ADMIN:DOCS

Linux Administration #1: Grundlagen der Linux-Administration

Grundlagen der Linux-Administration: Von der Systemarchitektur bis zur Rechteverwaltung - Lerne die fundamentalen Konzepte für den Einstieg in die Linux-Administration

Kategorie: Linux-Administration **Veröffentlicht:** 26.10.2024 **Zuletzt geprüft:** 02.11.2026

Willkommen zum ersten Teil unserer technischen Wiki-Serie über Linux-Administration!

In den kommenden acht Artikeln lernst du alle wichtigen Grundlagen, die du als Linux-Administrator benötigst. Wir beginnen mit den fundamentalen Konzepten und arbeiten uns Schritt für Schritt zu fortgeschrittenen Themen vor.

Was ist ein Linux-Administrator?

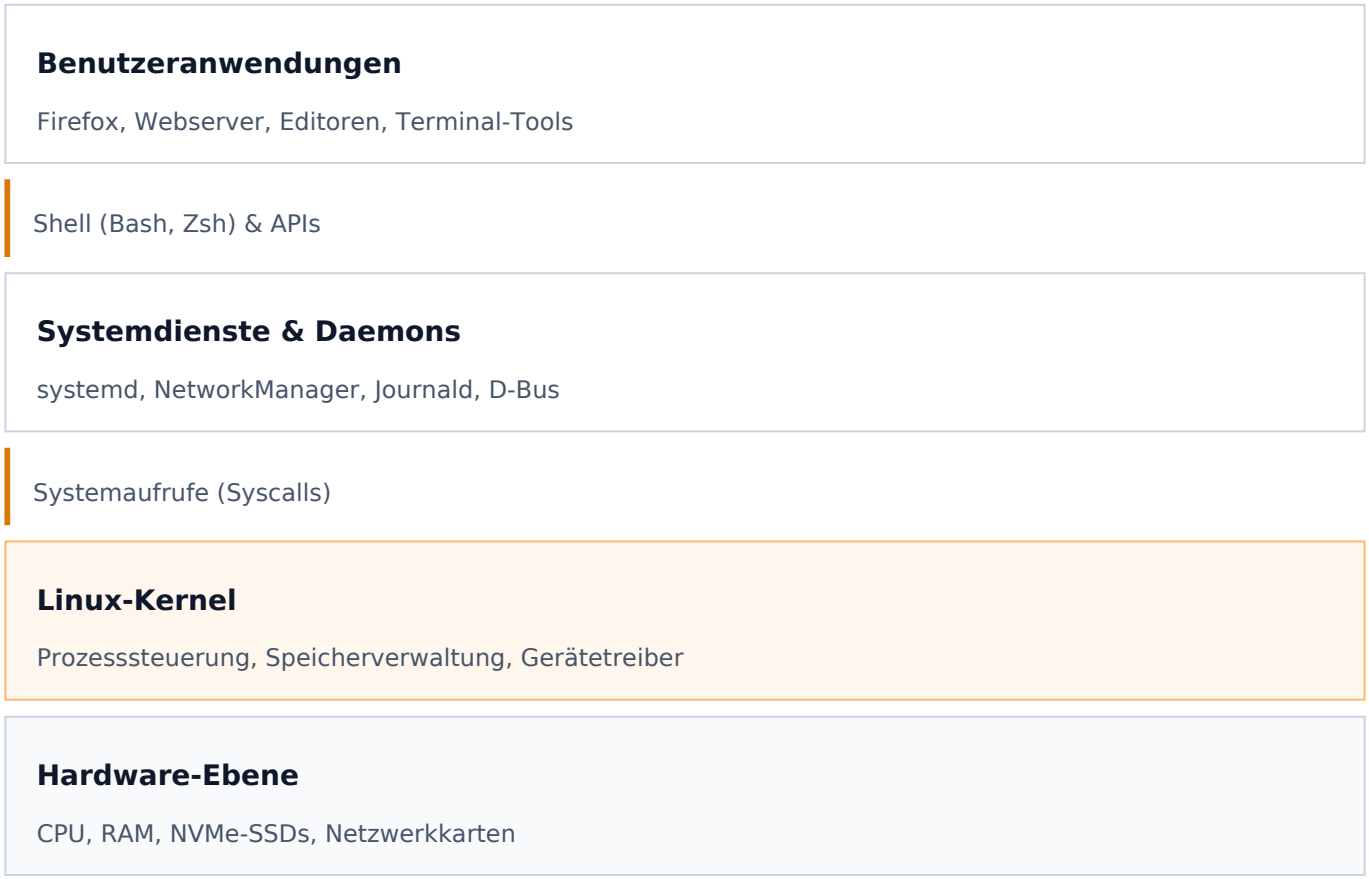
Als Linux-Administrator trägst du die Verantwortung für die Verwaltung, Wartung und Sicherheit von Linux-Systemen. Egal ob du dein eigenes Heimnetzwerk betreust oder in einem Unternehmen arbeitest – ein tiefes Verständnis der Linux-Administration ist heute wichtiger denn je.

