# Betriebssysteme

Übungen 04

Prof. Dr. Rainer Werthebach

Studiengang Informatik

Hochschule Aalen - Technik und Wirtschaft

# case-Anweisung

- allgemein:

```
case $variable in
   muster 1 ) kommandoliste ;;
   muster 2 ) kommandoliste ;;
   …
   muster n ) kommandoliste n ;;
esac
```

- Beispiele für Muster:

| | |
|---|---|
| 1) | → der Wert 1 |
| [abc]) | → a, b oder c |
| *.doc) | → Endung .doc |
| *) | → Default |
| *) | → der Wert * |

# case-Anweisung – ein Beispiel

```
#!/bin/bash
echo -n "Wollen Sie weitermachen? (J/N)? "
read x
case $x in
    [Jj]) echo "Ja" ;;
    [Nn]) echo "Nein" ;;
    *) echo "keine gültige Auswahl" ;;
esac
```

# test-Kommando

- **`test` *`Ausdruck`*** → Testet den Wahrheitswert des Ausdrucks

  wird **meistens im Zusammenhang mit** der **if-Anweisung** genutzt.
  Wahr, falls Rückgabewert 0 ist, sonst falsch.

  alternative Schreibweise: **`[` *`Ausdruck`* `]`**

- Wichtige Beispiele:

| | | |
|---|---|---|
| [ -d name ] | → | Verzeichnis existiert |
| [ -f name ] | → | Datei existiert |
| [ -s name ] | → | Datei existiert **und** ist nicht leer |
| [ -r name ] | → | Datei existiert **und** ist lesbar |
| [ -w name ] | → | Datei existiert **und** ist beschreibbar |
| [ -x name ] | → | Datei existiert **und** ist ausführbar |
| [ string ] | → | String ist nicht der Nullstring |
| [ name1 = name2 ] | → | Strings sind identisch |
| [ zahl1 -eq zahl2 ] | → | Zahlen sind gleich |

# test-Komando

- Verknüpfungen:

  -a → and

  -o → or

  ! → not

  Die Menge der Operatoren { and, or, not } ist eine Verknüpfungsbasis

- Beispiele:

```
[ -d name1 -a -f name2 ]
[ ! -r name ]
```

# if-Anweisung

- Syntax:

```
if Bedingung    # Rückgabewert eines Kommandos
then
    Kommandoliste
elif Bedingung
then
    Kommandoliste
else
    Kommandoliste
fi
```

# if-Anweisung - ein Beispiel

```
#!/bin/bash
for file # entspricht: for file in $@
do
    if [ -d $file ]
    then
        echo "$file ist ein Verzeichnis!"
    elif [ ! -s $file ]
    then
        echo "$file ist leer oder existiert nicht!"
    else
        echo "$file wird ausgegeben:"
        cat $file
    fi
done
```

# while-Schleife

- Syntax:

```
while Bedingung
do
    Kommandoliste
done
```

- Bedeutung:

  Solange Bedingung wahr ist, wird die Kommandoliste ausgeführt.

  Noch einmal: Bedingung ist der Rückgabewert eines Kommandos.

# until-Schleife

- Syntax:

```
until Bedingung
do
    Kommandoliste
done
```

- Bedeutung:

  Bis die Bedingung wahr ist, wird die Kommandoliste ausgeführt.

# while-Schleife – ein Beispiel

```
#!/bin/bash
while [ $1 ]
do
    Anzahl=$(cat $1 | wc -w)
    echo "Die Anzahl Woerter in $1 ist $Anzahl"
    shift 1
done
```

# Funktionen

- Syntax:

```
function Funktionsname {
    Kommandos
}

Funktionsname () {
    Kommandos
}
```

- Bash-Funktionen liegen im Hauptspeicher (schneller Zugriff).
- Bash-Prozeduren (externe Skripte) starten langsamer, da sie immer neu geladen werden.
- Parameter einer Funktion werden wie positionale Skript-Parameter behandelt.

# Funktionen (Fortsetzung)

- Funktionsaufruf:

```
Funktionsname [Parameter1 Parameter2 …]
```

Funktion muss vor dem Aufruf deklariert werden.

- Zugriff auf Übergabeparameter:

```
Jetzt gilt $1=Parameter1, $2=Parameter2 usw.
$*, $@ und $# beziehen sich jetzt auf
neue $1, $2 usw. Werte.
Ausnahme: $0 bleibt Skriptname und
nicht Funktionsname
```

# Funktionen (Fortsetzung)

Funktionsrückgabe:

- Mittels `return n` (n = Integer)
  Zugriff im Hauptprogramm über $?

```
funktionsaufruf
ergebnis = $?
```

- Mittels `echo`

```
halbieren() {
   val=`expr $1 / 2`
   echo $val
}
```

  Zugriff im Hauptprogramm über `...` bzw. $(...)

```
ergebnis = `halbieren 20`
```

# Funktionen (Fortsetzung)

Funktionsrückgabe:

- Mittels lokaler Variablen

```
halbieren() {
   value_1=`expr $1 / 2`
}
```

Zugriff im Hauptprogramm über lokale Variable

```
ergebnis = $value_1
```

# Steuerkommandos

- `exit [Status]` → beendet Shell mit `Status` oder falls `Status` nicht angegeben mit exit-Status des letzten Kommandos
- `break` → beendet aktuelle Schleife und setzt nach `done` fort
- `continue` → bricht den aktuellen Iterationsschritt ab und macht mit dem nächsten weiter
- `true` → immer wahr
- `false` → immer falsch

# Unbedingte / bedingte Ausführung

- unbedingte Ausführung:

  ```
  Kommando; Kommando; Kommando; …
  ```

  Beispiel: `cat datei1.txt; wc -w datei2.txt`

- Kurzschreibweise für bedingte Ausführungen:
  - Variante 1: Kommando_1 **&&** Kommando_2
    entspricht: **if** Kommando_1; **then** Kommando_2; **fi**

    Beispiele:
    ```
    [ -f datei ] && cat datei
    chmod u+x skript.sh && ./skript.sh
    ```

  - Variante 2: Kommando_1 **||** Kommando_2
    entspricht: **if !** Kommando_1; **then** Kommando_2; **fi**

    Beispiel: `cat datei1.txt || cat datei2.txt`

# bedingte Ausführung – ein Beispiel

```
#!/bin/bash
nummer=$#
if [ $nummer -eq 0 ]
then
    while true
    do
        read in || break
        echo $in >> temp$$
    done
    nummer=$(cat temp$$ | wc -w)
    rm temp$$
fi
echo "Die Anzahl Wörter ist $nummer."
```