

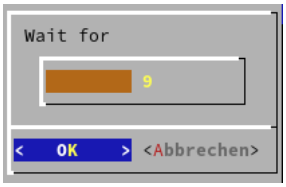

Abbildung 3.18: Dialog pause

3.1.19 Prgbox

Syntax: `dialog --pause "Textcommand" <höhe> <breite>`

```
uws@tux>dialog --prgbox "Prgbox: " "ls -l" 8 50
```

Listing 3.19: Prgbox

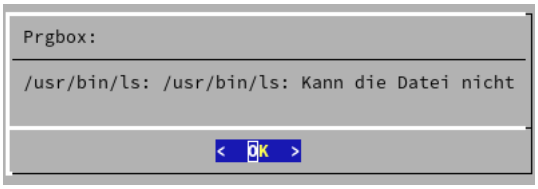

Abbildung 3.19: Dialog prgbox

3.1.20 Programbox

Syntax: `dialog --programbox "Text" <höhe> <breite>`

```
uws@tux>dialog --programbox "Command: " 0 0
```

Listing 3.20: Programbox

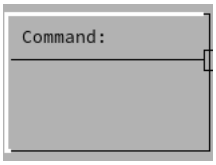

Abbildung 3.20: Dialog programbox

3.1.21 Progressbox

Syntax: `dialog --progressbox "Text" <höhe> <breite>`

```
uws@tux>dialog --progressbox "Fortschritt:" 10 60
```

Listing 3.21: Msgbox

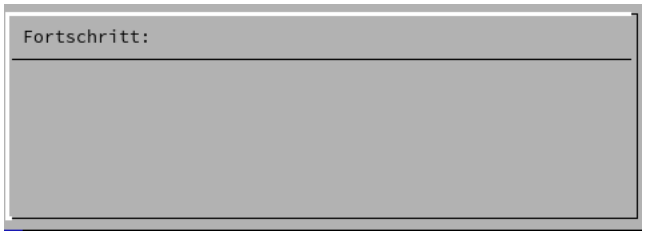

Abbildung 3.21: Dialog progressbox

3.1.22 Radiolist

Syntax: `dialog --radiolist "Text" <höhe> <breite> <list_höhe> <tag1> <item1> <status1>`

```
uws@tux>dialog --radiolist "Auswahl:" 0 0 0 "A" "Hund" "off" "02" "Katze" "
    On"
```

Listing 3.22: Radiolist

Abbildung 3.22: Dialog radiolist

3.1.23 Rangebox

Syntax: `dialog --rangebox "Text" <höhe> <breite> <min> <max> <default>`

```
uws@tux>dialog --rangebox "Range: " 0 5 1 20 5
```

Listing 3.23: Rangebox

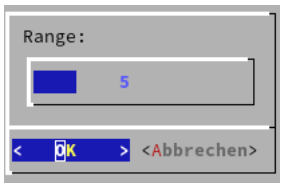

Abbildung 3.23: Dialog rangebox

3.1.24 Tailbox

Syntax: `dialog --tailbox <file> <höhe> <breite>`

```
uws@tux>dialog --tailbox "/home/uws/.bashrc" 10 40
```

Listing 3.24: tailbox

Abbildung 3.24: Dialog tailbox

3.1.25 Tailboxbg

Syntax: `dialog --tailboxbg <file> <höhe> <breite>`

```
uws@tux>dialog --tailboxbg "/home/uws/.bashrc" 10 40
```

Listing 3.25: tailbox

3.1.26 Textbox

Syntax: `dialog --textbox <file> <höhe> <breite>`

```
uws@tux>dialog --textbox "/home/uws/.bashrc" 25 70
```

Listing 3.26: Textbox

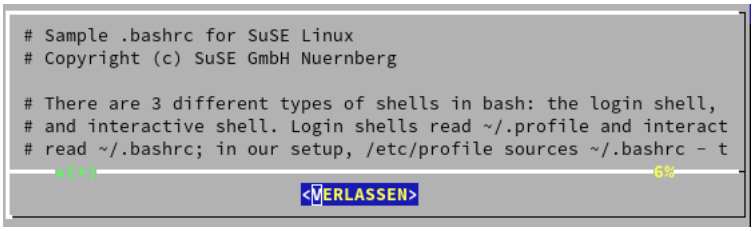

Abbildung 3.25: Dialog textbox

3.1.27 Timebox

Syntax: `dialog --timebox "Text" <höhe> <breite> <stunde> <minute> <sekunde>`

```
uws@tux>dialog --timebox "Aktuelle Zeit:" 0 30 13 09 22
```

