Inhaltsverzeichnis

Buchseiten

Seite	Inhalt	Beschreibung
180	SystemD	Diensteverwaltung und Init-System.
28	Verzeichnisbaum	Hierarchische Dateistruktur ab /.
161	Sudo	Befehle mit Root-Rechten ausführen.
63	Ls-Befehl	Verzeichnisinhalt auflisten.
90	Touch-Befehl	Neue, leere Datei erstellen.
65	Cp-Befehl	Dateien/Ordner kopieren.
66	Mv-Befehl	Verschieben oder Umbenennen.
67	Rm-Befehl	Dateien oder Ordner löschen.
72	Mkdir-Befehl	Neues Verzeichnis erstellen.
41	Umgebungsvariablen	Werte zur Systemkonfiguration (z. B. $PATH).
60	Man-Befehl	Handbuch für Befehle anzeigen.
109	Grep-Befehl	Text nach Mustern durchsuchen.
124	Awk-Befehl	Text- und Spaltenbearbeitung.
125	Sed-Befehl	Automatisierte Textveränderung.
100	Find-Befehl	Dateien im System suchen.
76	Head-Befehl	Dateianfang anzeigen (Standard: erste 10 Zeilen).
77	Tail-Befehl	Dateiende anzeigen (Standard: letzte 10 Zeilen).
86	Stat-Befehl	Zeigt detaillierte Datei-Metadaten an.
124	Tee-Befehl	Schreibt Ausgabe in Datei und zeigt sie zeitgleich an.

Grundlegende Linux-Kommandos

Navigation und Dateisystem

Befehl	Beschreibung
pwd	Aktuelles Arbeitsverzeichnis anzeigen
Sudo (-u)	Programm als root(anderer benutzer)
su	Benutzer wechseln
cd verzeichnis	Verzeichnis wechseln
cd ~ oder cd	Ins Home-Verzeichnis wechseln
cd ..	Eine Ebene höher
cd -	Ins vorherige Verzeichnis
ls	Verzeichnisinhalt anzeigen
ls -l	Langes Listing (Details)
ls -a	Alle Dateien inkl. versteckte (mit . beginnend)
ls -lR	Rekursiv mit Details
ls -li	Mit Inode-Nummern
ls -lh	Menschenlesbare Größen
ls -ltr	Nach Zeit sortiert, älteste zuerst
ls -lSr	Nach Größe sortiert, kleinste zuerst

Wildcards / Globbing

Muster	Bedeutung	Beispiel
`*`	Beliebig viele beliebige Zeichen	`ls *.txt`
`?`	Genau ein beliebiges Zeichen	`ls prog?.c`
`[abc]`	Genau ein Zeichen aus der Menge	`ls p[12].*`
`[a-z]`	Zeichenbereich	`ls [a-z]*`

Dateien erstellen, kopieren, verschieben, löschen

Befehl	**Beschreibung**
`touch datei`	Leere Datei erstellen / Zeitstempel ändern
`touch -am datei`	Zugriffs- und Änderungszeit ändern
`mkdir verz`	Verzeichnis erstellen
`mkdir -p eltern/kind`	Verschachtelte Verzeichnisse erstellen
`cp quelle ziel`	Datei kopieren
`cp -r quelle ziel`	Verzeichnis rekursiv kopieren
`mv alt neu`	Datei verschieben/umbenennen
`mv -i alt neu`	Mit Nachfrage
`rm datei`	Datei löschen
`rm -i datei`	Mit Nachfrage
`rm -r verzeichnis`	Verzeichnis rekursiv löschen
`rm -rf verzeichnis`	Erzwungen rekursiv löschen (VORSICHT!)
`rmdir verz`	Leeres Verzeichnis löschen

Dateiinhalte anzeigen

Befehl	**Beschreibung**
`cat datei`	Gesamte Datei anzeigen
`cat > datei`	Von Tastatur in Datei schreiben (Beenden mit STRG+D)
`cat >> datei`	An Datei anhängen
`less datei`	Seitenweise anzeigen
`more datei`	Seitenweise anzeigen
`head datei`	Erste 10 Zeilen
`head -n 20 datei`	Erste 20 Zeilen
`tail datei`	Letzte 10 Zeilen
`tail -n 20 datei`	Letzte 20 Zeilen
`tail -f datei`	Live-Überwachung einer Datei
`wc -l datei`	Zeilen zählen
`diff datei1 datei2`	Unterschiede zwischen Dateien

Systeminformationen

Befehl	Beschreibung
`uname -a`	System-Infos (Kernel, Architektur)
`uname -v`	Kernel-Version
`uname -i`	Hardware-Plattform
`whoami`	Aktueller Benutzername
`who`	Eingeloggte Benutzer
`id`	Benutzer-/Gruppen-ID anzeigen
`date`	Datum und Uhrzeit
`uptime`	Systemlaufzeit und Auslastung
`free -h`	Speicherauslastung
`df -h`	Festplattenbelegung
`du -sh *`	Verzeichnisgrößen

Umleitungen und Pipes

Operator	Beschreibung
`>`	Ausgabe in Datei (überschreiben)
`>>`	Ausgabe in Datei (anhängen)
`<`	Eingabe aus Datei
`|`	Pipe: Ausgabe als Eingabe für nächsten Befehl
`2>&1`	Fehlerausgabe zu Standardausgabe umleiten
`&>/dev/null`	Alle Ausgaben unterdrücken

cp -u

Grundprinzip

cp kopiert Dateien und Verzeichnisse. Die Option -u steht für **update** und verändert das Kopierverhaltens grundlegend:
Datei wird nur kopiert, wenn die **Quelldatei neuer** ist als die Zieldatei – oder wenn die Zieldatei **nicht existiert**.

```
cp -u quelle.txt ziel.txt
```

Warum ist das nützlich?

Ohne -u überschreibt cp immer. Mit -u werden nur wirklich geänderte Dateien kopiert. Das ist ideal für:

- Inkrementelle Backups
- Synchronisation von Verzeichnissen ohne rsync
- Skripte, die nur bei Änderungen reagieren sollen

Alle relevanten cp-Optionen im Überblick

Option	Langform	Bedeutung
-u	--update	Nur kopieren wenn Quelle neuer als Ziel (oder Ziel fehlt)
-r / -R	--recursive	Verzeichnisse rekursiv kopieren
-v	--verbose	Jede Aktion ausgeben
-i	--interactive	Vor dem Überschreiben fragen
-n	--no-clobber	Existierende Dateien niemals überschreiben
-p	--preserve	Metadaten (Zeitstempel, Rechte, Eigentümer) erhalten
-a	--archive	Entspricht -dR --preserve=all
-l	--link	Hardlinks statt Kopien erstellen
-s	--symbolic-link	Symlinks statt Kopien
-f	--force	Ziel-Datei entfernen wenn nicht schreibbar
-b	--backup	Backup der Zieldatei anlegen
--suffix=SUFFIX		Backup-Suffix festlegen (Standard: ~)

Beispiele

```
# Nur neuere Dateien kopieren
cp -u dokument.txt /backup/dokument.txt

# Ganzes Verzeichnis aktualisierend synchronisieren
cp -ru /home/user/projekt/ /backup/projekt/

# Mit Ausgabe, welche Dateien wirklich kopiert wurden
cp -ruv /home/user/daten/ /mnt/backup/daten/

# Metadaten erhalten und nur Updates kopieren
cp -uap /var/www/ /backup/www/
```

Wie entscheidet cp -u?

cp -u vergleicht **mtime** (Modification Time) der Dateien:

Quelldatei-mtime > Zieldatei-mtime → KOPIEREN
Quelldatei-mtime ≤ Zieldatei-mtime → ÜBERSPRINGEN
Zieldatei existiert nicht → KOPIEREN

tac

Grundprinzip

tac ist das Gegenteil von cat und gibt eine Datei **zeilenweise in umgekehrter Reihenfolge** aus. Der Name ist cat rückwärts gelesen.

```
tac datei.txt
```

Enthält datei.txt:

```
Zeile 1
Zeile 2
Zeile 3
```

Gibt tac datei.txt aus:

```
Zeile 3
Zeile 2
Zeile 1
```

Alle Optionen

Option	Langform	Bedeutung
-b	--before	Trennzeichen vor statt nach dem Datensatz einfügen
-r	--regex	Das Trennzeichen als regulären Ausdruck interpretieren
-s TRENN	--separator=TRENN	Eigenes Trennzeichen statt Zeilenumbruch verwenden
--help		Hilfe anzeigen
--version		Versionsnummer anzeigen

Standardverhalten

Ohne Optionen gilt \n (Zeilenumbruch) als Trenner, und tac gibt die letzte Zeile zuerst aus.

Beispiele

```
# Datei umgekehrt ausgeben
tac /var/log/syslog | less

# Letzten Log-Eintrag zuerst sehen
tac /var/log/auth.log | head -20

# Mit eigenem Trennzeichen (statt Zeilenumbruch)
echo "a:b:c:d" | tac -s ":"
# Ausgabe: d:c:b:a:

# Trennzeichen wird VOR dem Datensatz gesetzt (-b)
echo -e "---BLOCK1n---BLOCK2n---BLOCK3" | tac -b -s "---"

# Trennzeichen als Regex (-r), z.B. beliebige Zahl als Trenner
echo "Wort1Wort2Wort3" | tac -r -s "[0-9]"

# Kombiniert mit grep: letzte Fehlerzeile im Log
tac /var/log/syslog | grep -m1 "error"

# Skript rückwärts "lesen" (für Debugging)
tac skript.sh
```

Praktischer Nutzen

- Logs von hinten lesen (neueste Einträge zuerst)
- In Kombination mit head die letzten N Einträge effizient filtern
- Alternative zu tail, wenn man mehr Kontrolle braucht

basename

Grundprinzip

`basename` entfernt den Verzeichnisteil aus einem Dateipfad und gibt nur den **Dateinamen** zurück. Optional kann auch eine Dateiendung entfernt werden.

Grundsyntax:

```
basename PFAD [SUFFIX]
basename OPTION... PFAD... [SUFFIX]
```

Alle Optionen

Option	Langform	Bedeutung
`-a`	`--multiple`	Mehrere Pfade auf einmal verarbeiten
`-s SUFFIX`	`--suffix=SUFFIX`	Suffix vom Ergebnis entfernen
`--help`		Hilfe anzeigen
`--version`		Versionsnummer anzeigen

Beispiele

```
# Nur Dateiname aus Pfad
basename /home/user/dokumente/bericht.pdf
# → bericht.pdf

# Nur Name ohne Endung (Suffix angeben)
basename /home/user/dokumente/bericht.pdf .pdf
# → bericht

basename /usr/bin/python3
# → python3

# In Skript: aktuellen Skriptnamen
SKRIPTNAME=$(basename "$0")
echo "Dieses Skript heißt: $SKRIPTNAME"

# Dateiendung dynamisch entfernen
DATEI="/pfad/zur/datei.tar.gz"
basename "$DATEI" .tar.gz
# → datei

# Mehrere Pfade gleichzeitig (-a)
basename -a /etc/hosts /etc/passwd /etc/shadow
# → hosts
#    passwd
#    shadow

# Mehrere Pfade   Suffix entfernen (-a   -s)
basename -a -s .txt /daten/a.txt /daten/b.txt /daten/c.txt
# → a
#    b
#    c

# In Schleife: alle .sh-Dateien ohne Endung
for f in /skripte/*.sh; do
    name=$(basename "$f" .sh)
    echo "Verarbeite: $name"
done

# Gegenstück: dirname gibt das Verzeichnis zurück
dirname /home/user/dokumente/bericht.pdf
# → /home/user/dokumente

# Kombination basename   dirname
DATEI="/home/user/daten/report.csv"
DIR=$(dirname "$DATEI")
NAME=$(basename "$DATEI" .csv)
echo "Verzeichnis: $DIR"
echo "Dateiname:   $NAME"
# → Verzeichnis: /home/user/daten
# → Dateiname:   report
```

Häufiger Einsatz in Skripten

```
#!/bin/bash
# Skript, das sich selbst benennt und Locks verwendet

PROG=$(basename "$0")
LOCKFILE="/tmp/${PROG}.lock"

if [ -f "$LOCKFILE" ]; then
    echo "$PROG läuft bereits!" >&2
    exit 1
fi

touch "$LOCKFILE"
trap "rm -f $LOCKFILE" EXIT

echo "$PROG wird ausgeführt..."
```

Verzeichnisstruktur (FHS)

Filesystem Hierarchy Standard

```
/                   Wurzelverzeichnis (root)
├── /bin            Essentielle Programme (ls, bash, echo, cp, mv,
rm)
├── /sbin           System-Programme (für root: fdisk, mkfs, init)
├── /boot           Bootloader, Kernel (vmlinuz, initramfs, grub/)
├── /dev            Gerätedateien (sda, tty, null, zero)
├── /etc            Globale Konfigurationsdateien
│   ├── fstab       Dateisystem-Mountpoints
│   ├── passwd      Benutzerdatenbank
│   ├── shadow      Passwortdatenbank
│   ├── group       Gruppendatenbank
│   ├── hostname    Rechnername
│   ├── hosts       DNS-Einträge lokal
│   └── sudoers     Sudo-Regeln
├── /home           Benutzer-Homeverzeichnisse (/home/username)
├── /lib            Shared Libraries, Kernelmodule
├── /media          Automatische Mountpoints (USB, CD)
├── /mnt            Manuelle Mountpoints
├── /opt            Optionale Software
├── /proc           Virtuelles Dateisystem: Prozesse + Kernel-Info
│   ├── cpuinfo     CPU-Informationen
│   ├── meminfo     Speicherinformationen
│   └── [PID]/      Verzeichnis pro Prozess
├── /root           Home-Verzeichnis von root
├── /run            Runtime-Daten seit letztem Boot
├── /srv            Daten für Serverdienste
├── /sys            Virtuelles Dateisystem: Hardware + Kernel
├── /tmp            Temporäre Dateien (nach Neustart gelöscht)
├── /usr            Bibliotheken, Systemtools, installierte
Programme
│   ├── /usr/bin    Benutzer-Programme
│   ├── /usr/sbin   System-Programme
│   └── /usr/lib    Bibliotheken
└── /var            Variable Daten
    ├── /var/log    Logdateien (syslog, dmesg, auth.log)
    ├── /var/spool  Warteschlangen (Druckjobs, Cron, Mail)
    └── /var/tmp    Persistente temporäre Dateien
```

Wichtige Merkmale:

- **Keine Laufwerksbuchstaben** – stattdessen **Mountpoints**
- `/proc` und `/sys` sind **virtuelle Dateisysteme** → belegen keinen Speicherplatz
- `/proc` ändert sich ständig → **keine Backups davon erstellen!**
- Logdateien: `syslog` = allgemein, `dmesg` = Kernel-Log
- Ansehen: `cat /var/log/syslog` oder `less /var/log/syslog`

Zusammenfassung: Die /etc/sudoers Datei

- Zentrale Konfigurationsdatei für sudo
- Legt fest welche Benutzer welche Berechtigungen auf welchen Hosts haben
- Darf nicht mit normalem Texteditor geöffnet werden
    - Immer visudo: Prüft Syntax auf Fehler

Struktur

- Festes Muster: User Host=(RunAsUser:RunAsGroup) Commands
  - User: Der Benutzer oder die Gruppe (Gruppen werden mit % gekennzeichnet), der die Berechtigung erhält.
  - Host: Der Hostname, auf dem die Regel gilt (meist ALL für alle Rechner).
  - RunAsUser: Der Benutzer, als der der Befehl ausgeführt werden soll (meist root oder ALL).
  - RunAsGroup: Die Gruppe, als die der Befehl ausgeführt werden soll (meist ALL).
  - Commands: Die Liste der erlaubten Befehle (absolute Pfade verwenden!).
    -

Praktische Beispiele

A. Vollzugriff (Administrator-Status)

Erlaubt einem Benutzer alle Befehle mit Root-Rechten (erfordert Passwort).

```
# Benutzer 'max' darf alles
max ALL=(ALL:ALL) ALL

# Gruppe 'admin' darf alles (beachte das %)
%admin ALL=(ALL:ALL) ALL
```

B. Spezifische Befehle erlauben

Einschränkung auf nur einen einzigen Befehl (sehr wichtig für Sicherheit!).

```
# 'devuser' darf nur den Apache-Webserver neu starten
devuser ALL=(ALL) /usr/bin/systemctl restart apache2
```

C. Ausführung ohne Passwort (NOPASSWD)

Nützlich für Skripte oder Automatisierungen, aber ein Sicherheitsrisiko.

```
# 'automation-user' darf Updates machen, ohne nach einem Passwort gefragt zu werden
automation-user ALL=(ALL) NOPASSWD: /usr/bin/apt-get update, /usr/bin/apt-get upgrade
```

D. Verwendung von Pfaden und Wildcards

```
# Erlaubt das Ausführen von Backup-Skripten im Verzeichnis /opt/scripts/
backup-user ALL=(ALL) /opt/scripts/*.sh
```

Best Practices & Sicherheit

Principle of Least Privilege (Prinzip der geringsten Berechtigung): Erteile nur so viele Rechte, wie absolut notwendig sind. Vermeide ALL=(ALL) ALL für normale Benutzer.

Absolute Pfade nutzen: Gib Befehle immer mit ihrem vollständigen Pfad an (z. B. /usr/bin/apt statt nur apt), um "Path Injection" Angriffe zu verhindern.

Modularität nutzen (/etc/sudoers.d/): Anstatt die Hauptdatei /etc/sudoers zu verändern, erstelle separate Dateien im Verzeichnis /etc/sudoers.d/. Das ist sauberer und weniger fehleranfällig bei Systemupdates.

Beispiel: sudo visudo -f /etc/sudoers.d/mein-projekt

Gruppen statt Einzeluser: Nutze Gruppen (%gruppe), um die Verwaltung einfacher zu machen. Wenn ein neuer Admin dazukommt, musst du nur die Gruppe ändern, nicht die sudoers-Datei.

Kurzübersicht der Platzhalter

Platzhalter	**Bedeutung**
ALL	Gilt für alle Benutzer, alle Hosts oder alle Befehle.
%group	Bezieht sich auf eine Benutzergruppe (z. B. %sudo).
(ALL:ALL)	Erlaubt das Ausführen als jeder beliebige User und jede beliebige Gruppe.
NOPASSWD:	Erfordert keine Passworteingabe für die nachfolgenden Befehle.

Prinzipien

Das Letzte gewinnt. (The Last Match Wins)

Die sudoers-Datei wird von oben nach unten (sequenziell) eingelesen. Wenn eine Regel auf einen Benutzer zutrifft, werden die Berechtigungen geladen. Wenn eine weitere Regel weiter unten in der Datei ebenfalls auf denselben Benutzer zutrifft, wird diese neue Regel auf die bisherigen Regeln aufgeschlagen oder sie überschreibt spezifische Teile der vorherigen Regeln.

Hier ist die detaillierte Aufschlüsselung, wie sich das in verschiedenen Szenarien verhält:

1. Das Prinzip der Addition (Kumulativ)

Wenn zwei Regeln unterschiedliche Befehle erlauben, werden sie einfach kombiniert. Der Benutzer erhält die Summe aller erlaubten Befehle.

Beispiel:

```
# Regel 1 (oben)
max ALL=(ALL) /usr/bin/apt-get

# Regel 2 (weiter unten)
max ALL=(ALL) /usr/bin/systemctl
```

Ergebnis: max darf sowohl apt-get als auch systemctl ausführen. Es gibt hier keinen "Verlust", sondern eine Erweiterung.

2. Das Prinzip der Überschreibung (Override)

Ein Konflikt entsteht wirklich erst, wenn sich die Parameter (wie NOPASSWD oder der RunAsUser) für den gleichen Befehl widersprechen. In diesem Fall überschreibt die letzte gefundene Regel die vorherige.

Beispiel (Das Passwort-Dilemma):

```
# Regel 1: Erlaubt alles ohne Passwort
max ALL=(ALL) NOPASSWD: ALL

# Regel 2: Erlaubt spezifisch apt-get, aber ERZWINGT ein Passwort
max ALL=(ALL) /usr/bin/apt-get
```

Ergebnis: Wenn max versucht, apt-get auszuführen, wird er nach einem Passwort gefragt. Obwohl Regel 1 "alles ohne Passwort" erlaubt hat, hat die spätere Regel 2 die Anweisung für apt-arg mit der Passwort-Pflicht überschrieben.

3. Konflikte durch Gruppen vs. Benutzer (Spezifisch schlägt Generell)

Oft ist ein Benutzer Mitglied einer Gruppe (z. B. %admin). Hier greifen zwei Regeln gleichzeitig: die Gruppenregel und die Benutzerregel.

Beispiel:

```
# Regel 1: Die Gruppe 'admin' darf alles, aber MIT Passwort
%admin ALL=(ALL) ALL

# Regel 2: Der Benutzer 'max' (ist Mitglied von admin) darf alles OHNE Passwort
max ALL=(ALL) NOPASSWD: ALL
```

Ergebnis: Da die Regel für max spezifischer ist und nach der Gruppenregel kommt, gewinnt die NOPASSWD-Einstellung für ihn. Er kann Befehle ohne Passwort ausführen, während andere Mitglieder der Gruppe admin weiterhin nach einem Passwort gefragt werden.

4. Besonderheit: /etc/sudoers.d/

Das Verzeichnis /etc/sudoers.d/ wird am Ende der Haupt-sudoers-Datei per #includedir eingebunden.

Das bedeutet: Alles, was in den Dateien im sudoers.d-Ordner steht, hat Vorrang vor den Einträgen in der Haupt-sudoers-Datei, weil diese Dateien technisch gesehen "ganz unten" in der Konfigurationskette gelesen werden.

Pro-Tipp für die Praxis: Wenn du eine Berechtigung für einen Benutzer in der Hauptdatei einschränken willst, musst du diese Einschränkung in einer Datei innerhalb von /etc/sudoers.d/ platzieren, damit sie die (eventuell mächtigeren) Regeln aus der Hauptdatei überschreibt.

Zusammenfassung

Szenario	Verhalten
Unterschiedliche Befehle	Befehle werden addiert (User bekommt mehr Rechte).
Gleiche Befehle, andere Parameter	Die letzte Regel überschreibt die vorherige.
Gruppe vs. Einzelner User	Die letzte zutreffende Regel (meist die spezifischere) gewinnt.
Hauptdatei vs. sudoers.d/	Die Dateien in sudoers.d/ gewinnen (da sie zuletzt gelesen werden).