Die Grundlagen, die du in diesem Artikel lernst, bilden das Fundament für alle weiteren Administrationsaufgaben. Lass uns mit der Systemarchitektur beginnen – dem Herzstück jedes Linux-Systems.

Grundlegende Systemkonzepte

Systemarchitektur

Bevor du mit der Administration beginnst, ist es wichtig zu verstehen, wie ein Linux-System aufgebaut ist. Stell dir das System wie ein mehrstöckiges Gebäude vor, bei dem jede Ebene eine bestimmte Aufgabe hat.



Systemarchitektur im Detail

Die Linux-Systemarchitektur besteht aus mehreren Schichten, die aufeinander aufbauen. Jede Schicht hat spezifische Aufgaben:

Hardware-Ebene

- Physische Komponenten (CPU, RAM, NVMe/SSDs, Netzwerkkarten)
- Firmware (BIOS / UEFI)
- Bus-Systeme und Hardware-Controller

Kernel-Ebene

Der Kernel ist das Herzstück deines Linux-Systems. Er ist die zentrale Softwareschicht, die nach dem Bootvorgang geladen wird und die Ressourcen sowie Zugriffsrechte der Hardware

Quelle: https://admindocs.de/de/linux-administration/linux-administration-grundlagen-der-linux-administration

kontrolliert.

```
# Kernel-Version anzeigen
uname -r
# Beispielausgabe auf einem modernen LTS-System: 6.12.0-generic

# Kernel-Parameter anzeigen
sysctl -a

# Geladene Kernel-Module verwalten
lsmod

# Modul inklusive Abhängigkeiten laden
sudo modprobe bluetooth

# Modul entladen
sudo modprobe -r bluetooth
```

Wichtige Kernel-Aufgaben:

- Verwaltet den Zugriff auf die Hardware
- Steuert die Prozesse und deren Prioritäten
- Kontrolliert den Arbeitsspeicher

Kernel-Module

Kernel-Module sind wie Erweiterungen für deinen Kernel. Sie können bei Bedarf geladen und entladen werden:

```
# Alle geladenen Module anzeigen
lsmod
```

```
# Spezifisches Modul laden
sudo modprobe bluetooth

# Modul-Informationen anzeigen
modinfo bluetooth

# Modul sicher entladen
sudo modprobe -r bluetooth
```

Systemdienste-Ebene (systemd)

systemd ist das Standard-Init-System und der zentrale Dienstemanager fast aller modernen Linux-Distributionen (u. a. Debian, Ubuntu, Fedora, RHEL, Arch). Es ist verantwortlich für:

- Den parallelen und geordneten Start aller Systemdienste beim Booten
- Die Verwaltung und Überwachung von Hintergrunddiensten (Daemons) während des Betriebs
- Das moderne Ressourcen-Management und Prozess-Sandboxing über cgroups v2
- Das saubere Herunterfahren des Systems

```
# Systemd-Status
systemctl status

# Alle aktiven Dienste anzeigen
systemctl list-units --type=service --state=active

# Boot-Zeit analysieren
systemd-analyze
systemd-analyze blame

# Zeigt Boot-Zeit pro Dienst
```

Wichtig für Anfänger:

- Der Kernel läuft immer im Hintergrund
- Module können nach Bedarf geladen werden
- Systemd verwaltet fast alle Systemdienste
- Logging hilft bei der Fehlersuche

