A Way to (S)hell – Einstieg in die Programmierung mit Bash

Chemnitzer Linux-Tage 2025

22. März 2025

Susanne Schütze
Linux Consultant
B1 Systems GmbH
schuetze@b1-systems.de

Inhaltsverzeichnis

Downloads / Unterlagen

Die Folien und ein Handout sind hier zu finden:

https://share.b1-systems.de/index.php/s/DsNuPEV7zRM5413

Einkaufsliste

- aktuelles Linux (je nach Distro gibt es andere Eastereggs ;-))
- Terminal (`bash`)
- Texteditor (`vim`, `emacs`, `nano`, `geany`, `gedit`, `kate`, `Leafpad`, `Mousepad`...)
- LibreOfficeWriter ist kein Texteditor!
- Packages:
  - `shellcheck` (`https://www.shellcheck.net/`)
  - `sl`
  - `cowsay`
  - `figlet`
  - `fortune-mod`

Einkaufsliste

- aktuelles Linux (je nach Distro gibt es andere Eastereggs ;-))
- Terminal (`bash`)
- Texteditor (`vim`, `emacs`, `nano`, `geany`, `gedit`, `kate`, `Leafpad`, `Mousepad`...)
- LibreOfficeWriter ist kein Texteditor!
- Packages:
  - `shellcheck` (`https://www.shellcheck.net/`)
  - `sl`
  - `cowsay`
  - `figlet`
  - `fortune-mod`

Schreibe ein Rezept

Befehle zusammenpacken

- den Lieblings-Texteditor benutzen
- mehrere Befehle, die auf einmal bzw. direkt nacheinander ausgeführt werden sollen (eine Idee)
- Befehle in der Reihenfolge aufschreiben, in der sie ausgeführt werden sollen
- beim Speichern Dateiextension (`.sh`) nicht notwendig, kann aber bei der Übersicht helfen
- ausführen und Spaß haben

Skript ausführen

```
1 $ ./name #sourcen erfordert Ausführungsrechte
2 $ bash name #bash mach mal name
```

Schreibe ein Rezept

Befehle zusammenpacken

- den Lieblings-Texteditor benutzen
- mehrere Befehle, die auf einmal bzw. direkt nacheinander ausgeführt werden sollen (eine Idee)
- Befehle in der Reihenfolge aufschreiben, in der sie ausgeführt werden sollen
- beim Speichern Dateiextension (`.sh`) nicht notwendig, kann aber bei der Übersicht helfen
- ausführen und Spaß haben

Skript ausführen

```
1 $ ./name #sourcen erfordert Ausführungsrechte
2 $ bash name #bash mach mal name
```

Alles in einer Zeile

Beispiel: alles in einer Zeile

```
Zahl= ;Wort= ;i=; for ((i=1 ; i<=15 ; i++)); do Zahl=$((RANDOM%134+1)); Wort="$(sed -n "$Zahl p" liste.txt)"; echo $Wort; echo $Wort >> dat.txt; done; echo -e '\n \n \t' " Viel Glück beim Merken"; sleep 2m; clear; echo -e '\n' "Mach mal 10 min was Anderes!"; sleep 10m; echo -e '\n \a' "Es geht jetzt los" '\n'; Ergeb= ; r=0; f=0; echo "Schreibe die gemerkten Worte in der richtigen Reihenfolge auf"; for i in `seq 1 15`; do read Ergeb$i; if [ "$(eval echo \$Ergeb$i)" = "$(sed -n "$i p" dat.txt)" ]; then echo -e "Richtig!" '\n'; ((r++)); else echo -e "Falsch!" '\n'; ((f++)); fi; done; echo "Die richtigen Wörter waren:"; cat dat.txt; echo "Du hast " $r " Wörter richtig und " $f " Wörter falsch."; rm dat.txt
```