Dateien suchen: find, grep, sed

find – Dateien und Verzeichnisse suchen

find START_PFAD OPTIONEN AUSDRUCK

Wichtige Optionen:

Option	Beschreibung	Beispiel
`-name`	Nach Name suchen	`find / -name "*.txt"`
`-iname`	Name (case-insensitive)	`find . -iname "readme*"`
`-type f`	Nur Dateien	`find . -type f`
`-type d`	Nur Verzeichnisse	`find . -type d`
`-type l`	Nur Symlinks	
`-user`	Nach Besitzer	`find / -user root`
`-group`	Nach Gruppe	`find / -group admin`
`-size +2G`	Größer als 2 GB	`find / -size +200M`
`-size 0` / `-empty`	Leere Dateien	`find ~ -empty`
`-mtime -7`	Geändert in letzten 7 Tagen	
`-mtime +30`	Älter als 30 Tage	
`-maxdepth N`	Max. Suchtiefe	`find . -maxdepth 2 -name "*.log"`
`-not`	Negation	`find . -not -name "*.bak"`
`-exec`	Befehl auf Ergebnis ausführen	`find . -name "*.tmp" -exec rm {} \;`
`-delete`	Gefundene Dateien löschen	`find . -name "*.bak" -delete`

Beispiele:

```
# Alle .txt Dateien im aktuellen Verzeichnis
find . -name "*.txt" -type f

# Dateien mit Inhalt durchsuchen
find . -type f -exec grep "suchbegriff" '{}'
-print

# Dateien älter als 30 Tage löschen
find /home/user/logs/ -type f -mtime 30 -exec rm {}

# Leere Dateien im Home finden
find ~ -empty
```

grep – In Textdateien suchen

grep [OPTIONEN] 'MUSTER' DATEI

Wichtige Optionen:

Option	Beschreibung
`-i`	Case-insensitive
`-n`	Zeilennummern anzeigen
`-v`	Nur Zeilen die NICHT passen
`-c`	Anzahl der Treffer
`-r`	Rekursiv in Verzeichnissen
`-A1`	1 Zeile nach dem Treffer anzeigen
`-B2`	2 Zeilen vor dem Treffer anzeigen

Reguläre Ausdrücke für grep:

Ausdruck	Bedeutung
`.`	Ein beliebiges Zeichen
`*`	Vorheriges Zeichen beliebig oft
`.*`	Beliebig viele beliebige Zeichen
`^text`	Zeilenanfang
`text$`	Zeilenende
`[abc]`	Ein Zeichen aus der Menge
`[^abc]`	Kein Zeichen aus der Menge
`[a-z]`	Zeichenbereich

Beispiele:

```
# Zeilen die mit "König" beginnen
grep "^König" datei.txt

# Zeilen die mit "zz" enden
grep "zz$" datei.txt

# Case-insensitive Suche
grep -i "suchbegriff" datei.txt

# In /etc/passwd nach Bash-Benutzern suchen
grep "/bin/bash" /etc/passwd
```

sed – Stream Editor

Grundprinzip

`sed` (Stream Editor) bearbeitet Text zeilenweise. Es liest stdin oder Dateien, wendet Transformationsregeln an und schreibt das Ergebnis nach stdout.

Grundsyntax:

```
sed 'BEFEHL' datei
sed -e 'BEFEHL1' -e 'BEFEHL2' datei
```

Alle Kommandozeilen-Optionen

Option	Langform	Bedeutung
`-n`	`--quiet` / `--silent`	Keine automatische Ausgabe; `p`-Befehl nötig
`-e SKRIPT`	`--expression=SKRIPT`	Skript direkt in der Kommandozeile
`-f DATEI`	`--file=DATEI`	sed-Skript aus Datei lesen
`-i[SUFFIX]`	`--in-place[=SUFFIX]`	Datei direkt bearbeiten (optional Backup)
`-r` / `-E`	`--regexp-extended`	Erweiterte reguläre Ausdrücke (ERE)
`-s`	`--separate`	Dateien separat behandeln (NR wird zurückgesetzt)
`-z`	`--null-data`	NUL (`\0`) als Zeilentrenner statt `\n`
`--sandbox`		Sichere Ausführung (kein `e`/`r`/`w`)
`--posix`		POSIX-Striktmodus

sed-Befehle (innerhalb des Skripts)

Befehl	**Bedeutung**
`s/alt/neu/`	Substituieren (ersetzen), erste Vorkommen
`s/alt/neu/g`	Alle Vorkommen ersetzen
`s/alt/neu/i`	Groß-/Kleinschreibung ignorieren
`s/alt/neu/2`	Nur das 2. Vorkommen ersetzen
`s/alt/neu/gp`	Ersetzen und Zeile ausgeben (mit `-n`)
`d`	Zeile löschen
`p`	Zeile ausgeben (extra)
`q`	Verarbeitung beenden (quit)
`Q`	Sofort beenden ohne Ausgabe
`i\TEXT`	TEXT vor der Zeile einfügen
`a\TEXT`	TEXT nach der Zeile anhängen
`c\TEXT`	Zeile durch TEXT ersetzen
`y/abc/xyz/`	Zeichen 1:1 austauschen (transliterate)
`=`	Zeilennummer ausgeben
`n`	Nächste Zeile lesen
`N`	Nächste Zeile an Pattern Space anhängen
`r DATEI`	Inhalt von DATEI nach der Zeile einfügen
`w DATEI`	Matching-Zeilen in DATEI schreiben

Adressen (Adressierung von Zeilen)

```
sed '3 d'         # Nur Zeile 3 löschen
sed '3,7 d'       # Zeilen 3-7 löschen
sed '/muster/ d' # Zeilen löschen, die Muster enthalten
sed '3,/ende/ d' # Ab Zeile 3 bis zur Zeile mit "ende"
sed '$ d'         # Letzte Zeile löschen
sed '1~2 d'       # Jede 2. Zeile löschen (ab Zeile 1): 1,3,5,...
sed '0~2 d'       # Gerade Zeilen löschen: 2,4,6,...
```

Reguläre Ausdrücke in sed

Ausdruck	Bedeutung
`.`	Beliebiges Zeichen
`*`	0 oder mehr des vorherigen
`+`	1 oder mehr (ERE mit `-E`)
`?`	0 oder 1 (ERE mit `-E`)
`^`	Zeilenanfang
`$`	Zeilenende
`[abc]`	Zeichenklasse
`[^abc]`	Negierte Zeichenklasse
`\(…\)`	Gruppe (BRE)
`(…)`	Gruppe (ERE mit `-E`)
`\1`	Rückreferenz auf Gruppe 1
`&`	Gesamter gefundener Text im Ersatz

Beispiele

```
# Einfaches Ersetzen
sed 's/alt/neu/' datei.txt

# Alle Vorkommen ersetzen
sed 's/alt/neu/g' datei.txt

# Groß-/Kleinschreibung ignorieren
sed 's/fehler/FEHLER/gi' datei.txt

# Datei direkt bearbeiten (in-place)
sed -i 's/alt/neu/g' datei.txt

# Backup erstellen und in-place bearbeiten
sed -i.bak 's/alt/neu/g' datei.txt
# (erstellt datei.txt.bak als Sicherung)

# Zeilen löschen, die "kommentar" enthalten
sed '/kommentar/d' datei.txt

# Leerzeilen löschen
sed '/^$/d' datei.txt

# Leerzeilen und Zeilen mit nur Leerzeichen löschen
sed '/^s*$/d' datei.txt

# Kommentarzeilen (# am Anfang) entfernen
sed '/^#/d' datei.txt
sed '/^[[:space:]]*#/d' datei.txt
```

```
# Nur Zeilen ausgeben, die "Muster" enthalten (-n   p)
sed -n '/muster/p' datei.txt

# Leerzeichen am Zeilenende entfernen
sed 's/[[:space:]]*$//' datei.txt

# Führende Leerzeichen entfernen
sed 's/^[[:space:]]*//' datei.txt

# Beides (trim)
sed 's/^[[:space:]]*//; s/[[:space:]]*$//' datei.txt

# Zeilen 5-10 ausgeben
sed -n '5,10p' datei.txt

# Erste Zeile löschen (Header entfernen)
sed '1d' datei.txt

# Letzte Zeile löschen
sed '$d' datei.txt

# Zeilenumbruch nach ";" einfügen (Erweiterter Regex)
sed -E 's/;/;n/g' datei.txt

# Rückreferenz: Wort in Anführungszeichen einschließen
sed 's/(Wort)/"1"/' datei.txt
sed -E 's/(Wort)/"1"/' datei.txt # ERE-Variante

# Text nach Zeile 3 einfügen
sed '3aDies ist eine neue Zeile' datei.txt

# Text vor Zeile 3 einfügen
sed '3iDies kommt davor' datei.txt

# Zeile 3 ersetzen
sed '3cDiese Zeile ersetzt die alte' datei.txt

# Mehrere Befehle (mit -e oder Semikolon)
sed -e 's/foo/bar/g' -e 's/baz/qux/g' datei.txt
sed 's/foo/bar/g; s/baz/qux/g' datei.txt

# Skript aus Datei
sed -f mein_skript.sed datei.txt

# Nur erste Zeile ausgeben (wie head -1)
sed -n '1p' datei.txt

# Zeilenanzahl begrenzen (wie head -5)
sed '5q' datei.txt

# Zeichenersatz (y-Befehl): Kleinbuchstaben → Großbuchstaben
sed 'y/abcdefghijklmnopqrstuvwxyz/ABCDEFGHIJKLMNOPQRSTUVWXYZ/'
datei.txt

# Zeilennummern anzeigen (= gibt Nummer aus, p gibt Zeile aus)
sed '=' datei.txt | sed 'N; s/n/t/'

# HTML-Tags entfernen
sed 's/<[^>]*>//g' seite.html

# DOS-Zeilenenden (rn) in Unix (n) umwandeln
  sed 's/r//' datei.txt
```

cut – Spalten extrahieren

```
# 5. Feld aus /etc/passwd (Delimiter: Doppelpunkt)
cut -d: -f5 /etc/passwd

# 3. Spalte einer CSV-Datei
cut -d, -f3 datei.csv
```

awk – Textverarbeitung

Grundprinzip

`awk` ist eine vollständige Programmiersprache für Textverarbeitung. Es liest Dateien oder Streams zeilenweise, teilt jede Zeile in **Felder** auf und wendet ein Programm darauf an.

Grundsyntax:

```
awk 'MUSTER { AKTION }' datei
```

Felder und Variablen

Variable	Bedeutung
`$0`	Die gesamte aktuelle Zeile
`$1`	Erstes Feld
`$2`	Zweites Feld
`$NF`	Letztes Feld (Number of Fields)
`$(NF-1)`	Vorletztes Feld
`NR`	Aktuelle Zeilennummer (Number of Records)
`NF`	Anzahl der Felder in der aktuellen Zeile
`FS`	Feldtrenner (Field Separator, Standard: Leerzeichen/Tab)
`OFS`	Ausgabe-Feldtrenner (Output Field Separator)
`RS`	Datensatztrenner (Record Separator, Standard: `\n`)
`ORS`	Ausgabe-Datensatztrenner
`FILENAME`	Name der aktuellen Datei
`FNR`	Zeilennummer in der aktuellen Datei (bei mehreren Dateien)
`ARGC`	Anzahl der Argumente
`ARGV`	Array der Argumente

Kommandozeilen-Optionen

Option	Bedeutung
`-F TRENN`	Feldtrenner setzen (z.B. `-F:` für Doppelpunkt)
`-v VAR=WERT`	Variable vor dem Start setzen
`-f DATEI`	awk-Programm aus Datei lesen
`-W compat`	POSIX-kompatiblen Modus aktivieren (gawk)
`--posix`	Streng POSIX-konform (gawk)
`--re-interval`	Intervallausdrücke in Regex aktivieren (gawk)
`--sandbox`	Sicherheitsmodus (kein system(), getline etc.) (gawk)

Besondere Blöcke

```
BEGIN { ... }    # Wird einmal VOR dem Einlesen ausgeführt
END { ... }      # Wird einmal NACH dem letzten Datensatz ausgeführt
/Muster/ { ... } # Wird für jede Zeile ausgeführt, die Muster
enthält
```

Vergleichsoperatoren

Operator	Bedeutung
`==`	Gleich
`!=`	Ungleich
`<`, `>`, `<=`, `>=`	Vergleich
`~`	Regex-Match
`!~`	Kein Regex-Match

Arithmetik und Strings

```
# Arithmetik
$3 + $4         # Addition
$3 * 1.19       # Multiplikation
int($1 / 2)     # Ganzzahldivision

# String-Funktionen
length($0)              # Länge der Zeile
substr($1, 2, 4)        # Teilstring ab Position 2, Länge 4
split($1, arr, ":")     # Splitten in Array
gsub(/alt/, "neu")      # Globale Ersetzung in $0
sub(/alt/, "neu")       # Erste Ersetzung in $0
toupper($1)             # Großbuchstaben
tolower($1)             # Kleinbuchstaben
index($1, "suche")      # Position des Teilstrings
match($0, /regex/)      # Regex-Match, setzt RSTART/RLENGTH
sprintf("%.2f", $3)     # Formatierte Ausgabe
```

Beispiele

```
    # Alle Zeilen ausgeben (wie cat)
    awk '{ print }' datei.txt
    awk '{ print $0 }' datei.txt

    # Nur das erste Feld ausgeben
    awk '{ print $1 }' datei.txt

    # Spalten 1 und 3, durch Tab getrennt
    awk '{ print $1, $3 }' datei.txt

    # Mit Doppelpunkt als Trennzeichen (/etc/passwd)
    awk -F: '{ print $1 }' /etc/passwd

    # Benutzernamen und Shell ausgeben
    awk -F: '{ print $1, $7 }' /etc/passwd

    # Nur Zeilen mit mehr als 3 Feldern
    awk 'NF > 3' datei.txt

    # Zeilen die "error" enthalten
    awk '/error/ { print }' /var/log/syslog

    # Zeilen die NICHT "debug" enthalten
    awk '!/debug/' /var/log/syslog

    # Zeilennummer mit ausgeben
    awk '{ print NR": "$0 }' datei.txt

    # Nur Zeile 5 bis 10
    awk 'NR>=5 && NR<=10' datei.txt

    # Summe der dritten Spalte
    awk '{ sum += $3 } END { print "Summe:", sum }' zahlen.txt

    # Durchschnitt
    awk '{ sum += $3 } END { print "Avg:", sum/NR }' zahlen.txt

    # Anzahl Zeilen zählen (wie wc -l)
    awk 'END { print NR }' datei.txt

    # Duplikate entfernen (wie uniq, aber nicht sortiert)
    awk '!seen[$0]++' datei.txt

    # Leerzeilen entfernen
    awk 'NF > 0' datei.txt
    awk '/^./' datei.txt

    # Felder vertauschen (1. und 2. Spalte tauschen)
    awk '{ print $2, $1 }' datei.txt

    # Ausgabe-Feldtrenner setzen (Komma)
    awk 'BEGIN{OFS=","} { print $1,$2,$3 }' datei.txt

    # Variablen von außen übergeben
    awk -v grenze=100 '$3 > grenze { print }' datei.txt

    # if/else in awk
    awk '{ if ($3 > 50) print "groß:", $1; else print "klein:", $1
}' datei.txt

    # for-Schleife
    awk '{ for(i=1; i<=NF; i++) print $i }' datei.txt
```

```
    # Arrays verwenden
    awk '{ count[$1]++ } END { for(k in count) print k, count[k] }'
log.txt

    # Nur die letzte Zeile
    awk 'END { print }' datei.txt

    # Mehrzeiliger awk-Code aus Datei ausführen
    awk -f programm.awk datei.txt

    # CSV-Datei auswerten (Komma als Trenner, Anführungszeichen
ignorieren)
    awk -F'","' '{ print $2 }' daten.csv

    # Zwischen zwei Mustern ausgeben (inkl.)
    awk '/START/,/STOP/' datei.txt

    # BEGIN und END
    awk 'BEGIN { print "=== Start ===" } { print } END { print "===
Ende ===" }' datei.txt

    # Prozesse nach CPU-Nutzung filtern (ps aux)
    ps aux | awk '$3 > 1.0 { print $1, $2, $3, $11 }'

    # IP-Adressen aus Logdatei extrahieren
    awk '{ match($0, /[0-9]+\.[0-9]+\.[0-9]+\.[0-9]+/, arr); print
arr[0] }' access.log

    # /etc/passwd – UIDs über 1000 (normale User)
    awk -F: '$3 >= 1000 { print $1, $3 }' /etc/passwd

    # Größe einer Datei (du-Ausgabe aufsummieren)
    du -sh * | awk '{ sum += $1 } END { print sum }'
```

apt (High-Level – empfohlen!)

Befehl	Beschreibung
`sudo apt update`	Paketlisten aktualisieren
`sudo apt upgrade`	Installierte Pakete aktualisieren
`sudo apt install paket`	Paket installieren
`sudo apt remove paket`	Paket deinstallieren (Konfig bleibt)
`sudo apt purge paket`	Paket + Konfig entfernen
`apt search begriff`	Nach Paketen suchen
`apt list --installed`	Installierte Pakete anzeigen
`sudo apt autoremove`	Nicht benötigte Pakete entfernen
`apt source paket`	Quellcode herunterladen
`apt clean`	Apt cache leeren

Paketquellen: /etc/apt/sources.list

```
deb http://server.example.com/debian distribution component1 component2
```

```
deb-src http://server.example.com/debian distribution component1 component2
```

- `deb` = vorcompilierte Binary-Pakete
- `deb-src` = Source-Pakete
- Weitere Quellen in: `/etc/apt/sources.list.d/*.list`

Benutzer- und Gruppenverwaltung

Arten von Benutzern

Typ	Beschreibung	UID
Root	Superuser, uneingeschränkter Zugriff	0
Normale Benutzer	Begrenzte Rechte	> 1000
Systembenutzer	Für Dienste (z.B. `www-data`)	1-999

Wichtige Konfigurationsdateien

`/etc/passwd` – Benutzerdatenbank

```
Username:Password:UID:GID:Info:Home:Shell
```

- `Password` = `x` bedeutet: Passwort steht in `/etc/shadow`
- Beispiel: `bob:x:1001:1001:Bob Miller:/home/bob:/bin/bash`

`/etc/shadow` – Passwortdatenbank

```
Username:Password:DOC:MinD:MaxD:Warn:Exp:Dis
```

- `DOC` = Tag der letzten Passwortänderung (ab 1.1.1970)
- Editieren mit: `vipw -s`

`/etc/group` – Gruppendatenbank

```
Gruppenname:Passwort:GID:Mitgliederliste
```

`/etc/skel` – Vorlage für neue Benutzer

- Dateien aus diesem Verzeichnis werden bei `useradd -m` ins neue Home kopiert

Befehle zur Gruppenverwaltung

Befehl	**Beschreibung**
`groupadd gruppenname`	Neue Gruppe anlegen
`groupadd -g 1007 verkauf`	Gruppe mit bestimmter GID
`groupdel gruppenname`	Gruppe löschen
`groupmod -n neuer_name alter_name`	Gruppe umbenennen
`gpasswd -a benutzer gruppe`	Mitglied hinzufügen
`gpasswd -d benutzer gruppe`	Mitglied entfernen
`newgrp gruppenname`	Primäre Gruppe wechseln

Befehle zur Benutzerverwaltung

Befehl	Beschreibung
`useradd benutzername`	Neuen Benutzer anlegen
`useradd -m benutzername`	Mit Home-Verzeichnis anlegen
`useradd -m -u 502 -c "Info" name`	Mit UID und Kommentar
`userdel benutzername`	Benutzer löschen
`userdel -r benutzername`	Benutzer + Home löschen
`usermod -aG gruppe benutzer`	Benutzer zur Gruppe hinzufügen
`passwd benutzername`	Passwort setzen/ändern
`passwd -l benutzername`	Account sperren
`passwd -u benutzername`	Account freigeben
`id benutzername`	UID, GID und Gruppen anzeigen
`groups benutzername`	Gruppenmitgliedschaften anzeigen
`su benutzername`	Benutzer wechseln
`su -`	Zu root wechseln mit Umgebungswechsel

Zugriffsrechte und Dateiberechtigungen

Aufbau der Berechtigungen

```
-rwxr-xr--  1  besitzer  gruppe  größe  datum  dateiname
|┼┼┼┼┼
| |   |   └── Andere (Others): r--
| |   └────── Gruppe (Group):  r-x
| └────────── Besitzer (User): rwx
└──────────── Dateityp: - = Datei, d = Verzeichnis
```

Bedeutung der Rechte

Recht	Datei	Verzeichnis
r (read=4)	Dateiinhalt lesen	Verzeichnisinhalt auflisten
w (write=2)	Dateiinhalt ändern	Dateien erstellen/löschen/umbenennen
x (execute=1)	Programm/Skript ausführen	Verzeichnis betreten (cd)

chmod – Rechte ändern

Symbolische Methode:

```
chmod u+x datei.sh          # User: Ausführen hinzufügen
chmod g-w datei.txt          # Gruppe: Schreiben entfernen
chmod o=r datei.txt          # Andere: Nur Lesen
chmod a+x skript.sh          # Alle: Ausführen hinzufügen
chmod u=rwx,g=rx,o= datei    # Explizit setzen
```

Oktalnotation:

```
chmod 755 datei     # rwxr-xr-x
chmod 644 datei     # rw-r--r--
chmod 700 datei     # rwx------
chmod 000 datei     # ---------
```

Häufige Berechtigungssätze:

Oktal	Symbolisch	Verwendung
755	rwxr-xr-x	Programme, Verzeichnisse
644	rw-r--r--	Normale Dateien
700	rwx------	Private Verzeichnisse, .ssh/
600	rw-------	Private Dateien, SSH-Keys
777	rwxrwxrwx	Unsicher! Alle Rechte

chown / chgrp – Besitzer/Gruppe ändern

```
chown benutzer:gruppe datei     # Besitzer und Gruppe ändern
chown benutzer datei            # Nur Besitzer ändern
chgrp gruppe datei              # Nur Gruppe ändern
```

Spezialberechtigungen

Spezialbit	Oktal	Setzen	Beschreibung
SetUID	4000	chmod u+s datei	Programm läuft mit Rechten des Dateieigentümers (z.B. /usr/bin/passwd)
SetGID	2000	chmod g+s verz	Neue Dateien erben Gruppe des Verzeichnisses
Sticky Bit	1000	chmod +t verz	Nur Eigentümer darf eigene Dateien löschen (z.B. /tmp)

SetUID – Detailbeschreibung

- Gesetzt auf eine **ausführbare Datei**: Programm läuft immer mit der UID des **Datei-Eigentümers**, egal welcher Benutzer es startet
- Klassisches Beispiel: **/usr/bin/passwd** → muss **/etc/shadow** (Besitzer: root) schreiben, darf aber von jedem Benutzer aufgerufen werden
- Auf **Verzeichnisse**: In den meisten Linux-Distros **keine Wirkung**
- **Sicherheitsrisiko:** SetUID-root-Programme sind ein beliebtes Angriffsziel (Privilege Escalation)

```
chmod u+s /usr/bin/meinprog     # SetUID setzen
chmod 4755 /usr/bin/meinprog    # Oktal: 4 + 755
```

SetGID – Detailbeschreibung

- Gesetzt auf eine **ausführbare Datei**: Programm läuft mit der GID des **Datei-Eigentümers**
- Gesetzt auf ein **Verzeichnis**: Alle neu erstellten Dateien und Unterverzeichnisse erben automatisch die **Gruppe des Verzeichnisses** (sehr nützlich für Team-Ordner!)

```
chmod g+s /srv/team/            # SetGID auf Verzeichnis
chmod 2755 /srv/team/           # Oktal: 2 + 755
```

Sticky Bit – Detailbeschreibung

- Gesetzt auf ein **Verzeichnis**: Jeder kann Dateien erstellen, aber nur der **Eigentümer der Datei** (oder root) darf diese löschen/umbenennen
- Wichtigstes Beispiel: **`/tmp`** – alle Benutzer können dort Dateien anlegen, aber niemand darf fremde Dateien löschen
- Auf **Dateien**: Historisch bedeutete es „Programm im Swap behalten" – heute in Linux **keine Funktion mehr**

```
chmod +t /tmp/gemeinsam/        # Sticky Bit setzen
chmod 1777 /tmp/                # Oktal: 1 + 777  ← Typisch für /tmp
```

Verifizieren:

```
ls -ld /tmp
# drwxrwxrwt  → das 't' am Ende = Sticky Bit gesetzt
stat /tmp
```

