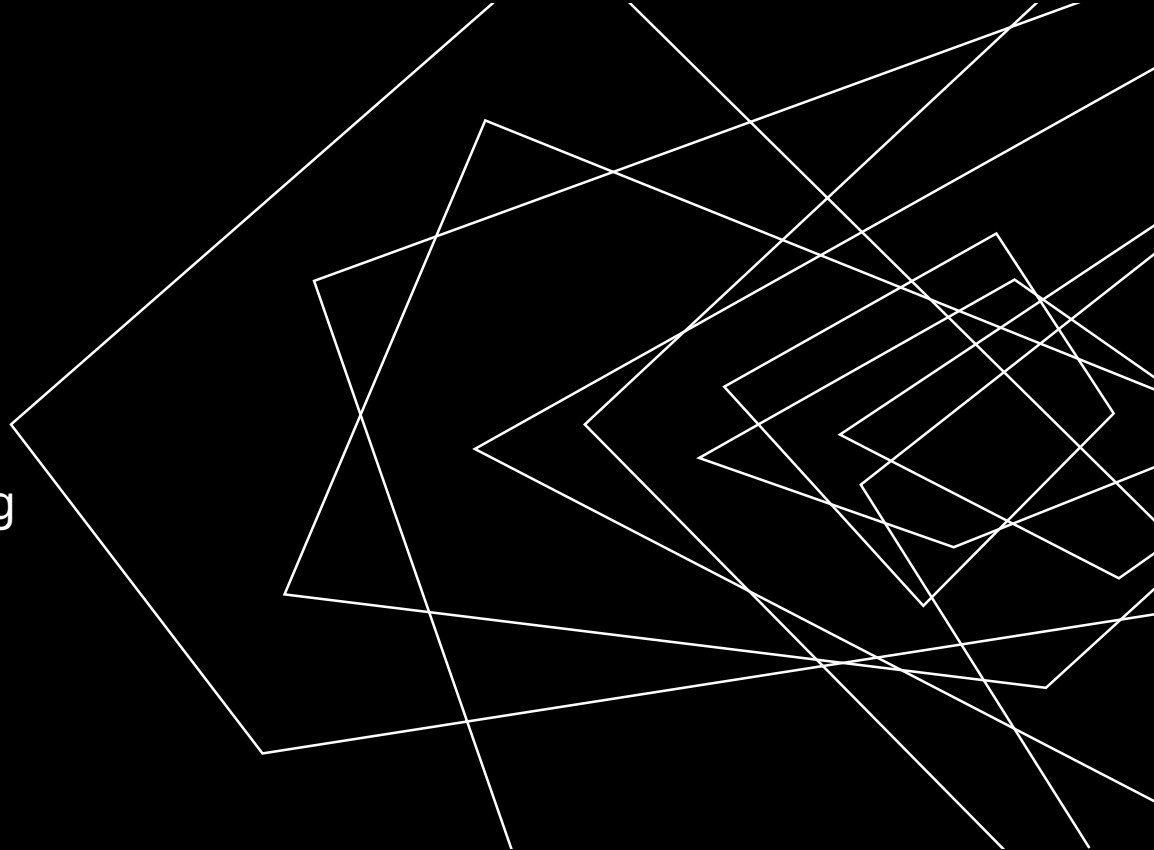
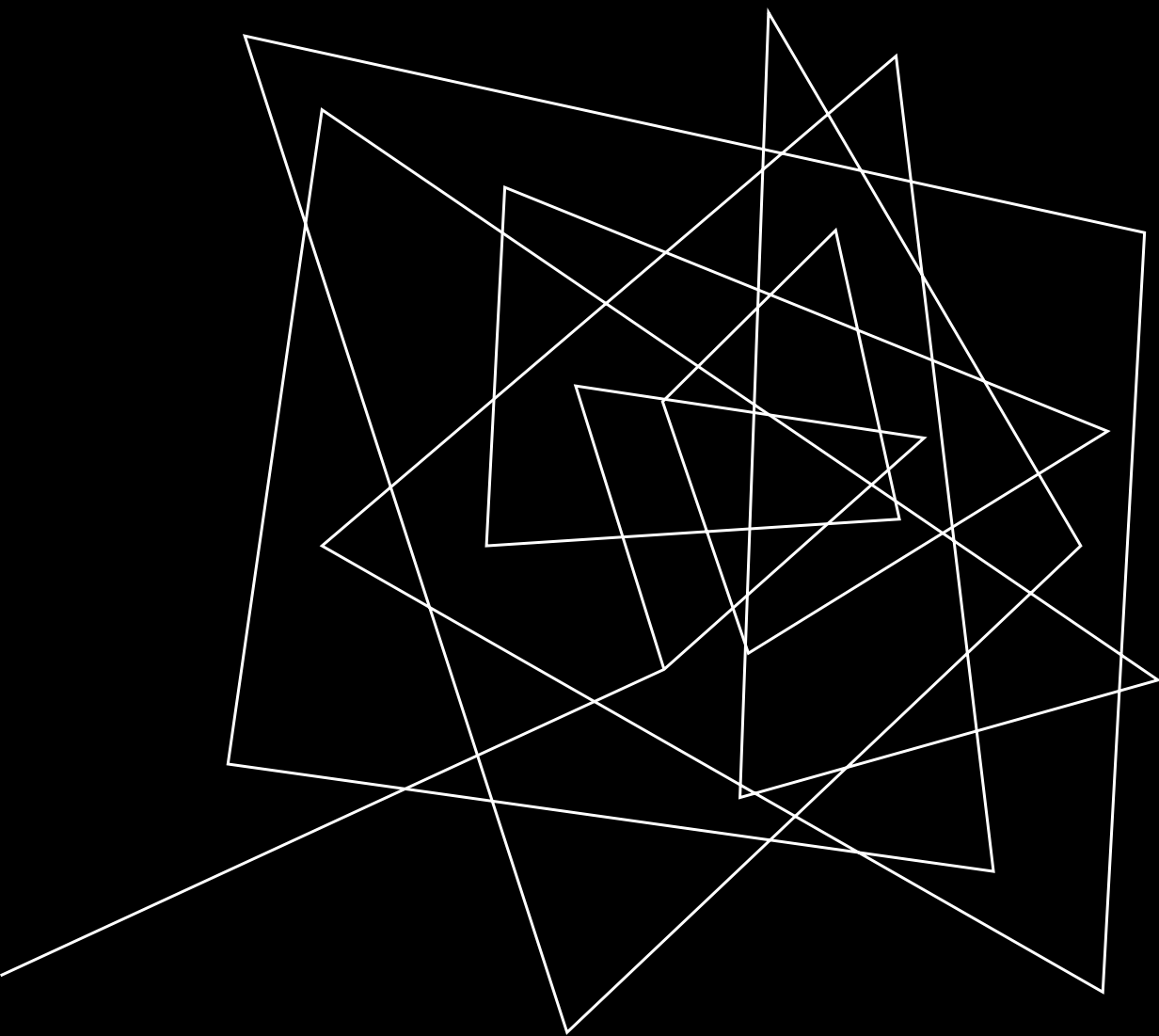