Eingerückt

Beispiel eingerückter Code 1/2

```
Z=  ; E=
W=  ; r=0
i=  ; f=0
for ((i=1 ; i<=15 ; i++)) ;do
  Z=$((RANDOM%134+1))
  W="$(sed -n "$Z p" liste.txt)"
  echo $W
  echo $W >> dat.txt
done
echo -e '\n \n \t'
  ↪  " Viel Glück beim Merken"
sleep 2m
clear
echo -e '\n' "10 min Warten!"
sleep 10m
```

Beispiel eingerückter Code 2/2

```
echo -e '\n \a' "Es geht jetzt los"
echo "Schreibe die Wörter auf:"
for i in `seq 1 15` ;do
  read E$i
  if [ "$(eval echo \$E$i)" = "$(sed -n
  ↪  "$i p" dat.txt)" ]
  then echo -e "Richtig!" '\n'
    ((r++))
  else echo -e "Falsch!" '\n'
    ((f++))
  fi
done
echo "Du hast $r richtig und $f falsch."
rm dat.txt
```

Sprechende Namen im Skript

Beispiel sprechende Namen im Code (gekürzt)

```
1 #!/bin/bash
2 Zahl=
3 Wort=
4 i=
5 for ((i=1 ; i<=15 ; i++))
6 do
7   Zahl=$((RANDOM%134+1))
8   Wort="$(sed -n "$Zahl p" liste.txt)"
9   echo $Wort
10  echo $Wort >> dat.txt
11 done
12 echo -e '\n \n \t' " Viel Glück beim Merken"
13 sleep 2m
14 clear
15 echo -e '\n' "Mach mal 10 min was Anderes!"
```

Wer braucht schon Erklärungen im Code

```
# Niemand hat die Absicht, Hash-Tags als Kommentar zu benutzen
Der Code dokumentiert sich stets von selbst
#Für Kommentare wird der ganze Latten-Zaun benutzt
#Die Kommentare waren zuerst da!
```

Wer braucht schon Erklärungen im Code

```
# Niemand hat die Absicht, Hash-Tags als Kommentar zu benutzen
Der Code dokumentiert sich stets von selbst
#Für Kommentare wird der ganze Latten-Zaun benutzt
#Die Kommentare waren zuerst da!
```

Wer braucht schon Erklärungen im Code

```
# Niemand hat die Absicht, Hash-Tags als Kommentar zu benutzen
Der Code dokumentiert sich stets von selbst
#Für Kommentare wird der ganze Latten-Zaun benutzt
#Die Kommentare waren zuerst da!
```

Mit Kommentaren

Beispiel mit Kommentaren (gekürzt)

```
1 #!/bin/bash
2 # Variablen-Deklaration
3 Zahl=
4 Wort=
5 i=
6 # For Schleife zum Durchzählen
7 for   ((i=1 ; i<=15 ; i++)) ;do
8   # Zufallszahl ausgeben
9   # echo $RANDOM # für das Debugging
10  Zahl=$((RANDOM%134+1))
11  #Die Zufallszahl wird in der Variable "Zahl" gespeichert. Nun soll die Zeile mit
    ↪  der entsprechenden Nummer ausgegeben werden
12  Wort="$(sed -n "$Zahl p" liste.txt)"
```

Typographische Konventionen

Sind nirgendwo eindeutig festgeschrieben. Folgende werden häufig eingesetzt:

- sprechende Namen für Funktionen und Variablen verwenden
- Quotes hilfreich wenn Variablen Leerzeichen enthalten; geschweifte Klammern nur für bestimmte Situationen erforderlich `"${variable}"`
- `test` wird häufig als `[ ]` oder `[[ ]]` geschrieben, wobei letzteres für die Bash sauberer ist
- Variablen nicht in Großbuchstaben, um nicht mit Umgebungsvariablen und OS-Variablen ins Gehege zu kommen
- `=` vs `==` – meistens wird `==` verwendet wie in anderen Programmiersprachen

bang!bang!bang!