Listing 3.27: Timebox

Abbildung 3.26: Dialog timebox

3.1.28 Treeview

Syntax: `dialog --treeview "Text" <höhe> <breite> <listheight> <tag1> <item1> <status1> <depth1>`

```
uws@tux>dialog --treeview "Liste" 0 0 0 "A" "root" off 1 "B" "usr" on 2
```

Listing 3.28: Treeview

3.1.29 YesNo

Syntax: `dialog --yesno "Text" <höhe> <breite>`

```
uws@tux>dialog --yesno "Steuern sparen?" 5 20
```

Listing 3.29: YesNo

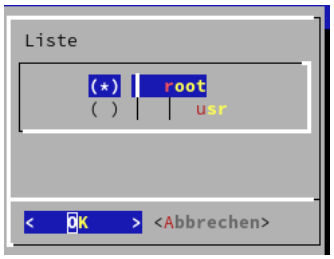

Abbildung 3.27: Dialog treeview

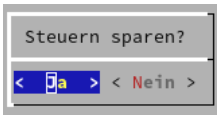

Abbildung 3.28: Dialog yesno

Kapitel 4

Beispiele

4.1 Backup Script

Mit diesem Script wird ein Backup von einem Verzeichnis ersteelt und die Sicherungsarchive werden hochgezählt. Ist das Sicherungsverzeichnis nicht vorhanden, so wird es angelegt.

```
uws@tux>cat backup.sh
#!/bin/env bash
#
# Seab@er Software - Backup Script
#
BACKUPDIR=/daten/backup
SOURCEDIR=/daten/bilder
TIMESTAMP=backup-timestamp.dat

# Backup Verzeichnis vorhanden?

if [ ! -d "${BACKUPDIR}" ]; then
  echo "Das Verzeichnis ${BACKUPDIR} ist nicht vorhanden"
  echo "und wird nun angelegt! "
  mkdir -p ${BACKUPDIR}
fi
set -- ${BACKUPDIR}/backup-???.tgz   # Alle Backups einlesen in $1, $2 usw.
lastname=${!#}                       # Letzter Backup Name
backupnr=${lastname##*backup-}       # Pfad und backup- entfernen
backupnr=${backupnr%%.*}             # Alles hinter dem ersten  "."
   Entfernen
backupnr=${backupnr//\?/0}           # Keine Backups vorhanden, dann 0
backupnr=$[10#${backupnr}]           # Fuehrende Nullen entfernen, (siehe in
                                     # der Bash-Manpage unter base#)

if [ "$[backupnr++]" -ge 999]; then # Erhoehen des  Wertes um 1
  echo "Error: Schon 999 Backups vorhanden! "
  exit 1
fi

backupnr=000${backupnr}    # Nullen voranstellen
backupnr=${backupnr: -3}  # Die letzten 3 Ziffern herausschneiden,
# das Leerzeichen vor dem - ist noetig.
filename=backup-${backupnr}.tgz
echo "Sichere veraenderte Daten in ${filename}."

# Es erfolgt ein inkrementelles Backup, Schalter -g

tar -czf ${BACKUPDIR}/${filename} -g ${BACKUPDIR}/$ {TIMESTAMP} ${SOURCEDIR}
```

Listing 4.1: Backup.sh

4.2 Benutzereingabe

Möchte man den Benutzer eine Eingabe ermöglichen, so wird hierzu der Befehl `read` verwendet. Werden mehr Worte als definierte Variablen eingegeben, so bekommt die letzte Variablen den Rest des Textes. Werden weniger Worte eingegeben, so bleiben die letzten Variablen leer.

```
uws@tux>cat DemoRead.sh
#!/bin/env bash
printf "\nBitte Werte eingeben: \n"
read str1 str2 str3 str4
printf "\nFolgende Werte wurden eingegeben: $str1 $str2 $str3 $str4\n"
```

Listing 4.2: DemoRead.sh

Für `read` gibt es folgende Optionen:

Optionen	*Beschreibung*
-n<anzahl>	Maximale Anzahl an Zeichen. Wurde die maximale Anzahl erreicht, so wird die Eingabe mit einem Enter automatisch abgeschlossen
-s	Silent Mode, die Eingabe wird nicht angezeigt. z.B. bei einer Passwort Abfrage
-t<sec>	Hiermit wird ein Timeout für die Eingabe gesetzt. Der Rückgabewert bei keiner Eingabe ist dann 0
-p	Ausgabe eines Textes

Tabelle 4.1: Optionen für Dialog

```
uws@tux>read -n1 -s -p "Press any key" CHOISE
```