LINUX - TAG 4

AGENDA

1. Einführung in Shell-Skripting
2. Fortgeschrittenes Shell-Skripting
3. Prozessmanagement und Systemüberwachung
4. Cron-Jobs und Automatisierung
5. Systemadministration Grundlagen





1. EINFÜHRUNG IN SHELL-SKRIPTING

Grundlagen der Shell-Skripte, Erstellung einfacher
Skripte, Variablen und Steuerstrukturen

1. EINFÜHRUNG IN SHELL-SKRIPTING

- 1.1. Was ist Shell-Skripting?
- 1.2. Grundlagen der Shell-Skripte
- 1.3. Variablen in Shell-Skripten
- 1.4. Steuerstrukturen: Bedingungen und Schleifen

1.1. WAS IST EIN SHELL-SKRIPT?

Definition:

- Ein Shell-Skript ist eine Textdatei, die eine Reihe von Befehlen enthält, die in der Shell ausgeführt werden.

Vorteile:

- Automatisierung wiederholter Aufgaben
- Batch-Verarbeitung

Aufbau:

- Shebang (!) gibt die zu verwendende Shell an.
- Skriptdateien werden ausführbar gemacht.

Beispiel:

- ```
#!/bin/bash
echo "Hallo, Welt!"
```

## 1.2. ERSTELLUNG EINES EINFACHEN SKRIPTS

### Schritte zur Erstellung:

1. **Skript schreiben:** Mit einem Texteditor ein Skript erstellen (z.B. nano script.sh).
2. **Ausführbar machen:** Mit `chmod +x script.sh` das Skript ausführbar machen.
3. **Skript ausführen:** Mit `./script.sh` das Skript starten.

### Beispiel-Skript:

```
#!/bin/bash
echo "Das ist ein einfaches Shell-Skript."
```

## 1.3. VARIABLEN IN SHELL-SKRIPTEN

### Verwendung von Variablen:

- Speichern von Daten, die während der Ausführung genutzt werden.

### Syntax:

- **Zuweisung:** `variable=Wert`
- **Zugriff:** `$variable`

### Beispiel:

```
#!/bin/bash
name="Max"
echo "Hallo, $name"
```

## 1.4. BENUTZERDEFINIERT EINGABEN

### Eingaben des Benutzers abfragen:

- Mit read Benutzereingaben in Variablen speichern.

### Syntax:

```
#!/bin/bash
echo "Wie heißt du?"
read name
echo "Hallo, $name"
```



# 1.5. STEUERSTRUKTUREN – BEDINGUNGEN

## if-Abfragen:

- Verwendet, um Bedingungen in einem Skript zu überprüfen.

## Syntax:

```
if [Bedingung]; then
 Befehle
else
 Befehle
fi
```

## Beispiel:

```
#!/bin/bash
if [$1 -gt 10]; then
 echo "Die Zahl ist größer als 10"
else
 echo "Die Zahl ist kleiner oder gleich 10"
fi
```

## 1.6. STEUERSTRUKTUREN – SCHLEIFEN

### **for-Schleife:**

- Wiederholt Befehle für eine Liste von Elementen.