- die Skript-Sprache verraten
- das Shebang ist wichtig, wenn Irrtümer vermieden werden sollen
- das Skript wird nicht unbedingt mit der Skriptsprache vorangestellt gestartet
- das Skript kann aus einer anderen Shell aufgerufen werden, z. B. `csh`, `fish`, `zsh`

Shebang

```
1    #!/bin/bash
```

Vorsicht! Der Pfad kann je nach Distribution anders sein: `whereis bash`

bang!bang!bang!

- die Skript-Sprache verraten
- das Shebang ist wichtig, wenn Irrtümer vermieden werden sollen
- das Skript wird nicht unbedingt mit der Skriptsprache vorangestellt gestartet
- das Skript kann aus einer anderen Shell aufgerufen werden, z. B. `csh`, `fish`, `zsh`

Shebang

```
1    #!/bin/bash
```

Vorsicht! Der Pfad kann je nach Distribution anders sein: `whereis bash`

bang!bang!bang!

- die Skript-Sprache verraten
- das Shebang ist wichtig, wenn Irrtümer vermieden werden sollen
- das Skript wird nicht unbedingt mit der Skriptsprache vorangestellt gestartet
- das Skript kann aus einer anderen Shell aufgerufen werden, z. B. `csh`, `fish`, `zsh`

Shebang

```
1    #!/bin/bash
```

Vorsicht! Der Pfad kann je nach Distribution anders sein: `whereis bash`

bang!bang!bang!

- die Skript-Sprache verraten
- das Shebang ist wichtig, wenn Irrtümer vermieden werden sollen
- das Skript wird nicht unbedingt mit der Skriptsprache vorangestellt gestartet
- das Skript kann aus einer anderen Shell aufgerufen werden, z. B. `csh`, `fish`, `zsh`

Shebang

```
1    #!/bin/bash
```

Vorsicht! Der Pfad kann je nach Distribution anders sein: `whereis bash`

Lies das Handbuch

```
man man
man bash
info bash
apropos / whatis
```

Lies das Handbuch

```
man man
man bash
info bash
apropos / whatis
```

Lies das Handbuch

```
man man
man bash
info bash
apropos / whatis
```

Lies das Handbuch

```
man man
man bash
info bash
apropos / whatis
```

Anfangen zu Programmieren

- an die Shells
- auf die Texteditoren
- mitmachen!

Anfangen zu Programmieren

- an die Shells
- auf die Texteditoren
- mitmachen!

Anfangen zu Programmieren

- an die Shells
- auf die Texteditoren
- mitmachen!

Was ist in der Tasse?

- denk dir ein Wort oder Buchstaben aus
- weise diesem Wort oder Buchstaben etwas zu
- keine Gedanken um Datentypen machen

Variablen – Deklaration

```
1 Tasse=10
2 B1-Tasse=Pinguin
3 command=$(fortune)
```

Variablen – Anwendung

```
1 echo $Tasse
2 echo $B1-Tasse
3 echo $command
```

Was ist in der Tasse?

- denk dir ein Wort oder Buchstaben aus
- weise diesem Wort oder Buchstaben etwas zu
- keine Gedanken um Datentypen machen

Variablen – Deklaration

```
1 Tasse=10
2 B1-Tasse=Pinguin
3 command=$(fortune)
```

Variablen – Anwendung

```
1 echo $Tasse
2 echo $B1-Tasse
3 echo $command
```

Variablen

Was besser nicht gemacht wird:

Unpassender Variablen-Name

```
1 $ apt=moo
2 $ echo $apt
3 moo
```

Leerzeichen um das Gleichheitszeichen

```
1 $ Tasse = lecker
2 command Tasse not found
```

Aufgabe 5 min

- Texteditor öffnen (nicht LibreOfficeWrite!)
- schreibe folgende Variablen auf:

Monat	mit dem Befehl `date` die Zahl des aktuellen Monats ermitteln
Tag	mit dem Befehl `date` die Zahl des aktuellen Tages ermitteln
Jahr	mit dem Befehl `date` die Zahl des aktuellen Jahres ermitteln
Event	leere Variable
Ostersonntag	leere Variable
Ostermonat	leere Variable