Vollständige Übersicht: Was `ls -l` bei Berechtigungen anzeigt

Das Berechtigungsfeld hat **10 Zeichen**: 1 Dateityp + 3×3 Rechte (user/group/others)

```
Position:  1   234   567   890
           |   ├─┤   ├─┤   ├─┤
Beispiel:  d   rwx   r-s   r-t
           |    |     |     |
           |    |     |     └── Others
           |    |     └──────── Group
           |    └────────────── User
           └─────────────────── Dateityp
```

Position 1: Dateityp

Zeichen	Dateityp
-	Reguläre Datei
d	Verzeichnis (directory)
l	Symbolischer Link (symlink)
b	Block-Gerätedatei (z.B. Festplatte)
c	Zeichen-Gerätedatei (z.B. Terminal)
p	Named Pipe (FIFO)
s	Socket

Positionen 2–4 (User), 5–7 (Group), 8–10 (Others): Basisrechte

Zeichen	Position	Bedeutung
r	1. Stelle	Lesen erlaubt
-	1. Stelle	Lesen verboten
w	2. Stelle	Schreiben erlaubt
-	2. Stelle	Schreiben verboten
x	3. Stelle	Ausführen erlaubt
-	3. Stelle	Ausführen verboten

Position 4 (User), 7 (Group), 10 (Others): Spezialbit-Überlagerung

Zeichen	Position	Bedeutung
s	User (Pos. 4)	SetUID gesetzt und x vorhanden
S	User (Pos. 4)	SetUID gesetzt, aber kein x (unüblich!)
s	Group (Pos. 7)	SetGID gesetzt und x vorhanden
S	Group (Pos. 7)	SetGID gesetzt, aber kein x
t	Others (Pos. 10)	Sticky Bit gesetzt und x vorhanden
T	Others (Pos. 10)	Sticky Bit gesetzt, aber kein x (selten)

Merkregel: *Kleinbuchstabe (**s**, **t**) = Spezialbit **UND x**. Großbuchstabe (**S**, **T**) = Spezialbit **OHNE x**.*

Alle möglichen Zeichen auf einen Blick

```
Dateityp: - d l b c p s

User:    r w x s S
Group:   r w x s S
Others:  r w x t T
         (- für jeweils "nicht gesetzt")
```

Praxisbeispiele mit `ls -l`

Ausgabe von `ls -l`	Bedeutung
`-rw-r--r--`	Normale Datei: User lesen+schreiben, Group+Others nur lesen
`-rwxr-xr-x`	Programm: User alle Rechte, Group+Others lesen+ausführen
`-rwx------`	Private Datei: nur Owner alle Rechte
`drwxr-xr-x`	Verzeichnis: Standard (755)
`drwxrwxrwt`	`/tmp`: Alle Rechte + Sticky Bit (1777)
`-rwsr-xr-x`	SetUID gesetzt (z.B. `/usr/bin/passwd`)
`drwxr-sr-x`	SetGID auf Verzeichnis (Gruppe wird vererbt)
`-rwSr--r--`	SetUID, aber kein execute-Bit → sinnlos/Fehler!
`lrwxrwxrwx`	Symbolischer Link (Rechte immer 777)

Oktalwerte der Spezialbits

Oktal	Bedeutung	Beispiel
4xxx	SetUID	`chmod 4755 datei` → `-rwsr-xr-x`
2xxx	SetGID	`chmod 2755 verz/` → `drwxr-sr-x`
1xxx	Sticky Bit	`chmod 1777 /tmp` → `drwxrwxrwt`
6xxx	SetUID + SetGID	`chmod 6755 datei`
7xxx	Alle drei	`chmod 7777 datei` (sehr selten)
0xxx	Keine Spezialbits	`chmod 0755 datei` (normales 755)

Spezialberechtigungen prüfen:

```
# Alle SetUID-Dateien auf dem System finden:
find / -perm -4000 -type f 2>/dev/null

# Alle SetGID-Dateien finden:
find / -perm -2000 -type f 2>/dev/null

# Alle Verzeichnisse mit Sticky Bit:
find / -perm -1000 -type d 2>/dev/null

# Detailansicht mit stat:
stat /tmp
stat /usr/bin/passwd
```

umask – Standard-Berechtigungen

umask (user file creation mask) legt fest, welche Rechte beim Erstellen neuer Dateien und Verzeichnisse **weggenommen** werden. Es ist eine **Maske**, keine direkte Rechtevergabe.

Berechnung

```
Neue Rechte = Maximalrechte  -  umask
```

	Datei	**Verzeichnis**
Maximalrechte	**666** (rw-rw-rw-)	**777** (rwxrwxrwx)
umask (Standard)	**022**	**022**
Ergebnis	**644** (rw-r--r--)	**755** (rwxr-xr-x)

Warum haben Dateien maximal 666?
*Linux vergibt **niemals** automatisch das execute-Bit (**x**) auf neue Dateien, da eine normale Datei kein Programm ist. Verzeichnisse brauchen **x** um betreten werden zu können, daher Maximal **777**.*

Wichtiger Hinweis zur Berechnung

Die Subtraktion ist **bitweise** (nicht arithmetisch):

Beispiel: **umask 027**

```
Datei:       666 (rw-rw-rw-)
umask:       027 (----w-rwx)  → wird weggenommen
Ergebnis:    640 (rw-r-----)
```

Verzeichnis:

```
Verzeichnis: 777 (rwxrwxrwx)
umask:       027 (----w-rwx)  → wird weggenommen
Ergebnis:    750 (rwxr-x---)
```

Alle wichtigen umask-Werte

umask	**Datei**	**Verzeichnis**	**Verwendung**
000	666 (rw-rw-rw-)	777 (rwxrwxrwx)	Alle Rechte (unsicher!)
002	664 (rw-rw-r--)	775 (rwxrwxr-x)	Gruppenarbeit
022	644 (rw-r--r--)	755 (rwxr-xr-x)	Standard für normale Benutzer
027	640 (rw-r-----)	750 (rwxr-x---)	Erhöhte Sicherheit (Other kein Zugriff)
077	600 (rw-------)	700 (rwx------)	Private Dateien (nur Eigentümer)
177	400 (r--------)	600 (rw-------)	Schreibgeschützte private Dateien

Befehle

```
umask                   # Aktuelle umask anzeigen (oktal)
umask -S                # Symbolische Anzeige (z.B. u=rwx,g=rx,o=rx)
umask 022               # umask für aktuelle Shell-Sitzung setzen
umask 027               # Strengere umask setzen
```

umask dauerhaft setzen

Datei	Gültig für
~/.bashrc	Normaler Benutzer (interaktive Shell)
~/.profile	Normaler Benutzer (Login-Shell)
/etc/profile	Alle Benutzer (System-weit)
/etc/login.defs	System-Standard (UMASK-Eintrag)

```
# In ~/.bashrc eintragen:
umask 022
```

Praxisbeispiel: Warum umask wichtig ist

```
# Ohne umask 022 (Standard):
touch neue_datei.txt    →  -rw-r--r-- (644)
mkdir neuer_ordner/     →  drwxr-xr-x (755)

# Mit umask 077 (privat):
umask 077
touch geheime_datei.txt →  -rw------- (600)
mkdir privater_ordner/  →  drwx------ (700)
```

Umgebungsvariablen

Wichtige Standard-Variablen

Variable	Beschreibung
`$HOME`	Home-Verzeichnis
`$PATH`	Suchpfade für Programme
`$USER`	Aktueller Benutzername
`$SHELL`	Aktuelle Shell
`$PWD`	Aktuelles Verzeichnis
`$?`	Exit-Code des letzten Befehls
`$$`	PID des aktuellen Prozesses

Variablen anzeigen/setzen

```
echo $HOME # Variable ausgeben
env        # Alle Umgebungsvariablen anzeigen
printenv   # Alternative
set        # Alle Variablen   Funktionen

# Variable setzen (nur lokal/aktuelle Shell)
myvar="Hallo"

# Variable exportieren (für Kindprozesse)
export myvar="Hallo"

# Variable löschen
unset myvar

# PATH erweitern
export PATH="/home/user/bin:$PATH"
```

Quoting-Regeln

Typ	Beschreibung	Beispiel
Doppelte Anführungszeichen `"`	Variablen werden aufgelöst	`echo "$HOME"` → `/home/user`
Einfache Anführungszeichen `'`	Alles wird wörtlich genommen	`echo '$HOME'` → `$HOME`
Backslash `\`	Nächstes Zeichen literal	`echo \$HOME` → `$HOME`

Prozessmanagement und Systemüberwachung

Prozess-Grundlagen

- Ein **Prozess** = ein Programm in Ausführung
- Jeder Prozess hat: **PID** (Prozessnummer), **PPID** (Elternprozess-ID), Benutzer
- **PID 1** = erster Prozess (`/sbin/init` bzw. `systemd`)

Prozesszustände

Status	Zeichen	Beschreibung
Running	`R`	Prozess läuft/kann Rechenzeit erhalten
Sleeping	`S`	Wartet auf Ereignis
Uninterruptible Sleep	`D`	Wartet, kann nicht gestoppt werden
Stopped	`T`	Angehalten, kann fortgesetzt werden
Zombie	`Z`	Beendet, Rückgabewert nicht abgeholt

Befehle zur Prozessverwaltung

Befehl	Beschreibung
`ps`	Prozesse des aktuellen Terminals
`ps aux`	Alle Prozesse anzeigen
`ps -ef`	Alle Prozesse mit PPID
`ps -l`	Langes Listing (mit Nice-Wert)
`pstree`	Baumstruktur
`pstree -p`	Baumstruktur mit PIDs
`top`	Dynamische Prozesstabelle
`htop`	Farbige Prozesstabelle
`kill PID`	Prozess beenden (SIGTERM)
`kill -9 PID`	Prozess erzwungen beenden (SIGKILL)
`kill %jobnr`	Job beenden
`killall programmname`	Alle Prozesse mit dem Namen beenden

Vorder- und Hintergrundprozesse

Befehl	Beschreibung
`kommando &`	Im Hintergrund ausführen
`Strg+Z`	Aktuellen Prozess in Hintergrund + stoppen
`bg`	Gestoppten Prozess im Hintergrund weiterlaufen lassen
`fg`	Prozess in den Vordergrund holen
`fg %1`	Job 1 in den Vordergrund
`jobs`	Hintergrundjobs anzeigen

Prozesspriorität (nice-Wert)

Wert	Priorität	Wer darf?
`-20`	Höchste Priorität	Nur root
`-20` bis `-1`	Hoch	Nur root
`0`	Standard	Alle
`1` bis `19`	Niedrig	Alle Benutzer
`19`	Niedrigste Priorität	Alle

```
nice -n 10 kommando  # Programm mit niedrigerer Priorität starten
nice -n -10 kommando # Höhere Priorität (nur root)
renice 5 -p PID      # Priorität eines laufenden Prozesses ändern
```

Systemüberwachung

Befehl	Beschreibung
`free -h`	Arbeitsspeicher-Auslastung
`uptime`	Laufzeit und Lastdurchschnitt
`df -h`	Festplattenbelegung
`du -sh *`	Verzeichnisgrößen
`top` / `htop`	CPU und RAM live
`docker stats`	Docker-Container-Ressourcen

Das `/proc`-Verzeichnis

Jeder Prozess hat ein Verzeichnis `/proc/PID/`:

```
/proc/
├── PID/
│   ├── cmdline     → Befehl
│   ├── cwd         → Link zum Arbeitsverzeichnis
│   ├── exe         → Link zum Programm
│   └── status      → Statusinformationen
├── meminfo          → Speicherinfo
└── cpuinfo          → CPU-Info
```

Systemstart: Init-Systeme

Bootvorgang (Überblick)

```
BIOS/UEFI → GRUB Bootloader → Kernel + initramfs laden
→ Root-Dateisystem mounten → init/systemd starten
→ Systemdienste laden → Benutzeranmeldung
```

Ablauf

1. Bootloader: Kernel und initramfs in RAM laden
2. Kernel laden: Hardware wird initialisiert
3. InitRamFS: Temporäres Root-Dateisystem zum laden von Modulen und Zugriff aufs echte FS
4. Mount Root FS: Kernel mountet Dateisystem nach /etc/fstab
5. Init starten: Erstes Prozessprogramm wird ausgeführt (init/systemd)
6. Dienste starten
7. Boot abgeschlossen: Bereit zur Anmeldung

sysinit vs. systemd – Vergleich

Eigenschaft	**sysvinit**	**systemd**
Konzept	Skriptbasiert, Runlevels	Units (services, sockets, timers, targets)
PID 1	`/sbin/init`	`systemd`
Start	Seriell (nacheinander)	Parallel (schneller!)
Konfiguration	`/etc/init.d/`, `/etc/rc*.d/`	Unit-Dateien in `/etc/systemd/system/`
Abhängigkeiten	Wenig, manuell	Umfassend (Wants, Requires, After, Before)
Logging	Syslog-basiert	journald + `journalctl`
Prozessverwaltung	Skripte nacheinander	Units, Restarts, cgroups
Sicherheit	Wenig eingebettet	Sandbox: PrivateTmp, ProtectSystem, NoNewPrivileges
Status	Veraltet	Standard in modernen Distros

systemd-Dienstverwaltung

systemctl – Dienste verwalten

Befehl	**Beschreibung**
`systemctl start dienst`	Dienst starten

Befehl	Beschreibung
`systemctl stop dienst`	Dienst stoppen
`systemctl restart dienst`	Dienst neustarten
`systemctl reload dienst`	Konfiguration neu laden (ohne Neustart)
`systemctl status dienst`	Status anzeigen
`systemctl enable dienst`	Beim Boot automatisch starten
`systemctl disable dienst`	Autostart deaktivieren
`systemctl is-enabled dienst`	Prüfen ob Autostart aktiv
`systemctl is-active dienst`	Prüfen ob Dienst läuft
`systemctl list-units --type service`	Alle laufenden Dienste
`systemctl list-unit-files --type=service`	Alle Dienste (auch inaktive)
`systemctl --failed`	Fehlgeschlagene Dienste
`systemctl daemon-reload`	Unit-Dateien neu einlesen

journalctl – Logging

Befehl	Beschreibung
`journalctl`	Alle Logs anzeigen
`journalctl -u nginx`	Logs eines bestimmten Dienstes
`journalctl -b`	Logs seit letztem Boot
`journalctl -f`	Live-Logs (wie `tail -f`)
`journalctl -p err`	Nur Fehler anzeigen
`journalctl --since "1 hour ago"`	Zeitgefiltert
`journalctl -xe`	Letzte Fehler mit Erklärung

systemd Unit-Dateien

Speicherorte:

- `/usr/lib/systemd/system/` – Standard-Units (vom Paketmanager)
- `/etc/systemd/system/` – Eigene/überschriebene Units (höhere Priorität)

Aufbau einer Service-Unit:

```
[Unit]
Description=Mein Beispielservice
Requires=network.target # Harte Abhängigkeit
After=network.target    # Startreihenfolge
Wants=redis.service     # Weiche Abhängigkeit

[Service]
Type=simple
ExecStart=/usr/bin/python3 /opt/app/server.py
ExecStop=/bin/kill -SIGTERM $MAINPID
User=nobody
Restart=on-failure
RestartSec=3
StartLimitBurst=4
StartLimitIntervalSec=30
WorkingDirectory=/opt/app/

# Sicherheitshärtung:
PrivateTmp=true      # Isoliertes /tmp
ProtectSystem=full   # System read-only
ProtectHome=true     # Kein Zugriff auf /home
NoNewPrivileges=true # Keine Privileg-Eskalation

# Ressourcenlimits:
MemoryMax=200M
CPUQuota=20%
TasksMax=100

[Install]
WantedBy=multi-user.target # Beim Boot starten
```

Unit aktivieren und starten:

```
sudo systemctl daemon-reload
sudo systemctl enable mein.service
sudo systemctl start mein.service
```

Systemanalyse

```
systemd-analyze blame                 # Bootzeit pro Dienst
systemd-analyze critical-chain        # Kritischer Pfad
systemd-analyze plot >boot.svg        # Visualisierung
systemd-analyze verify dienst.service # Syntax prüfen
```

cron

Grundprinzip

`cron` ist ein Dienst (Daemon) zum **zeitgesteuerten Ausführen von Befehlen**. Die Aufgaben werden in der **crontab** (cron table) definiert.

```
crontab -e          # Eigene crontab bearbeiten
crontab -l          # Eigene crontab anzeigen
crontab -r          # Eigene crontab löschen
crontab -u user -e # crontab eines bestimmten Benutzers bearbeiten (root)
```

Das Zeitformat (5-Felder-Cron)

```
┌──────────── Minute        (0 - 59)
│ ┌────────── Stunde        (0 - 23)
│ │ ┌──────── Tag (Monat)   (1 - 31)
│ │ │ ┌────── Monat         (1 - 12 oder jan,feb,...)
│ │ │ │ ┌──── Wochentag     (0 - 7, 0 und 7 = Sonntag, oder sun,mon,...)
│ │ │ │ │
* * * * *  Befehl
```

Sonderzeichen in cron

Zeichen	Bedeutung	Beispiel
`*`	Jeden möglichen Wert	`*` in Stunde = jede Stunde
`,`	Liste von Werten	`1,3,5` = Minute 1, 3 und 5
`-`	Bereich	`9-17` = von 9 bis 17 Uhr
`/`	Schrittweite (Step)	`*/5` = alle 5 Minuten

Das `/` (Slash / Schrittweite) – ausführlich erklärt

Der Slash definiert eine Schrittweite innerhalb eines Wertebereichs.

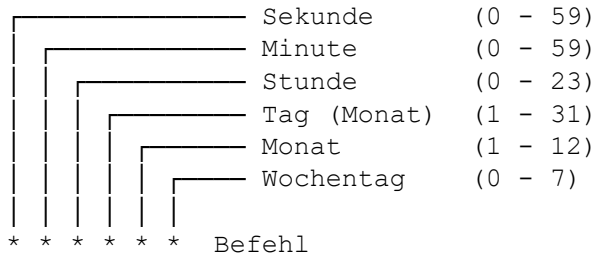
```
BEREICH/SCHRITT
```

- **`*/4`** im Minutenfeld: „Von 0 bis 59, jeden 4. Wert" → Minute 0, 4, 8, 12, 16, 20, ...56
- **`0/4`** im Minutenfeld: „Starte bei 0, dann alle 4" → Minute 0, 4, 8, 12, ...56 (identisch zu `*/4`)
- **`2/4`** im Minutenfeld: „Starte bei 2, dann alle 4" → Minute 2, 6, 10, 14, ...58
- **`10-30/5`** im Minutenfeld: „Von 10 bis 30, alle 5" → Minute 10, 15, 20, 25, 30

Fazit: `0/4` und `*/4` sind im Minutenfeld identisch, da `*` für 0-59 steht und `0/4` explizit bei 0 startet.

Das 6-Felder-Format (Quartz/Spring)

Manche Systeme (Java Quartz Scheduler, Spring Boot Cron, manche DevOps-Tools) verwenden ein **6-Felder-Format mit Sekunden**:



`0/4 * * * * *` (6 Felder, Quartz-Format):

- Sekunden-Feld: `0/4` → Starte bei Sekunde 0, alle 4 Sekunden → 0, 4, 8, 12, ...56
- Alle anderen Felder: `*` → immer
- **Bedeutung: Alle 4 Sekunden, jede Minute, jede Stunde, jeden Tag...**

Wichtig: Standard-Linux-cron (Vixie Cron, cronie) verwendet **5 Felder** ohne Sekunden. Die 6-Felder-Variante ist bei systemd-Timer oder Spring `@Scheduled` üblich.

Vordefinierte Makros (nur 5-Felder-cron)

Makro	**Bedeutung**	**Äquivalent**
`@reboot`	Bei jedem Start	—
`@yearly` / `@annually`	Einmal jährlich	`0 0 1 1 *`
`@monthly`	Einmal monatlich	`0 0 1 * *`
`@weekly`	Einmal wöchentlich	`0 0 * * 0`
`@daily` / `@midnight`	Täglich um Mitternacht	`0 0 * * *`
`@hourly`	Jede Stunde	`0 * * * *`

Beispiele

```
# Jede Minute
* * * * * /skript.sh

# Alle 5 Minuten
*/5 * * * * /skript.sh

# Alle 4 Minuten (ab Minute 0)
0/4 * * * * /skript.sh
# oder äquivalent:
*/4 * * * * /skript.sh

# Alle 4 Minuten, aber STARTEND bei Minute 2
2/4 * * * * /skript.sh # → 2, 6, 10, 14, ...

# Täglich um 3:30 Uhr
30 3 * * * /backup.sh

# Montag bis Freitag um 8:00 Uhr
0 8 * * 1-5 /benutzer_check.sh

# Jeden 1. des Monats um Mitternacht
0 0 1 * * /monatsabschluss.sh

# Stündlich in der Geschäftszeit (9-17 Uhr, Mo-Fr)
0 9-17 * * 1-5 /pruefe_server.sh

# Um 6:00 und 18:00 Uhr täglich
0 6,18 * * * /check.sh

# Alle 15 Minuten zwischen 8 und 20 Uhr
*/15 8-20 * * * /check.sh

# Beim Systemstart
@reboot /home/user/start_dienst.sh

# Wöchentlich (Sonntag 0:00 Uhr)
@weekly /backup_woche.sh

# Ausgabe in Logdatei umleiten
0 2 * * * /backup.sh >>/var/log/backup.log 2>&1

# Ausgabe unterdrücken
0 * * * * /check.sh >/dev/null 2>&1

# Mit Benutzerumgebung (PATH explizit setzen)
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
0 3 * * * /eigenes/backup.sh
```

cron-Umgebungsvariablen

```
# Am Anfang der crontab-Datei setzbar:
SHELL=/bin/bash
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin
MAILTO=admin@example.com # Ausgabe per Mail senden (leer = kein
Mail)
HOME=/root
```

Systemweite cron-Verzeichnisse

```
/etc/crontab        # Systemweite crontab (hat Benutzer-Spalte!)
/etc/cron.d/        # Weitere systemweite Cron-Dateien
/etc/cron.hourly/   # Skripte die stündlich laufen
/etc/cron.daily/    # Skripte die täglich laufen
/etc/cron.weekly/   # Skripte die wöchentlich laufen
/etc/cron.monthly/ # Skripte die monatlich laufen
```

Ausgabe umleiten:

```
# In Datei umleiten
* * * * * echo "test" >/home/user/output.txt 2>&1

# An Terminal senden
* * * * * echo "test" >/dev/pts/1
```

rsync

Grundprinzip

`rsync` (Remote Sync) ist ein Werkzeug zum **effizienten Synchronisieren von Dateien und Verzeichnissen**, lokal oder über Netzwerk (SSH). Es überträgt nur geänderte Teile von Dateien (**Delta-Transfer-Algorithmus**), was es sehr schnell macht.

Grundsyntax:

```
rsync [OPTIONEN] QUELLE ZIEL
```

Wie rsync intern funktioniert (Delta-Algorithmus)

1. Zieldatei wird in Blöcke (Chunks) aufgeteilt
2. Von jedem Block wird eine **schwache Prüfsumme** (rolling checksum) und eine **starke Prüfsumme** (MD4/MD5) berechnet
3. Die Prüfsummen werden an den Sender geschickt
4. Der Sender findet passende Blöcke in der Quelldatei
5. Nur **neue/geänderte Blöcke** werden übertragen
6. Das Ziel wird aus der Kombination alter und neuer Blöcke rekonstruiert

Ergebnis: Statt einer 100 MB Datei werden vielleicht nur 2 KB Unterschied übertragen.

