

**ZSL**

Zentrum für Schulqualität  
und Lehrerbildung  
Baden-Württemberg

# LINUX ESSENTIALS

## Skript-Grundlagen



**Tobias Heine**  
[tobias.heine@springer-schule.de](mailto:tobias.heine@springer-schule.de)

**Andreas Mundt**  
[a.mundt@lehrerfortbildung-bw.de](mailto:a.mundt@lehrerfortbildung-bw.de)

**Jan Nathan**  
[jan.nathan@lehrerfortbildung-bw.de](mailto:jan.nathan@lehrerfortbildung-bw.de)

### 3.3 Von Befehlen zum Skript

<b>Gewichtung</b>	4
<b>Beschreibung</b>	Kandidaten sollten aus sich wiederholenden Befehlen einfache Skripte erstellen können.

#### **Hauptwissensgebiete:**

- Grundlagen von Shell-Skripten
- Wissen über die gängigen Texteditoren (vi and nano)

#### **Hier ist eine auszugsweise Liste der verwendeten Dateien, Begriffe und Hilfsprogramme:**

- #! (shebang)
- /bin/bash
- Variablen
- Argumente
- for-Loops
- echo
- Exit-Status

Quelle: [https://wiki.lpi.org/wiki/LinuxEssentials\\_Objectives\\_V1.6\(DE\)#3.3\\_Von\\_Befehlen\\_zum\\_Skript](https://wiki.lpi.org/wiki/LinuxEssentials_Objectives_V1.6(DE)#3.3_Von_Befehlen_zum_Skript) Stand 08.12.22

- Skript = Textdatei mit Befehlen (*ohne Formatierung!*)
- Erstellung mit Texteditor
  - Grafische Editoren gibt es viele, sind aber nicht in LinEss
  - In der Konsole gibt es sehr mächtige Editoren, die leider eine steile Lernkurve haben: vi / **vim** sowie emacs
  - Einfach bedienbar: pico und dessen Fork **nano** (GNU-Tool)

- Im Screenshot sieht man oben Informationen, unten eine Befehls-Kurzreferenz.  
  ^ steht für die Strg-Taste → Beenden mit Strg-X

```
GNU nano 5.4          test.txt

Hallo Welt! Moege der Tux mit eu█

^G Hilfe      ^O Speichern    ^W Wo ist      ^K Ausschneiden
^X Beenden    ^R Datei öffne  ^\ Ersetzen    ^U Einfügen
```

1. Textdatei anlegen: `nano test.sh`
2. Den Befehl `echo "Hallo allerseits"` hinein schreiben
3. Mit Strg-S speichern (mit Enter bestätigen)
4. Ausführen mit `sh test.sh`
  - Startet die Shell `sh` und lässt diese den Befehl ausführen

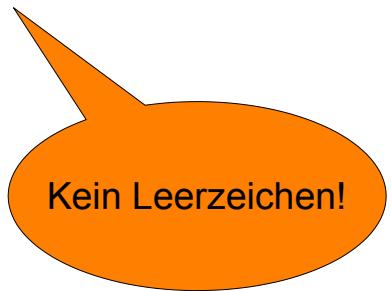
Alternativ: Skriptdatei mit `chmod` ausführbar machen

- `chmod +x test.sh`
- Ausführen mit `./test.sh`
  - `./` notwendig, weil aktuelles Verzeichnis `.` nicht im Suchpfad `$PATH`

- Mit den Zeichen **#!** ganz am Anfang eines Skripts kann man die zu verwendende Shell angeben
  - `#!/bin/bash` oder `#!/bin/sh`
  - `#!/usr/bin/python`
- **#!** nennt man Shebang
  - Kombination aus Hash **#** und Bang **!**

- sleep → eine bestimmte Anzahl an Sekunden schlafen
  - `sleep 10` wartet 10 Sekunden
- date → aktuelles Datum und Uhrzeit ausgeben
- echo → Ausgabe von Texten und Variablen
  - `echo Hallo Welt!`
  - `echo "Hallo Welt ich bin $USER und befinde mich aktuell in $PWD"`

- Shellvariablen sind nur lokal innerhalb der aktuellen Shell sichtbar – sie werden nicht an Kindprozesse weitergegeben
  - Mit dem Befehl `export variablename` kann eine Shellvariable zur Umgebungsvariablen „befördert“ werden
- Umgebungsvariablen werden an Kindprozesse weitergegeben. Sie bilden gemeinsam die Umgebung eines laufenden Programms (engl. Environment)
  - Befehl `env` (Environment): alle Variablen anzeigen lassen
  - `variablename=wert` : Shellvariablen anlegen / ändern
  - `$variablename` : Variable auswerten / verwenden
  - `unset variablename`: Variable löschen



Kein Leerzeichen!