- abspeichern und testen mit `bash name-deiner-Datei`

Kommunikation mit Usern

Wenn Text im Terminal ausgegeben werden soll, wird `echo` als Befehl verwendet:

Beispiel Ausgabe

```
1 echo "Was macht ein Pirat am Computer?"
2 echo "Enter drücken"
```

Wenn du den Nutzer des Programms etwas eingeben lassen willst, kannst du `read` verwenden:

Beispiel Eingabe

```
1 echo "Was macht ein Pirat am Computer?"
2 read Antwort
3 echo "Enter drücken"
4 echo "deine Antwort war $Antwort"
```

Kommunikation mit Usern

Wenn Text im Terminal ausgegeben werden soll, wird `echo` als Befehl verwendet:

Beispiel Ausgabe

```
1 echo "Was macht ein Pirat am Computer?"
2 echo "Enter drücken"
```

Wenn du den Nutzer des Programms etwas eingeben lassen willst, kannst du `read` verwenden:

Beispiel Eingabe

```
1 echo "Was macht ein Pirat am Computer?"
2 read Antwort
3 echo "Enter drücken"
4 echo "deine Antwort war $Antwort"
```

Aufgabe 5 min

- frag den User deines Skriptes, ob er die Tage bis Weihnachten oder Ostern berechnen will
- speichere die Antwort deines Users in eine Variable
- gib die Entscheidung deines Users in der Shell aus

f(x)

Funktionen können:

- Code-Wiederholungen vermeiden
- übersichtlichere Strukturen schaffen
- die Lesbarkeit erleichtern

Beispiel Funktion

```
1 #/bin/bash
2 datumsfrage()
3 {
4   cowsay -f dragon "Haben wir heute den $(date)?"
5 }
6 datumsfrage
```

f(x)

Funktionen können:

- Code-Wiederholungen vermeiden
- übersichtlichere Strukturen schaffen
- die Lesbarkeit erleichtern

Beispiel Funktion

```
1 #/bin/bash
2 datumsfrage()
3 {
4   cowsay -f dragon "Haben wir heute den $(date)?"
5 }
6 datumsfrage
```

Aufgabe 5 min

- erstelle eine Funktion, die die bereits geschriebene Frage/Antwort eures Users umschließt
- erstelle einen Funktionsaufruf

Wie funktioniert die Logik?

Warum testen?

- ich will wissen, ob etwas richtig ist
- ich will abhängig vom Wert einer Variablen etwas tun
- ich will etwas nicht tun, wenn ...

Beispiel Test

```
1 read -p "Was packst du in die Tasse? " Tasse
2 test $Tasse = "Kaffee"
3 echo $?
4 [ $Tasse = "Kaffee" ] # andere Schreibweise
5 echo $?
6 [[ $Tasse = "Kaffee" ]] # andere Schreibweise
7 echo $?
```

Wie funktioniert die Logik?

Warum testen?

- ich will wissen, ob etwas richtig ist
- ich will abhängig vom Wert einer Variablen etwas tun
- ich will etwas nicht tun, wenn ...

Beispiel Test

```
1 read -p "Was packst du in die Tasse? " Tasse
2 test $Tasse = "Kaffee"
3 echo $?
4 [ $Tasse = "Kaffee" ] # andere Schreibweise
5 echo $?
6 [[ $Tasse = "Kaffee" ]] # andere Schreibweise
7 echo $?
```

Aufgabe 5 min

- schreibe eine neue Funktion, in der du:
- prüfst, ob dein User „Ostern“ eingegeben hat
- prüfst, ob dein User „Weihnachten“ eingegeben hat

Wie funktioniert die Logik 1/3

UND prüfen, ob alle Tests wahr sind

Beispiel UND

```
1 B1_Tasse="Tee"
2 Tasse="Kaffee"
3 test $B1_Tasse = "Tee" && [ $Tasse = "Kaffee" ]
4 echo $?
```