Alle wichtigen Optionen

Option	Langform	Bedeutung
`-a`	`--archive`	Archivmodus: `-rlptgoD` zusammengefasst
`-r`	`--recursive`	Verzeichnisse rekursiv übertragen
`-l`	`--links`	Symlinks als Symlinks kopieren
`-p`	`--perms`	Berechtigungen übertragen
`-t`	`--times`	Zeitstempel übertragen

Option	Langform	Bedeutung
`-g`	`--group`	Gruppeninfo übertragen
`-o`	`--owner`	Eigentümer übertragen (root nötig)
`-D`		Gerätedateien + Sonderdateien
`-v`	`--verbose`	Ausführliche Ausgabe
`-vv`		Sehr ausführliche Ausgabe
`-q`	`--quiet`	Keine Ausgabe außer Fehlern
`-n`	`--dry-run`	Simulation: Zeigt was passieren würde, ohne etwas zu tun
`-z`	`--compress`	Daten komprimieren (bei langsamen Verbindungen)
`-P`		`--partial --progress` zusammen: Fortschritt und Wiederaufnahme
`--progress`		Fortschrittsanzeige
`--partial`		Teilübertragungen behalten (für Wiederaufnahme)
`-e`	`--rsh=BEFEHL`	Remote Shell angeben (z.B. `-e ssh`)
`--delete`		Dateien im Ziel löschen, die in der Quelle fehlen
`--delete-before`		Löschen vor dem Übertragen
`--delete-after`		Löschen nach dem Übertragen
`--exclude=MUSTER`		Dateien/Verzeichnisse ausschließen
`--exclude-from=DATEI`		Ausschlussliste aus Datei
`--include=MUSTER`		Ausschluss aufheben
`--filter=REGEL`		Flexible Filterregel
`--backup`		Sicherungskopien erstellen
`--backup-dir=DIR`		Backup-Verzeichnis für geänderte Dateien
`--suffix=SUFFIX`		Backup-Suffix (Standard: `~`)
`-u`	`--update`	Neuere Zieldateien nicht überschreiben
`-c`	`--checksum`	Vergleich per Prüfsumme statt Größe+Zeit
`-H`	`--hard-links`	Hardlinks erhalten
`-A`	`--acls`	ACLs übertragen
`-X`	`--xattrs`	Erweiterte Attribute übertragen

Option	Langform	Bedeutung
`--chmod=RECHTE`		Berechtigungen beim Ziel setzen
`--chown=USER:GRUPPE`		Eigentümer beim Ziel setzen
`--max-size=GRÖSSE`		Maximale Dateigröße (z.B. `--max-size=100M`)
`--min-size=GRÖSSE`		Mindestdateigröße
`--bwlimit=KBPS`		Bandbreite begrenzen (KB/s)
`--timeout=SEKS`		Verbindungs-Timeout setzen
`--stats`		Übertragungsstatistik anzeigen
`-h`	`--human-readable`	Größen leserlich darstellen
`--log-file=DATEI`		Log in Datei schreiben
`--password-file=DATEI`		Passwort aus Datei lesen
`--port=PORT`		Alternativen Port verwenden

Der Slash-Trick (sehr wichtig!)

```
rsync -av /quelle /ziel  # Kopiert den Ordner "quelle" IN "ziel" → /ziel/quelle/...
rsync -av /quelle/ /ziel # Kopiert den INHALT von "quelle" → /ziel/...
```

Der abschließende `/` an der Quelle macht den Unterschied!

Beispiele

```
# Lokale Synchronisation (einfach)
rsync -av /home/user/daten/ /backup/daten/

# Mit Simulation zuerst überprüfen (-n = dry run)
rsync -avn /home/user/daten/ /backup/daten/

# Remote-Backup über SSH
rsync -avz /home/user/dokumente/ benutzer@server:/backup/dokumente/

# Remote → Lokal (pull)
rsync -avz benutzer@server:/var/www/html/ /lokal/webseite/

# Backup mit --delete (spiegelt exakt)
rsync -av --delete /home/user/ /backup/user/

# SSH-Port 2222 verwenden
rsync -avz -e "ssh -p 2222" /daten/ user@server:/backup/

# SSH-Key angeben
rsync -avz -e "ssh -i ~/.ssh/mein_key" /daten/ user@server:/backup/
```

```
# Bestimmte Typen ausschließen
rsync -av --exclude="*.tmp" --exclude="*.log" /daten/ /backup/daten/

# Ausschlussliste aus Datei (eine Regel pro Zeile)
rsync -av --exclude-from=/home/user/.rsync-exclude /daten/ /backup/

# Nur .jpg Dateien übertragen
rsync -av --include="*.jpg" --exclude="*" /fotos/ /backup/fotos/

# Bandbreite begrenzen (500 KB/s)
rsync -avz --bwlimit=500 /daten/ user@server:/backup/

# Große Dateien ausschließen
rsync -av --max-size=50M /daten/ /backup/

# Vollständiges Backup mit Versionierung
rsync -av --backup --backup-dir=/backup/$(date %Y-%m-%d) /home/
/backup/aktuell/

# Fortschritt anzeigen
rsync -av --progress /quelle/ /ziel/
# Oder kürzer:
rsync -avP /quelle/ /ziel/

# Übertragungsstatistiken
rsync -av --stats /quelle/ /ziel/

# Dateien nur nach Prüfsumme vergleichen (langsamer aber genauer)
rsync -avc /quelle/ /ziel/
```

rsync in cron-Jobs

```
# Tägliches Backup um 2:30 Uhr

30 2 * * * rsync -az --delete /home/ /backup/home/
>>/var/log/backup.log 2>&1
```

Backup mit tar

tar – Archivierung

```
# Archiv erstellen
tar cvf archiv.tar datei1 datei2 verzeichnis/

# Archiv mit Komprimierung (gzip)
tar czf archiv.tar.gz /verzeichnis

# Archiv entpacken
tar xvf archiv.tar

# Komprimiertes Archiv entpacken
tar xzf archiv.tar.gz

# Inhalt anzeigen
tar tf archiv.tar
```

Optionen:

Option	Beschreibung
`c`	Create (Archiv erstellen)
`x`	Extract (Entpacken)
`v`	Verbose (Dateiliste anzeigen)
`f`	File (Archivdatei angeben)
`z`	gzip-Komprimierung
`j`	bzip2-Komprimierung
`t`	Inhalt auflisten

Backup-Skript (Beispiel):

```
#!/usr/bin/env bash
set -euo pipefail

BACKUP_SRC=("/etc" "/var/www")
BACKUP_DEST="/backup"
RETENTION_DAYS=7
LOGFILE="/var/log/backup.log"
LOCKFILE="/tmp/backup.lock"

log() {
    echo "$(date ' %F %T') $1" | tee -a "$LOGFILE"
}

cleanup() {
    rm -f "$LOCKFILE"
}
trap cleanup EXIT

[[ -f "$LOCKFILE" ]] && {
    echo "Already running"
    exit 1
}
touch "$LOCKFILE"

TIMESTAMP=$(date %F_%H-%M-%S)
TARGET="$BACKUP_DEST/backup_$TIMESTAMP.tar.gz"

log "Starting backup..."
tar -czf "$TARGET" "${BACKUP_SRC[@]}"

log "Cleaning old backups..."
find "$BACKUP_DEST" -type f -mtime $RETENTION_DAYS -name "*.tar.gz"
-delete

log "Backup finished."
```

Bash-Skripting

http://steve-parker.org/sh/sh.shtml v1.1 – 7 Aug 2007

UNIX / Linux Shell Cheat Sheet

steve-parker.org

File Manipulation

> file	create (overwrite) file
>> file	append to file
>file 2>&1	both output and errors to file
< file	read from file
a | b	pipe output from 'a' as input to 'b'

Common Constructs

while read f do echo "Line is $f" done < file	read text file line by line
$ grep foo myfile afoo foo foobar	find matching lines
$ cut -d: -f5 /etc/passwd Dilbert	get field with delimiter
foo=`ls`	get output of command

case $foo in
  a)
  echo "foo is A"
  ;;
  b)
  echo "foo is B"
  ;;
  *)
  echo "foo is not A or B"
  ;;
esac
	case is a good way to avoid iterating through many if/elif/elif/elif constructs.

doubleit() {
  expr $1 * 2
}
doubleit 3 # returns 6
	function declaration and calling syntax

for i in * do echo "File is $i" done	A for loop iterates through its input (which is subject to globbing)

Useful Variables

$IFS	Internal File Separator
$?	return code from last program
$SHELL	what shell is running this script?
LANG	Language; C is US English

Test Operators

```
if [ "$x" -lt "$y" ]; then
  # do something
fi
```

Numeric Tests

lt	less than
gt	greater than
eq	equal to
ne	not equal
ge	greater or equal
le	less or equal

File Tests

nt	newer than
d	is a directory
f	is a file
r	readable
w	writeable
x	executable

String Tests

=	equal to
z	zero length
n	not zero length

Logical Tests

&&	logical AND
||	logical OR
!	logical NOT

Argument Variables

$0	program name
$1	1st argument
$2	2nd argument
...	...
$9	9th argument
$*	all arguments
$#	No. of arguments

Variable Substitution

${V:-def}	$V, or "def" if unset
${V:=def}	$V (set to "def" if unset)
${V:?err}	$V, or "err" if unset

Conditional Execution

c1 || c2	run c1; if it fails, run c2
c1 && c2	run c1; if it works, run c2

Common utilities and switches

ls -lSr	list files, biggest last
ls -ltr	list files, newest last
ls -lh	human-readable filesizes
du -sk *	directory sizes (slow)
sort -n	sort numerically (not alpha)
ps -ef	list my commands
wget URL	download URL
time cmd	stopwatch on `cmd`
touch file	create file
read x	read "x" from keyboard
cmd | \ tee file.txt	cmd output to stdout and also to file.txt
nice cmd	run cmd with low priority

Networking

ifconfig -a	list all network interfaces
netstat -r	show routers
ssh u@host	log in to host as user "u"
scp file.txt \ u@host:	copy file.txt to host as user "u"

General Admin

less file	display file, page by page
alias l='ls -l'	create "l" as alias for "ls -l"
tar cf t.tar \ list_of_files	create a tar archive t.tar from the listed dirs/files
cal 3 1973	display a calendar (Mar 73)
df -h	show disk mounts
truss -p PID	show syscalls of PID

Files: Contents / Attributes

find . -size 10k -print	files over 10Kb
find . -name "*.txt" -print	find text files
find /foo -type d -ls	ls all directories under /foo
three=`expr 1 + 2`	simple maths
echo "scale = 5 ; 5121 / 1024" | bc	better maths
egrep "(foo|bar)" file	find "foo" or "bar" in file
awk '{ print $5 }' file	print the 5th word of each line
sed s/foo/bar/g file	replace "foo" in file with "bar"

Skript-Grundlagen & Datenfluss

- **Shebang:** Beginne ein Skript immer mit #! (z. B. #!/bin/bash), um die Shell festzulegen.
- **Ausführbar machen**: chmod +x <scriptname>.sh
- **Ausgabe überschreiben:** > file erstellt oder überschreibt eine Datei.
- **Ausgabe anhängen:** >> file fügt Text ans Ende einer Datei an.
- **Eingabe lesen:** < file liest den Inhalt einer Datei ein.
- **Pipe:** a | b leitet die Ausgabe von Befehl "a" als Eingabe an Befehl "b" weiter.
- **Kommentare**: # am Beginn der Zeile
- **Variablen-Zuweisung**: variablenname=”<wert>” (Kein Leerzeichen um das =)
- **Variablen-Aufruf**: $variablenname (Case-Sensitive)
- **Umgebungsvariable**: export global=”<wert>”
  - Für Unterprozesse sichtbar
- **Script-Aufruf**: ./scriptname.sh [param1]
- **User-Eingabe während des Scripts**: read [-s] [-r] [-p “<prompt>”] <variablenname>
  - -p “<prompt>”: Prompt vor der Eingabe
  - -s: verdeckte Eingabe (Passwörter)
  - -r: verhindert Escape-Sequenzen

Wichtige Variablen

- **$0**: Der Name des aufgerufenen Programms.
- **$1, $2, …**: Das erste und zweite übergebene Argument.
- **$#**: Die Anzahl der übergebenen Argumente.
- **$***: Alle übergebenen Argumente auf einmal.
- **$?**: Der Rückgabecode des letzten Befehls (0 = Fehlerfrei).
- **$$**: Die Prozess-ID (PID) des aktuellen Skripts.
- **${V:-default}**: Gibt den Wert von $V aus, oder "default", falls die Variable leer ist.

Rechnen mit Variablen

In Bash sind Variablen standardmäßig Text. Wenn mit Zahlen gerechnet werden soll, muss Bash ausdrücklich gesagt werden, dass eine arithmetische Berechnung gemeint ist.

Variable setzen und ausgeben

```
zahl=5
echo "$zahl"
```

Wichtig:

```
zahl=5   # richtig
zahl = 5 # falsch!
```

Bei Variablenzuweisungen dürfen keine Leerzeichen um das = stehen.

Rechnen mit $((...))

Die empfohlene Schreibweise für Ganzzahlrechnung ist:

```
a=10
b=3

echo $((a + b))   # 13
echo $((a - b))   # 7
echo $((a * b))   # 30
echo $((a / b))   # 3
echo $((a % b))   # 1
```

Operator	Bedeutung	Beispiel
+	Addition	$((a + b))
-	Subtraktion	$((a - b))
*	Multiplikation	$((a * b))
/	Division	$((a / b))
%	Modulo / Rest	$((a % b))

Ergebnis in Variable speichern

```
a=7
b=4

ergebnis=$((a + b))

echo "Das Ergebnis ist: $ergebnis"
```

Ausgabe:

```
Das Ergebnis ist: 11
```

Variable hochzählen

```
zahl=1

zahl=$((zahl + 1))

echo "$zahl"
```

Kurzformen:

```
((zahl++))     # um 1 erhöhen
((zahl--))     # um 1 verringern
((zahl += 5))  # 5 addieren
((zahl -= 2))  # 2 abziehen
```

Beispiel:

```
counter=0

((counter++))
((counter++))

echo "$counter"
```

Ausgabe:

```
2
```

Rechnen in Bedingungen

```
alter=20

if ((alter >= 18)); then
    echo "Volljährig"
else
    echo "Minderjährig"
fi
```

Bei Zahlenvergleichen ist diese Schreibweise oft lesbarer als:

```
if [ "$alter" -ge 18 ]; then
    echo "Volljährig"
fi
```

Achtung: Bash rechnet nur mit Ganzzahlen

```
echo $((5 / 2))
```

Ausgabe:

```
2
```

Bash rundet nicht mathematisch, sondern schneidet Nachkommastellen ab. Für Kommazahlen nutzt man z. B. bc:

```
echo "scale=2; 5 / 2" | bc
```

Ausgabe:

```
2.50
```

Falsch gesetzte Klammern beim Rechnen

Beim Rechnen mit Variablen müssen die Klammern exakt zusammenpassen. Bash unterscheidet zwischen arithmetischer Expansion $((...)), normaler Kommando-Ersetzung $(...) und Bedingungen mit ((...)). Eine falsch gesetzte oder vergessene Klammer führt meistens zu einem Syntaxfehler oder zu einem ganz anderen Verhalten.

Richtig: arithmetische Expansion

```
a=5
b=2

echo $((a + b))        # 7
echo $((a * (b + 3))) # 25
```

Die äußeren doppelten Klammern $((...)) bedeuten: Bash soll den Inhalt als Rechnung auswerten. Innere Klammern können wie in der Mathematik verwendet werden, um die Reihenfolge festzulegen.

Falsch: schließende Klammer fehlt

```
a=5
b=2

echo $((a + b)
```

Typische Folge: Bash meldet einen Syntaxfehler, weil die arithmetische Expansion nicht korrekt abgeschlossen wurde, z. B. "unexpected EOF" oder "syntax error".

Falsch: $() statt $(( )) verwendet

```
a=5
b=2

echo $(a + b)
```

Das ist keine Rechnung. $(...) bedeutet Kommando-Ersetzung. Bash versucht also, den Inhalt als Befehl auszuführen. Dadurch entstehen Fehler wie "command not found".

Falsch: Klammern verändern die Rechenreihenfolge

```
a=10
b=2
c=3

echo $((a + b * c))   # 16, weil Multiplikation vor Addition gilt
echo $(((a + b) * c)) # 36, weil zuerst a + b gerechnet wird
```

Klammern sind also nicht nur für die Syntax wichtig, sondern verändern auch das Ergebnis. Wie in der Mathematik wird das berechnet, was in Klammern steht, zuerst.

Falsch: einfache eckige Klammern für Rechnungen erwarten

```
a=5
b=2

if [ a + b -gt 6 ]; then
    echo "größer"
fi
```

Die einfache eckige Klammer [...] ist ein Test-Befehl, aber keine Rechenumgebung. Für Rechnungen in Bedingungen ist ((...)) besser geeignet:

```
if ((a + b > 6)); then
    echo "größer"
fi
```

Merksatz zu Klammern beim Rechnen

```
$(( ... )) = Rechnung ausführen
$( ... )   = Befehl ausführen und Ausgabe einsetzen
(( ... ))  = Rechnung/Bedingung auswerten, oft bei if oder Schleifen
[ ... ]    = Test-Befehl, z. B. für Dateien, Strings oder klassische Zahlenvergleiche
```

Listen, Strings und Arrays

In Bash ist wichtig zu unterscheiden, ob etwas als ein zusammenhängender Text, als mehrere Wörter oder als Array mit mehreren Elementen behandelt wird.

"123" ist ein String

```
wert="123"

echo "$wert"
```

Hier ist 123 ein String, also Text. Trotzdem kann Bash damit rechnen, wenn der Inhalt nur aus Zahlen besteht:

```
wert="123"

echo $((wert + 1))
```

Ausgabe:

```
124
```

Merke:

```
"123"      # ein Text/String mit drei Zeichen
123        # kann in Rechnungen als Zahl verwendet werden
```

"1 2 3" ist ein String mit Leerzeichen

```
werte="1 2 3"

echo "$werte"
```

Ausgabe:

```
1 2 3
```

Mit Anführungszeichen bleibt es ein einziger Wert:

```
for x in "$werte"; do
    echo "$x"
done
```

Ausgabe:

```
1 2 3
```

Die Schleife läuft hier nur einmal, weil "$werte" als ein kompletter String behandelt wird.

Ohne Anführungszeichen wird daraus eine Liste

```
werte="1 2 3"

for x in $werte; do
    echo "$x"
done
```

Ausgabe:

```
1
2
3
```

Ohne Anführungszeichen wird der Inhalt an Leerzeichen getrennt. Aus "1 2 3" werden also drei einzelne Werte.

Unterschied: "123" und "1 2 3"

```
a="123"
b="1 2 3"

for x in $a; do
    echo "$x"
done
```

Ausgabe:

```
123
```

123 enthält keine Leerzeichen, also bleibt es ein Wert.

```
for x in $b; do
    echo "$x"
done
```

Ausgabe:

```
1
2
3
```

1 2 3 enthält Leerzeichen, also wird es in mehrere Werte aufgeteilt.

Arrays in Bash

Ein Array ist eine echte Liste mit mehreren Elementen.

```
zahlen=(1 2 3)

echo "${zahlen[0]}"
echo "${zahlen[1]}"
echo "${zahlen[2]}"
```

Ausgabe:

```
1
2
3
```

Wichtig: Bash-Arrays beginnen bei Index 0.

Zugriff	Wert
${zahlen[0]}	erstes Element
${zahlen[1]}	zweites Element
${zahlen[2]}	drittes Element

Alle Array-Elemente ausgeben

```
zahlen=(1 2 3)

  echo "${zahlen[@]}"
```

Ausgabe:

```
  1 2 3
```

Über Array iterieren

```
zahlen=(1 2 3)

for zahl in "${zahlen[@]}"; do
    echo "$zahl"
done
```

Ausgabe:

```
1
2
3
```

Diese Schreibweise ist sicher und empfohlen:

```
  "${array[@]}"
```

Unterschied zwischen "$array" und "${array[@]}"

```
werte=("Apfel Birne" "Banane" "Kirsche")
```

Richtig:

```
for wert in "${werte[@]}"; do
    echo "$wert"
done
```

Ausgabe:

```
Apfel Birne
Banane
Kirsche
```

Falsch bzw. oft problematisch:

```
for wert in ${werte[@]}; do
    echo "$wert"
done
```

Ausgabe:

```
Apfel
Birne
Banane
Kirsche
```

Ohne Anführungszeichen wird "Apfel Birne" in zwei Wörter zerlegt. Deshalb sollten Arrays fast immer mit "${array[@]}" durchlaufen werden.

Merksätze

- "123" ist ein String, kann aber als Zahl verwendet werden, wenn nur Ziffern enthalten sind.
- "1 2 3" ist ein String mit Leerzeichen.
- Ohne Anführungszeichen wird an Leerzeichen getrennt.
- Mit Anführungszeichen bleibt der Inhalt ein zusammenhängender Wert.
- Arrays sind echte Listen und werden mit Klammern geschrieben: array=(wert1 wert2 wert3).
- Beim Durchlaufen von Arrays ist "${array[@]}" die sichere Standardform.

Logik & Bedingungen

Befehle direkt verketten:

- cmd1 && cmd2: Führe cmd1 aus; wenn er erfolgreich war, führe cmd2 aus.
- cmd1 || cmd2: Führe cmd1 aus; wenn er fehlschlägt, führe cmd2 aus.

If-Abfrage:

Bash

```
if [ "$x" -lt "$y" ]; then
    # do something
fi
```

Die Case-Anweisung (Mehrfach-Auswahl):

Bash

```
case $foo in
  a) echo "foo is A" ;; #(foo = a)
  b) echo "foo is B" ;; #(foo = b)
  *) echo "foo is not A or B" ;; #(alles anderes)
Esac
```

(Wichtig: Das ;; am Ende jedes Blocks ist zwingend erforderlich).

for – loop Beispiel: Über eine Liste iterieren

Dies ist der klassische Anwendungsfall: Du gehst eine Liste von Dateien oder Werten nacheinander durch.

Bash
```bash
#!/bin/bash

MEIN_ARRAY=("Apfel Birne" "Banane" "Kirsche")
MEINE_LISTE="Apfel Birne Banane"
for obst in $MEINE_LISTE; do
    echo "Ich mag $obst"
done

# Liste von Werten durchgehen
for tier in Hund Katze Maus; do
    echo "Das ist ein(e): $tier"
done
```

Beispiel: Über einen Zahlenbereich iterieren

Wenn du eine Aktion eine bestimmte Anzahl an Malen wiederholen willst (z. B. 5-mal):

Bash
```bash
#!/bin/bash

# Zählen von 1 bis 5
for i in {1..5}; do
    echo "Durchlauf Nummer: $i"
done
```

Profi-Tipp: Dateien im Verzeichnis verarbeiten

Ein sehr häufiger Anwendungsfall in der Shell ist es, alle Dateien eines bestimmten Typs zu bearbeiten:

Bash
```bash
#!/bin/bash

# Alle .txt Dateien im aktuellen Verzeichnis finden und ausgeben
for datei in *.txt; do
    echo "Verarbeite Datei: $datei"
    # Hier könnte z.B. ein grep oder cat Befehl folgen
done
```

Zusammenfassung der Struktur:

- **`for`**: Startet die Schleife.
- **`variable`**: Ein frei wählbarer Name, der bei jedem Durchlauf den aktuellen Wert annimmt.
- **`in`**: Definiert die Liste oder den Bereich, der abgearbeitet wird.
- **`do`**: Leitet den Codeblock ein, der für jedes Element ausgeführt wird.
- **`done`**: Beendet den Schleifenblock.

for	Wenn die Anzahl der Elemente feststeht.
While	Wenn du auf einen Zustand wartest.
Continue	Wenn ein einzelnes Element ignoriert werden soll.
Break	Wenn die Arbeit vorzeitig erledigt ist.

