Linux und Shell-Programmierung – Teil 6

Prof. Dr. Christian Baun

Fachhochschule Frankfurt am Main
Fachbereich Informatik und Ingenieurwissenschaften
christianbaun@fb2.fh-frankfurt.de

Heute

- Shell-Skripting (Teil 2)
  - Arithmetik auswerten (`expr`)
  - Zufallszahlen berechnen (`RANDOM`)
  - Funktionen
  - Lokale und globale Variablen (`local`, `typeset`)
  - Funktionsbibliotheken
  - Auswahlmenüs (`select`)

Arithmetik auswerten mit `expr`

- Arithmetische Ausdrücke können in Shell-Skripten mit dem Kommando `expr` ausgewertet werden
- Als Zahlen können 32-Bit-Integerzahlen verwendet werden. Auf manchen Systemen sind auch 64-Bit-Integerzahlen zulässig
- `expr` beherrscht die vier Grundrechenarten

`+`	Addition
`-`	Subtraktion
`*`	Multiplikation
`/`	Division
`%`	Divisionsrest (Modulo-Operator)

- Auch bei `expr` gilt die Priorität **Punkt vor Strich**

expr beherrschen

- Klammern können gesetzt werden
- Da die Klammern und der Stern (Multiplikation) auch von der Shell verwendet werden, müssen diese Operationszeichen immer durch einen Backslash geschützt werden: `\*`, `\(`, `\)`
- Damit die Shell die Operatoren erkennt, müssen diese von Leerzeichen umgeben sein

```
#!/bin/bash
X=3
XPLUS1=‘expr $X + 1‘
XMAL2=‘expr $X \* 2‘
echo $XPLUS1
echo $XMAL2
```

```
$ ./expr
4
6
```

Zufallszahlen erzeugen

- Die Shell (Bash und Korn-Shell) verfügt über einen Zufallszahlengenerator
- Bei jedem Zugriff auf die Umgebungsvariable `RANDOM` wird eine neue, zufällige Zahl im Zahlenraum von 0 bis 32767 zurückgegeben

```
$ echo $RANDOM
12958
$ echo $RANDOM
32418
$ echo $RANDOM
24436
```

- Zufallszahlen im Bereich zwischen 0 und 10 ausgeben:

```
$ echo `expr $RANDOM % 10`
7
```

Funktionen in Shell-Skripten

- Mit Funktionen können Kommandos zu Blöcken zusammengefasst und unter einem gemeinsamen Funktionsnamen aufgerufen werden
- Funktionen können auch bei Shell-Skripten außerhalb des Hauptprogramms definiert werden. Somit kann im Hauptprogramm eine gute Lesbarkeit erhalten werden
- Syntax für die Definition von Funktionen:
  `Funktionsname() { Kommando1; Kommando2; ... ; }`
- Aufruf einer Funktion:
  `Funktionsname`
- Funktionen können beim Aufruf Argumente übergeben werden:
  `Funktionsname Argument1 Argument2 ...`

Einfaches Beispiel für eine Funktion

```
#!/bin/bash
# Multiplikation als Funktion

multiplikation() {
  ergebnis=‘expr $1 \* $2‘
  echo "Ergebnis: $ergebnis"
}

multiplikation $1 $2
multiplikation 15 13
multiplikation 20 $1
multiplikation $2 10
```

```
$ ./mult 5 3
Ergebnis: 15
Ergebnis: 195
Ergebnis: 100
Ergebnis: 30
```

Beispiel für Datenrückgabe in Funktionen

```
#!/bin/bash
# Datenrückgabe bei Funktionen

multiplikation() {
  ergebnis=‘expr $1 \* $2‘
  echo $ergebnis
}

rechnung1=‘multiplikation 15 13‘
rechnung2=‘multiplikation 20 10‘
rechnung3=‘multiplikation 10 25‘

echo "1.Rechnung: $rechnung1"
echo "2.Rechnung: $rechnung2"
echo "3.Rechnung: $rechnung3"
```

```
$ ./mult
1.Rechnung: 195
2.Rechnung: 200
3.Rechnung: 250
```

Lokale und globale Variablen

- Variablen im Hauptprogramm eines Shell-Skripts mit Funktionen sind immer globale Variablen
- Variablen des Hauptprogramms sind auch in Funktionen zugänglich
- Bei kleinen Skripten ist dieser Umstand bequem
  - Bei größeren Projekten kann es aber zu Schwierigkeiten kommen, weil der Überblick verloren geht und es zur doppelten Verwendung von Variablennamen kommen kann
- In der Bash-Shell existieren die Kommandos `local` und `typeset`, mit der lokale Variablen innerhalb von Funktionen deklariert werden können

```
local Variable1=Wert1 Variable2=Wert2 ...
typeset Variable1=Wert1 Variable2=Wert2 ...
```