Benutzerebene

```
# Aktuelle Shell anzeigen
echo $SHELL

who

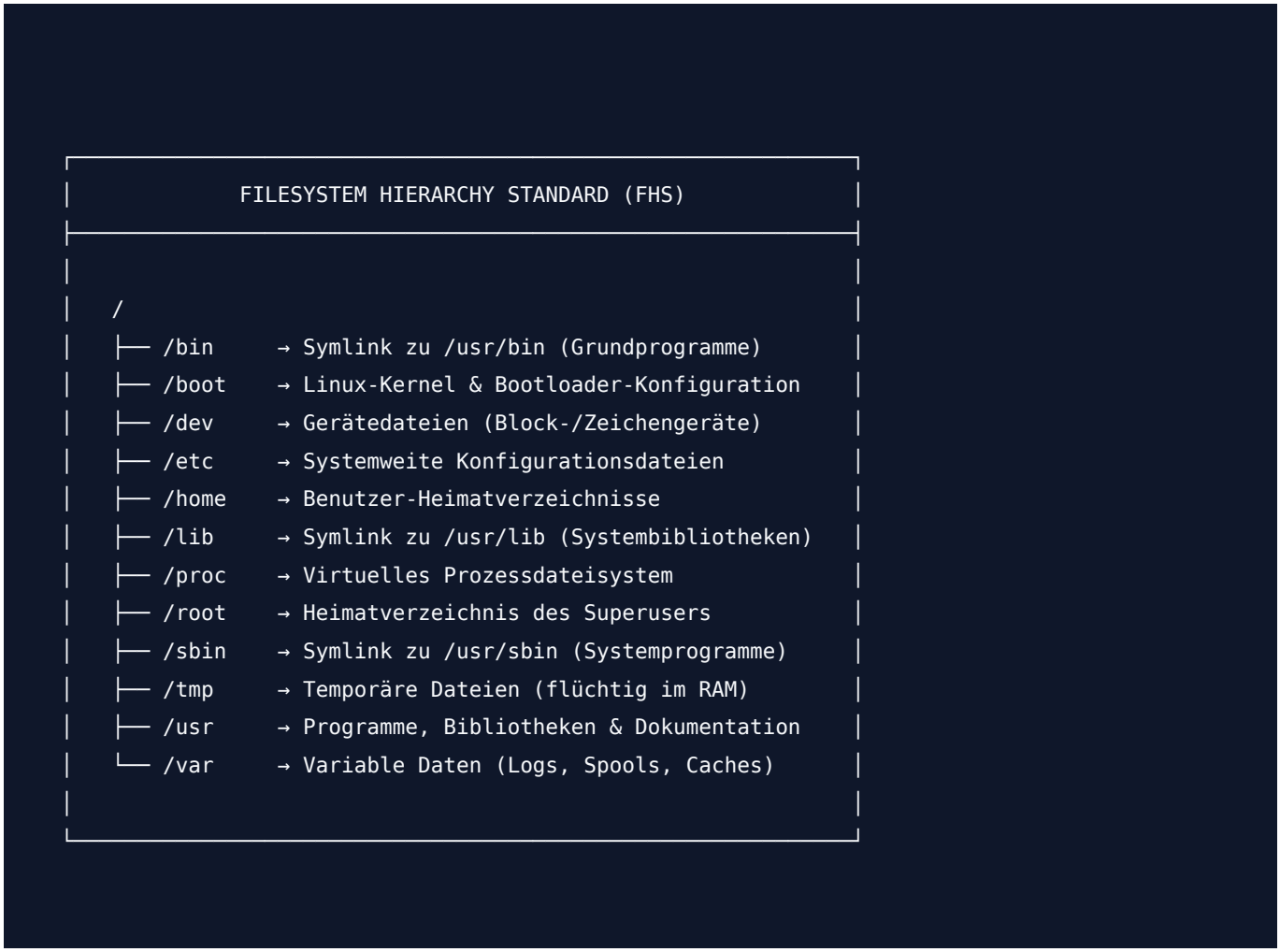
# Eingeloggte Benutzer anzeigen
w

# Detailliertere Informationen
# Prozesse des aktuellen Benutzers
ps aux | grep $USER
```

Dateisystemhierarchie (FHS) und der moderne /usr-Merge

Unter Linux ist alles in einer hierarchischen Verzeichnisstruktur organisiert, beginnend mit dem Stammverzeichnis (`/`):

> **Standard auf modernen Distributionen – Der /usr-Merge:** Auf zeitgemäßen Systemen (wie Debian 12+, Ubuntu 24.04+, RHEL 9+) ist der sogenannte **/usr-Merge** vollständig abgeschlossen. Die Verzeichnisse `/bin` , `/sbin` und `/lib` sind keine separaten Partitionen oder getrennten Ordner mehr, sondern symbolische Links auf ihre Gegenstücke unter `/usr` (`/usr/bin` , `/usr/sbin` , `/usr/lib`). Sämtliche Systemprogramme liegen damit zentral in `/usr` , während `/etc` für Konfiguration und `/var` für variable Daten reserviert sind.