Listing 4.3: Textausgabe

Wenn nach `read` keine Variable angegeben wird, so kann man den eingegebenen Wert mit `reply` auswerten.

```
uws@tux>cat DemoReadReply.sh
#!/bin/env bash
# Default Value for read is reply
printf "\nWas ist deine Lieblingsfarbe? \n"
read
printf "\nDeine Lieblingsfarbe ist: $REPLY\n"
printf "\nWas ist dein Lieblingsverein? \n"
read VEREIN
printf "\nDein Lieblingsverein ist: $VEREIN\n"
printf "und deine Lieblingsfarbe ist: $REPLY\n"
exit 0
```

Listing 4.4: DemoReadReply

4.3 Teilstring

Einen String kann man mit verschiedenen Techniken zerlegen. Mit % wird der String von rechts abgeschnitten und bei # von links.

```
uws@tux>FOO='1234567890'
uws@tux>echo ${FOO:0:6}
123456

uws@tux>echo ${FOO:2:3}
345

uws@tux>FOO=mein.txt.dat
echo ${FOO%.*}
mein.txt

uws@tux>echo ${FOO%%.*}
mein

uws@tux>FOO=/daten/test/mein.txt.dat # schneidet bis zum ersten / ab.
uws@tux>echo ${FOO#*/}
daten/text/mein.txt.dat

uws@tux>echo ${FOO#*.} # schneidet alles ab, bis zum ersten Punkt
txt.dat

uws@tux>echo ${FOO##*/} # schneidet alles ab, bis zum letzen /
mein.txt.dat

uws@tux>echo ${FOO##*.} # schneidet alles bis zum letzten Punkt ab
dat

uws@tux>BAT_STAT=99%
uws@tux>BAT_STAT=${BAT_STAT%%%*
uws@tux>echo $BAT_STAT
99
```

Listing 4.5: Teilstring Examples

4.4 Eval

Mit `Eval` is es möglich, Kommandos auszuführen, als würden sie in der Shell ausgeführt.

```
uws@tux>cat commands.sh
#!/bin/env bash
while true
do
  printf "\tCommand: "
  read
  eval $REPLY
done

uws@tux>./commands.txt
  Command: ls *.sh
commands.sh DemoEval.sh
  Comand: echo $$
16061
  Command: exit
uws@tux>
```

Listing 4.6: Commands.sh

Es besteht die Möglichkeit, indirekt auf eine Variable zugreifen zu können. In dem nächsten Beispiel wird dieses gemacht.

```
uws@tux>cat commands.sh
#!/bin/env bash
Mo=backupMo
Di=backupDi
Mi=backupMi
Do=backupDo
Fr=backupFr
Sa=backupSa
So=backupSo

tag=‘date +"%a"‘
eval backup=$$tag
printf "\n\tDas Backup Script $backup wird ausgefuehrt.\n\n"
./$backup

uws@tux>./DemoEval.sh

  Das Backup Script: backupDi wird ausgefuehrt.
./backupDi
```

Listing 4.7: DemoEval.sh

Im ersten Durchlauf wird aus `$$tag => $Di`. Die Variable `$Di` hat den Vert `backupDi` und dieses wird im zweiten Durchlauf der Variable `backup` zugewiesen.

4.5 Auswahlmenu mit Select

Mit einem select in einem Script kann man ein Auswahlmenu darstellen.

```
uws@tux>cat DemoSelect.sh
#!/bin/env bash
declare -Aa WERT # Array deklarieren
WERT["Montag Backup"]="backup.Mo.sh"
WERT["Dienstag Backup"]="backupDi.sh"
WERT["Mittwoch Backup"]="backupMi.sh"

printf "\n\tBitte Backup ausw"ahlen: \n\n"
#
# Das Ausrufezeichen vor der Array Variable gibt die Array Index No.
# aus und das @ Zeichen zwischen den [] gibt dann alle Index No. aus.
#
select Backupin "${!WERT[@]}";
do
  BACKUPSCRIPT=${WERT[${BACKUP}]}
  TITLE="${BACKUP}"
  printf "\n\tDas ${TITLE} mit dem Script ${BACKUPSCRIPT} wird ausgefuehrt.\
     n\n"
  ~/bin/${BACKUPSCRIPT}
  exit 0
done

uws@tux>./DemoSelect.sh
  Bitte Backup auswaehlen:

1) Dienstag Backup
2) Montag Backup
3) Mittwoch Backup
#? 2

  Das Montag Backup wird mit dem Script backupMo.sh ausgefuehrt.
```