- Lokale Variablen, die in einer Funktion definiert werden verfallen, wenn die Funktion endet

Funktionsbibliotheken

- Soll aus verschiedenen Skripten auf häufig verwendete Funktionen zurückgegriffen werden, macht es Sinn, diese Funktionen in eigenen Dateien, sogenannten Funktionsbibliotheken, zu sammeln
- Beispiel für eine Funktionsbibliothek:

```
# Funktionsbibliothek funktionen.bib
# (enthält nur Funktionen)

addition() {
  adderg=‘expr $1 + $2‘
  echo $adderg
}
multiplikation() {
  multerg=‘expr $1 \* $2‘
  echo $multerg
}
```

Funktionsbibliotheken nutzen

- Damit die Funktionen aus einer Funktionsbibliothek in einem Shell-Skript nutzbar sind, muss die Funktionsbibliothek eingelesen werden

```
#!/bin/bash

. ./funktionen.bib

echo "Multiplikation: ‘multiplikation 10 20‘"
echo "Addition: ‘addition 5 10‘"
```

- Das Ergebnis:

```
$ ./grundrechenarten
Multiplikation: 200
Addition: 15
```

Auswahlmenüs erzeugen mit `select`

```
select Variable in Auswahlliste
do
   ...
done
```

- Mit `select` ist es auf einfach Art und Weise möglich, Auswahlmenüs zu erzeugen
- Um mit `select` zu arbeiten sind zwei Dinge notwendig:
  - Eine Liste von Auswahlmöglichkeiten
  - Eine Variable, in der die ausgewählte Option gespeichert wird
- `select` funktioniert wie eine Endlosschleife
  - Nach einer Eingabe erfolgt ein neuer Schleifendurchlauf
- Soll `select` beendet werden, ist es notwendig mit `Strg-D` die Schleife zu beenden oder ein `break` einzubauen

Beispiel zu select

```
#!/bin/bash

select eingabe in x y z
do
  echo "Ihre Auswahl war: $eingabe"
done
```

```
$ ./select
1) x
2) y
3) z
#? 1
Ihre Auswahl war: x
#? 3
Ihre Auswahl war: z
#?
```

Aufforderungs-Prompt bei `select` ändern

- Um den Aufforderungs-Prompt (Standardwert: #?) zu ändern, muss die Variable PS3 einen anderen Wert erhalten

```
#!/bin/bash

PS3="Bitte wählen Sie aus: "

select eingabe in x y z
do
  echo "Ihre Auswahl war: $eingabe"
done
```

```
$ ./selectprompt
1) x
2) y
3) z
Bitte wählen Sie aus: 2
Ihre Auswahl war: y
Bitte wählen Sie aus:
```

Beispiel zu select und break

```
#!/bin/bash
PS3="Bitte wählen Sie aus: "

select eingabe in lesen schreiben beenden
do
  echo "Ihre Auswahl war: $eingabe"
  if [ "$eingabe" = "lesen" ]
  then
    ...
  fi
  if [ "$eingabe" = "schreiben" ]
  then
    ...
  fi
  if [ "$eingabe" = "beenden" ]
  then
    break
  fi
done
```

Beispiel für Dateiauswahlen mit `select`

- `select` ist sehr flexibel was die Generierung der Auswahlliste angeht

```
#!/bin/bash

PS3="Bitte wählen Sie eine Datei aus: "

select file in beenden *.txt
do
  if [ "$file" = "beenden" ]
  then
    break
  fi
  echo "Datei: $file"
done
```

Ergebnis des Beispiels mit `select`

```
$ ./selectdatei
1) beenden          3) testdatei.txt   5) umlaute.txt
2) sedtest.txt      4) test.txt        6) ausgabe.txt
7) log.txt
Bitte wählen Sie eine Datei aus: 3
Datei: testdatei.txt
Bitte wählen Sie eine Datei aus: 6
Datei: ausgabe.txt
Bitte wählen Sie eine Datei aus: 1
$
```

Möglichkeit eines Datei-Browsers mit `select`

- Es ist nicht schwierig mit `select` einen Datei-Browser zu schreiben
- Dafür muss mit `test` getestet werden, ob eine ausgewählten Datei eine normale Datei oder ein Verzeichnis ist
- Wenn es ein Verzeichnis ist, muss in dieses gewechselt und die Dateiliste ausgegeben werden