Wichtige Verzeichnisse im Detail:

```
# /etc - Systemkonfiguration
ls -l /etc/

# Beispiele wichtiger Dateien:
# - /etc/passwd    (Benutzerinformationen)
# - /etc/fstab     (Dateisystem-Konfiguration)
# - /etc/hosts     (Hostname-Zuordnung)
# /var - Variable Daten
ls -l /var/

# Beispiele wichtiger Unterverzeichnisse:
# - /var/log      (Systemprotokolle)
```

```
# - /var/spool    (Drucker- und Mail-Warteschlangen)
# - /var/www      (Webserver-Dateien)
```

Dateisystemtypen

Linux unterstützt verschiedene Dateisystemtypen für unterschiedliche Anforderungen:

```
# Verfügbare Dateisysteme anzeigen
cat /proc/filesystems

# Aktuell gemountete Dateisysteme
df -Th
```

Häufig verwendete Dateisysteme:

- **ext4**: Bewährter Standard für Linux-Distributionen und alltägliche Server
- **xfs**: Hochperformant und Standard in Enterprise-Distributionen (z. B. RHEL) für große Speichermengen
- **btrfs**: Modernes CoW-Dateisystem (Copy-on-Write) mit integrierten Prüfsummen und Snapshot-Funktionalität (Standard in Fedora und openSUSE)
- **bcachefs**: Modernes CoW-Dateisystem mit nativer Multi-Device- und Caching-Architektur (im Linux-Kernel integriert)
- **tmpfs**: Temporäres Dateisystem im flüchtigen RAM (für schnelle Zugriffe wie `/tmp` oder `/run`)

Mount-Punkte verwalten

```
# Alle Mount-Punkte anzeigen
mount

# Eine neue Festplatte mounten
sudo mount /dev/sdb1 /mnt/dataplatte

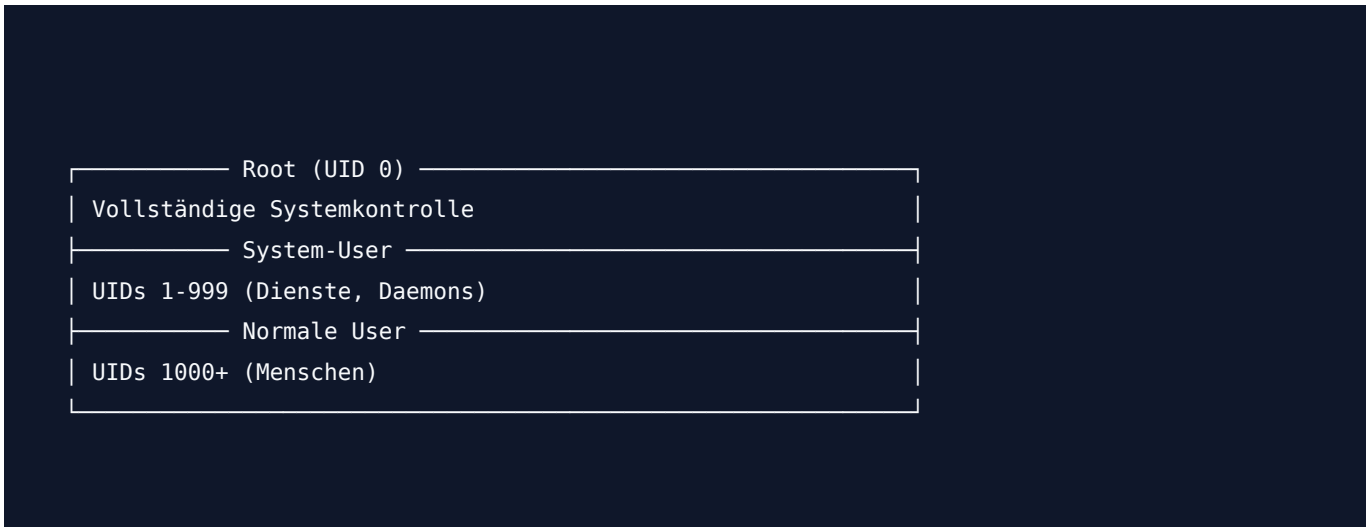
# Automatisches Mounten in fstab konfigurieren
sudo nano /etc/fstab

# Beispieleintrag:
# /dev/sdb1  /mnt/dataplatte  ext4  defaults  0  2
```