Listing 4.8: DemoSelect.sh

Hier ist noch ein weiteres Beispiel für eine select Anweisung.

```
uws@tux>cat DemoSelect1.sh
#!/bin/env bash
WDR2="http://www.wdr2.de"
SPORT="http://www.sportschau.de"

printf "\n\tBitte Web Seite auswaehlen: \n\n"
select A in WDR2\ Seite Sportschau\ Seite;
do
  case $A in
  WDR*)
    printf "\n\tFolgende URL wird aufgerufen: ${WDR2}.\n\n"
    firefox ${WDR2}
    exit 0
    ;;
  SPORT*)
    printf "\n\tFolgende URL wird aufgerufen: ${SPORT}.\n\n"
    firefox ${SPORT}
    exit 0
    ;;
  esac
done

uws@tux>./DemoSelect1.sh
```

```
  Bitte Web Seite auswaehlen:

1) WDR Seite
2) Sportschau Seite
#? 1

  Folgende URL wird aufgerufen: http://www.wdr2.de.
```

Listing 4.9: DemoSelect1.sh

4.6 Datei zur Laufzeit erstellen

Eine Datei zur Laufzeit kann mit der sogenannten `Here-Dokumentation` erstellt werden. Als Trenner kann jede Zeichenfolge verwendet werden, meistens wird `EOF` als Trenner verwendet. Alles was zwischen den Trenner steht, wird in eine neue Datei ausgegeben.

```
uws@tux>cat DemoCreateFile.sh
#!/bin/env bash
cat <<EOF > "NewScript"
#!/bin/env bash
# Laufzeiterstelltes Script
#
printf "\n\tHier geht es los!\n"
ls -lash
EOF
```

Listing 4.10: DemoCreateFile.sh

Wird in einem Script Taps verwendet, so muss ein - Zeichen vor dem Trenner gesetzt werden. Dann werden die Taps ignoriert.

```
uws@tux>cat DemoCreateFileTaps.sh
#!/bin/env bash
cat <<-EOF > "NewScriptTaps"
#!/bin/bash
# Laufzeiterstelltes Script mit Taps
#
if test -r /home/uws/bin/test.txt
then
  printf "\n\tHier geht es los!\n"
  ls -lash
fi
EOF
```

Listing 4.11: DemoCreateFileTaps.sh

4.7 Scriptoptionen

Möchte man in einem Script Optionen / Parameter übergeben und auswerten, so geschieht das mit `getopts`. Alle Optionen / Parameter können in beliebiger Reihenfolge gesetzt werden. Erfolgt nach einer Option / Parameter ein `Doppelpunkt`, wie im unteren Beispiel nach dem `f`, so wird für diese Option / Parameter ein Wert benötigt.

```
uws@tux>cat DemoOption.sh
#!/bin/env bash
while getops "abf:hv" Arg; do
  case ${Arg} in
    a) printf "Die Option -a wurde "ubergeben.";;
    b) printf "Die Option -b wurde "ubergeben.";;
    f) if [ -f ${OPTARG} ]; then
         filename=${OPTARG}
       fi
       ;;
    h) printf "Die Syntax lautet: ...";;
    v) verbose=y;;
  esac
done
```

Listing 4.12: DemoOption.sh

Script Optionen / Parameter können auch mit einer Kombination von `while` und `case` ausgewertet werden.

```
uws@tux>cat DemoDemoOptionWithCase.sh
#!/bin/env bash
VERBOSE="0"
OUTFILE=""

while [ $# -gt 0 ]
do
  case $1 in
    -h|--help)
      printf "\n"
      printf "Folgende Optionen sind zulaessig:\n"
      printf "-h|--help           : Diese Hilfe\n"
      printf "-v|--verbose        : Statusinformatione n\n"
      printf "-o|--outfile <file> : Ausgabe in <file>\ n\n"
      exit 0
      ;;
    -v|--verbose)
      VERBOSE="1"
      ;;
    -o|--outfile)
      shift # Parameter $2 wird $1
      if [ -z "$1" ]
      then
        printf "Error: No File Name\n" 1>&2
        exit 1
      fi
      OUTFILE="$1"
      ;;
    *)
      printf "Unbekannte Option: $1\n" 1>&2
      exit 1
      ;;
  esac
  shift
```

```
done
```