### **Syntax:**

```
for variable in Liste; do
 Befehle
done
```

### **Beispiel:**

```
#!/bin/bash
for i in 1 2 3; do
 echo "Zahl: $i"
done
```

## 1.6. STEUERSTRUKTUREN – SCHLEIFEN

### **while-Schleife:**

- Führt Befehle aus, solange eine Bedingung wahr ist.

### **Syntax:**

```
while [Bedingung]; do
 Befehle
done
```

### **Beispiel:**

```
#!/bin/bash
counter=1
while [$counter -le 3]; do
 echo "Zähler: $counter"
 counter=$((counter+1))
done
```

## 1.7. PRAXISBEISPIELE

### 1. Einfaches Skript:

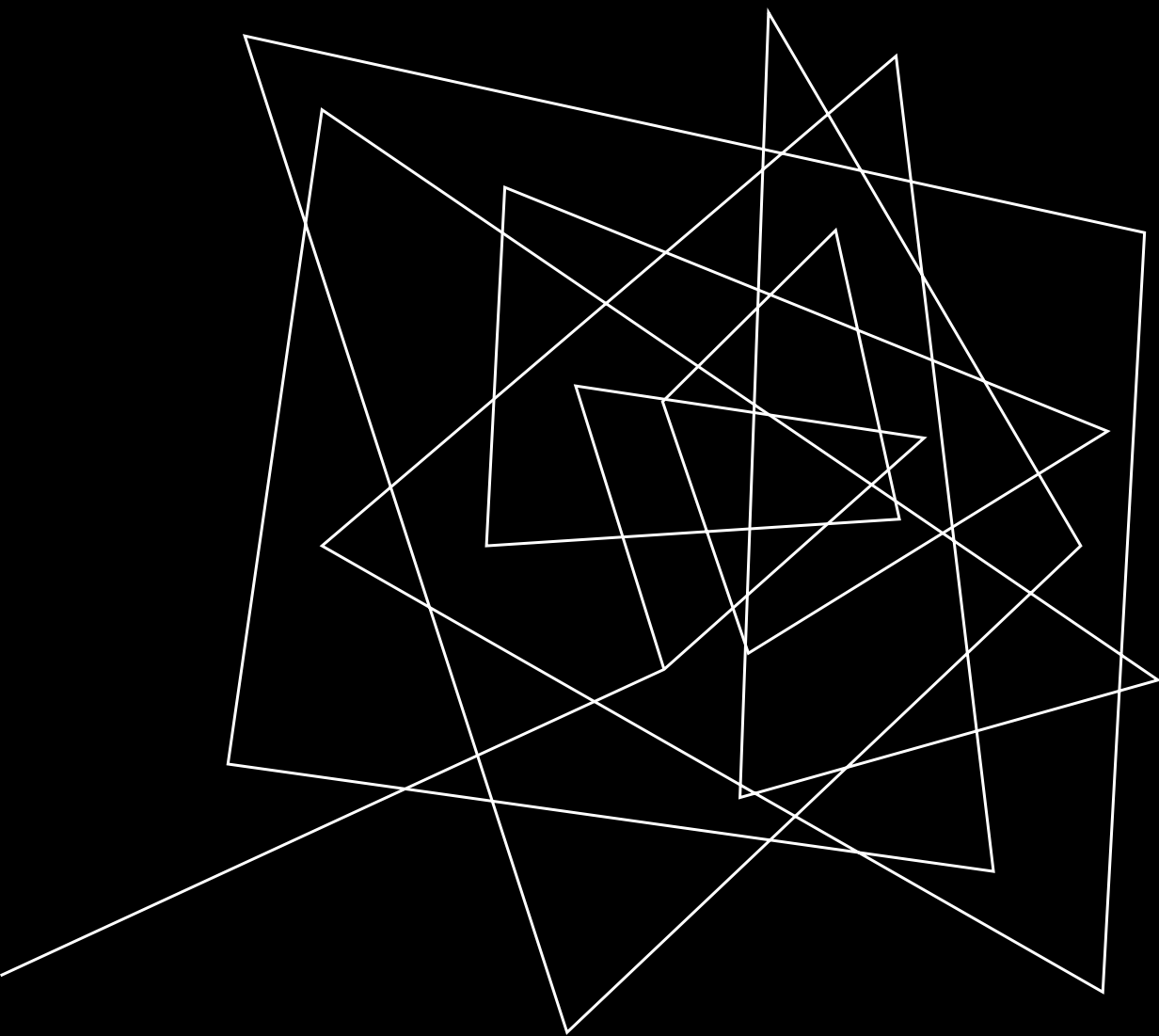
- Erstellt ein Skript, das den Namen des Benutzers abfragt und eine Begrüßung ausgibt.

### 2. Bedingungen und Schleifen:

- Ein Skript, das prüft, ob eine Zahl größer oder kleiner als 10 ist.
- Ein Skript, das Zahlen von 1 bis 3 ausgibt.