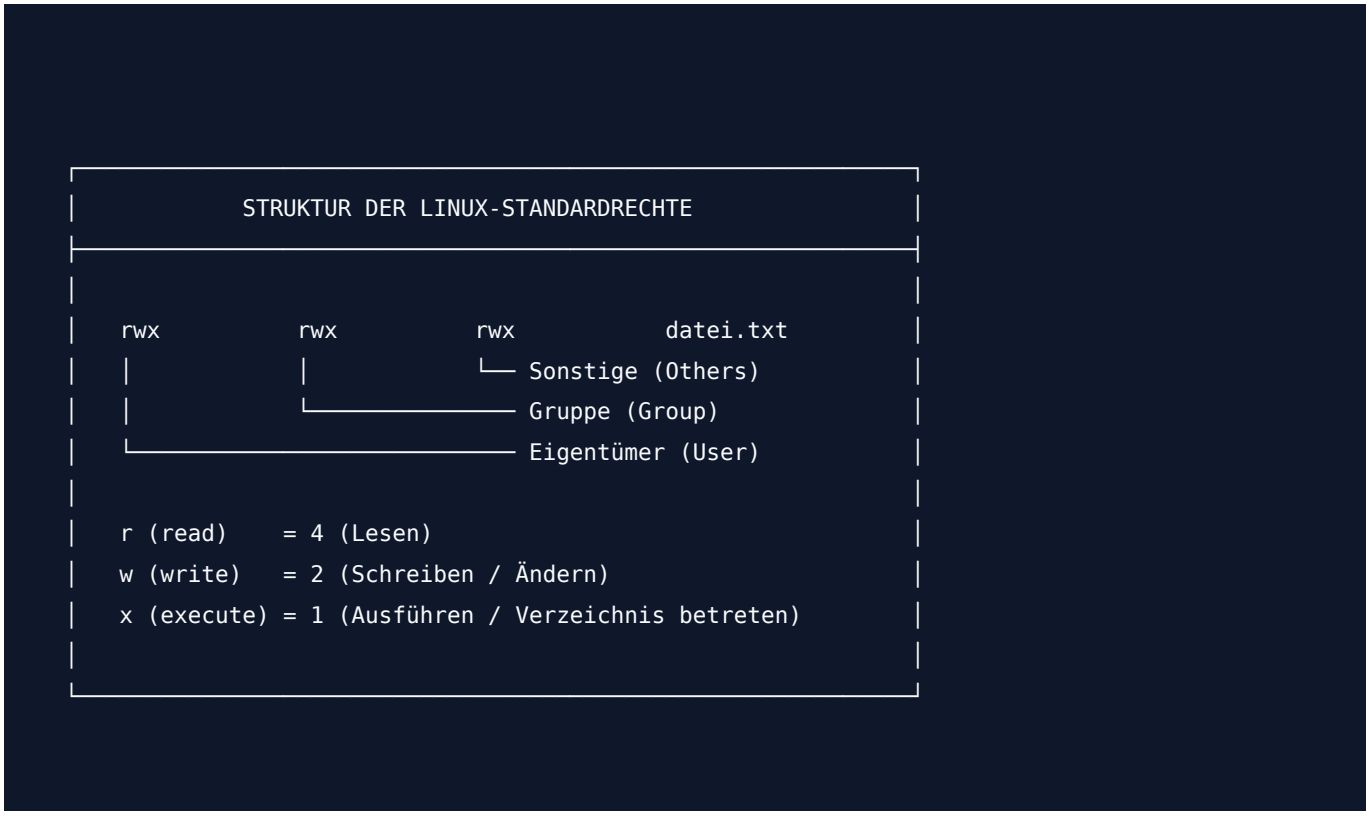
Benutzer- und Rechteverwaltung

Als Linux-Administrator ist eine deiner wichtigsten Aufgaben, Benutzer und deren Rechte zu verwalten. Stell dir das wie ein Gebäude vor, wo verschiedene Personen unterschiedliche Schlüssel und Zugangsberechtigungen haben.