Listing 4.13: DemoOptionWithCase.sh

4.8 Ausgabe Version

Die Ausgabe der Version seines Scripts könnte folgenden Inhalt haben.

```
uws@tux>./DemoVersion.sh -V
DemoVersion.sh  1.2
Copyright Â© 2018 Seab@er Software
Lizenz GPLv3+: GNU GPL Version 3 oder hoeher <http://gnu.org/licenses/gpl.
   html>
Dies ist eine freie Software: Sie koennen sie aendern  und weitergeben.
Es gibt keinerlei Garantien, soweit wie es das Gesetz erlaubt.

Geschrieben von Uwe Schimanski
```

Listing 4.14: DemoVersion.sh

4.9 Ausgabe Datei Name

Die Ausgabe des Script Namens wird mit `basename $0` gemacht.

```
uws@tux>cat DemoBasename.sh
#!/bin/env bash
echo Script Name: $(basename $0)
```

Listing 4.15: DemoBasename.sh

Möchte man aus einer Pfadangabe nur den Dateiname haben, so wird hierzu nur `basename` mit dem Pfad aufgerufen. Soll auch die Extension abgeschnitten werden, so wird nach der Angabe der Datei noch die Extension geschrieben.

```
uws@tux>basename /daten/script/MyScript.py
MyScript.py

uws@tux>basename /daten/script/MyScript.py .py
MyScript
```

Listing 4.16: DemoBasename1.sh

4.10 Let

Mit `let` kann man Berechnungen durchführen.

```
uws@tux>let "m=4*1024" && echo -e "\tErgebnis: $m"
  Ergebnis: 4096

@uws@tux>cat DemoLet.sh
#!/bin/env bash
let wert1=20
let wert2=10

let m=$wert1+$wert2 && echo -e "Ergebnis: $m"
uws@tux>bash DemoGerade.sh

Wert von Day: 326
Wert von MDAY: 0
```

Listing 4.17: DemoLet.sh

4.11 Network

4.11.1 Up / Down

Ein einfacher Netzwerktest, ob das Netz vorhanden ist..

```
uws@tux>bash DemoNetwork.sh
#!/bin/env bash
HOST_NAME=\$1

printf "Warte auf das Netz"

while [ $(ping -w1 -c1 $HOST_NAME | grep -c "0 received") -eq 1 ]; do
  printf "."
done
printf "\nNetz ist up.\n"
```

Listing 4.18: DemoNetwork.sh

4.12 Generate MAC-Adresse / UUID

Braucht man eine MAC-Adresse für eine Netzwerkkonfiguration von virtuellen Maschinen und auch eine UUID dazu, so kann man das mit dem folgenden Script machen.

```
uws@tux>cat GenMacUuid.sh
#!/usr/bin/env bash
# Generate MAC-Address and UUID
#  Author: Uwe Schimanski
# Version: 19.01.08
#
   ------------------------------------------------------------------------------


printf "\nGenereate MAC-Address with prefix\n
   ==================================\n"
printf "MAC: 00-60-2F-%02X-%02X-%02X\n" $[RANDOM%256] $[RANDOM%256] $[RANDOM
   %256]
printf "MAC: 00-60-2F-" && cut -b 7-11,24-26 /proc/sys/kernel/random/uuid
printf "\nGenerate MAC-Address without prefix\n"
printf "===================================\n"
printf "MAC: %02X-%02X-%02X-%02X-%02X-%02X\n"  $[RANDOM%256] $[RANDOM%256] $
   [RANDOM%256] $[RANDOM%256] $[RANDOM%256] $[RANDOM%256]

UUID1=$(tr -dc a-h0-9 < /dev/random | head -c 8 | xargs)
UUID2=$(tr -dc a-h0-9 < /dev/random | head -c 4 | xargs)
UUID3=$(tr -dc a-h0-9 < /dev/random | head -c 4 | xargs)
UUID4=$(tr -dc a-h0-9 < /dev/random | head -c 4 | xargs)
UUID5=$(tr -dc a-h0-9 < /dev/random | head -c 12 | xargs)
printf "\nGenerate UUID\n"
printf "=============\n"
printf "UUID: $UUID1-$UUID2-$UUID3-$UUID4-$UUID5\n"
printf "UUID: " && cat /proc/sys/kernel/random/uuid

uws@tux>bash GenMacUuid.sh

Genereate MAC-Address with prefix
==================================
MAC: 00-60-2F-90-FB-BB
MAC: 00-60-2F-86-bc-60

Generate MAC-Address without prefix
===================================
MAC: E0-AF-5F-91-7E-30

Generate UUID
=============
UUID: 5df61c8g-7f5e-46d6-621b-ddgcg3a4gegf
UUID: cf11bf04-e9bc-484f-8642-92ca5c636167
```

Listing 4.19: Generiere MAC / UUID