Test-Operatoren (Bedingungen prüfen)

Typ	Operatoren	Bedeutung
Zahlen	-eq / -ne	Gleich / Ungleich
	-lt / -le	Kleiner als / Kleiner oder gleich
	-gt / -ge	Größer als / Größer oder gleich
Text	=	Strings sind gleich
	-z / -n	Länge ist Null (leer) / Nicht leer
Dateien	-d / -x	Ist ein Verzeichnis / Ist ausführbar
	-r / -w	Ist lesbar / Ist schreibbar
	-nt	Ist neuer als (newer than)
Logik	&& / `	
	!	Logisches NICHT

While-Schleife (Datei zeilenweise einlesen):

Bash

```
while read f; do
    echo "Line is $f"
done <dateiname.txt

while read benutzer; do
    mkdir "/home/$benutzer"
    echo "Ordner für $benutzer wurde erstellt."
done <benutzerliste.txt #Datei einlesen wird als 1. Ausgeführt
```

IF-Abfrage (ausführlich)

Die einfache if-Anweisung

Grundsätzlich hat die if-Anweisung der Bourne-Shell eine sehr einfache Form. Nach dem if steht ein Befehl, der ausgeführt wird. Gibt dieses Kommando eine 0 als Rückgabewert zurück, so gilt die Bedingung als erfüllt und die Aktionen, die zwischen dem folgenden then und fi stehen werden ausgeführt.

```
if Kommando
then
  Aktion
  Aktion
  ...
 fi
```

Natürlich sind die Aktionen auch wieder normale Unix-Befehle. Das „fi“, das den Block beendet, der durch „if ... then“ begonnen wurde, ist einfach nur das „if“ rückwärts geschrieben.

Das Programm test

Damit es jetzt sinnvolle Möglichkeiten gibt, Bedingungen zu überprüfen brauchen wir ein Programm, das verschiedene Tests durchführt und jeweils bei gelungenem Test eine 0 als Rückgabewert zurückgibt, bei mislingenem Test eine 1. Dieses Programm heißt *test* und ermöglicht alle wesentlichen Bedingungsüberprüfungen, die für das Shell-Programmieren notwendig sind.

Damit wir nicht jedesmal schreiben müssen

```
if test ...
```

gibt es einen symbolischen Link auf das Programm test, der einfach [heißt. Allerdings verlangt das Programm test, wenn es merkt, dass es als [aufgerufen wurde, auch als letzten Parameter eine eckige Klammer zu. Damit ist es also möglich zu schreiben:

```
if [ ... ]
```

Wichtig ist hierbei, dass unbedingt ein Leerzeichen zwischen if und der Klammer und zwischen der Klammer und den eigentlichen Tests stehen muß. Es handelt sich bei der Klammer ja tatsächlich um einen Programmaufruf!

```
DATEI="test.txt" # Prüfen, ob die Datei existiert UND sowohl lesbar als auch beschreibbar ist
if [ -f "$DATEI" ] && [ -r "$DATEI" ] && [ -w "$DATEI" ]; then
```

Die verschiedenen Bedingungsüberprüfungen mit test bzw. [(Buch Seite 312)

-r *Dateiname: if [-r file.txt]; then:* Die Datei *Dateiname* existiert und ist lesbar

-w *Dateiname*: Die Datei *Dateiname* existiert und ist beschreibbar

-x *Dateiname*: Die Datei *Dateiname* existiert und ist ausführbar

-d *Dateiname*: Die Datei *Dateiname* existiert und ist ein Verzeichnis

-s *Dateiname*: Die Datei *Dateiname* existiert und ist nicht leer

-b *Dateiname*: Die Datei *Dateiname* existiert und ist ein blockorientiertes Gerät

-c *Dateiname*: Die Datei *Dateiname* existiert und ist ein zeichenorientiertes Gerät

-g *Dateiname*: Die Datei *Dateiname* existiert und das SGID-Bit ist gesetzt

-k *Dateiname*: Die Datei *Dateiname* existiert und das Sticky-Bit ist gesetzt

-u *Dateiname*: Die Datei *Dateiname* existiert und das SUID-Bit ist gesetzt

-p *Dateiname*: Die Datei *Dateiname* existiert und ist ein Named Pipe

-e *Dateiname*: Die Datei *Dateiname* existiert

-f *Dateiname*: Die Datei *Dateiname* existiert und ist eine reguläre Datei

-L *Dateiname*: Die Datei *Dateiname* existiert und ist ein symbolischer Link

-S *Dateiname*: Die Datei *Dateiname* existiert und ist ein Socket

-O *Dateiname*: Die Datei *Dateiname* existiert und ist Eigentum des Anwenders, unter dessen UID das test-Programm gerade läuft

-G *Dateiname*: Die Datei *Dateiname* existiert und gehört zu der Gruppe, zu der der User gehört, unter dessen UID das test-Programm gerade läuft

Datei1* -nt *Datei2: *Datei1* ist neuer als *Datei2* (newer than)

Datei1* -ot *Datei2: *Datei1* ist älter als *Datei2* (older than)

Datei1* -ef *Datei2: *Datei1* und *Datei2* benutzen die gleiche I-Node (equal file)

-z *Zeichenkette*: Wahr wenn *Zeichenkette* eine Länge von Null hat.

-n *Zeichenkette*: Wahr wenn *Zeichenkette* eine Länge von größer als Null hat.

Zeichenkette1* = *Zeichenkette2: Wahr wenn *Zeichenkette1* gleich *Zeichenkette2*

Zeichenkette1* != *Zeichenkette2: Wahr wenn *Zeichenkette1* ungleich *Zeichenkette2*

Wert1* -eq *Wert2: Wahr, wenn *Wert1* gleich *Wert2* (equal)

Wert1* -ne *Wert2: Wahr, wenn *Wert1* ungleich *Wert2* (not equal)

Wert1* -gt *Wert2: Wahr, wenn *Wert1* größer *Wert2* (greater than)

Wert1* -ge *Wert2: Wahr, wenn *Wert1* größer oder gleich *Wert2* (greater or equal)

Wert1* -lt *Wert2: Wahr, wenn *Wert1* kleiner *Wert2* (less than)

Wert1* -le *Wert2: Wahr, wenn *Wert1* kleiner oder gleich *Wert2* (less or equal)

!*Ausdruck*: Logische Verneinung von *Ausdruck*

Ausdruck* -a *Ausdruck: Logisches UND. Wahr, wenn beide Ausdrücke wahr sind

Ausdruck* -o *Ausdruck: Logisches ODER. Wahr wenn mindestens einer der beiden Ausdrücke wahr ist

Mit diesen Tests sind so ziemlich alle denkbaren Bedingungsüberprüfungen möglich, die in einem Shellscript notwendig sind.

Die erweiterte if-else Anweisung

Natürlich bietet die if-Anweisung auch eine Erweiterung zur normalen Form, die sogenannte if-else Anweisung. Es ist also möglich zu schreiben:

```
if [ Ausdruck ];
 then
   Kommandos
 else
   Kommandos
 fi
```

Die if-elif-else Anweisung

Um noch einen Schritt weiterzugehen bietet die if-Anweisung sogar ein weiteres if im else, das sogenannte elif, das wieder eine Bedingung überprüft:

```
if [ Ausdruck ];
 then
   Kommandos
 elif [ Ausdruck ];
 then
   Kommandos
 else
   Kommandos
 fi
```

Mehrfachauswahl mit case

Oft kommt es vor, dass eine Variable ausgewertet werden muß und es dabei viele verschiedenen Möglichkeiten gibt, welche Werte diese Variable annehmen kann. Natürlich wäre es mit einer langen if-elif-elif-elif... Anweisung möglich, so etwas zu realisieren, das wäre aber sowohl umständlich, als auch schwer zu lesen. Damit solche Fälle einfacher realisiert werden können, gibt es die Mehrfachauswahl mit case. Der prinzipielle Aufbau einer case-Entscheidung sieht folgendermaßen aus:

```
case Variable in
   Muster1) Kommando1 ;;
   Muster2) Kommando2 ;;
   Muster3) Kommando3 ;;
   ...
 esac
```

Essenzielle Werkzeuge

- **Suchen:** grep foo myfile findet Zeilen mit dem Text "foo" in einer Datei.
- **Dateien finden:** find . -name "*.txt" -print sucht nach Textdateien im aktuellen Verzeichnis.
- **Text filtern:** awk '{print $5}' file gibt nur das 5. Wort jeder Zeile aus.
- **Text ersetzen:** sed s/foo/bar/g file ersetzt "foo" durch "bar".

Funktionen

1. Der Exit-Status (Die Rückgabe von Erfolg/Fehler)

Jeder Befehl in Linux – und somit auch jede Funktion – endet mit einem Exit-Status. Dieser ist eine Zahl zwischen **0 und 255**.

- **0**: Der Befehl war erfolgreich.
- **1–255**: Es trat ein Fehler auf (1 ist oft ein allgemeiner Fehler).

Du setzt diesen Status explizit mit dem Befehl `return`.

Bash

```bash
check_file() {
    if [ -f "$1" ]; then
        return 0  # Datei existiert
    else
        return 1  # Datei existiert nicht
    fi
}

# Aufruf:
if check_file "test.txt"; then
    echo "Alles okay."
else
    echo "Fehler: Datei nicht gefunden."
fi
```

Wichtig: Der Status wird in der Shell-Variable `$?` gespeichert. Er ist **ausschließlich** für die Logik (Erfolg/Fehler) gedacht, nicht um Daten zu übergeben.

2. Die Standard-Ausgabe (Die Rückgabe von Daten)

Wenn du einen **Wert** (z. B. einen Dateinamen, eine Berechnung oder einen String) aus der Funktion an dein Skript zurückgeben willst, musst du die **Standard-Ausgabe (stdout)** nutzen.

Die Funktion „schreibt" das Ergebnis einfach auf die Konsole (mit `echo`), und du fängst es mit der **Command Substitution** `$(...)` ab.

Bash

```bash
# Funktion berechnet etwas und "gibt es aus"
addiere() {
    local summe=$(( $1 + $2 ))
    echo $summe  # Das ist die Rückgabe der Daten
}

# Aufruf: Fange die Ausgabe der Funktion in einer Variable ab
ergebnis=$(addiere 10 20)

echo "Das Ergebnis ist: $ergebnis"
```

```bash
# Definition
meine_funktion() {
    echo "Hallo $1!"
}

# Aufruf
meine_funktion "Welt"

# Mit Rückgabewert (Exit-Code 0-255)
ist_gerade() {
    if (($1 % 2 == 0)); then
        return 0
    else
        return 1
    fi
}

# Werte "zurückgeben" über echo
berechne() {
    echo $(($1 * $1))
}
ergebnis=$(berechne 5)

# Lokale Variablen
funktion() {
    local zahl=5 # Nur innerhalb der Funktion sichtbar
    echo $zahl
}
```

Best Practices

Fehlerbehandlung mit trap

```bash
cleanup() {
    rm -f "$LOCKFILE"
    echo "Aufgeräumt"
}
trap cleanup EXIT INT TERM
```

Locking (verhindert doppelte Ausführung)

```bash
LOCKFILE="/tmp/myscript.lock"
exec 200>"$LOCKFILE"
flock -n 200 || {
    echo "Already running"
    exit 1
}
```

main()-Struktur (empfohlen)

```bash
#!/bin/bash
set -euo pipefail

check_input() {
    if [ -z "$1" ]; then
        echo "Kein Parameter!"
        exit 1
    fi
}

verarbeite() {
```

```
    echo "Verarbeite: $1"
}

main() {
    check_input "$1"
    verarbeite "$1"
}

main "$@"
```

VM vs. Docker

Virtualisierung

Eigenschaft	Virtuelle Maschine (VM)	Docker Container
Isolation	Vollständig (eigener Kernel)	Prozess-Level (teilt Kernel mit Host)
Ressourcen	Braucht viel RAM/CPU (eigenes OS)	Leichtgewichtig
Startzeit	Minuten	Sekunden
Größe	GB (volles Betriebssystem)	MB (nur App + Dependencies)
Portabilität	Image-basiert	Container-Image (Docker Hub)
Hypervisor	Ja (VMware, VirtualBox, KVM)	Docker Engine
Use Case	Verschiedene OS gleichzeitig	Microservices, CI/CD

Begriffe (Virtualisierung):

- **Hostsystem** = System, auf dem die Virtualisierung läuft
- **Gastsystem** = System in der VM
- **Hypervisor / VMM** = Virtualisierungssoftware

VirtualBox Netzwerktypen:

Typ	Beschreibung
NAT	Standard, VM nutzt Host als Router
NAT-Netzwerk	VMs können untereinander + nach außen kommunizieren
Bridged	VM bekommt direkten Netzwerkzugang
Internes Netzwerk	Nur VMs untereinander
Host-only	VMs + Host können kommunizieren

Docker

Grundbegriffe

Begriff	Beschreibung
Image	Bauplan/Vorlage ("eingepackte Applikation")
Container	Laufende Instanz eines Images
Dockerfile	Bauanleitung für ein Image
Docker Hub	Zentrale Registry für Images
Volume	Persistenter Speicher

Wichtige Docker-Befehle

Images verwalten

```
docker image pull nginx            # Image herunterladen
docker image build -t myapp:1.0 . # Image aus Dockerfile bauen
docker image ls                    # Alle Images anzeigen
docker image rm nginx              # Image löschen
docker images prune                # Ungenutzte Images löschen
```

Container verwalten

```
# Container erstellen und starten
docker run -d -p 8080:80 --name myweb nginx
docker run -it ubuntu bash          # Interaktiv starten
docker run --rm alpine echo "Hello" # Nach Beenden löschen

# Container verwalten
docker container ls             # Laufende Container
docker container ls -a          # Alle Container
docker container stop myweb     # Stoppen
docker container start myweb    # Starten
docker container rm myweb       # Löschen
docker container rm -f myweb    # Erzwungen löschen
docker container inspect myweb # Details anzeigen

# Im Container arbeiten
docker exec -it myweb bash # Shell in laufendem Container
docker logs myweb          # Logs anzeigen
docker logs -f myweb       # Live-Logs
docker stats               # Ressourcen-Monitoring
```

docker run Optionen

Option	Beschreibung
`-d systemctl`	Detached (Hintergrund)
`-it`	Interaktiv mit Terminal
`-p host:container`	Port-Mapping
`-v host:container`	Volume/Bind Mount
`--name name`	Container-Name vergeben

Option	Beschreibung
`--network netz`	Mit Netzwerk verbinden
`--rm`	Container nach Beenden löschen
`-e VAR=WERT`	Umgebungsvariable setzen

Volumes

```
docker volume create myvolume  # Volume erstellen
docker volume ls               # Alle Volumes
docker volume inspect myvolume # Details
docker volume rm myvolume      # Löschen

# Mit Volume starten
docker run -v myvolume:/data ubuntu
# Bind Mount (lokales Verzeichnis)
docker run -v /home/user/html:/usr/share/nginx/html nginx
```

Netzwerke

```
docker network create mynetwork # Netzwerk erstellen
docker network rm mynetwork     # Netzwerk löschen

# Container im gleichen Netzwerk können sich über Namen erreichen:
docker run -d --network mynetwork --name db mongo
docker run -d --network mynetwork --name app node-app
# → app kann db über "db" als Hostname ansprechen
```

Dockerfile – Image erstellen

```
# Basis-Image
FROM python:3.10-slim

# Arbeitsverzeichnis setzen
WORKDIR /app

# Abhängigkeiten zuerst kopieren (Layer-Cache!)
COPY requirements.txt .
RUN pip install -r requirements.txt

# Quellcode kopieren
COPY . .

# Benutzer erstellen (Sicherheit!)
RUN addgroup app && adduser -S -G app app
USER app

# Port dokumentieren
EXPOSE 5000

# Umgebungsvariable
ENV APP_ENV=production

# Startbefehl (Exec-Format bevorzugt!)
CMD ["python", "app.py"]
```

Wichtige Dockerfile-Anweisungen:

Anweisung	Beschreibung
`FROM`	Basis-Image
`WORKDIR`	Arbeitsverzeichnis im Container
`COPY`	Dateien in Container kopieren
`RUN`	Befehl beim Image-Build ausführen
`CMD`	Startbefehl (überschreibbar)
`ENTRYPOINT`	Startbefehl (schwer überschreibbar)
`EXPOSE`	Port dokumentieren
`ENV`	Umgebungsvariable
`ARG`	Build-Argument
`USER`	Benutzer wechseln
`.dockerignore`	Dateien vom Build ausschließen

Multi-Stage Build (Image-Optimierung):

```
# Stage 1: Build
FROM node:14 AS build
WORKDIR /app
COPY . .
RUN npm install && npm run build

# Stage 2: Production (nur fertiges Ergebnis!)
FROM nginx:alpine
COPY --from=build /app/build /usr/share/nginx/html
```

Layer-Cache-Optimierung:

- **Stabile Dateien** (package.json) **vor** veränderlichen Dateien (Quellcode) kopieren
- So werden Abhängigkeiten nur bei Änderung neu installiert

Container-Lebenszyklus:

```
Erstellt → docker run → Running → docker stop → Stopped
→ docker kill → Gelöscht
Running → docker pause → Paused → docker unpause → Running
```

Docker Compose

Grundprinzip

`docker compose` (früher: `docker-compose`) ermöglicht die **Definition und Verwaltung mehrerer zusammengehöriger Container** (Multi-Container-Anwendungen) über eine einzige YAML-Konfigurationsdatei.

```
docker compose [OPTIONEN] BEFEHL
# Konfigurationsdatei: docker-compose.yml (oder docker-compose.yaml)
```

Alle docker compose Befehle

Befehl	Bedeutung
`up`	Container erstellen und starten
`down`	Container stoppen und entfernen
`start`	Gestoppte Container starten
`stop`	Container graceful stoppen
`restart`	Container neu starten
`pause`	Container einfrieren (SIGSTOP)
`unpause`	Container fortsetzen
`build`	Images neu bauen
`pull`	Images herunterladen
`push`	Images in Registry hochladen
`ps`	Status der Container anzeigen
`logs`	Container-Logs anzeigen
`exec`	Befehl in laufendem Container ausführen
`run`	Einmaligen Befehl in neuem Container ausführen
`config`	Konfiguration anzeigen/validieren
`images`	Images der Services auflisten
`rm`	Gestoppte Container entfernen
`kill`	Container mit Signal beenden
`top`	Prozesse in Containern anzeigen
`events`	Echtzeit-Ereignisse ausgeben
`port`	Port-Mapping eines Services anzeigen
`version`	Versionsinformationen

Optionen für `docker compose up`

Option	Bedeutung
`-d` / `--detach`	Im Hintergrund starten (detached mode)
`--build`	Images vor dem Start neu bauen
`--no-build`	Images nicht neu bauen
`--no-recreate`	Existierende Container nicht neu erstellen
`--force-recreate`	Container immer neu erstellen
`--remove-orphans`	Container für nicht mehr definierte Services entfernen
`--scale SERVICE=N`	Service auf N Instanzen skalieren
`--timeout N`	Timeout in Sekunden (Standard: 10)
`--wait`	Warten bis alle Container healthy sind

Optionen für `docker compose down`

Option	Bedeutung
`-v` / `--volumes`	Volumes ebenfalls löschen
`--rmi all`	Alle Images entfernen
`--rmi local`	Nur lokal gebaute Images entfernen
`--remove-orphans`	Orphan-Container entfernen
`-t N` / `--timeout N`	Timeout setzen

Optionen für `docker compose logs`

Option	Bedeutung
`-f` / `--follow`	Log live verfolgen
`--tail=N`	Nur letzte N Zeilen anzeigen
`-t` / `--timestamps`	Zeitstempel anzeigen
`--no-log-prefix`	Service-Name nicht voranstellen

Optionen für `docker compose exec`

Option	Bedeutung
`-it`	Interaktives Terminal
`-u USER`	Als Benutzer ausführen
`-e VAR=WERT`	Umgebungsvariable setzen
`-w DIR`	Arbeitsverzeichnis setzen

Allgemeine Optionen (vor dem Befehl)

Option	Bedeutung
`-f DATEI`	Andere Compose-Datei verwenden
`-p NAME`	Projektname setzen (Standard: Verzeichnisname)
`--profile PROFIL`	Service-Profile aktivieren
`--env-file DATEI`	Andere .env-Datei verwenden
`--progress STRING`	Ausgabestil: `auto`, `tty`, `plain`, `quiet`

Aufbau einer docker-compose.yml

```yaml
version: '3.9'
services:
  webserver:
    image: 'nginx:alpine'
    container_name: mein-nginx
    hostname: webserver
    restart: always
    ports:
      - '80:80'
      - '443:443'
    volumes:
      - './html:/usr/share/nginx/html'
      - 'nginx_conf:/etc/nginx'
    environment:
      - NGINX_HOST=example.com
    env_file:
      - .env
    networks:
      - frontend
    depends_on:
      - app
    healthcheck:
      test:
        - CMD
        - curl
        - '-f'
        - 'http://localhost'
      interval: 30s
      timeout: 10s
```

```
      retries: 3
      start_period: 40s
    labels:
      - traefik.enable=true
    logging:
      driver: json-file
      options:
        max-size: 10m
        max-file: '3'
  app:
    build:
      context: ./app
      dockerfile: Dockerfile.prod
      args:
        - BUILD_VERSION=1.0
    image: 'meine-app:latest'
    restart: unless-stopped
    ports:
      - '3000:3000'
    volumes:
      - 'app_data:/app/data'
    environment:
      DATABASE_URL: 'postgres://user:pass@db:5432/mydb'
      NODE_ENV: production
    networks:
      - frontend
      - backend
    depends_on:
      db:
        condition: service_healthy
  db:
    image: 'postgres:15'
    restart: always
    volumes:
      - 'db_data:/var/lib/postgresql/data'
    environment:
      POSTGRES_DB: mydb
      POSTGRES_USER: user
      POSTGRES_PASSWORD: pass
    networks:
      - backend
    healthcheck:
      test:
        - CMD-SHELL
        - pg_isready -U user -d mydb
      interval: 10s
      timeout: 5s
      retries: 5
volumes:
  nginx_conf: null
  app_data: null
  db_data:
    driver: local
    driver_opts:
      type: none
      o: bind
      device: /mnt/data/postgres
networks:
  frontend:
    driver: bridge
  backend:
    driver: bridge
    internal: true
  extern_netz:
```

```
    external: true
    name: mein-netz
```

restart-Policies

Wert	Bedeutung
`no`	Nie neu starten (Standard)
`always`	Immer neu starten
`on-failure`	Nur bei Fehler (Exit-Code ≠ 0) neu starten
`unless-stopped`	Immer neu starten, außer wenn manuell gestoppt

.env-Datei

```
# .env (im selben Verzeichnis wie docker-compose.yml)
POSTGRES_PASSWORD=geheim123
APP_PORT=3000
IMAGE_TAG=1.5.2
```

Im docker-compose.yml verwendbar als:

```
services:
  app:
    image: 'meineapp:${IMAGE_TAG}'
    ports:
      - '${APP_PORT}:3000'
```

