LU01b - Überblick

2. Die Grundlagen eines Bash-Skripts

Ein Bash-Skript beginnt normalerweise mit einer sogenannten **Shebang**-Zeile, die angibt, welche Shell zur Ausführung verwendet werden soll. Für Bash-Skripte sieht die Shebang wie folgt aus:

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

Ein einfaches Beispiel:

```bash #!/bin/bash # Mein erstes Bash-Skript echo „Hallo, Welt!“ ```

Speichern Sie dieses Skript in einer Datei, z. B. `hallo.sh`, und machen Sie die Datei ausführbar:

```bash chmod +x hallo.sh ./hallo.sh ```

Das Skript gibt dann den Text „Hallo, Welt!“ aus.

3. Variablen und ihre Verwendung

In Bash können Sie Variablen verwenden, um Daten zu speichern und diese im Skript wiederzuverwenden.

Beispiel:

```bash #!/bin/bash NAME=„Max“ echo „Hallo, $NAME!“ ```

Variablen werden ohne das `$`-Zeichen deklariert (`NAME="Max"`), aber mit `$` aufgerufen (`$NAME`).

Eingebaute Variablen:

Bash enthält viele vordefinierte Variablen. Einige wichtige Beispiele:

- `$0`: Der Name des Skripts. - `$1`, `$2`, …: Übergabeparameter an das Skript. - `$#`: Anzahl der Übergabeparameter. - `$?`: Rückgabewert des letzten Befehls.

Beispiel:

```bash #!/bin/bash echo „Das Skript heißt $0“ echo „Der erste Parameter ist $1“ echo „Anzahl der Parameter: $#“ ```

4. Kontrollstrukturen

Bedingte Anweisungen (if-else)

Mit `if`-Bedingungen lassen sich Verzweigungen definieren:

```bash #!/bin/bash NUMBER=5 if [ $NUMBER -gt 3 ]; then

```
  echo "$NUMBER ist größer als 3"
```

else

```
  echo "$NUMBER ist nicht größer als 3"
```

fi ```

Vergleichsoperatoren: - `-eq`: gleich (= - `-ne`: ungleich (! - `-lt`: kleiner als (<) - `-gt`: größer als (>) - `-le`: kleiner oder gleich (⇐) - `-ge`: größer oder gleich (>

Schleifen (for, while)

for-Schleife:

```bash #!/bin/bash for i in 1 2 3 4 5; do

```
  echo "Zahl: $i"
```

done ```

**while-Schleife:**

```bash #!/bin/bash COUNTER=1 while [ $COUNTER -le 5 ]; do

```
  echo "Durchlauf $COUNTER"
  ((COUNTER++))
```

done ```

5. Funktionen

Funktionen sind nützlich, um wiederverwendbare Codeblöcke zu definieren:

```bash #!/bin/bash # Funktion definieren begruesse() {

```
  echo "Hallo, $1!"
```

}

# Funktion aufrufen begruesse „Maria“ begruesse „Max“ ```

### 6. Nützliche Befehle in Bash-Skripten

- **`echo`**: Gibt Text aus. - **`read`**: Liest Benutzereingaben. - **`expr`**: Rechnet einfache arithmetische Ausdrücke. - **`test`** oder `[ ... ]`: Prüft Bedingungen (Dateien, Strings, Zahlen). - **`exit`**: Beendet das Skript und gibt einen Status zurück.

Beispiel für Benutzereingabe und Arithmetik:

```bash #!/bin/bash echo „Gib eine Zahl ein:“ read num1 echo „Gib eine weitere Zahl ein:“ read

num2 sum=$(expr $num1 + $num2) echo „Die Summe ist: $sum“ ```

### 7. Fehlerbehandlung

Mit `$?` kann der Rückgabewert des letzten Befehls überprüft werden. Ein Wert von `0` bedeutet Erfolg, jeder andere Wert zeigt einen Fehler an.

```bash #!/bin/bash mkdir /testverzeichnis if [ $? -ne 0 ]; then

```
  echo "Fehler beim Erstellen des Verzeichnisses"
```

else

```
  echo "Verzeichnis erfolgreich erstellt"
```

fi ```

8. Praktische Tipps

- Verwenden Sie **Kommentare** (`#`), um den Code lesbarer zu machen. - Testen Sie Skripte Schritt für Schritt, insbesondere bei komplexen Aufgaben. - Nutzen Sie **Debugging**-Modi mit `bash -x script.sh`, um zu sehen, wie Befehle ausgeführt werden.

Fazit

Die Shellprogrammierung mit Bash bietet eine Vielzahl von Möglichkeiten, wiederkehrende Aufgaben zu automatisieren und effiziente Workflows zu schaffen. Mit grundlegenden Kenntnissen in Variablen, Kontrollstrukturen und Funktionen können Sie bereits erste nützliche Skripte erstellen.
```