Programmiermethoden und -werkzeuge 1

Woche 8 - Advanced Bash Scripting

Jochen Hosenfeld

jochen.hosenfeld@informatik.hs-fulda.de

Fachbereich Angewandte Informatik

December 12, 2025

Inhalte des Moduls

Grundlegendes Arbeiten mit der Kommandozeile

Umgang mit Text-Editoren im Terminal

Umgebungsvariablen und Shell-Skripte

Laden von Software-Projekten aus Open-Source-Quellen

Build-Systeme und Paketmanager

Installation und De-Installation von selbst übersetztem Code

Inhalte des Moduls

Grundlegendes Arbeiten mit der Kommandozeile

Umgang mit Text-Editoren im Terminal

Umgebungsvariablen und Shell-Skripte

Laden von Software-Projekten aus Open-Source-Quellen

Build-Systeme und Paketmanager

Installation und De-Installation von selbst übersetztem Code

Indizierte Arrays

Einführung in Arrays

- Was sind Arrays?
  - → Datenstruktur zur Speicherung mehrerer Werte in einer einzigen Variablen
  - → Verwendet in der Shell-Programmierung zur Bearbeitung von Listen von Elementen
- Deklaration eines Arrays:
  - → Einfache Syntax: `array_name=(Element1 Element2 Element3)`

```
1  colors=("red" "green" "blue")
```

Zugriff auf Array-Elemente

- Zugriff auf einzelne Elemente:
  - → Syntax: `${array_name[Index]}`
  - → Indizes beginnen bei 0
- Zugriff auf alle Elemente:
  - → Syntax: `${array_name[@]}` oder `${array_name[*]}`

```
1 echo "Erster Farbton: ${colors[0]}"
2 Erster Farbton: red
3
4 echo "Alle Farbtöne: ${colors[@]}"
5 Alle Farbtöne: red green blue
```

Exkurs: Geschweifte Klammern {}

- Vielseitige Verwendung in Bash in verschiedenen Kontexten
- Geschweifte Klammern helfen bei der Strukturierung von Variablen, Arrays und Befehlen
- Erhöhen Lesbarkeit und Effizienz von Bash-Skripten

Variablennamen

```
1 name="Sarah"
2 document="Stundenplan"
3 echo "Hallo $name, Sie finden Ihr Dokument unter $document_2025.pdf"
```

Variablennamen

```
1 name="Sarah"
2 document="Stundenplan"
3 echo "Hallo $name, Sie finden Ihr Dokument unter $document_2025.pdf"
4 Hallo Sarah, Sie finden Ihr Dokument unter .pdf
```

Parametererweiterung

Parameter Expansion

- Verwendung, um die Grenze von Variablennamen zu kennzeichnen, besonders in Kombination mit anderen Zeichen
- Syntax: `${variablenname}text`

```
1 name="Sarah"
2 document="Stundenplan"
3 echo "Hallo $name, Sie finden Ihr Dokument unter $document_2025.pdf"
4 Hallo Sarah, Sie finden Ihr Dokument unter .pdf
5
6 echo "Hallo $name, Sie finden Ihr Dokument unter ${document}_2025.pdf"
7 Hallo Sarah, Sie finden Ihr Dokument unter Stundenplan_2025.pdf
```

Listenerweiterung

Brace Expansion

- Erzeugt Listen oder Sequenzen von Zeichenketten
- Syntax: `{start..end}` oder `{item1,item2,item3}`

```
1  echo {1..5}              # Sequenz, Ergebnis: 1 2 3 4 5
2  mv file.{txt,md}         # Optionen: Verschiebt 'file.txt' und 'file.md'
```

String-Manipulation

- Geschweifte Klammern erlauben verschiedene String-Operationen
- Beispiele sind das Kürzen von Präfixen/Suffixen und Ersetzen innerhalb von Strings

```
1 dateiname="bericht.txt"
2 echo ${dateiname%.txt}      # Entfernt '.txt', Ergebnis: bericht
```