## 1.8. ZUSAMMENFASSUNG

- **Shell-Skripte** ermöglichen die Automatisierung von Aufgaben in der Linux-Shell.
- **Variablen** speichern Daten, die zur Laufzeit verwendet werden.
- **Steuerstrukturen** wie if-Bedingungen und Schleifen steuern den Ablauf von Skripten.
- Shell-Skripting ist ein mächtiges Werkzeug, um wiederkehrende Aufgaben effizienter zu erledigen.



## 2. FORTGESCHRITTENES SHELL-SKRIPTING

Schleifen, Bedingungen, Skriptparameter, Arbeiten mit  
Dateien in Skripten

## 2. FORTGESCHRITTENES SHELL-SKRIPTING

- 2.1. Übersicht
- 2.2. Schleifen – Vertiefung
- 2.3. Bedingungen – Vertiefung
- 2.4. Arbeiten mit Skriptparametern
- 2.5. Arbeiten mit Dateien
- 2.6. Lesen von Dateien
- 2.7. Fehlerbehandlung
- 2.8. Praxisbeispiele
- 2.9. Zusammenfassung

## 2.1. ÜBERSICHT

- Fortgeschrittene Steuerstrukturen
- Arbeiten mit Skriptparametern
- Schleifen in der Tiefe
- Manipulation von Dateien in Shell-Skripten



## 2.2.1. SCHLEIFEN – VERTIEFUNG

### **for-Schleife:**

- Iteriert durch Listen oder Sequenzen.

### **Beispiel:**

```
#!/bin/bash
for datei in *.txt; do
 echo "Bearbeite Datei: $datei"
done
```

## 2.2.2. SCHLEIFEN – VERTIEFUNG

### **while-Schleife:**

- Lässt Skripte laufen, solange eine Bedingung erfüllt ist.

### **Beispiel:**

```
#!/bin/bash
counter=5
while [$counter -gt 0]; do
 echo "Countdown: $counter"
 counter=$((counter-1))
done
```

## 2.3.1. BEDINGUNGEN – VERTIEFUNG

### **if-else-Anweisungen:**

- Mehrfachverschachtelungen und Prüfung mehrerer Bedingungen.

### **Beispiel:**

```
#!/bin/bash
zahl=$1
if [$zahl -gt 10]; then
 echo "Größer als 10"
elif [$zahl -eq 10]; then
 echo "Gleich 10"
else
 echo "Kleiner als 10"
fi
```

## 2.3.2. BEDINGUNGEN – VERTIEFUNG

### Case-Anweisung:

- Verwenden, um mehrere Bedingungen elegant zu prüfen.

### Beispiel:

```
#!/bin/bash
case $1 in
 start)
 echo "Starten..."
 ;;
 stop)
 echo "Stoppen..."
 ;;
 restart)
 echo "Neustarten..."
 ;;
 *)
 echo "Unbekannter Befehl!"
 ;;
esac
```

## 2.4. ARBEITEN MIT SKRIPTPARAMETERN

### **Skriptparameter:**

- Parameter, die beim Aufruf des Skripts übergeben werden, sind zugänglich durch \$1, \$2, etc.

### **Beispiel:**

```
#!/bin/bash
echo "Das erste Argument ist: $1"
echo "Das zweite Argument ist: $2"
```

## 2.5. ARBEITEN MIT DATEIEN

### Dateien erstellen und bearbeiten:

- **touch [Dateiname]**: Erstellt eine neue Datei.
- **cat**: Gibt den Inhalt einer Datei aus.
- **echo** mit Umleitung:
  - `echo "Text" > datei.txt`: Schreibt Text in eine Datei (Überschreiben).
  - `echo "Text" >> datei.txt`: Fügt Text an eine Datei an.

### Beispiel:

```
#!/bin/bash
echo "Erstelle eine Datei..."
touch neue_datei.txt
echo "Hallo Welt" > neue_datei.txt
```

## 2.6. LESEN VON DATEIEN

### Inhalt von Dateien lesen:

- **cat [Dateiname]**: Zeigt den Inhalt einer Datei an.
- **while-Schleife** zum Zeilenweise Lesen:

### Beispiel:

```
#!/bin/bash
while IFS= read -r zeile; do
 echo "Gelesene Zeile: $zeile"
done < datei.txt
```

## 2.7. FEHLERBEHANDLUNG

### Überprüfen von Fehlern:

- `$?`: Gibt den Rückgabewert des letzten Befehls zurück.

### Beispiel:

```
#!/bin/bash
mkdir testverzeichnis
if [$? -eq 0]; then
 echo "Verzeichnis erfolgreich erstellt"
else
 echo "Fehler beim Erstellen des Verzeichnisses"
fi
```



## 2.8. PRAXISBEISPIELE

### 1. Skript mit Parametern:

- Ein Skript, das zwei Zahlen als Parameter nimmt und ihre Summe berechnet.