Berechtigungshierarchie:



Dateiberechtigungen:



Root-Benutzer

Der Root-Benutzer (auch Superuser genannt) hat absolute Kontrolle über das System. Das ist wie ein Generalschlüssel, der alle Türen öffnet.

Eigenschaften des Root-Accounts:

- Root besitzt unveränderlich die Benutzer-ID `(UID) 0`.
- Root unterliegt keinen Dateizugriffsbeschränkungen und darf alle Systemprozesse steuern.
- Mit großer Macht kommt große Verantwortung: Befehle werden ohne Rückfrage mit vollen Rechten ausgeführt.
- Administrationsaufgaben sollten stets über `sudo` oder moderne Werkzeuge wie `run0` ausgeführt werden, statt sich dauerhaft direkt als Root anzumelden.

Normale Benutzer

```
# Informationen über deinen Benutzer anzeigen
id

# Beispielausgabe:
# uid=1000(max) gid=1000(max) groups=1000(max),27(sudo)
```

Was bedeutet diese Ausgabe?

- `uid=1000`: Deine eindeutige Benutzer-ID
- `gid=1000`: Deine primäre Gruppen-ID
- `groups=...`: Alle Gruppen, in denen du Mitglied bist

Benutzer erstellen und verwalten

Lass uns das Schritt für Schritt durchgehen:

Neuen Benutzer anlegen

```
# Benutzer erstellen
sudo useradd -m -s /bin/bash anna
```

Bedeutung der Optionen:

- `-m`: Erstellt das persönliche Home-Verzeichnis (`/home/anna`)
- `-s`: Legt die Standard-Shell fest (`/bin/bash`)

Home-Verzeichnis verstehen:

```
ls -la /home/anna

# Zeigt:
# drwxr-xr-x 2 anna anna 4096 Feb 20 10:00 .
# drwxr-xr-x 4 root root 4096 Feb 20 10:00 ..
# -rw-r--r-- 1 anna anna  220 Feb 20 10:00 .bash_logout
# -rw-r--r-- 1 anna anna 3526 Feb 20 10:00 .bashrc
# -rw-r--r-- 1 anna anna  807 Feb 20 10:00 .profile
```

Passwort setzen:

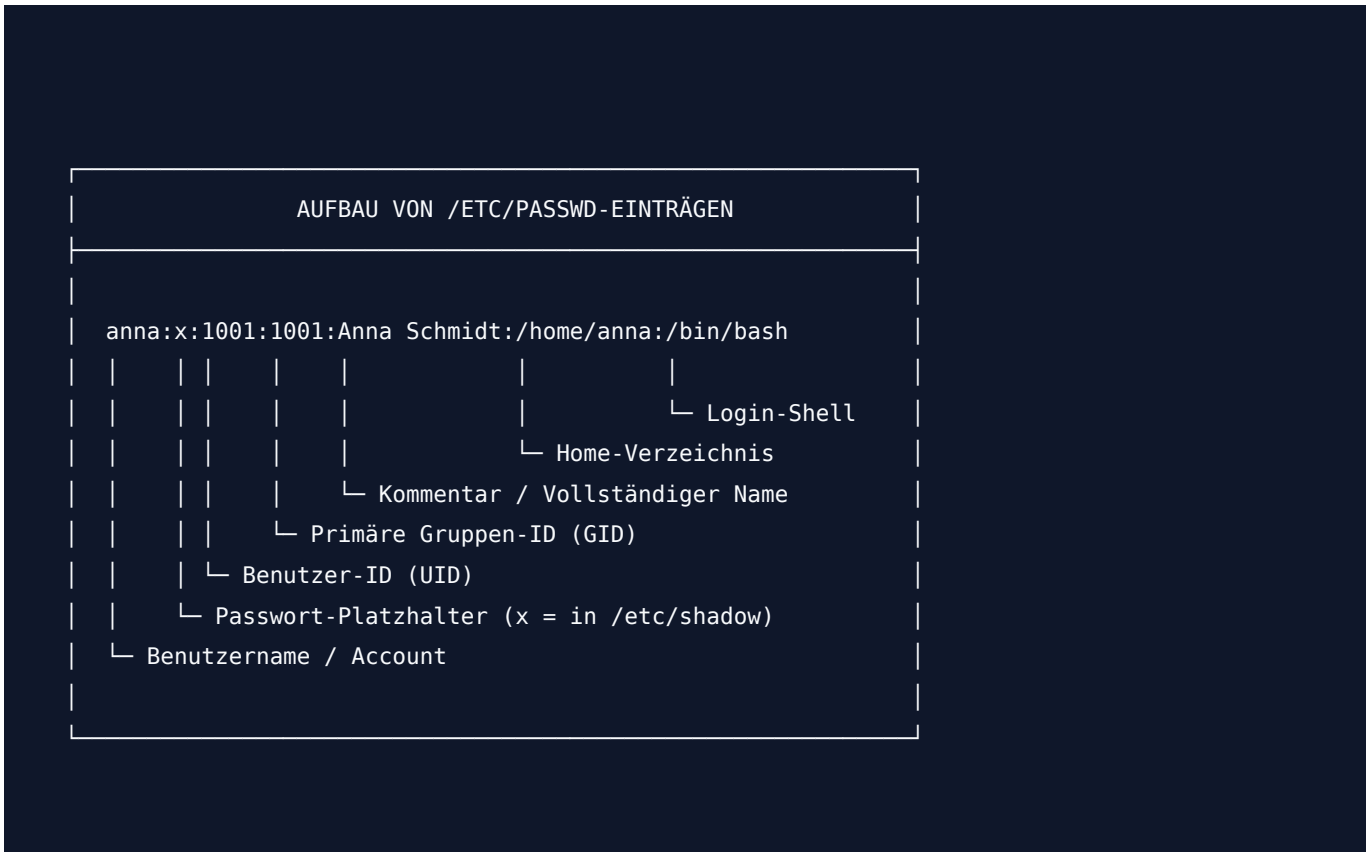
```
sudo passwd anna

# Eingabe: Neues Passwort
# Eingabe: Passwort wiederholen
```