Befehlsgruppierung

Command Grouping

- Gruppiert mehrere Befehle, die zusammen ausgeführt werden sollen
- Syntax: `{ befehl1; befehl2; }`

```
1 { echo "Beginn der Sequenz"; echo "Ende der Sequenz"; }
2 # Mehrere Befehle in einer Zeile werden mit Semikolon getrennt
3
4 Beginn der Sequenz
5 Ende der Sequenz
```

Ende des Exkurses: Zugriff auf Array-Elemente

```
1 colors=("red" "green" "blue")
2
3 for color in "${colors[@]}"; do
4     echo "Farbe: $color"
5 done
```

Ende des Exkurses: Zugriff auf Array-Elemente

```
1 colors=("red" "green" "blue")
2
3 for color in "${colors[@]}"; do
4     echo "Farbe: $color"
5 done
6
7 Farbe: red
8 Farbe: green
9 Farbe: blue
```

Ende des Exkurses: Zugriff auf Array-Elemente

```
1  colors=("red" "green" "blue")
2
3  for color in "${colors[@]}"; do
4      echo "Farbe: $color"
5  done
6
7  Farbe: red
8  Farbe: green
9  Farbe: blue
10
11 for color in "${colors[*]}"; do
12     echo "Farbe: $color"
13 done
```

Ende des Exkurses: Zugriff auf Array-Elemente

```
1  colors=("red" "green" "blue")
2
3  for color in "${colors[@]}"; do
4      echo "Farbe: $color"
5  done
6
7  Farbe: red
8  Farbe: green
9  Farbe: blue
10
11 for color in "${colors[*]}"; do
12     echo "Farbe: $color"
13 done
14
15 Farbe: red green blue
```

Bearbeiten von Arrays

- An letztes Element anhängen: `array_name+=(NeuesElement)`

```
colors=("red" "green" "blue")
colors+=("yellow")
echo "Aktualisierte Farben: ${colors[@]}"
Aktualisierte Farben: red green blue yellow
```

- Setzen eines Elements: `array_name[Index]=NeuerWert`

```
colors[1]="purple"
echo "Aktualisierte Farben: ${colors[@]}"
Aktualisierte Farben: red purple blue yellow
```

Entfernen von Array-Elementen

- Einzelnes Element löschen: `unset array_name[Index]`

```
1 colors=("red" "green" "blue" "yellow")
2 unset colors[1]
3 echo "Farben nach Löschen: ${colors[@]}"
4 Farben nach Löschen: red blue yellow
```

Entfernen von Array-Elementen

- Einzelnes Element löschen: `unset array_name[Index]`

```
colors=("red" "green" "blue" "yellow")
unset colors[1]
echo "Farben nach Löschen: ${colors[@]}"
Farben nach Löschen: red blue yellow

echo "Farbe auf Position 1: ${colors[1]}"
Farbe auf Position 1:
# Anderen Elemente behalten ihre Position
```

- Gesamtes Array löschen: `unset array_name`

Nützliche Array-Operationen

- Länge eines Arrays bestimmen:

```
1  echo "Anzahl der Farben: ${#colors[@]}"
```

- Schleife über ein Array:

```
1  for color in "${colors[@]}"; do
2    echo "Farbe: $color"
3  done
```

Assoziative Arrays

Assoziative Arrays

- Assoziativ: “Verbinden, vereinigen”
- Erlauben Zeichenketten als Indizes
- Ermöglichen die Speicherung von Key-Value-Pairs
- Benötigen die `declare -A`-Anweisung zur Deklaration

```
1  declare -A color_codes
```

Initialisierung von assoziativen Arrays

- Mit einem einzelnen Key-Value-Pair initialisieren:

```
1  color_codes["rot"]="#FF0000"
```

- Mit mehreren Key-Value-Pairs initialisieren:

```
1  color_codes=(["rot"]="#FF0000" ["blau"]="#0000FF")
```