### 2. Bearbeitung von Dateien:

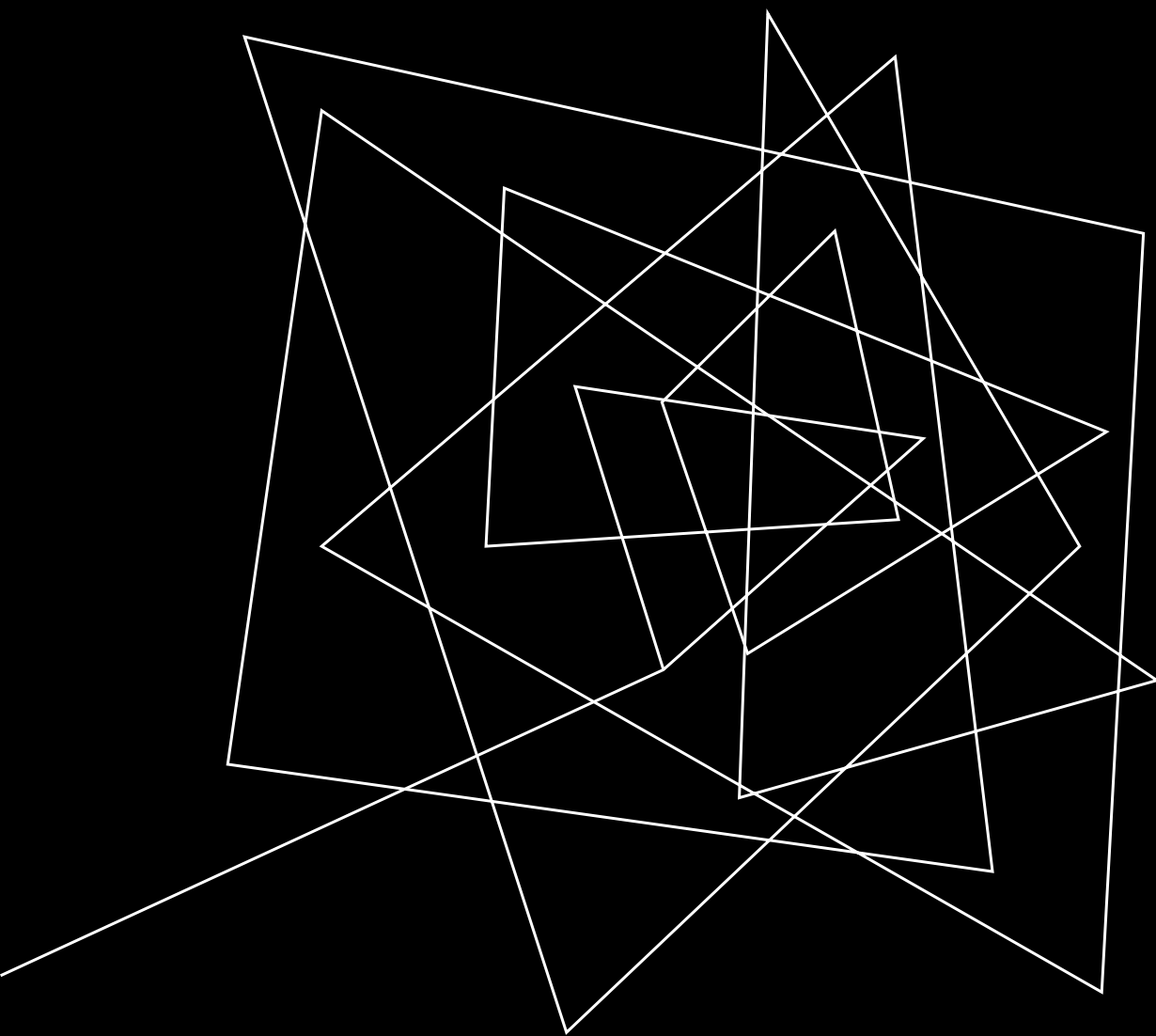
- Ein Skript, das durch alle .txt-Dateien in einem Verzeichnis iteriert und Text hinzufügt.

### 3. Fehlerbehandlung:

- Ein Skript, das prüft, ob eine Datei erfolgreich erstellt wurde.

## 2.9. ZUSAMMENFASSUNG

- **Erweiterte Schleifen** erlauben die Verarbeitung von Listen, Dateien und Zahlen.
- **Bedingungen** und case-Strukturen bieten vielseitige Kontrollmöglichkeiten.
- **Skriptparameter** machen Skripte dynamischer und flexibler.
- **Arbeiten mit Dateien** ist eine wichtige Fähigkeit für die Automatisierung in Shell-Skripten.



### 3. PROZESSMANAGEMENT UND SYSTEMÜBERWACHUNG

Prozessanzeige und -steuerung mit ps, top,  
kill, jobs und Hintergrundprozesse

# 3. PROZESSMANAGEMENT UND SYSTEMÜBERWACHUNG

- 3.1. Übersicht
- 3.2. Prozesse unter Linux
- 3.3. Der Befehl ps
- 3.4. Der Befehl top
- 3.5. Prozesse beenden mit kill
- 3.6. Jobs und Hintergrundprozesse
- 3.7. Praktische Anwendung
- 3.8. Zusammenfassung

## 3.1. ÜBERSICHT

- **Prozesse unter Linux**
- **Überwachen von Systemressourcen**
- **Befehle zur Prozesssteuerung: ps, top, kill**
- **Jobs und Hintergrundprozesse**

## 3.2. PROZESSE UNTER LINUX

### Was ist ein Prozess?

- Ein laufendes Programm mit einem eigenen Speicherbereich.