Praktische Beispiele

```
# Alle Services starten (Hintergrund)
docker compose up -d

# Services starten und Live-Logs beobachten
docker compose up

# Nur bestimmten Service starten
docker compose up -d db

# Images neu bauen und starten
docker compose up -d --build

# Status anzeigen
docker compose ps

# Logs aller Services
docker compose logs -f

# Logs nur für "app" Service, letzte 50 Zeilen
docker compose logs --tail=50 -f app

# In Container einloggen
docker compose exec app bash
```

```
docker compose exec db psql -U user -d mydb

# Einmaligen Befehl ausführen
docker compose run --rm app npm test

# Alles stoppen und entfernen (inkl. Volumes)
docker compose down -v

# Bestimmte Services neu starten
docker compose restart app

# Scale (mehrere Instanzen)
docker compose up -d --scale app=3

# Mit anderem Dateinamen
docker compose -f docker-compose.prod.yml up -d

# Projektname setzen
docker compose -p meinprojekt up -d

# Konfiguration validieren
docker compose config

# Alle Images bauen ohne zu starten
docker compose build

# Aktuellen Stand der Container
docker compose top
```

Aufgaben und Lösungen aus dem Unterricht

DockerComposeZusatzaufgaben.pdf

Quelle: DockerComposeZusatzaufgaben.pdf - 9 Seite(n)

Seite 1

```
Thema: Linux
Technikerschule Erlangen



📝     Docker-Compose Übungsblatt

✅   Aufgabe 1: Webserver mit Docker Compose
Ziel: Erstelle mit Docker Compose einen Webserver, der über den Browser erreichbar ist.

📋   Aufgabenstellung:
    1. Erstelle ein neues Projektverzeichnis mit einer index.html-Datei.

    2. Setze mit Docker Compose einen Nginx-Container auf.
    3. Der Webserver soll unter http://localhost:8080 erreichbar sein.

💡   Hinweise:
    • Verwende das Nginx-Image (nginx:latest).

    • Binde das Verzeichnis mit index.html als Volume ein.

    • Port 80 im Container soll auf Port 8080 des Hosts weitergeleitet werden.

✅   Lösung:
Dateistruktur:
webserver/
├── docker-compose.yml
└── html/
    └── index.html

index.html:
<h1>Hello from Docker Compose!</h1>

docker-compose.yml:
version: '3.8'
services:
  web:
    image: nginx:latest
    ports:
       - "8080:80"
    volumes:
       - ./html:/usr/share/nginx/html:ro
```

Seite 2

```
Thema: Linux


✅
Technikerschule Erlangen


    Aufgabe 2: PHP + MySQL mit Docker Compose
Ziel: Starte eine einfache PHP-Webanwendung, die sich mit einer MySQL-Datenbank verbindet.

📋   Aufgabenstellung:
    1. Erstelle eine PHP-Datei, die sich mit einer MySQL-Datenbank verbindet.
    2. Setze zwei Container auf: einen für PHP (z. B. php:8.2-apache) und einen für MySQL.

    3. Konfiguriere docker-compose.yml so, dass die Dienste korrekt verbunden sind.

💡   Hinweise:
    • In PHP ist der MySQL-Host db (der Name des Services).

    • Nutze Umgebungsvariablen für MySQL (Benutzer, Passwort, Datenbank).

✅   Lösung:
```

```
Dateistruktur:
php-mysql/
├── docker-compose.yml
└── src/
    └── index.php

index.php:
<?php
$mysqli = new mysqli("db", "user", "password", "testdb");
if ($mysqli->connect_error) {
    die("Connection failed: " . $mysqli->connect_error);
}
echo "Connected to MySQL successfully!";
?>

docker-compose.yml:

version: '3.8'
services:
  web:
    image: php:8.2-apache
    volumes:
       - ./src:/var/www/html
    ports:
       - "8000:80"
    depends_on:
       - db
  db:
    image: mysql:8.0
    environment:
       MYSQL_DATABASE: testdb
       MYSQL_USER: user
       MYSQL_PASSWORD: password
       MYSQL_ROOT_PASSWORD: root
```

Seite 3

```
Thema: Linux


✅
Technikerschule Erlangen


    Aufgabe 3: Node.js mit MongoDB
Ziel: Erstelle eine Node.js-App, die eine Verbindung zu MongoDB aufnimmt und einfache HTTP-
Antworten liefert.

📋   Aufgabenstellung:
    1. Erstelle einen Node.js-Service mit Express.
    2. Verbinde dich mit einer MongoDB-Instanz (in einem eigenen Container).
    3. Zeige über einen HTTP-Endpunkt eine Erfolgsmeldung, wenn MongoDB verbunden ist.

💡   Hinweise:
    • Du brauchst eine Dockerfile, package.json und index.js.

    • Verwende Mongoose zur Verbindung mit MongoDB.
    • MongoDB läuft unter dem Service-Namen mongo.

✅   Lösung:
Dateistruktur:
node-mongo/
├── docker-compose.yml
├── app/
│   ├── Dockerfile
│   ├── index.js
│   └── package.json

Dockerfile:
FROM node:18
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
CMD ["node", "index.js"]

index.js:
const express = require('express');
```

```
const mongoose = require('mongoose');
const app = express();

mongoose.connect('mongodb://mongo:27017/test')
  .then(() => console.log('Connected to MongoDB'))
  .catch(err => console.error('MongoDB connection error:', err));

app.get('/', (req, res) => {
  res.send('Hello from Node.js and MongoDB!');
});

app.listen(3000, () => console.log('Server running on port 3000'));
```

Seite 4

```
Thema: Linux
Technikerschule Erlangen


package.json:
{
    "name": "node-mongo-app",
    "version": "1.0.0",
    "dependencies": {
      "express": "^4.18.2",
      "mongoose": "^7.0.0"
    }
}

docker-compose.yml:
version: '3.8'
services:
  app:
    build: ./app
    ports:
       - "3000:3000"
    depends_on:
       - mongo

    mongo:
      image: mongo
      ports:
        - "27017:27017"
```

Seite 5

```
Thema: Linux



🛠️
Technikerschule Erlangen



      Aufgabe 4: Laravel + MySQL Umgebung
📋    Aufgabenstellung:
Setze eine vollständige Laravel-Entwicklungsumgebung mit MySQL und Docker Compose auf.

💡     Hinweise:
    • Verwende laravel/laravel und mysql:8.

    • Benutze Volumes für persistente Daten.
    • Öffne Laravel auf Port 8000.

✅     Lösung:
docker-compose.yml
version: '3.8'
services:
  app:
    image: laravelsail/php82-composer
    container_name: laravel_app
    working_dir: /var/www/html
    volumes:
       - .:/var/www/html
    ports:
       - "8000:8000"
    depends_on:
       - db
```

```
    command: bash -c "composer install && php artisan serve --host=0.0.0.0 --
port=8000"

  db:
    image: mysql:8.0
    container_name: mysql_db
    environment:
      MYSQL_DATABASE: laravel
      MYSQL_ROOT_PASSWORD: root
      MYSQL_USER: user
      MYSQL_PASSWORD: password
    volumes:
      - dbdata:/var/lib/mysql
    ports:
      - "3306:3306"

volumes:
  dbdata:
```

Seite 6

Thema: Linux

🛠️
Technikerschule Erlangen

Aufgabe 5: Redis als Cache-Service

📋 Aufgabenstellung:
Erstelle einen Redis-Service mit Docker Compose und verbinde ihn mit einer einfachen Node.js-App als Cache Layer.

💡 Hinweise:

- Verwende das Redis-Image.
- Die App soll bei / prüfen, ob Daten gecached sind.
- Wenn nicht, speichert sie etwas in Redis.

✅ Lösung:
index.js

```
const express = require('express');
const redis = require('redis');

const client = redis.createClient({ url: 'redis://redis:6379' });
client.connect();

const app = express();

app.get('/', async (req, res) => {
  const value = await client.get('message');
  if (value) {
    res.send(`Cached: ${value}`);
  } else {
    await client.set('message', 'Hello Redis');
    res.send('Set cache: Hello Redis');
  }
});

app.listen(3000, () => console.log('Server on 3000'));
```

docker-compose.yml

```
version: '3.8'
services:
  app:
    build: .
    ports:
       - "3000:3000"
    depends_on:
       - redis

  redis:
    image: redis:alpine
    ports:
      - "6379:6379"
```

Seite 7

Thema: Linux

🛠️
Technikerschule Erlangen

Aufgabe 6: PostgreSQL + Adminer (Datenbank GUI)

📋 Aufgabenstellung:
Starte eine PostgreSQL-Datenbank mit Adminer zur Verwaltung über den Browser.

💡 Hinweise:

- PostgreSQL Port: 5432
- Adminer Port: 8080

✅ Lösung:
docker-compose.yml

```
version: '3.8'
services:
  db:
    image: postgres:15
    environment:
      POSTGRES_DB: example
      POSTGRES_USER: user
      POSTGRES_PASSWORD: password
    volumes:
      - pgdata:/var/lib/postgresql/data
    ports:
      - "5432:5432"

  adminer:
    image: adminer
    ports:
      - "8080:8080"

volumes:
  pgdata:
```

Seite 8

Thema: Linux

🛠️
Technikerschule Erlangen

Aufgabe 7: RabbitMQ Messaging Queue

📋 Aufgabenstellung:
Starte einen RabbitMQ-Container und sende eine Nachricht von einer Node.js-App.

💡 Hinweise:

- RabbitMQ läuft auf 5672, Web-UI auf 15672
- Nutze das Paket amqplib

✅ Lösung:
docker-compose.yml

```
version: '3.8'
services:
  rabbitmq:
    image: rabbitmq:3-management
    ports:
      - "5672:5672"
      - "15672:15672"

  sender:
    build: ./sender
    depends_on:
      - rabbitmq
```

sender/index.js

```
const amqp = require('amqplib');

(async () => {
  const conn = await amqp.connect('amqp://rabbitmq');
  const ch = await conn.createChannel();
  const q = 'task_queue';
  await ch.assertQueue(q);
  ch.sendToQueue(q, Buffer.from('Hello RabbitMQ'));
  console.log('Message sent');
  await ch.close();
  await conn.close();
})();

sender/Dockerfile
FROM node:18
WORKDIR /app
COPY . .
RUN npm install
CMD ["node", "index.js"]
```

Seite 9

```
Thema: Linux
Technikerschule Erlangen


sender/package.json
{
    "name": "rabbit-sender",
    "dependencies": {
      "amqplib": "^0.10.0"
    }
}
```

sudoers_antworten_mit_fragen.pdf

Quelle: sudoers_antworten_mit_fragen.pdf - 9 Seite(n)

Seite 1

```
Thema: Linux

Technikerschule Erlangen

               Prufungsfragen zur sudoers-Datei - Antworten
Grundlagen
1. Was ist die Datei /etc/sudoers?
   Antwort: Die Datei /etc/sudoers legt fest, welche Benutzer oder Gruppen sudo benutzen
durfen,
   als welcher Zielbenutzer Befehle laufen und welche Kommandos erlaubt oder verboten sind.
2. Warum sollte man die sudoers-Datei nicht direkt mit einem Editor bearbeiten?
   Antwort: Weil schon kleine Syntaxfehler sudo unbrauchbar machen konnen. Dann konnen
   Administratoren sich im schlimmsten Fall aussperren oder Rechte falsch vergeben.
3. Welchen Befehl nutzt man zum sicheren Bearbeiten der sudoers-Datei?
   Antwort: Dafur nutzt man visudo. Das Tool sperrt die Datei wahrend der Bearbeitung und
pruft
   die Syntax vor dem Speichern.

Syntax & Struktur
4. Wie ist eine typische sudoers-Regel aufgebaut?
   Antwort: Typisch ist: Benutzer Host=(Runas) Optionen: Befehl. Beispiel: max ALL=(root)
   /usr/bin/systemctl restart apache2.
5. Was bedeutet ALL in der sudoers-Datei?
   Antwort: ALL bedeutet immer eine Freigabe fur alle Werte im jeweiligen Feld. Je nach
Position
   kann es also alle Hosts, alle Zielbenutzer oder alle Befehle meinen.
6. Was bedeutet (ALL) in einer Regel?
   Antwort: Das Runas-Feld (ALL) bedeutet, dass der Befehl als jeder beliebige Zielbenutzer
   ausgefuhrt werden darf, also nicht nur als root.

Rechte & Sicherheit
7. Was bewirkt die Option NOPASSWD?
   Antwort: NOPASSWD erlaubt die Ausfuhrung der angegebenen Befehle ohne Passwortabfrage.
   Das ist bequem, sollte aber nur sehr gezielt fur wenige sichere Kommandos eingesetzt
werden.
8. Was ist der Unterschied zwischen sudo und su?
   Antwort: sudo fuhrt einzelne Befehle mit erhohten Rechten aus und kann fein granular
geregelt
   werden. su wechselt direkt in ein anderes Benutzerkonto, oft in eine komplette Root-Shell.
```

```
Aliase
9. Welche Alias-Typen gibt es in der sudoers-Datei?
   Antwort: Es gibt User_Alias fur Benutzer, Runas_Alias fur Zielbenutzer, Host_Alias fur
Rechner
   und Cmnd_Alias fur Befehlsgruppen.

Praktische Fragen
10. Wie erlaubst du einem Benutzer nur den Neustart des Systems?
   Antwort: Zum Beispiel mit: max ALL=(root) /usr/sbin/reboot. Noch besser ist die exakte
Freigabe
   nur des einen benotigten Befehls statt allgemeiner Systemrechte.
11. Wie erlaubst du einer Gruppe sudo-Zugriff?
   Antwort: Gruppen werden mit einem Prozentzeichen geschrieben, zum Beispiel: %sudo
   ALL=(ALL) ALL. Dann erhalten alle Mitglieder dieser Gruppe die erlaubten Rechte.

Sicherheit & Best Practices
```

Seite 2

```
12. Warum ist es gefahrlich, ALL=(ALL) ALL zu vergeben?
   Antwort: Weil der Benutzer damit praktisch jede Aktion als jeder Benutzer ausfuhren kann.
Das
   entspricht fast vollem Administrationszugriff und vergroBert das Missbrauchs- und
Fehlerrisiko
   stark.
13. Was macht die Direktive Defaults?
   Antwort: Defaults setzt Standardoptionen fur sudo, zum Beispiel Umgebungsvariablen,
   Passwortverhalten oder TTY-Anforderungen. Solche Vorgaben konnen global oder
   benutzerbezogen gelten.

Erweiterte Fragen
14. Was ist der Unterschied zwischen /etc/sudoers und /etc/sudoers.d/?
   Antwort: /etc/sudoers ist die zentrale Hauptdatei. In /etc/sudoers.d/ liegen zusatzliche
   Einzeldateien, mit denen Rechte sauber modular und besser wartbar verteilt werden konnen.
15. Wie uberpruft man die Syntax der sudoers-Datei?
   Antwort: Mit visudo -c. Der Befehl pruft die Syntax der Hauptdatei und normalerweise auch
die
   eingebundenen Dateien.
```

Seite 3

```
Thema: Linux

Technikerschule Erlangen

                    Szenario- & Analysefragen - Antworten
Szenario- & Analysefragen
16. Was bewirkt folgende Regel genau?
max ALL=(root, www-data) /usr/bin/systemctl restart apache2
   Antwort: Benutzer max darf auf allen Hosts genau den Befehl /usr/bin/systemctl restart
apache2
   ausfuhren, und zwar als root oder als www-data. Andere Befehle sind dadurch nicht erlaubt.
17. Was ist der Unterschied zwischen diesen beiden Regeln?
max ALL=(ALL) ALL
max ALL=ALL ALL
   Antwort: Inhaltlich fast keiner: beide erlauben max alle Befehle als alle Zielbenutzer. Die
   Schreibweise mit Klammern ist die ubliche und klarere Form; ohne Klammern wird historisch
ein
   Standard-Runas angenommen.
18. Welche Regel hat Prioritat und warum?
max ALL=(ALL) ALL
max ALL=(ALL) !/bin/bash
   Antwort: Die einschrankende Regel mit !/bin/bash ist wichtiger, weil Negationen verbotene
   Befehle explizit ausschlieBen. Trotzdem sind solche Mischregeln heikel, weil man Bash oft
uber
   andere Programme indirekt wieder erreichen kann.
19. Was passiert bei folgender Konfiguration?
Cmnd_Alias DANGER = /bin/rm, /bin/dd
max ALL=(ALL) ALL, !DANGER
   Antwort: max darf grundsatzlich alle Befehle ausfuhren, auBer /bin/rm und /bin/dd. Sicher
ist das
   trotzdem nicht, weil ahnliche Wirkungen oft uber andere Werkzeuge oder Shells erreicht
werden
   konnen.

Tiefergehende Syntaxfragen
20. Wie wirken NOPASSWD und PASSWD in Kombination innerhalb einer Regel?
max ALL=(ALL) NOPASSWD: /bin/ls, PASSWD: /bin/cat
   Antwort: Die Option wirkt nur fur die danach genannten Befehle. /bin/ls ware ohne Passwort
```

```
   erlaubt, wahrend fur /bin/cat weiterhin eine Passwortabfrage erfolgt.
21. Was bewirkt folgende Defaults-Zeile?
Defaults:max !requiretty
   Antwort: Fur den Benutzer max wird die TTY-Pflicht deaktiviert. Er darf sudo also auch ohne
   interaktives Terminal verwenden, etwa in Skripten oder per Fernaufruf.
22. Was ist der Unterschied zwischen folgenden Regeln?
max ALL=(ALL:ALL) ALL
max ALL=(ALL) ALL
   Antwort: Mit (ALL:ALL) werden sowohl Zielbenutzer als auch Zielgruppen frei wahlbar. Mit
(ALL)
   ist nur der Zielbenutzer explizit frei; die Gruppenseite bleibt standardmaBig
eingeschrankter.

Edge Cases & Fallen
23. Warum kann folgende Regel ein Sicherheitsrisiko darstellen?
max ALL=(ALL) /usr/bin/vim
```

Seite 4

```
Antwort: Weil man aus vim heraus Shell-Befehle starten oder Dateien mit hohen Rechten
  bearbeiten kann. Dadurch wird aus einem scheinbar harmlosen Editor schnell ein Weg zur
  Privilege Escalation.
24. Was passiert bei dieser Regel und wo liegt ein mogliches Problem?
max ALL=(ALL) /usr/bin/less /var/log/*
   Antwort: max darf Logdateien unter /var/log mit less lesen. Problematisch ist, dass less
externe
   Programme aufrufen kann und Wildcards zusammen mit Dateilinks oder Sonderfallen unerwartete
   Zugriffe erlauben konnen.
25. Warum ist folgende Regel problematisch?
max ALL=(ALL) /usr/bin/python3
   Antwort: Ein Interpreter wie python3 kann beliebige Programme starten, Dateien schreiben
oder
   Shells offnen. Damit entspricht die Freigabe praktisch sehr weitreichendem Vollzugriff.
```

Seite 5

```
Thema: Linux

Technikerschule Erlangen

                  Kombinations- & Logikfragen - Antworten
Kombinations- & Logikfragen
26. Welche Berechtigungen ergeben sich aus folgender Konfiguration?
User_Alias ADMINS = max, anna
Cmnd_Alias WEB = /usr/bin/systemctl restart apache2
ADMINS ALL=(root) WEB
   Antwort: Die Benutzer max und anna durfen auf allen Hosts als root genau den Befehl
   /usr/bin/systemctl restart apache2 ausfuhren. Weitere Kommandos werden dadurch nicht
   freigegeben.
27. Was passiert, wenn mehrere Dateien im Verzeichnis /etc/sudoers.d/ vorhanden sind?
   Antwort: Alle eingebundenen Dateien werden ausgewertet. Deshalb muss man auf Reihenfolge,
   Namensgebung und mogliche Regelkonflikte achten, damit keine unbeabsichtigten Rechte
   entstehen.

Sicherheitsbewertung
28. Welche der folgenden Regeln ist sicherer und warum?
max ALL=(ALL) ALL
vs.
max ALL=(root) /usr/bin/systemctl
   Antwort: Die zweite Regel ist deutlich sicherer, weil sie nur einen bestimmten Befehl als
root
   erlaubt. Das Prinzip der minimalen Rechte wird damit viel besser umgesetzt.

Praxisaufgabe
29. Formuliere eine sudoers-Regel mit folgenden Anforderungen: Benutzer max,
Ausfuhrung als root, apt update und apt upgrade, ohne Passwort
   Antwort: Eine passende Regel ist: max ALL=(root) NOPASSWD: /usr/bin/apt update,
/usr/bin/apt
   upgrade. Wichtig ist die exakte Angabe der erlaubten Kommandos.
```

Seite 6

```
Thema: Linux

Technikerschule Erlangen

      Komplexe Prufungsaufgabe: sudoers Analyse - Antworten
Teil 1: Verstandnisfragen
1. Welche Rechte haben Benutzer aus der Gruppe ADMINS grundsatzlich?
   Antwort: ADMINS umfasst max und anna. Durch die Regel ADMINS ALL=(ALL) ALL durfen
   beide grundsatzlich alle Befehle als alle Zielbenutzer ausfuhren.
```

2. Welche Befehle darf tom konkret ausfuhren?
   Antwort: tom gehort zu DEV und darf dadurch als root die Befehle aus SYSTEM und EDIT
   ausfuhren, also systemctl, journalctl, vim und nano. Zusatzlich darf er laut letzter Regel
less auf
   Dateien unter /var/log verwenden.
3. Darf tom den Befehl /bin/rm ausfuhren? Begrunde.
   Antwort: Nach der DEV-Regel nicht direkt, weil !DANGEROUS den Alias mit /bin/rm und /bin/dd
   ausschlieBen soll. Praktisch bleibt die Konfiguration aber unsicher, weil erlaubte Editoren
oder
   andere Programme dennoch Missbrauch ermoglichen konnen.

Teil 2: Prioritat & Regelkonflikte
4. Welche Regel gilt fur max beim Ausfuhren von Befehlen allgemein? Begrunde anhand
der Reihenfolge.
   Antwort: Fur max gilt insgesamt die Kombination aller passenden Regeln. Allgemein hat er
durch
   ADMINS ALL=(ALL) ALL Vollzugriff; die spateren PASSWD- und NOPASSWD-Regeln steuern
   vor allem das Passwortverhalten fur bestimmte Befehle nach.
5. Kann max den Apache-Service ohne Passwort neu starten? Warum oder warum nicht?
   Antwort: Ja. Fur genau /usr/bin/systemctl restart apache2 existiert explizit eine NOPASSWD-
   Regel, die die Passwortabfrage fur diesen Befehl aufhebt.
6. Welche Auswirkung hat die Kombination aus NOPASSWD: /usr/bin/systemctl restart
apache2 und PASSWD: ALL?
   Antwort: Der Apache-Neustart bleibt ohne Passwort erlaubt, alle anderen per sudo erlaubten
   Befehle erfordern ein Passwort. Die speziellere Ausnahme greift also nur fur das exakt
genannte
   Kommando.

Teil 3: Sicherheitsanalyse
7. Warum ist die EDIT-Alias-Definition kritisch im Kontext von sudo?
   Antwort: Weil Editoren wie vim oder nano nicht nur Text bearbeiten, sondern oft Shell-
Aufrufe,
   Dateizugriffe und Plugins erlauben. Damit lassen sich Beschrankungen leicht umgehen.
8. Welche Sicherheitsprobleme entstehen durch die Regel: anna ALL=(root)
/usr/bin/vim /etc/*
   Antwort: anna darf damit als root beliebige Dateien unter /etc mit vim offnen und
verandern. Aus
   vim heraus kann sie zudem unter Umstanden Shell-Kommandos starten und so weit mehr als nur
   Konfigurationen bearbeiten.
9. Warum ist folgende Regel potenziell unsicher, obwohl sie eingeschrankt wirkt? tom
ALL=(ALL) /usr/bin/less /var/log/*
   Antwort: less wirkt wie ein Lesewerkzeug, kann aber externe Programme aufrufen. Auerdem
   konnen Wildcards und besondere Dateien oder Symlinks zu unerwarteten Zugriffen fuhren.