Zugriff auf Elemente in assoziativen Arrays

- Zugriff durch Angabe des Keys:

```
1  echo "Die Farbe Rot ist: ${color_codes["rot"]}"
2  Die Farbe Rot ist: #FF0000
```

Zugriff auf Elemente in assoziativen Arrays

- Zugriff durch Angabe des Keys:

```
1 echo "Die Farbe Rot ist: ${color_codes["rot"]}"
2 Die Farbe Rot ist: #FF0000
3
4 echo "Die Farbe auf Position 1 ist: ${color_codes[1]}"
5 Die Farbe auf Position 1 ist:
```

Durchlaufen der Elemente in assoziativen Arrays

```
for key in "${!color_codes[@]}"; do
  echo "Farbe $key: ${color_codes[$key]}"
done

Farbe rot: #FF0000
Farbe blau: #0000FF
```

Durchlaufen der Elemente in assoziativen Arrays

```
for key in "${!color_codes[@]}"; do
  echo "Farbe $key: ${color_codes[$key]}"
done

Farbe rot: #FF0000
Farbe blau: #0000FF


for key in "${color_codes[@]}"; do
  echo "Farbe $key: ${color_codes[$key]}";
done

Farbe #FF0000:
Farbe #0000FF:
```

Prüfen auf Existenz eines Keys

```
1 if [[ -v color_codes["grün"] ]]; then
2   echo "Grün existiert im Array"
3 else
4   echo "Grün existiert nicht"
5 fi
```

Löschen von Keys und Arrays

- Einzelnes Element löschen:

```
1  unset color_codes["rot"]
```

- Gesamtes Array löschen:

```
1  unset color_codes
```

Nützliche Operationen mit assoziativen Arrays

- Anzahl der Elemente in einem assoziativen Array bestimmen:

```
1  echo "Anzahl der Farben: ${#color_codes[@]}"
```

Einschränkungen bei Arrays

- Unterstützung für mehrdimensionale Arrays:
  → Bash unterstützt keine mehrdimensionalen Arrays direkt
  → Workarounds mit assoziativen Arrays oder anderen Techniken erforderlich

- Speichergrenzen:
  → Große Arrays können viel Speicher verwenden und Bash hat möglicherweise Einschränkungen durch Systemlimits

Typisierung in Bash

- Dynamische Typisierung:

  → Variablen müssen vor der Verwendung nicht deklariert werden.

```
1  var="Hallo, Welt"
2  echo $var  # Ausgabe: Hallo, Welt
3
4  var=42
5  echo $var  # Ausgabe: 42
```

Variable Typen in Bash

- Häufig verwendete Typen:
  - → Strings: Zeichenfolgen von Text
  - → Ganzzahlen: Numerische Werte für Berechnungen
- Keine strikte Typengrenze:
  - → Eine Variable kann beliebig zwischen String und Ganzzahl wechseln.

Besonderheiten der Typisierung

- Zahlenerkennung:

  → Bash kann numerische Operationen durchführen, wenn der Wert numerisch ist.

```
1  zahl1=10
2  zahl2=5
3  summe=$((zahl1 + zahl2))
4  echo "Summe: $summe"
```

Besonderheiten der Typisierung

- Verwendung von `declare`:

  → Optionale Typisierung und Einschränkungen

```
1  declare -i nummer  # Deklariert "nummer" als Ganzzahl
2  nummer=5
3  nummer="Text"  # Führt zu einem Fehler, da "Text" keine Zahl ist
4
5  echo $nummer
6  0
7
8  declare -a      # Deklariert ein indiziertes Array
9  declare -A      # ... ein assoziatives Array (ab Bash 4)
10 declare -r      # ... eine schreibgeschützte (readonly) Variable
```

Umgang mit Arrays in der Typisierung

- Einfache Arrays:

  → Werte sind untypisiert und können Strings oder Ganzzahlen sein

```
1  mixed=("Text" 3 "Mehr Text" 7)
2  for i in "${mixed[@]}"; do
3    echo "Element: $i"
4  done
```

Umgang mit Arrays in der Typisierung

- Einfache Arrays:

  → Werte sind untypisiert und können Strings oder Ganzzahlen sein

```
1  mixed=("Text" 3 "Mehr Text" 7)
2  for i in "${mixed[@]}"; do
3    echo "Element: $i"
4  done
5
6  Element: Text
7  Element: 3
8  Element: Mehr Text
9  Element: 7
```

Typkonvertierung

- Konvertierung von Strings in Zahlen

  → Bei arithmetischen Operationen in `$(( ))` wird eine Konvertierung automatisch versucht.