Wie funktioniert die Logik 2/3

ODER prüfen, ob ein Test von mehreren Werten wahr ist

Beispiel ODER

```
1 B1_Tasse="Tee"
2 Tasse="Kaffee"
3 test $B1_Tasse = "1x" || [ $Tasse = "Kaffee" ]
4 echo $?
```

Wie funktioniert die Logik 3/3

`nicht` prüfen, ob ein Test falsch ist

Beispiel NICHT

```
1 Tasse="Tee"
2 test $Tasse != "Kaffee"
3 echo $?
```

Die Logik kann beliebig verknüpft werden – also viel Spaß!

Aufgabe 5 min

- nutzt die Funktion, die ihr zuletzt geschrieben habt
- prüft, ob euer User „Ostern“ oder „Weihnachten“ eingegeben hat
- prüft, ob euer User Blödsinn geschrieben hat

Wenn Du nicht sofort tust, was ich sage, dann ...

`if`

Wenn, prüfe ob das wahr ist, dann...

Beispiel Verzweigung

```
1 if test "$Tasse" = "Kaffee"; then echo "lecker"; fi
2 if [[ $Tasse == "Tee" ]] ; then echo "lecker" ; else cowsay
  ↪  "Ostfriese";fi
3 if [[ -z $Tasse ]] ; then echo "Tasse leer"; elif test $Tasse =
  ↪  "Kaffee" ; then cowsay "lecker $Tasse " ; else cowsay
  ↪  "So'n Tee";fi
```

`else` muss nicht unbedingt verwendet werden

`elif Bedingung` mehrfach Verkettungen

`-z` die Länge der ZEICHENKETTE ist Null

`=` vs `==` egal, solange Leerzeichen verwendet werden

Wenn Du nicht sofort tust, was ich sage, dann ...

`if`

Wenn, prüfe ob das wahr ist, dann...

Beispiel Verzweigung

```
1 if test "$Tasse" = "Kaffee"; then echo "lecker"; fi
2 if [[ $Tasse == "Tee" ]] ; then echo "lecker" ; else cowsay
  ↪  "Ostfriese";fi
3 if [[ -z $Tasse ]] ; then echo "Tasse leer"; elif test $Tasse =
  ↪  "Kaffee" ; then cowsay "lecker $Tasse " ; else cowsay
  ↪  "So'n Tee";fi
```

`else` muss nicht unbedingt verwendet werden

`elif` Bedingung mehrfach Verkettungen

`-z` die Länge der ZEICHENKETTE ist Null

`=` vs `==` egal, solange Leerzeichen verwendet werden

Wenn Du nicht sofort tust, was ich sage, dann ...

`if`

Wenn, prüfe ob das wahr ist, dann...

Beispiel Verzweigung

```
1 if test "$Tasse" = "Kaffee"; then echo "lecker"; fi
2 if [[ $Tasse == "Tee" ]] ; then echo "lecker" ; else cowsay
  ↪  "Ostfriese";fi
3 if [[ -z $Tasse ]] ; then echo "Tasse leer"; elif test $Tasse =
  ↪  "Kaffee" ; then cowsay "lecker $Tasse " ; else cowsay
  ↪  "So'n Tee";fi
```

`else` muss nicht unbedingt verwendet werden

`elif` Bedingung mehrfach Verkettungen

`-z` die Länge der ZEICHENKETTE ist Null

`=` vs `==` egal, solange Leerzeichen verwendet werden

Wenn Du nicht sofort tust, was ich sage, dann ...

`if`

Wenn, prüfe ob das wahr ist, dann...