Teil 4: Syntax & Logikfehler

Seite 7

10. Gibt es einen logischen oder sicherheitstechnischen Fehler in dieser Zeile? DEV
ALL=(root) SYSTEM, EDIT, !DANGEROUS
   Antwort: Ja. Die Regel erlaubt sehr starke Werkzeuge wie Editoren und versucht nur einzelne
   gefahrliche Befehle auszuschlieBen. Das ist logisch schwach, weil viele andere erlaubte
   Programme ebenfalls missbraucht werden konnen.
11. Wird !DANGEROUS hier zuverlassig durchgesetzt? Begrunde.
   Antwort: Nur fur exakt die dort genannten Pfade. Zuverlassig sicher ist das nicht, weil
dieselben
   Wirkungen uber andere Programme, Shells oder alternative Pfade erreichbar bleiben konnen.

Teil 5: Fehler finden
12. Identifiziere mindestens zwei versteckte Probleme oder Risiken in der gesamten
Konfiguration.
   Antwort: Erstens fuhrt ADMINS ALL=(ALL) ALL praktisch zu Vollzugriff. Zweitens sind vim,
nano,
   less und ahnliche Programme als sudo-Befehle riskant, weil sie Shell-Eskalation oder
   Dateimanipulation erlauben.
13. Welche Regel(n) konnten unbeabsichtigt zu vollstandigem Root-Zugriff fuhren?
   Antwort: Sicher die ADMINS-Regel. Praktisch ebenfalls kritisch sind die Freigaben fur vim,
nano
   oder pythonartige Werkzeuge, weil sie sehr leicht zu einer Root-Shell fuhren konnen.

Teil 6: Transfer / Praxis
14. Formuliere eine sichere Alternative fur die DEV-Regel, sodass nur systemctl restart
apache2 erlaubt ist und keine Shell-Eskalation moglich ist.
   Antwort: Eine sichere Alternative ware: tom ALL=(root) /usr/bin/systemctl restart apache2.
Damit
   wird nur ein exakt benannter Befehl freigegeben und keine Shell oder kein Editor erlaubt.
15. Wie wurdest du verhindern, dass Editoren wie vim zur Privilege Escalation genutzt
werden?
   Antwort: Am besten gar keine allgemeinen Editoren per sudo freigeben. Stattdessen nur

einzelne, exakt definierte Verwaltungsbefehle erlauben oder mit sudoedit arbeiten und Shell-
Funktionen vermeiden.

Seite 8

Thema: Linux

Technikerschule Erlangen

Prufungsaufgabe: sudoers mit Syntaxfehlern & Fallen -
Antworten

Teil 1: Syntaxfehler erkennen

1. Identifiziere alle Syntaxfehler, die dazu fuhren, dass visudo die Datei ablehnt.
   Antwort: Fehlerhaft sind zum Beispiel: User_Alias ADMINS = max, anna, wegen des abschlieBenden Kommas; Cmnd_Alias EDIT = /usr/bin/vim /usr/bin/nano wegen fehlendem Komma; Cmnd_Alias DANG = /bin/rm, /bin/dd, wegen des abschlieBenden Kommas; ADMINS ALL=(ALL ALL) ALL wegen falscher Runas-Syntax; DEV ALL=(root) SYS, EDIT !DANG wegen fehlendem Komma; max ALL=(ALL) PASSWD ALL wegen fehlendem Doppelpunkt nach PASSWD.
2. Welche Zeilen enthalten unguItige Alias-Definitionen?
   Antwort: Unzulassig sind User_Alias ADMINS = max, anna, sowie Cmnd_Alias EDIT = /usr/bin/vim /usr/bin/nano und Cmnd_Alias DANG = /bin/rm, /bin/dd,.
3. Wo fehlen notwendige Trennzeichen oder Operatoren?
   Antwort: Es fehlt ein Komma zwischen vim und nano, ein Komma vor !DANG und ein Doppelpunkt nach PASSWD. Genau solche kleinen Zeichen sind in sudoers besonders wichtig.

Teil 2: Tricky Syntax

4. Ist folgende Zeile gultig oder fehlerhaft? Begrunde.
Defaults:anna ! requiretty
   Antwort: Sie ist fehlerhaft, weil zwischen ! und requiretty kein Leerzeichen stehen darf. Korrekt
   ware: Defaults:anna !requiretty.
5. Was ist das Problem bei: ADMINS ALL=(ALL ALL) ALL
   Antwort: Im Runas-Teil fehlt der korrekte Trenner zwischen Benutzer und Gruppe. Richtig ware
   entweder (ALL) oder (ALL:ALL).
6. Warum ist diese Zeile syntaktisch oder logisch problematisch?
DEV ALL=(root) SYS, EDIT !DANG
   Antwort: Syntaktisch fehlt das Komma vor !DANG. Logisch ist die Regel auBerdem problematisch, weil sie mit EDIT sehr machtige Programme erlaubt und nur einzelne Gefahren ausschlieBt.

Teil 3: Semantik trotz Syntax

7. Welche effektiven Rechte hatte max am Ende wirklich?
   Antwort: Wenn alle Syntaxfehler behoben werden, hatte max durch die ADMINS-Regel im Kern Vollzugriff. Zusatzlich ware der Apache-Neustart ohne Passwort erlaubt.
8. Welche Regel wurde fur Passwortabfragen bei max gelten?
   Antwort: Fur /usr/bin/systemctl restart apache2 galt NOPASSWD, fur alle anderen Befehle PASSWD. Also genau ein freigestellter Befehl, sonst Passwortpflicht.

Teil 4: Sicherheitsanalyse

9. Welche Regeln ermoglichen trotz Einschrankungen eine Root-Shell?
   Antwort: Vor allem Regeln mit vim, nano oder ahnlichen interaktiven Programmen. Auch Interpreter oder falsch eingeschrankte Werkzeuge konnen trotz scheinbarer Grenzen schnell zu
   einer Root-Shell fuhren.
10. Warum ist die Kombination aus EDIT und sudo grundsatzlich kritisch?

Seite 9

Antwort: Weil Editoren nicht nur Dateien andern, sondern oft Shell-Kommandos, Plugins oder Dateibrowser enthalten. Damit werden sie unter sudo zu sehr machtigen Werkzeugen.

Teil 5: Fehlerbehebung

11. Korrigiere folgende Zeile vollstandig: Cmnd_Alias EDIT = /usr/bin/vim /usr/bin/nano
   Antwort: Korrektur: Cmnd_Alias EDIT = /usr/bin/vim, /usr/bin/nano
12. Korrigiere die Alias-Definition mit minimaler Anderung: User_Alias ADMINS = max, anna,
   Antwort: Korrektur: User_Alias ADMINS = max, anna
13. Formuliere die DEV-Regel korrekt und sicher (nur Syntax, keine Sicherheitsoptimierung).
   Antwort: Syntaktisch korrekt ware: DEV ALL=(root) SYS, EDIT, !DANG

Teil 6: Transfer

14. Nenne zwei Grunde, warum syntaktisch korrekte sudoers-Dateien trotzdem unsicher sein konnen.
   Antwort: Erstens konnen zu breite Rechte wie ALL=(ALL) ALL vergeben sein. Zweitens konnen scheinbar harmlose Programme wie Editoren, Pager oder Interpreter indirekt eine Rechteausweitung erlauben.

15. Erklare, warum visudo zwar notwendig, aber nicht ausreichend fur Sicherheit ist.
   Antwort: visudo pruft nur die Syntax, nicht die fachliche Sicherheit der Regeln. Eine Datei kann
   also formal korrekt sein und trotzdem gefahrliche oder zu weitreichende Berechtigungen
   enthalten.

TechnikerVorbereitung.pdf

Quelle: TechnikerVorbereitung.pdf - 5 Seite(n)

Seite 1

```
Aufgabe 1: System
11 Punkte
Welcher Verzeichniszweig ist der, bei dem eine Datensicherung am wenigstens Sinn macht?
1P
Kreuzen Sie die richtige Antwort an! Begründen Sie Ihre Antwort!

   /tmp
   /etc
 x /proc
   /var


Werfen Sie einen Blick auf die folgende Ausgabe von top und beantworten Sie die folgenden
Fragen:



Welche Prozesse wurden vom Benutzer carol gestartet?
1P
Nur einer: top. (887)

Welches virtuelle Verzeichnis von /proc sollten Sie aufrufen, um nach Daten des Befehls top zu
suchen?

2P
/proc/887

Welcher Prozess wurde als erstes gestartet? Woher wissen Sie das?
2P

systemd, weil es die PID 1 hat.

Wie könnte der Befehl telinit verwendet werden, um das System neu zu starten?
2P
Der Befehl telinit 6 wechselt zu Runlevel 6, d.h. das System wird neu gestartet.
Was passiert mit den Diensten, die sich auf die Datei /etc/rc1.d/K90network beziehen, wenn das
System
Runlevel 1 aktiviert?
1P
Aufgrund des Buchstabens K am Anfang des Dateinamens werden die entsprechenden Dienste
beendet.
Wie könnte ein Benutzer mit dem Befehl systemctl überprüfen, ob die Unit sshd.service läuft?
Mit dem Befehl systemctl status sshd.service oder systemctl is-active sshd.service.
1P


Basierend auf der Nutzung von systemd: Welcher Befehl muss ausgeführt werden, um die
Aktivierung von
sshd.service während der Systeminitialisierung zu ermöglichen?
1P
Der Befehl systemctl enable sshd.service wird von root ausgeführt.
```

Seite 2

```
Aufgabe 2: Shell-Umgebung
7 Punkte
Geben Sie im Folgenden die entsprechenden Befehle für die angegebene Aufgabe an.


Erzeugen Sie eine lokale Variable namens mammal und weisen Sie ihr den Wert gnu zu:
1P
mammal="gnu"


Die Variable var_sub soll einen String in folgendem Format enthalten:
1P
        The value of mammal is gnu
Geben Sie die Befehlszeile an die, die Variable var_sub auf diesen Wert setzt. Verwenden Sie
dabei die
Variable mammal, um den String gnu zu erhalten.
var_sub="The value of mammal is $mammal" oder var_sub="The value of mammal is " $mammal
```

Machen Sie mammal zu einer Umgebungsvariablen:
1P
export mammal

Suchen Sie mit grep nach dieser Umgebungsvariable:
1P
set | grep mammal oder env | grep mammal

Sie sind als user2 eingeloggt. Erstellen Sie ein Verzeichnis namens bin in Ihrem Homeverzeichnis. 1P
mkdir ~/bin oder mkdir /home/user2/bin oder mkdir $HOME/bin

Das gerade angelegte Verzeichnis soll das Verzeichnis werden, in dem sie als erstes nach ausführbaren
Dateien suchen. Geben Sie den Befehl, der die entsprechende Umgebungsvariable anpasst.
1P
PATH="$HOME/bin:$PATH" PATH=~/bin:$PATH oder PATH=/home/user2/bin:$PATH

Geben Sie eine if-Anweisung an, die Sie in ~/.profile einfügen, um sicherzustellen, dass der Wert von der
gerade veränderten Umgebungsvariable über Neustarts hinweg unverändert bleibt.
1P

```
if [ -d "$HOME/bin" ] ; then
        PATH="$HOME/bin:$PATH"
fi
```

Seite 3

Aufgabe 3: Befehle
17 Punkte

Legen Sie mit genau einem Befehlsaufruf 5 nummerierte Dateien mit dem Präfix games an (games1, games2,…).
1,5 P

touch games{1..5}

Löschen Sie mit genau einem Befehlsaufruf die gerade erstellten 5 Dateien.
1,5 P

rm games? Oder rm games[12345]

Entfernen Sie die gesamten Verzeichnisbäume unterhalb der Verzeichnisse, die mit Test_Verzeichnis begin-
nen (z.B. Test_Verzeichnis1 oder auch Text_Verzeichnis_x uvm.) mit nur einem Befehl. 1P

rm -r Test_Verzeichnis*

Piping ist eine Möglichkeit zwei Befehle miteinander interagieren zu lassen. Erklären Sie das Prinzip. 2P

Piping sendet stdout von einem Befehl als stdin des anderen Befehls.

Erklären Sie folgende Befehlszeile möglichst genau:
2P

ls -l | head | wc -w

ls -l gibt langform des Verzeichnislisting an head weiter, der die ersten 10 Zeilen nimmt und an wc gibt, der
Wörter zählt.

Suchen Sie im Verzeichnisbaum nach allen Dateien, die mit einer Ziffer enden.
2P

find ~ -name "*[0-9]" -type f
oder ls | find -name "*[0-9]"

Erklären Sie folgende Befehlszeile möglichst genau:
2P

sort /etc 2 > /dev/null

Fehlerausgabe (2) wird in die Mülltonne gelenkt

Hängen Sie die letzten 9 Zeilen der Datei contents.txt, an die Datei dump.txt an
2P

tail -n 9 contents.txt » dump.txt

Schreiben Sie einen Befehl, der im aktuellen Verzeichnis nach Dateien mit der Endung .c sucht, in denen im
Dateiinhalt die Zeichenfolge „apple" enthalten ist. (Ignorieren Sie dabei Groß- und Kleinschreibung) 3P

find ./ -name "*.c" | grep -i "apple"

Seite 4

Aufgabe 4: Benutzerverwaltung
8 Punkte

Gegeben ist die folgende Ausgabe. Beantworten Sie folgende Fragen:

Wie lauten die Benutzer-ID (UID) und die Gruppen-ID (GID) von root und catherine?
1P
Die UID und GID von root sind 0 und 0, während die UID und GID von catherine 1030 und 1025 sind.

Wie lautet der Name der primären Gruppe von kevin? Geben Sie weitere Mitglieder der Gruppe an.
1,5 P
Der Gruppenname lautet db-admin. Auch emma und grace sind in dieser Gruppe.

Welche Shell ist für mail eingestellt? Warum ist die Shell für Benutzer mail auf diesen Wert gesetzt? 1,5 P

mail ist ein Systembenutzerkonto und seine Shell ist /sbin/nologin. Tatsächlich werden Systembenutzerkon-
ten wie mail, ftp, news und daemon für administrative Aufgaben verwendet und daher sollte die normale An-
meldung für diese Konten verhindert werden. Aus diesem Grund wird die Shell normalerweise auf /sbin/nolo-
gin oder /bin/false gesetzt.

Seite 5

Fortsetzung Aufgabe 4:

Geben Sie die Mitglieder der Gruppe app-developer an! Welche davon sind Gruppenadministratoren und
welche sind normale Mitglieder?
1P
Die Mitglieder sind catherine, dave und christian — alle sind ordentliche Mitglieder.

Welche IDs werden per Konvention den Systemkonten und welche den normalen Benutzern zugewiesen?
Welche ID hat root?
2P

Systemkonten haben in der Regel UIDs unter 100 oder zwischen 500 und 1000, während normale Benutzer
UIDs haben, die bei 1000 beginnen, obwohl einige Altsysteme die Nummerierung bei 500 beginnen können.
Der Benutzer root hat die UID 0.

Ihr System verwendet Shadow-Passwörter. Was heisst das?
1P

Wenn Shadow-Passwörter verwendet werden, enthält das zweite Feld in /etc/passwd für jedes

```
Benutzerkon-
to das Zeichen x, da die verschlüsselten Benutzerpasswörter in /etc/shadow gespeichert werden.
```

TechnikerVorbereitungawk2Lsg.pdf

Quelle: TechnikerVorbereitungawk2Lsg.pdf - 4 Seite(n)

Seite 1

Wiederholung awk

1. Was ist AWK und wofür wird es verwendet?
AWK ist eine Skriptsprache zur Textverarbeitung, die besonders für das Filtern, Analysieren und Formatieren
von strukturierten Daten (z. B. Tabellen, Logfiles) verwendet wird.

2. Wie ist ein AWK-Programm grundsätzlich aufgebaut?
Schema:
pattern { action }

- pattern: Bedingung (wann wird etwas ausgeführt)
- action: Anweisung (was wird gemacht)

3. Was bewirkt folgender Befehl?

```
awk '{print $1}' datei.txt
```

Gibt die erste Spalte jeder Zeile aus.

4. Bedeutung von $0, $1, $NF?

- $0 → ganze Zeile
- $1 → erstes Feld
- $NF → letztes Feld

5. Was macht dieser Ausdruck?

```
awk '$3 > 100'
```

Gibt alle Zeilen aus, bei denen das 3. Feld größer als 100 ist.

6. Wie filtert man nach einem bestimmten Wort?

```
awk '/Fehler/'
```

Zeigt alle Zeilen, die „Fehler" enthalte

7. Was macht folgendes Skript?

```
awk '{sum += $2} END {print sum}'
```

Summiert die zweite Spalte und gibt das Ergebnis am Ende aus.

Seite 2

8. Unterschied zwischen BEGIN und END?

- BEGIN → wird vor der Verarbeitung ausgeführt
- END → wird nach der Verarbeitung ausgeführt

9. Wie ändert man das Feldtrennzeichen?

```
awk -F ":" '{print $1}'
```

Trennt Felder anhand von : statt Leerzeichen.

10. Was macht dieses Beispiel?

```
awk '{if ($2 > 50) print $1, $2}'
```

Gibt nur Zeilen aus, bei denen Spalte 2 > 50 ist, und zeigt Spalte 1 und 2.

11. Wie zählt man die Anzahl der Zeilen?

```
awk 'END {print NR}'
```

NR = Number of Records (Zeilenanzahl)

12. Wie gibt man nur eindeutige Werte aus?

```
awk '!seen[$1]++'
```

Gibt jede erste Spalte nur einmal aus.

13. Aufgabe:
Gegeben ist eine Datei mit Name und Punktzahl. Berechne den Durchschnitt.

```
awk '{sum += $2; count++} END {print sum/count}'
```

14. Was ist der Unterschied zwischen AWK und grep?

- grep → nur Suchen
- AWK → Suchen + Verarbeiten + Berechnen

Seite 3

15. Logdateien auswerten.

Gegeben: Mehrere Logdateien in /var/log/app/

Ziel:

- Nur Dateinamen (ohne Pfad) ausgeben
- Fehlerzeilen (ERROR) filtern
- Anzahl Fehler pro Datei berechnen
- Ergebnis gleichzeitig anzeigen und in report.txt speichern

```
for file in /var/log/app/*.log; do
  name=$(basename "$file")
  count=$(awk '/ERROR/ {c++} END {print c+0}' "$file")
  echo "$name: $count"
done | tee report.txt
```

Was wird geprüft:

- basename → entfernt Pfad
- awk → zählt Fehler
- tee → Ausgabe + Speicherung gleichzeitig

15. CSV analysieren + bedingte Ausgabe

Gegeben: Datei daten.csv

```
id,name,punkte
1,Anna,45
2,Bob,78
3,Chris,30
```

Ziel:

- Nur Datensätze mit Punkte > 50
- Ausgabeformat: Dateiname: Name (Punkte)
- Ergebnis in Datei und Konsole

Seite 4

Lösung:

```
file="daten.csv"
awk -F, '$3 > 50 {print $2 " (" $3 ")"}' "$file" \
| sed "s/^/$(basename "$file"): /" \
| tee output.txt
```

16. Mehrere Dateien + Summenbildung

👉 Ziel:

- Für jede Datei die Summe der 2. Spalte
- Nur Dateiname anzeigen
- Gesamtsumme aller Dateien am Ende

Lösung:

```
total=0

for f in *.txt; do
  name=$(basename "$f")
  sum=$(awk '{s += $2} END {print s+0}' "$f")
  echo "$name: $sum"
  total=$((total + sum))
done | tee result.txt

echo "TOTAL: $total" | tee -a result.txt
```

17. Analysiere folgendes:

```
for f in *.txt; do
  base=$(basename "$f" .txt)
  awk 'NF < 3' "$f" | tee "${base}_errors.txt"
done
```

TechnikerVorbereitungsedLsg.pdf

Quelle: TechnikerVorbereitungsedLsg.pdf - 3 Seite(n)

Seite 1

Wiederholung sed

1. Nur bestimmte Zeilen ändern

Gegeben: Datei mit Logeinträgen
Ersetze „ERROR" durch „WARNUNG", aber nur in Zeilen 5–10

```
sed '5,10 s/ERROR/WARNUNG/' logfile.txt
```

2. Nur erste Übereinstimmung pro Zeile ersetzen

Ersetze nur das erste Vorkommen von „foo" in jeder Zeile

```
sed 's/foo/bar/'
```

3. Alle Vorkommen ersetzen

Jetzt aber alle „foo" ersetzen

```
sed 's/foo/bar/g'
```

4. Nur Zeilen anzeigen, die NICHT passen

Zeige alle Zeilen, die kein „ERROR" enthalten

```
sed '/ERROR/d'
```

5. Bestimmte Zeilen extrahieren

Nur Zeilen 10–20 anzeigen

```
sed -n '10,20p' file.txt
```

- -n unterdrückt Standardausgabe
- p druckt explizit
-

Seite 2

6. Mehrere Befehle kombinieren

Ersetze „foo" durch „bar" UND lösche leere Zeilen

```
sed -e 's/foo/bar/g' -e '/^$/d'
```

7. In-place bearbeiten

Datei direkt ändern

sed -i 's/foo/bar/g' file.txt

8. Mit Capture Groups arbeiten

Format ändern: Name: Max → Max (Name)

sed 's/\(.*\): \(.*\)/\2 (\1)/'

9. Nur bestimmte Spalte ersetzen

Ersetze nur im 2. Feld (durch Leerzeichen getrennt)

sed 's/^\([^]* \)foo/\1bar/'

10. Zeilen einfügen

Füge nach jeder Zeile mit „START" eine neue Zeile „---" ein

sed '/START/a ---'

Seite 3

1. Unterschied zwischen sed und AWK?

- sed → Stream Editor (Text ersetzen, löschen)
- AWK → datenorientierte Verarbeitung (Spalten, Berechnungen)

2. Was macht -n?
Unterdrückt automatische Ausgabe

3. Unterschied p vs. d?

p → drucken

d → löschen + nächste Zeile

4. „Zeilen, die nicht mit # beginnen" ersetzen.

techniker_vorbereitung2_mit_antworten.pdf

Quelle: techniker_vorbereitung2_mit_antworten.pdf - 4 Seite(n)

Seite 1

```
Aufgabe 4: crontab
8 Punkte

Folgendes Shellskript clean_backup.sh liegt vor:
   #!/bin/bash
   find "/backupfolder" -type f -mtime +5 -exec rm {} \;
a) Wozu dient die erste Zeile des Skripts?
  Antwort: Die erste Zeile ist der sogenannte Shebang. Sie legt fest, dass das Skript mit
/bin/bash
  ausgeführt werden soll. Dadurch wird sichergestellt, dass die Bash als Interpreter verwendet
wird und
  die Shell-Befehle korrekt verarbeitet werden.
b) Was sind die Auswirkungen des Skripts?
  Antwort: Das Skript durchsucht das Verzeichnis /backupfolder nach normalen Dateien. Alle
Dateien,
  deren Änderungsdatum mehr als 5 Tage zurückliegt, werden mit rm gelöscht. Unterordner selbst
  werden dabei nicht gelöscht, sondern nur passende Dateien.
c) Für die Ausführung des Shellskript clean_backup.sh sind Root-Rechte notwendig. Geben Sie
die Befehle an, um root zum Besitzer der Datei clean_backup.sh zu machen und die Ausführung
unter root – also unter den Rechten des Besitzers – zu gewährleisten.
  Antwort: Möglich wäre:
  sudo chown root clean_backup.sh
  sudo chmod u+s clean_backup.sh
  Der erste Befehl setzt root als Eigentümer. Der zweite setzt das Setuid-Bit, sodass das
Skript mit den
  Rechten des Besitzers laufen soll. In der Praxis ist Setuid bei Shellskripten jedoch oft aus
  Sicherheitsgründen deaktiviert; sicherer wäre die Ausführung über sudo oder einen root-
Cronjob.
d) Sie wollen das Skript clean_backup.sh automatisch ausführen lassen. Notieren Sie den
notwendigen Eintrag in der Datei crontab, um das Skript jeden Dienstag und Freitag einmal pro
Stunde zwischen 15:00 und 17:00 auszuführen.
  Antwort: Ein möglicher Eintrag lautet:
  0 15,16,17 * * 2,5 /pfad/zu/clean_backup.sh
  Das bedeutet: Minute 0, also jeweils zur vollen Stunde, an den Stunden 15, 16 und 17, an
jedem Tag
  und Monat, aber nur an den Wochentagen Dienstag (2) und Freitag (5).
e) Mit welchem Befehl geben Sie Ihre aktuelle crontab am Bildschirm aus? Leiten Sie die
Ausgabe in die Datei crontab.bak um.
  Antwort:
  crontab -l > crontab.bak
  Mit crontab -l wird die aktuelle Crontab angezeigt. Durch die Umleitung mit > wird die
Ausgabe in die
  Datei crontab.bak geschrieben.



                                                Seite 1 von 4
```