- Zahlen zu Strings

  → Werden standardmäßig als Strings betrachtet

```
1  string_number="8"
2  echo $((string_number + 2))  # Automatische Konvertierung: Ergebnis 10
3
4  natural_number=10
5  echo "Nummer ist $natural_number"  # Wird als String betrachtet
```

String-Manipulation: Substitution

- Ersetzen von Text:
  - → Syntax: `${variable/expr/replacement}`

```
1  text="Hallo Welt"
2  echo ${text/Welt/Freunde}
3
4  Hallo Freunde
```

String-Manipulation: Teilstrings

- Extrahieren von Teilstrings:

  → `${variable:offset:length}`

```
1  text="Hallo Welt"
2  echo ${text:0:5}
3
4  Hallo
```

Weitere String-Manipulationen

- Länge eines Strings:
  → `${#variable}`
- Entfernen von Präfixen/Suffixen:
  → Präfix: `${variable#pattern}`
  → Suffix: `${variable%pattern}`

```
1  string="Programmcode"
2  echo ${#string}              # Länge: 12
3  echo ${string#Programm}      # Ausgabe: code
4  echo ${string%code}          # Ausgabe: Programm
```

`break`-Anweisung

- Verwendungszweck:
  - → Beendet Schleifen vorzeitig basierend auf einer Bedingung
  - → Erhöht Effizienz durch Vermeidung unnötiger Iterationen
  - → Kann in `for`, `while` und `until`-Schleifen verwendet werden

```
1 for zahl in {1..5}; do
2   if [[ $zahl -eq 3 ]]; then
3     break
4   fi
5   echo "Zahl: $zahl"
6 done
7
8 Zahl: 1
9 Zahl: 2
```

`break` in verschachtelten Schleifen

- `break` verlässt nur die innerste Schleife.
- Mit `break N` können mehrere Schleifenebenen verlassen werden.

```
1 for i in {1..3}; do
2   echo "Äußere Schleife: $i"
3   for j in {1..3}; do
4     echo "  Innere Schleife: $j"
5     if [[ $j -eq 2 ]]; then
6       break 2
7     fi
8   done
9 done
```

`continue` in Schleifen

- Verwendungszweck:

  → Bestimmte Durchläufe der Schleife zu überspringen und trotzdem weiterzulaufen

```
1 for zahl in {1..5}; do
2   if [[ $zahl -eq 3 ]]; then
3     continue
4   fi
5   echo "Zahl: $zahl"
6 done
```

ShellCheck

- Tool zur statischen Code-Analyse von Shell-Skripten
- Prüfung auf:
  - → Syntaxfehler
  - → Best Practices
  - → Potenzielle Sicherheitsprobleme

```
sudo apt install shellcheck     # Installation (falls nicht vorhanden)
shellcheck script.sh            # Check eines Skripts

# Mögliche Fehlerausgabe
In script.sh line 14:
        if (( $RANDOM % 2 )); then
              ^-----^ SC2004 (style): $/${} is unnecessary on arithmetic variables.
```

ShellCheck

- Ausgabe enthält hilfreiche Erklärungen und Links
- ShellCheck-Webseite für Online-Überprüfungen: www.shellcheck.net
- Integration in IDEs: Unterstützung für VSCode, Atom und mehr

Zusammenfassung

Indizierte und assoziative Arrays

Brace Expansion mit {}

String-Manipulation

Typisierung

continue und break

ShellCheck