Beispiel Verzweigung

```
1 if test "$Tasse" = "Kaffee"; then echo "lecker"; fi
2 if [[ $Tasse == "Tee" ]] ; then echo "lecker" ; else cowsay
  ↪ "Ostfriese";fi
3 if [[ -z $Tasse ]] ; then echo "Tasse leer"; elif test $Tasse =
  ↪ "Kaffee" ; then cowsay "lecker $Tasse " ; else cowsay
  ↪ "So'n Tee";fi
```

`else` muss nicht unbedingt verwendet werden

`elif` Bedingung mehrfach Verkettungen

`-z` die Länge der ZEICHENKETTE ist Null

`=` vs `==` egal, solange Leerzeichen verwendet werden

Wenn Du nicht sofort tust, was ich sage, dann ...

`if`

Wenn, prüfe ob das wahr ist, dann...

Beispiel Verzweigung

```
1 if test "$Tasse" = "Kaffee"; then echo "lecker"; fi
2 if [[ $Tasse == "Tee" ]] ; then echo "lecker" ; else cowsay
  ↪  "Ostfriese";fi
3 if [[ -z $Tasse ]] ; then echo "Tasse leer"; elif test $Tasse =
  ↪  "Kaffee" ; then cowsay "lecker $Tasse " ; else cowsay
  ↪  "So'n Tee";fi
```

`else` muss nicht unbedingt verwendet werden

`elif` Bedingung mehrfach Verkettungen

`-z` die Länge der ZEICHENKETTE ist Null

`=` vs `==` egal, solange Leerzeichen verwendet werden

Aufgabe 5 min

- erstelle eine Funktion Ostern und eine Funktion Weihnachten, die jeweils ihre Funktionsnamen ausgeben
- erweitere die Funktion, in der du Sachen getestet hast, um ein `if`
- wenn dein User „Ostern“ eingegeben hat, rufe die Funktion Ostern auf
- wenn dein User „Weihnachten“ eingegeben hat, rufe die Funktion Weihnachten auf
- wenn dein User Blödsinn geschrieben hat, dann rufe die Funktion mit der Frage auf

Schleifen binden und durch Loopings fliegen

solange etwas stimmt, mach etwas!

Beispiel solange

```
1 Tasse="5"
2 while [[ $Tasse -gt 3 ]]
3 do
4   fortune
5   ((Tasse--))
6   echo "es sind $Tasse Murmeln in der Tasse"
7 done
```

Schleifen binden und durch Loopings fliegen

bis etwas stimmt, mach etwas!

Beispiel bis

```
1 Tasse="7"
2 until [[ $Tasse -lt 5 ]]
3 do
4   ((Tasse--))
5   apropos brain?
6   echo "es sind $Tasse Murmeln in der Tasse"
7 done
```

Inkrement erhöhe um 1 ((i++))

Dekrement verringere um 1 ((i--))

Schleifen binden und durch Loopings fliegen

bis etwas die Zahl n erreicht hat, mach etwas!
Wird häufig für Arrays benutzt

Beispiel for

```
1 for i in ${seq 4 1 7}
2 do
3   sl -$i
4 done
```

Aufgabe 5 min

- nutze die Funktion, in der du ermittelst, welche verbleibenden Tage berechnet werden sollen
- baue hier eine Schleife ein
- mit Hilfe der Schleife wird die Frage solange wiederholt, bis dein Nutzer eine sinnvolle Antwort gibt

Mathe und Bash

. . . geht, aber . . .

- `expr 2 \* 4`
- `let x=2*4 ; echo $x`
- `echo "2*4" | bc` bitte das Packet `bc` installieren
- `echo $((5+4))`

Aufgabe 15 min

- erstelle eine Funktion, in der du die verbleibenden Tage bis Weihnachten berechnest
- baue diese Funktion sinnvoll in das Skript ein
- gib dem User die Antwort, wie viele Tage es noch bis Weihnachten sind