Seite 2

```
Aufgabe 5: Software Installation
14 Punkte

Advanced Package Tool (APT) ist der Paketmanager für Debian- und Ubuntu-Linux-Distributionen.
a) Welche Datei regelt dabei, welche Online-Repositories bei der Software-Installation von APT
verwendet werden?
  Antwort: Die zentrale Datei ist /etc/apt/sources.list. Zusätzlich können weitere
Paketquellen in Dateien
  unter /etc/apt/sources.list.d/ eingetragen sein.
b) Wie sollten Sie vorgehen, wenn Sie den Webserver apache2 installieren wollen? Geben Sie die
Kommandos an.
  Antwort:
  sudo apt update
  sudo apt install apache2
  Zuerst wird die Paketliste aktualisiert, damit der Paketmanager den aktuellen Stand der
Repositories
  kennt. Anschließend wird das Paket apache2 installiert.
c) Nennen Sie drei distributionsunabhängige Paketformate und erklären Sie sie stichpunktartig.
  Antwort:
  AppImage: Eine einzelne ausführbare Datei, die meist ohne klassische Installation gestartet
werden
  kann.
```

Snap: Ein von Canonical unterstütztes Paketformat mit Sandbox-Ansatz und zentralem Snap Store.
Flatpak: Ebenfalls ein distributionsübergreifendes Format mit Fokus auf Desktop-Anwendungen und
isolierter Ausführung in einer Sandbox.
d) Wie installieren Sie mit einem Befehl spotify im Snap-Format?
Antwort:
sudo snap install spotify
Damit wird das Spotify-Paket aus dem Snap-Store installiert.
e) Welche Aufgabe hat die Datei /etc/sudoers?
Antwort: Die Datei /etc/sudoers legt fest, welche Benutzer oder Gruppen sudo verwenden dürfen und
welche Befehle sie mit erhöhten Rechten ausführen dürfen. Sie ist damit ein zentrales Sicherheits-
und Berechtigungskonfigurationsfile.
f) Was bedeutet die folgende Zeile aus der /etc/sudoers-Datei?
%admin ALL=(ALL) ALL
Antwort: Alle Benutzer, die Mitglied der Gruppe admin sind, dürfen auf allen Hosts bzw. in allen
passenden Kontexten alle Befehle als alle Benutzer ausführen. In der Praxis bedeutet das meist
weitreichende Administratorrechte, typischerweise auch Root-Rechte.
g) Mit welchem Befehl bearbeitet man die /etc/sudoers-Datei?
Antwort:
visudo
Dieser Befehl öffnet die sudoers-Datei in einem sicheren Editor und prüft beim Speichern direkt die
Syntax.
h) Was sind die Unterschiede der Befehle su und sudo in der Anwendung?
Antwort: su wechselt in der Regel auf einen anderen Benutzer, meist root, und verlangt normalerweise
das Passwort des Zielbenutzers. sudo führt dagegen gezielt einen einzelnen Befehl mit erhöhten
Rechten aus und verwendet normalerweise das eigene Benutzerpasswort. sudo ist feiner
konfigurierbar und protokollierbarer, während su eher einen vollständigen Benutzerwechsel bis zum
Abmelden oder bis exit darstellt.

Seite 3

Aufgabe 6: Shellskript
12 Punkte

a) Gegeben sei folgendes Shellskript:

```
#! /bin/sh
readcmd() {
 echo "Anzahl der Parameter in der Kommandozeile : $#"
 for var in $*
 do
  echo "$i. Parameter : $var"
  i=`expr $i + 1`
 done
}
echo "Vor der Funktion ..."
readcmd $*
echo "... nach der Funktion"
```

Geben Sie an, was bei folgender Ausführung ausgegeben wird:
you@host > ./afunc7 eins zwei drei vier
Antwort:
Vor der Funktion ...
Anzahl der Parameter in der Kommandozeile : 4
. Parameter : eins
1. Parameter : zwei
2. Parameter : drei
3. Parameter : vier
... nach der Funktion
Begründung: Es werden vier Parameter übergeben. Die Variable i ist anfangs leer, daher beginnt die
erste Ausgabe mit einem leeren Zähler vor dem Punkt. Danach wird i schrittweise erhöht.
b) Schreiben Sie ein Shellskript, welches Zeilenduplikate in einer Datei erkennt und löscht. Dabei
wird der Dateiname eingelesen und überprüft, ob die Datei vorhanden ist. Anschließend werden die Zeilenduplikate gelöscht, sodass jede Zeile nur einmal vorhanden ist. Dann wird das Ergebnis
in die Datei sorted.txt geschrieben. Die ursprüngliche Datei bleibt unverändert erhalten. (Tipp:

```
Reihenfolge der Zeilen in sorted.txt ist egal)
  Antwort:
  #! /bin/sh
  echo -n "Enter Filename-> "
  read filename
  if [ -f "$filename" ]; then
      sort "$filename" | uniq > sorted.txt
  else
      echo "No $filename in $PWD...try again"
  fi
  exit 0
  Erläuterung: Der Dateiname wird eingelesen. Mit -f wird geprüft, ob eine normale Datei
existiert. sort
  sortiert die Zeilen, uniq entfernt danach doppelte Zeilen. Das Ergebnis wird in sorted.txt
gespeichert,
  die Originaldatei bleibt unverändert.
```


Seite 4

```
Aufgabe 7: Docker
18 Punkte

a) Erklären Sie die Unterschiede zwischen virtueller Maschine und Docker-Container bezüglich
Ressourcenverteilung.
  Antwort: Eine virtuelle Maschine bringt ein komplettes Gastbetriebssystem mit. Dadurch
benötigt sie
  mehr Arbeitsspeicher, mehr Speicherplatz und meist auch mehr Startzeit. Docker-Container
teilen sich
  dagegen den Kernel des Host-Systems und kapseln nur die Anwendung samt benötigter
Bibliotheken.
  Deshalb sind Container leichter, starten schneller und nutzen die vorhandenen Ressourcen in
der
  Regel effizienter.
b) Schreiben Sie ein Dockerfile für ein Docker-Image für einen Apache-Webserver unter Port 80
in
einem Ubuntu-Docker-Container. Dazu apache2 installieren und folgende Umgebungsvariablen
setzen: APACHE_RUN_USER=www-data, APACHE_RUN_GROUP=www-data,
APACHE_LOG_DIR=/var/log/apache2. Darin starten Sie beim Erstellen des Containers den
apache2-Dienst.
  Antwort:
  FROM ubuntu
  RUN apt update && apt-get install -y apache2 && apt-get clean
  ENV APACHE_RUN_USER=www-data
  ENV APACHE_RUN_GROUP=www-data
  ENV APACHE_LOG_DIR=/var/log/apache2
  EXPOSE 80
  CMD ["apachectl", "-D", "FOREGROUND"]
  Hinweis: Für Container ist es besser, den Webserver im Vordergrund laufen zu lassen. service
  apache2 start startet den Dienst nur kurzzeitig und beendet den Container meist wieder.
c) Schreiben Sie folgende Docker-Container-Einstellung als Docker Compose um und verwenden
Sie Version 2.
docker run --name apache -p 80:8080 -p 443:8443 -v /path/to/app:/app bitnami/apache:latest
  Antwort:
  version: "2"
  services:
   apache:
    image: bitnami/apache:latest
    container_name: apache
    ports:
      - "80:8080"
      - "443:8443"
    volumes:
      - /path/to/app:/app
  Diese Compose-Datei bildet dieselben Einstellungen wie der docker-run-Befehl ab:
Containername,
  Image, Portweiterleitungen und Volume-Mount.
```

techniker_vorbereitung_find_mit_antworten.pdf

Quelle: techniker_vorbereitung_find_mit_antworten.pdf - 1 Seite(n)

Seite 1

Wiederholung find

1. Was ist die Hauptfunktion des Befehls find in Linux?
   Antwort: Der Befehl find dient dazu, Dateien und Verzeichnisse rekursiv zu durchsuchen. Dabei kann
   nach verschiedenen Kriterien wie Name, Typ, Größe, Änderungszeit, Besitzer oder Rechten gesucht
   werden.
2. Welche Option wird mit find verwendet, um nach einem Dateinamen zu suchen? A) -user B) -size
C) -name D) -exec
   Antwort: Richtig ist C) -name. Mit -name wird nach einem konkreten Dateinamen oder nach Mustern
   mit Platzhaltern wie *.txt oder backup* gesucht.
3. Was bewirkt folgender Befehl? find /home -type f -name "*.pdf"
   Antwort: Der Befehl durchsucht /home und alle Unterverzeichnisse nach normalen Dateien (-type f),
   deren Name auf .pdf endet. Es werden also alle PDF-Dateien unterhalb von /home gefunden.
4. Finde alle Dateien im aktuellen Verzeichnis, die größer als 10 MB sind.
   Antwort: Befehl: find . -type f -size +10M. Der Punkt steht für das aktuelle Verzeichnis, -type f
   beschränkt die Suche auf Dateien und +10M bedeutet größer als 10 Megabyte.
5. Finde alle Dateien mit der Endung .log in /var/log und lösche sie.
   Antwort: Befehl: find /var/log -type f -name "*.log" -delete. Damit werden alle .log-Dateien in /var/log
   und darunter direkt gelöscht. Vorsicht: -delete sollte nur verwendet werden, wenn das Suchmuster
   sicher stimmt.
6. Suche im Verzeichnis /etc alle Dateien, die in den letzten 7 Tagen geändert wurden.
   Antwort: Befehl: find /etc -type f -mtime -7. Die Option -mtime -7 findet Dateien, deren Änderungszeit
   weniger als 7 Tage zurückliegt.
7. Finde alle Verzeichnisse, deren Name mit „backup“ beginnt.
   Antwort: Befehl: find . -type d -name "backup*". Mit -type d werden nur Verzeichnisse gesucht und das
   Muster backup* findet alle Namen, die mit backup anfangen.
8. Suche nach Dateien, die genau 100 KB groß sind.
   Antwort: Befehl: find . -type f -size 100k. Damit werden Dateien gesucht, deren Größe genau 100
   Kilobyte beträgt. Das kleine k steht hier für Kilobyte in der find-Syntax.
9. Finde alle Dateien, die dem Benutzer student gehören.
   Antwort: Befehl: find . -type f -user student. Damit werden alle Dateien im aktuellen Verzeichnis und
   seinen Unterordnern gesucht, deren Besitzer der Benutzer student ist.
10. Führe für jede gefundene Datei mit Endung .tmp den Befehl rm aus.
   Antwort: Befehl: find . -type f -name "*.tmp" -exec rm {} \;. Für jede gefundene .tmp-Datei wird der
   Befehl rm einzeln ausgeführt. {} steht dabei für die jeweils gefundene Datei.
11. Was bedeutet {} \; in folgendem Befehl? find . -name "*.sh" -exec chmod +x {} \;
   Antwort: Die geschweiften Klammern {} sind ein Platzhalter für jede gefundene Datei. Das
   abschließende \; beendet den mit -exec gestarteten Befehl. Dadurch wird chmod +x nacheinander auf
   jede gefundene .sh-Datei angewendet.
12. Wie kann man mit find nur in einem bestimmten Verzeichnistiefenbereich suchen? (z. B. nur erste
Ebene)
   Antwort: Dafür verwendet man zum Beispiel -maxdepth und -mindepth. Mit find . -maxdepth 1 -name
   "*.txt" wird nur im aktuellen Verzeichnis gesucht, also ohne tiefer in Unterordner zu gehen.

techniker_vorbereitung_grep_mit_antworten.pdf

Quelle: techniker_vorbereitung_grep_mit_antworten.pdf - 2 Seite(n)

Seite 1

Wiederholung grep
1. Was ist der Zweck des Befehls grep?
   Antwort: grep dient dazu, Textdateien oder Eingaben nach einem bestimmten Suchmuster zu
   durchsuchen. Das Muster kann ein normales Wort, eine Zeichenfolge oder auch ein regulärer
Ausdruck
   sein. So lassen sich passende Zeilen schnell finden und ausgeben.
2. Welcher Befehl sucht nach dem Wort "Fehler" in der Datei log.txt, ohne auf
Groß-/Kleinschreibung zu achten?
   Antwort: Richtige Antwort: B) grep -i Fehler log.txt
   Die Option -i sorgt dafür, dass grep nicht zwischen Groß- und Kleinschreibung
unterscheidet. Dadurch
   werden zum Beispiel sowohl „Fehler" als auch „fehler" oder „FEHLER" gefunden.
3. Was bewirkt grep -v "root" /etc/passwd?
   Antwort: Der Befehl gibt alle Zeilen aus der Datei /etc/passwd aus, in denen das Wort root
nicht
   vorkommt. Die Option -v kehrt die Suche also um und filtert passende Zeilen heraus.
4. Suchen Sie in der Datei system.log nach allen Zeilen, die den Begriff „error" enthalten.
   Antwort: grep "error" system.log
   Dieser Befehl zeigt alle Zeilen der Datei system.log an, in denen der Text error vorkommt.
5. Zeigen Sie alle Zeilen in users.txt, die mit dem Buchstaben „A" beginnen.
   Antwort: grep "^A" users.txt
   Das Zeichen ^ steht für den Zeilenanfang. Deshalb werden nur Zeilen angezeigt, deren erstes
Zeichen
   ein großes A ist.
6. Suchen Sie in der Datei daten.csv alle Zeilen, die mit einer Zahl beginnen.
   Antwort: grep "^[0-9]" daten.csv
   Der Ausdruck [0-9] steht für eine beliebige Ziffer. Zusammen mit ^ bedeutet das: Zeige alle
Zeilen, die
   direkt am Anfang mit einer Zahl starten.
7. Suchen Sie rekursiv im Verzeichnis /var/log nach dem Begriff „failed".
   Antwort: grep -r "failed" /var/log
   Mit der Option -r werden das angegebene Verzeichnis und alle Unterverzeichnisse rekursiv
durchsucht.
   grep sucht dabei in allen passenden Dateien nach dem Begriff failed.
8. Zählen Sie, wie oft das Wort „login" in der Datei auth.log vorkommt.
   Antwort: grep -o "login" auth.log | wc -l
   Die Option -o gibt jeden einzelnen Treffer separat aus. Diese Treffer werden an wc -l
weitergegeben,
   das dann die Anzahl der Zeilen und damit die Anzahl der Vorkommen zählt.
9. Wie würden Sie alle Zeilen mit genau 8 Zeichen anzeigen? (Zeichenanzahl = 8)
   Antwort: grep -x ".\{8\}" datei.txt
   Der Ausdruck .\{8\} steht für genau acht beliebige Zeichen. Die Option -x sorgt dafür, dass
die gesamte
   Zeile genau diesem Muster entsprechen muss.
10. Verwenden Sie grep, um in einer Datei users.txt nur Zeilen auszugeben, die NICHT das
Wort "admin" enthalten.
  Antwort: grep -v "admin" users.txt
  Auch hier bedeutet -v, dass alle Zeilen ohne den Suchbegriff ausgegeben werden. Zeilen mit
dem Wort
  admin werden also ausgeschlossen.
11. Wozu dient der Befehl grep -l ?
  Antwort: Die Option -l gibt nur die Namen der Dateien aus, in denen ein Suchmuster gefunden
wurde.
  Die eigentlichen Trefferzeilen werden dabei nicht angezeigt. Das ist praktisch, wenn man nur
wissen
  möchte, in welchen Dateien ein Begriff vorkommt.
12. Geben Sie ein Beispiel an.

Seite 1

Seite 2

Antwort: grep -l "Fehler" *.log
  Dieser Befehl durchsucht alle Dateien mit der Endung .log im aktuellen Verzeichnis.
Anschließend
  werden nur die Dateinamen ausgegeben, in denen das Wort Fehler gefunden wurde.
13. Was macht folgender Befehl? grep -rl "TODO" .
  Antwort: Der Befehl durchsucht das aktuelle Verzeichnis . und alle Unterordner rekursiv nach
dem
  Begriff TODO. Durch die Kombination aus -r und -l werden nur die Dateinamen angezeigt, in

```
denen
  mindestens ein Treffer vorkommt.
14. Was macht folgender Befehl? grep -li "warnung" *.txt
  Antwort: Dieser Befehl durchsucht alle .txt-Dateien im aktuellen Verzeichnis nach dem Wort
warnung,
  ohne zwischen Groß- und Kleinschreibung zu unterscheiden. Wegen -l werden nur die Dateinamen
  ausgegeben, nicht die einzelnen Zeilen mit dem Treffer.



Seite 2
```

techniker_vorbereitung_systemd_mit_antworten.pdf

Quelle: techniker_vorbereitung_systemd_mit_antworten.pdf - 2 Seite(n)

Seite 1

Prüfungsfragen zu systemd
1. Was ist systemd und welche Aufgabe erfüllt es?
Antwort: systemd ist das Init-System und der Service-Manager vieler Linux-Distributionen. Es startet
beim Booten die notwendigen Dienste, überwacht laufende Prozesse und verwaltet deren Status während des Betriebs. Außerdem stellt es Werkzeuge bereit, um Dienste kontrolliert zu starten, zu
stoppen, neu zu laden oder beim Systemstart automatisch zu aktivieren.
2. Was ist eine Unit-Datei? Nenne Beispiele.
Antwort: Eine Unit-Datei ist eine Konfigurationsdatei, mit der systemd festlegt, wie eine bestimmte
Ressource oder Funktion verwaltet wird. Beispiele sind .service für Dienste, .socket für Socket-
Aktivierung, .target für Zielzustände ähnlich den früheren Runleveln, .mount für Einhängepunkte
und .timer für zeitgesteuerte Aufgaben.
3. Wo befinden sich systemd-Unit-Dateien?
Antwort: Häufig liegen benutzerdefinierte oder lokal angepasste Units unter /etc/systemd/system. Vom
Paketmanager installierte Standard-Units liegen meist unter /usr/lib/systemd/system oder je nach
Distribution unter /lib/systemd/system. Änderungen in /etc haben in der Praxis Vorrang, weil dort lokale
Anpassungen vorgenommen werden.
4. Wie startest du einen Dienst manuell?
Antwort: Mit einem Befehl wie systemctl start nginx. Dadurch wird der Dienst sofort gestartet, aber noch
nicht automatisch für den nächsten Bootvorgang aktiviert.
5. Wie aktivierst du einen Dienst beim Booten?
Antwort: Mit systemctl enable nginx. Damit wird der Dienst so eingerichtet, dass er beim Systemstart
automatisch geladen und gestartet wird. Der Befehl ändert also die Boot-Konfiguration, startet den
Dienst aber nicht zwingend sofort.
6. Unterschied zwischen start und enable?
Antwort: start wirkt sofort und startet den Dienst in der aktuellen Sitzung. enable sorgt dafür, dass der
Dienst beim nächsten Systemstart automatisch startet. In der Praxis werden oft beide Befehle kombiniert, wenn ein Dienst sofort laufen und dauerhaft aktiviert sein soll.
7. Wie prüfst du den Status eines Dienstes?
Antwort: Mit systemctl status nginx. Der Befehl zeigt unter anderem, ob der Dienst aktiv ist, wann er
gestartet wurde, ob Fehler aufgetreten sind und welche letzten Log-Ausgaben zugeordnet wurden.
8. Was ist ein Target?
Antwort: Ein Target ist eine Sammlung von Units, die einen bestimmten Systemzustand beschreibt.
Beispiele sind multi-user.target für einen Mehrbenutzerbetrieb ohne grafische Oberfläche oder graphical.target für den grafischen Modus. Targets ersetzen damit weitgehend das frühere Runlevel-
Konzept.
9. Wie wechselst du das Target zur Laufzeit?
Antwort: Mit systemctl isolate multi-user.target. Dadurch versucht systemd, sofort in den angegebenen
Zielzustand zu wechseln und nur die dafür benötigten Units aktiv zu lassen. Das kann laufende Dienste
stoppen, die im neuen Target nicht mehr benötigt werden.
10. Was ist der Unterschied zwischen restart und reload?
Antwort: restart beendet einen Dienst vollständig und startet ihn anschließend neu. reload lädt nur die
Konfiguration neu, ohne den Prozess komplett zu beenden, sofern der Dienst diese Funktion unterstützt.
reload ist daher oft schonender, weil laufende Verbindungen oder Zustände eher erhalten bleiben.

Seite 1

Seite 2

11. Erstelle einen einfachen Service: Datei: /etc/systemd/system/meinservice.service
Antwort: Eine einfache Unit-Datei könnte den Dienst mit ExecStart=/usr/bin/sleep 1000 definieren und
mit WantedBy=multi-user.target beim Mehrbenutzerstart einbinden. Nach dem Anlegen oder Ändern

der
Datei sollte man systemctl daemon-reload ausführen, damit systemd die neue Unit-Datei einliest.
Anschließend kann man den Dienst mit systemctl enable meinservice dauerhaft aktivieren und mit systemctl start meinservice sofort starten.
12. Was macht systemctl daemon-reload?
Antwort: Dieser Befehl liest die Unit-Dateien erneut ein, wenn Konfigurationen neu angelegt oder
verändert wurden. Er ist notwendig, damit systemd Änderungen an Service-Dateien, Timern oder anderen Units überhaupt erkennt. Ohne daemon-reload arbeitet systemd oft noch mit der alten Version
der Datei.
13. Was ist journalctl?
Antwort: journalctl ist das Werkzeug zum Auslesen des systemd-Journals. Darüber lassen sich zentrale
Logeinträge des Systems und einzelner Dienste anzeigen, filtern und zeitlich eingrenzen. Es ist
besonders nützlich für die Fehlersuche und die Analyse von Startproblemen.
14. Beispiel: Logs eines Dienstes anzeigen
Antwort: Mit journalctl -u nginx. Der Parameter -u filtert die Einträge auf eine bestimmte Unit, sodass nur
die Protokolle des betreffenden Dienstes angezeigt werden. So lassen sich Fehler, Warnungen und
Startmeldungen gezielt untersuchen.
15. Was sind Timer-Units?
Antwort: Timer-Units sind die systemd-Variante für zeitgesteuerte Aufgaben und damit oft ein Ersatz für
klassische Cronjobs. Ein Timer startet zu bestimmten Zeitpunkten oder nach bestimmten Intervallen eine
zugehörige Service-Unit. Der Vorteil liegt darin, dass Zeitsteuerung und Dienstverwaltung direkt
innerhalb von systemd zusammenarbeiten.