Moderne Richtlinien für Passwörter:

- **Länge vor Komplexität:** Mindestens 12 bis 16 Zeichen (nach BSI- und NIST-Standards); ideal sind einprägsame Passphrasen.
- **Passwort-Manager & SSH-Keys:** Passwörter niemals mehrfach nutzen; für Remote-Serverzugänge SSH-Keys oder FIDO2/MFA bevorzugen.
- **Modernes Hashing in `/etc/shadow`:** Auf modernen Distributionen (Debian 11+, Ubuntu 22.04+) werden Passwörter standardmäßig mit **`yescrypt`** gehasht (erkennbar am Präfix `$y$`), was durch hohe Speicherhärte Schutz vor schnellen GPU-/ASIC-Cracking-Angriffen bietet (im Gegensatz zu veraltetem `$6$` SHA-512).

Benutzerinformationen prüfen:



Gruppen und Berechtigungen

Gruppenkonzept verstehen

Stell dir ein Bürogebäude vor, in dem verschiedene Abteilungen arbeiten. Jede Abteilung (`Gruppe`) hat Zugang zu bestimmten Räumen (`Verzeichnissen`) und Dokumenten (`Dateien`).

Gruppenstruktur:



```
└──────────────────────────────────────────────────────┘
```

Benutzer-Gruppen-Beziehung:

```
┌─── Anna ─────────────────────────────────────────────┐
│ Primär:    │──→ marketin                             g
│ Sekundär:  │──→ entwickler, suppor                   t
└──────────────────────────────────────────────────────┘
```

Was sind Systemgruppen?

- `sudo`: Mitglieder dürfen Admin-Befehle ausführen
- `adm`: Zugriff auf Systemprotokolle
- `www-data`: Für Webserver-Prozesse
- `etc.`: Andere Systemgruppen

Diese Gruppen werden automatisch erstellt und vom System verwendet

Was sind Benutzergruppen?

- Von Administratoren erstellt
- Für spezifische Zwecke (z.B. Projekte, Abteilungen)
- Helfen bei der Organisation von Zugriffsrechten