### Prozessarten:

- **Vordergrundprozesse:** Vom Benutzer direkt gestartet.
- **Hintergrundprozesse:** Laufen im Hintergrund ohne direkte Benutzereingaben.

### Prozess-IDs (PIDs):

- Jeder Prozess erhält eine eindeutige ID.

## 3.3. DER BEFEHL PS

### Verwendung:

- Zeigt eine Momentaufnahme der laufenden Prozesse an.

### Syntax:

- **ps [Optionen]**

### Häufig verwendete Optionen:

- **ps aux:** Zeigt alle Prozesse an, inklusive CPU- und Speicherverbrauch.
- **ps -ef:** Detaillierte Ansicht der Prozesse mit Eltern-Kind-Beziehungen.

### Beispiel:

```
ps aux | grep apache
```

## 3.4. DER BEFEHL TOP

### **Verwendung:**

- Echtzeitüberwachung von Prozessen, CPU-Auslastung, Speicherverbrauch.

### **Wichtige Tasten:**

- **q**: Beendet top.
- **k**: Beenden eines Prozesses.
- **h**: Zeigt Hilfe an.

### **Beispiel:**

top



## 3.5. PROZESSE BEENDEN MIT KILL

### Verwendung:

- Beendet einen Prozess durch Senden eines Signals.

### Syntax:

- **kill** [Signal] [PID]

### Häufige Signale:

- **SIGTERM (15)**: Standardmäßiges Beenden eines Prozesses.
- **SIGKILL (9)**: Erzwingt das sofortige Beenden eines Prozesses.

### Beispiel:

```
kill -9 1234
```

## 3.6. JOBS UND HINTERGRUNDPROZESSE

---

### Hintergrundprozesse starten:

- Prozesse können mit **&** im Hintergrund ausgeführt werden.

### Beispiel:

```
./script.sh &
```

### Überwachung von Jobs:

- **jobs**: Zeigt laufende und gestoppte Jobs.
- **fg [Jobnummer]**: Bringt einen Hintergrundprozess in den Vordergrund.
- **bg [Jobnummer]**: Führt einen gestoppten Job im Hintergrund weiter.

### Beispiel:

```
jobs
fg %1
```

## 3.7. PRAKTISCHE ANWENDUNG

### 1. Prozessliste anzeigen:

- Mit **ps aux** alle Prozesse überwachen.

### 2. Prozesse in Echtzeit:

- Mit **top** CPU- und Speicherverbrauch verfolgen.

### 3. Prozesse beenden:

- Mit **kill** und der PID störende Prozesse stoppen.

### 4. Jobs verwalten:

- Einen Prozess im Hintergrund starten und verwalten.

## 3.8. ZUSAMMENFASSUNG

---

- **Prozessanzeige:** Mit **ps** eine Momentaufnahme der Prozesse erhalten.
- **Echtzeitüberwachung:** Mit **top** Prozesse und Systemressourcen in Echtzeit beobachten.
- **Prozesssteuerung:** Prozesse mit **kill** beenden.
- **Jobs und Hintergrundprozesse:** Jobs mit **fg**, **bg** und **jobs** verwalten.



## 4. CRON-JOBS UND AUTOMATISIERUNG

Planen und Automatisieren von Aufgaben mit cron,  
Erstellen von crontab Einträgen.

# 4. CRON-JOBS UND AUTOMATISIERUNG

- 4.1. Übersicht
- 4.2. Was ist cron?
- 4.3. Die Crontab-Datei
- 4.4. Crontab-Syntax
- 4.5. Platzhalter in Crontab
- 4.6. Automatisierung mit Cron-Jobs
- 4.7. Fehlerbehebung bei Cron-Jobs
- 4.8. Praxisbeispiele
- 4.9. Zusammenfassung

## 4.1. ÜBERSICHT

- **Was ist cron?**
- **Einsatz von Cron-Jobs**
- **Syntax von Crontab-Einträgen**
- **Automatisierung von Aufgaben mit cron**

## 4.2. WAS IST CRON?

### Definition:

- **cron** ist ein Dienst auf Linux- und Unix-Systemen, der die **zeitgesteuerte Ausführung** von Aufgaben ermöglicht.

### Typische Anwendungen:

- Automatisierung von Backups
- Systemwartung
- Periodische Ausführung von Skripten



## 4.3. DIE CRONTAB-DATEI

### Was ist crontab?

- Die Crontab-Datei enthält die Aufgaben (Jobs), die vom cron-Dienst ausgeführt werden sollen.

### Wichtige Befehle:

- **crontab -e**: Crontab-Datei bearbeiten.
- **crontab -l**: Liste der aktuellen Crontab-Einträge anzeigen.
- **crontab -r**: Alle Crontab-Einträge löschen.

