Shellskripting – Syntax und Grundlagen

Einführung

Ein Shellskript ist eine Textdatei mit einer Abfolge von Befehlen, die von der Shell (z. B. Bash) interpretiert und ausgeführt werden. Es dient zur Automatisierung wiederkehrender Aufgaben.

---

1. Shebang-Zeile

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

- Diese Zeile steht ganz oben im Skript.
- Sie gibt an, welche Shell zur Ausführung verwendet werden soll.

---

2. Kommentare

```
# Dies ist ein Kommentar
```

- Kommentare beginnen mit `#` (außer in der Shebang-Zeile).
- Sie werden beim Ausführen ignoriert.

---

3. Skript ausführbar machen

```
chmod +x mein_skript.sh
./mein_skript.sh
```

- Mit `chmod +x` wird das Skript ausführbar.
- Durch `./` wird das Skript im aktuellen Verzeichnis ausgeführt.

---

4. Variablen

```
name="Max"
echo "Hallo $name"
```

- Keine Leerzeichen um das `=`.
- Zugriff mit `$variablenname`.
- In `"`-Strings werden Variablen ersetzt.

5. Einfache Benutzereingaben

```
echo "username:"
read user
echo "hello $user"
```

- `read` liest eine Eingabe von der Tastatur.
- Interaktive Skripte sind dadurch möglich.

---

6. Bedingungen (if-Anweisung)

```
if [ "$name" = "Max" ]; then
  echo "Willkommen, Max!"
else
  echo "Du bist nicht Max."
fi
```

- Achte auf Leerzeichen innerhalb der `[...]`.

---

7. Schleifen

for-Schleife:

```
for i in 1 2 3
do
  echo "Zahl: $i"
done
```

while-Schleife:

```
count=1
while [ $count -le 3 ]
do
  echo "Durchlauf $count"
  count=$((count + 1))
done
```

C-ähnliche For-Schleife

```
for ((i=0; i<5; i++)); do
    echo "Index $i"
done
```

---

8. Funktionen

```
func_foo() {
  echo "hello, $1!"
}

func_foo "Anna"
```

- `$1`, `$2`, ... sind Argumente.
- Funktionsdefinition mit `name() { ... }`

---

9. Stringoperationen

Vergleich:

```
a="abc"
b="abc"
if [ "$a" = "$b" ]; then
  echo "equal."
else
  echo "not equal"
fi
```

Leer-Check:

```
str="Hello"
if [ -z "$str" ]; then
  echo "String is empty."
fi
```

Stringlänge:

```
str="Hello BSY3"
echo "length: ${#str}"
```

Substring:

```
text="Beispieltext"
echo "${text:0:7}" # Ausgabe: Beispiel
```

Ersetzen:

```
text="I love Bash"
echo "${text/Bash/Linux}" # Ausgabe: I love Linux
```

---

10. Skripte einbinden

```
source hilfe.sh
# oder
. hilfe.sh
```

- Der Code wird im aktuellen Kontext ausgeführt.

---

11. Menüs mit `select`

```
echo "Waehle eine Option:"
select opt in Start Stop Beenden
do
  case $opt in
    Start) echo "Gestartet";;
    Stop) echo "Gestoppt";;
    Beenden) break;;
    *) echo "Ungueltige Auswahl";;
  esac
done
```

---

12. Dateiumleitungen

```
ls > ausgabe.txt      # stdout in Datei
ls >> ausgabe.txt     # stdout anhängen
ls nicht_da 2> fehler.txt   # stderr in Datei
ls nicht_da &> alles.txt    # stdout und stderr zusammen
```

---

13. Exit-Status prüfen

```
cp quelle.txt ziel.txt
if [ $? -eq 0 ]; then
  echo "Kopieren erfolgreich"
else
  echo "Fehler beim Kopieren"
fi
```

- `$?` gibt den Exit-Code des letzten Befehls zurück.
- `0` bedeutet erfolgreich, alles andere = Fehler.

---

14. Fallunterscheidung mit `case`

```
echo "Geben Sie eine Zahl ein:"
read zahl

case $zahl in
  1) echo "Eins";;
  2) echo "Zwei";;
  3) echo "Drei";;
  *) echo "Unbekannt";;
esac
```

---

15. Dateitests

```
datei="bericht.txt"

if [ -f "$datei" ]; then
  echo "regulaere Datei"
fi

if [ -d "$datei" ]; then
  echo "Es ist ein Verzeichnis"
fi

if [ -e "$datei" ]; then
  echo "Datei oder Verzeichnis existiert"
fi
```

- `-f` prüft auf reguläre Datei.
- `-d` prüft auf Verzeichnis.
- `-e` prüft auf Existenz allgemein.