Osterberechnung

Osterformel nach Harold Spencer Jones

```
1  a=$(expr $Jahr % 19)
2  b=$(expr $Jahr / 100)
3  c=$(expr $Jahr % 100)
4  d=$(expr $b / 4)
5  e=$(expr $b % 4)
6  f=$(expr $(expr $b + 8) / 25)
7  g=$(expr $(expr $b - $f + 1) / 3)
8  h=$(expr $(expr 19 \* $a + $b - $d - $g + 15) % 30 )
9  i=$(expr $c / 4)
10 k=$(expr $c % 4)
11 l=$(expr $(expr 32 + 2 \* $e + 2 \* $i - $h - $k) % 7 )
12 m=$(expr $(expr $a + 11 \* $h + 22 \* $l) / 451)
13 Ostermonat=$(expr $(expr $h + $l - 7 \* $m + 114) / 31)
14 Ostersonntag=$(expr $(expr $h + $l - 7 \* $m + 114 ) % 31)
15 Ostersonntag=$(expr $Ostersonntag + 1)
16 echo "Ostersonntag ist am $Ostersonntag'.'$Ostermonat"
```

Aufgabe 15 min

- erstelle eine Funktion, in der die verbleibenden Tage bis Ostern berechnet werden
- baue diese Funktion sinnvoll in das Skript ein
- gib dem User die Antwort, wie viele Tage es noch bis Ostern sind

Tipps und Tricks

script	zeichnet auf, was in der Shell passiert (inklusive der Typos)
scriptreplay	spielt wieder ab, was du aufgezeichnet hast, mit Timing-Datei sogar in Echtzeit
Passwörter und Passphrases	haben in Skripten nichts verloren
root-Rechte in Skripten erlangen	lass es sein

Bitte vergiss nicht, Fehler einzubauen, das hat den größten Lerneffekt!

Tipps und Tricks

script	zeichnet auf, was in der Shell passiert (inklusive der Typos)
scriptreplay	spielt wieder ab, was du aufgezeichnet hast, mit Timing-Datei sogar in Echtzeit
Passwörter und Passphrases	haben in Skripten nichts verloren
root-Rechte in Skripten erlangen	lass es sein

Bitte vergiss nicht, Fehler einzubauen, das hat den größten Lerneffekt!

Vielen Dank für Ihre Aufmerksamkeit!

Bei weiteren Fragen wenden Sie sich bitte an info@b1-systems.de oder +49 (0)8457 - 931096

Anhang

Auf der Leitung stehen

- die Ausgabe eines Befehls an einen anderen weitergeben
- | wird durch folgende Tasten erstellt:

+

Pipe-Beispiel

```
1 fortune | cowsay | lolcat
```

das geht beliebig oft

Im Fall von, tue . . .

Wenn `if` für die Verschachtelungen unübersichtlich wird:

case-select Beispiel

```
1 case $Tasse in
2   Kaffee)
3     cowsay -f tux "Lecker Kaffee"
4   ;;
5   Tee)
6     cowsay -f turtle "Lecker Tee"
7   ;;
8   *)
9     cowsay -f ghostbusters "uaaahhh Gespenster"
10  ;;
11 esac
```

lässt sich nur im Skript benutzen, nicht in der Kommandozeile

Im Fall von, tue . . .

Wenn `if` für die Verschachtelungen unübersichtlich wird:

case-select Beispiel

```
1 case $Tasse in
2   Kaffee)
3     cowsay -f tux "Lecker Kaffee"
4   ;;
5   Tee)
6     cowsay -f turtle "Lecker Tee"
7   ;;
8   *)
9     cowsay -f ghostbusters "uaaahhh Gespenster"
10  ;;
11 esac
```

lässt sich nur im Skript benutzen, nicht in der Kommandozeile

Boxen

Was sind Arrays?
eine Box für mehrere Werte

Beispiel Arrays

```
1 Box=(1 pinguin 3 4)
2 echo $Box
3 #mit index
4 echo ${Box[2]}
5 #gesamtes Array
6 echo ${Box[*]}
7 echo ${Box[@]}
```

hint: * und @ unterscheiden sich.
@ hat Newline zwischen den Index-Werten (wird beim Looping fliegen relevant).