## 4.4. CRONTAB-SYNTAX

### Grundstruktur eines Crontab-Eintrags:

\* \* \* \* \* /pfad/zum/skript.sh

- **Minute (0-59)**
- **Stunde (0-23)**
- **Tag des Monats (1-31)**
- **Monat (1-12)**
- **Wochentag (0-7, wobei 0 und 7 Sonntag darstellen)**

### Beispiel:

30 2 \* \* \* /home/user/backup.sh

- Führt das Skript **täglich** um **02:30 Uhr** aus.

## 4.5. PLATZHALTER IN CRONTAB

### Spezielle Zeichen:

- \*: Beliebiger Wert (jede Minute, jede Stunde, etc.).
- ,: Mehrere Werte (z.B. "1,15" für Minute 1 und 15).
- -: Bereich von Werten (z.B. "1-5" für Montag bis Freitag).
- /: Schrittweite (z.B. "\* /5" für alle 5 Minuten).

### Beispiel:

```
* /10 * * * * /pfad/zum/skript.sh
```

- Führt das Skript **alle 10 Minuten** aus.

## 4.6. AUTOMATISIERUNG MIT CRON-JOBS

---

### Häufige Automatisierungsaufgaben:

- **Backups:** Regelmäßige Sicherung von Daten.
- **Log-Analyse:** Automatische Überprüfung und Archivierung von Log-Dateien.
- **Systemwartung:** Automatisiertes Update von Paketen oder Systemchecks.

### Beispiel:

```
0 0 * * 7 /home/user/system_cleanup.sh
```

- Führt ein Skript **jeden Sonntag um Mitternacht** aus, um das System zu bereinigen.

## 4.7. FEHLERBEHEBUNG BEI CRON-JOBS

### Protokollierung von Ausgaben:

- Um Fehler oder Ausgaben von Cron-Jobs zu prüfen, können diese in eine Datei umgeleitet werden.

### Beispiel:

```
* * * * * /pfad/zum/skript.sh >> /var/log/cronlog.log 2>&1
```

- Alle Ausgaben werden in **/var/log/cronlog.log** gespeichert.

### Cron-Protokoll anzeigen:

- **/var/log/syslog** unter Debian/Ubuntu oder **/var/log/cron** unter CentOS.

## 4.8. PRAXISBEISPIELE

### **1. Tägliches Backup-Skript:**

- Führe ein Skript jeden Tag um Mitternacht aus, um Daten zu sichern.

### **2. Systemupdates automatisieren:**

- Automatisiere die Installation von Systemupdates alle zwei Wochen.

### **3. Log-Dateien archivieren:**

- Verschiebe und archiviere Log-Dateien jeden Sonntag.

## 4.9. ZUSAMMENFASSUNG

---

- **Cron-Jobs** ermöglichen die **Automatisierung** von Aufgaben auf Linux- und Unix-Systemen.
- **Crontab**-Einträge folgen einer festgelegten Zeitsyntax und können verschiedene Automatisierungsaufgaben steuern.
- Typische Anwendungsfälle umfassen **Backups, Systemwartung, und Log-Archivierung.**



# 5. SYSTEMADMINISTRATION GRUNDLAGEN

Verwalten von Systemdiensten, Systemstart und -stop  
mit systemctl, Überblick über Systemprotokolle



# 5. SYSTEMADMINISTRATION GRUNDLAGEN

- 5.1. Übersicht
- 5.2. Systemdienste unter Linux
- 5.3. Dienstverwaltung mit systemctl
- 5.4. Systemstart und -stop von Diensten
- 5.5. Überblick über Systemprotokolle
- 5.6. Wichtige Systemprotokoll-Befehle
- 5.7. Fehlerbehebung mit Systemprotokollen
- 5.8. Praxisbeispiele
- 5.9. Zusammenfassung

## 5.1. ÜBERSICHT

- Systemdienste unter Linux
- Dienstverwaltung mit **systemctl**
- System**start** und **-stop** von Diensten
- Überblick über **Systemprotokolle**

## 5.2. SYSTEMDIENSTE UNTER LINUX

### Was sind Systemdienste?

- Programme oder Prozesse, die im Hintergrund laufen und bestimmte Aufgaben erfüllen.
- Beispiele: **Webserver, Datenbankdienste, Druckerwarteschlangen.**

### Arten von Diensten:

- **Systemdienste:** Laufen unabhängig von einem Benutzer (z.B. sshd).
- **Benutzerdienste:** Werden von einem Benutzer gestartet.

## 5.3. DIENSTVERWALTUNG MIT SYSTEMCTL

---

### **systemctl:**

- Ein zentrales Werkzeug zur Verwaltung von Systemd-Diensten.

### **Wichtige Befehle:**

- **systemctl start [Dienst]:** Startet einen Dienst.
- **systemctl stop [Dienst]:** Stoppt einen Dienst.
- **systemctl restart [Dienst]:** Startet einen Dienst neu.
- **systemctl status [Dienst]:** Zeigt den Status eines Dienstes an.