- Mit `read` kann ein Wert vom Benutzer erfragt werden:

```
read -p "Gib Deinen Namen ein: " name
echo "Hallo $name!"
```

- `$0`: Name des Skripts / Programms
- `$1, $2, $3,..., $9`: Kommandozeilenargument Nr. 1 bis 9
- `$?`: Rückgabewert („exit code“) des zuletzt ausgeführten Befehls
  - 0 → „alles OK“, Befehl erfolgreich
  - 1 bis 255: einer von 255 Fehlercodes (siehe manpage des jeweiligen Befehls)
- `variablename=`befehl``: Ausgabe des Befehls in Variable abspeichern

- Wie in allen Programmiersprachen gibt es Verzweigungen
- ```
if <Bedingung>; then
    echo "Bedingung ist wahr"
else
    echo "Bedingung ist falsch"
fi
```
- Helferprogramm `test` prüft verschiedene Bedingungen
  - `if test -f dateiname; then ...`: Existiert die Datei? Dann führe ... aus
  - Test bietet Operatoren für Dateien, Verzeichnisse, Zahlenvergleiche, Text-/Stringvergleiche → NDG 9.4.2 bzw. `man test`

- Bei `if` können mit `elif` (else if) Alternativen geprüft werden  
`if ... then ... elif ... then ... elif ... then ... else ... fi`
- Für Fallunterscheidungen gibt es `case`

```
read -p "Gib eine Grussformel ein: " gruss
case $gruss in
    hallo|hi)
        echo "Selber hi"
        ;;
    bye|ciao)
        echo "Machs gut"
        ;;
    *)
        echo "Die Floskel ist mir neu ; - )"
esac
```

Kurze Demo

- Statt `test` kann man auch die Kurzform `[` verwenden, muss aber nach dem Ausdruck auch die Klammer schließen

- `if test -f datei` entspricht `if [ -f datei ]`
  - `if test $anz -gt 10` entspricht `if [ $anz -gt 10 ]`

- Ausrufezeichen am Anfang dreht Bedeutung um:

- `if test ! -f datei` oder `if [ ! -f datei ]`
  - `if test ! $anz -gt 10` oder `if [ ! $anz -gt 10 ]`

- Beispiele für Tests (nur ein Auszug)

- `-f name / -d name` → Datei/Verzeichnis vorhanden?
  - `-x name` → Datei vorhanden und ausführbar?
  - `-eq / -lt / -gt` → Zahlenvergleich: equal, less or greater than
  - `=` → Zeichenketten auf Gleichheit testen

- Im Gegensatz zu vielen Programmiersprachen geht die for-Schleife der bash nicht über Zahlen, sondern über eine Liste von Strings (Zeichenketten)

```
for name in Lars Steffi Kurt
do
    echo "Hallo $name"
done
```

- Mit Variablen:

```
PCS="192.168.1.1 192.168.1.2 192.168.1.3"
for pc in $PCS
do
    ping -c1 $pc
done
```

- for-Schleife mit Datei-Globbing  
(Shell ersetzt \* durch alle Dateinamen)

```
for datei in *
do
    echo "Die Datei $datei wird bearbeitet"
done
```

- Praxisbeispiel: Fingerprints der SSH-Host-Keys anzeigen:

```
for i in /etc/ssh/*.pub; do ssh-keygen -lf $i; done
256 SHA256:m49h3ugLJFCvkYVneyGPQvxB2oW7qkUAS8LFZ6UNMII root@bgthnk (ECDSA)
256 SHA256:mu+Vqc2grwaC0GiTdYd4lEAea2ipusCuTaFI0jQ3DvQ root@bgthnk (ED25519)
3072 SHA256:xPkUtXv0xNW15Y6ytAh/mQ22x09Pl7mqKilURwqNwX0 root@bgthnk (RSA)
```

- Mit einer while-Schleife kann man die klassische for-Zählschleife nachbilden

```
➤ i=0
while [ $i -lt 10 ]; do
    echo $i
    i=$(( $i + 1 ))
done
echo "Fertig :)"
```

- Die Schreibweise `$(( Ausdruck ))` sorgt für arithmetische Auswertung eines Ausdrucks – hier kann die Shell mit den üblichen Rechenoperatoren rechnen

## Anmerkungen

- Shell-Skripte sind überaus mächtig!
- Das Thema ist absolut oberflächlich behandelt. Viele wesentliche Inhalte sind nicht abgedeckt (Funktionen, getopt, ...)
- Nicht nur das Thema hier abhaken ... sondern:  
Probieren und Erfahrungen sammeln!
- Empfehlung: Eigene Skripte mit `shellcheck` auf Fehler prüfen.

## Credits

- Die Präsentation wurde ursprünglich von Michael Krüger ([mich@elkrueger.de](mailto:mich@elkrueger.de)) erstellt
- Aktualisierungen (Screenshots, LinuxEssentials Objektives-Neuerungen) wurden von Jan Nathan ([jan.nathan@lehrerfortbildung-bw.de](mailto:jan.nathan@lehrerfortbildung-bw.de)) hinzugefügt. Stand 2022-12-08