### **Beispiel:**

```
systemctl status apache2
```

## 5.4. SYSTEMSTART UND -STOPP VON DIENSTEN

---

### Dienste beim Systemstart:

- Dienste können so konfiguriert werden, dass sie beim Booten automatisch starten.

### Befehle:

- **systemctl enable [Dienst]:** Aktiviert den Dienst für den automatischen Start.
- **systemctl disable [Dienst]:** Deaktiviert den automatischen Start.

### Beispiel:

```
systemctl enable apache2
```

- **systemctl is-enabled [Dienst]:** Überprüft, ob ein Dienst aktiviert ist.

# 5.5. ÜBERBLICK ÜBER SYSTEMPROTOKOLLE

---

## Systemprotokolle (Logs):

- Enthalten Informationen über Systemereignisse, Fehler und Statusmeldungen.

## Log-Dateien:

- **/var/log/syslog**: Allgemeine Systemprotokolle.
- **/var/log/auth.log**: Protokoll für Authentifizierungen.
- **/var/log/dmesg**: Kernel-Meldungen.

## Anzeige von Logs:

- **journalctl**: Befehl zur Anzeige von Systemd-Logs.

## Beispiele:

```
journalctl -xe
```

- Zeigt die letzten Fehlermeldungen im Systemd-Protokoll.

## 5.6. WICHTIGE SYSTEMPROTOKOLL-BEFEHLE

---

### **journalctl-Befehle:**

- **journalctl -b:** Zeigt Logs des aktuellen Bootvorgangs an.
- **journalctl -u [Dienst]:** Zeigt Logs eines bestimmten Dienstes.

### **Beispiel:**

```
journalctl -u apache2
```

- Zeigt die Protokolle des Apache-Webservers.

## 5.7. FEHLERBEHEBUNG MIT SYSTEMPROTOKOLLEN

---

### **Protokolle zur Diagnose:**

- Logs sind entscheidend für die Fehlersuche und Systemdiagnose.

### **Beispiel:**

- Ein Dienst startet nicht? Überprüfe die Protokolle mit `journalctl -u [Dienst]`.



## 5.8. PRAXISBEISPIELE

### **1. Dienst starten und stoppen:**

- Verwende `systemctl start` und `systemctl stop`, um Dienste zu verwalten.

### **2. Automatischen Start aktivieren:**

- Mit `systemctl enable` einen Dienst beim Booten starten lassen.

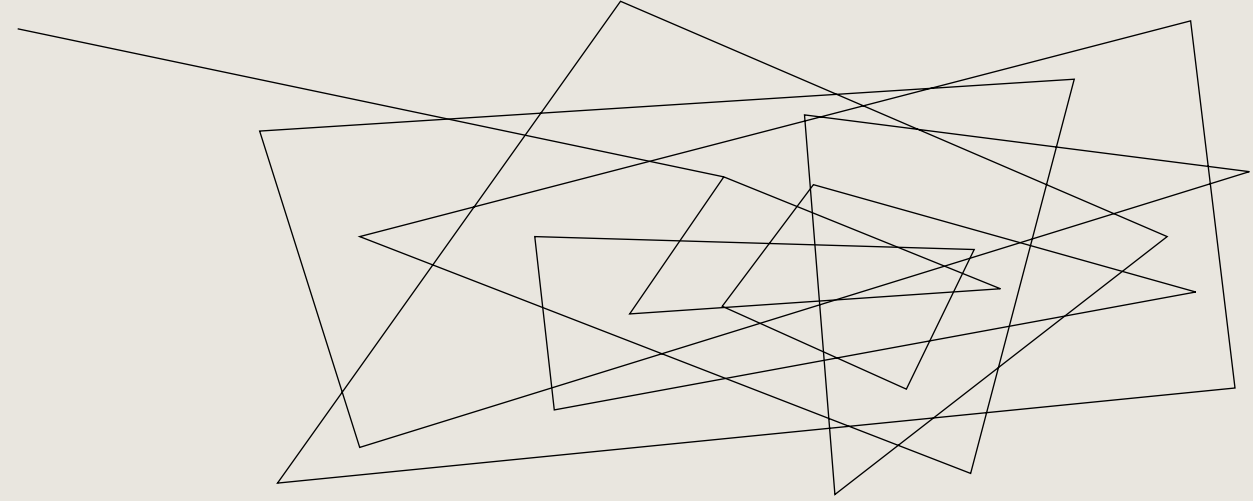
### **3. Fehlerprotokolle überprüfen:**

- Mit `journalctl` Protokolle durchsuchen, um Probleme zu beheben.

## 5.9. ZUSAMMENFASSUNG

---

- **systemctl** ist das zentrale Werkzeug zur Verwaltung von System-Diensten.
- Dienste können beim Systemstart aktiviert oder deaktiviert werden.
- **Systemprotokolle** bieten Einblick in die Aktivitäten und Fehler des Systems.
- **Protokoll-Tools** wie `journalctl` sind entscheidend für die Fehlerbehebung.





# DANKE

Costis Aivalis

[costis.Aivalis@gmail.com](mailto:costis.Aivalis@gmail.com)

[aivalis.eu](http://aivalis.eu)