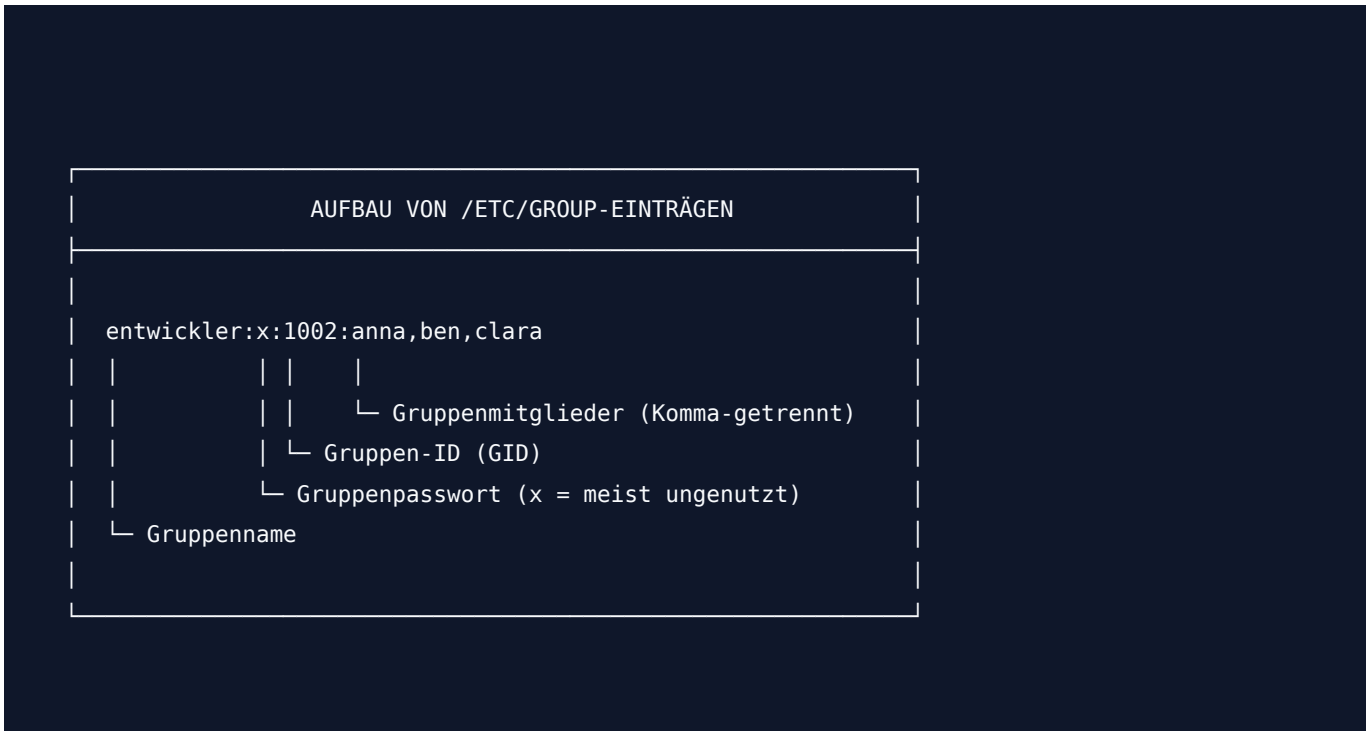
Gruppen erstellen und verwalten

```
# Neue Gruppe erstellen
```

```
sudo groupadd entwickler
```

Was passiert im Hintergrund?

1. Eintrag in `/etc/group`:



Wichtig für Anfänger

- Gruppennamen sollten aussagekräftig sein
- Keine Leerzeichen oder Sonderzeichen verwenden
- Kleinbuchstaben bevorzugen
- IDs über 1000 für eigene Gruppen verwenden

Benutzer zu Gruppen hinzufügen

```
# Anna zur Entwickler-Gruppe hinzufügen
```

```
sudo usermod -aG entwickler anna

# Was bedeuten die Optionen?
# -a: append (hinzufügen, nicht überschreiben)
# -G: sekundäre Gruppe(n)
# Gruppenmitgliedschaft prüfen
groups anna

# Beispielausgabe: anna : anna entwickler
```

Häufige Fehler und Lösungen:

```
# Fehler: Gruppe existiert nicht
sudo groupadd entwickler

# groupadd: group 'entwickler' already exists
# Lösung: Gruppe prüfen
getent group entwickler

# Fehler: Benutzer nicht in Gruppe
groups anna

# anna : anna
# Lösung: Benutzer zur Gruppe hinzufügen
sudo usermod -aG entwickler anna
```

Wichtig für Anfänger

- Verwende `-a` immer zusammen mit `-G`, sonst überschreibst du bestehende Gruppen
- Neue Gruppenmitgliedschaften werden erst nach Ab- und Anmeldung aktiv
- Alternativ aktiviert `newgrp` die neue Gruppe sofort
- Änderungen anschließend mit `groups` überprüfen

Best Practices:

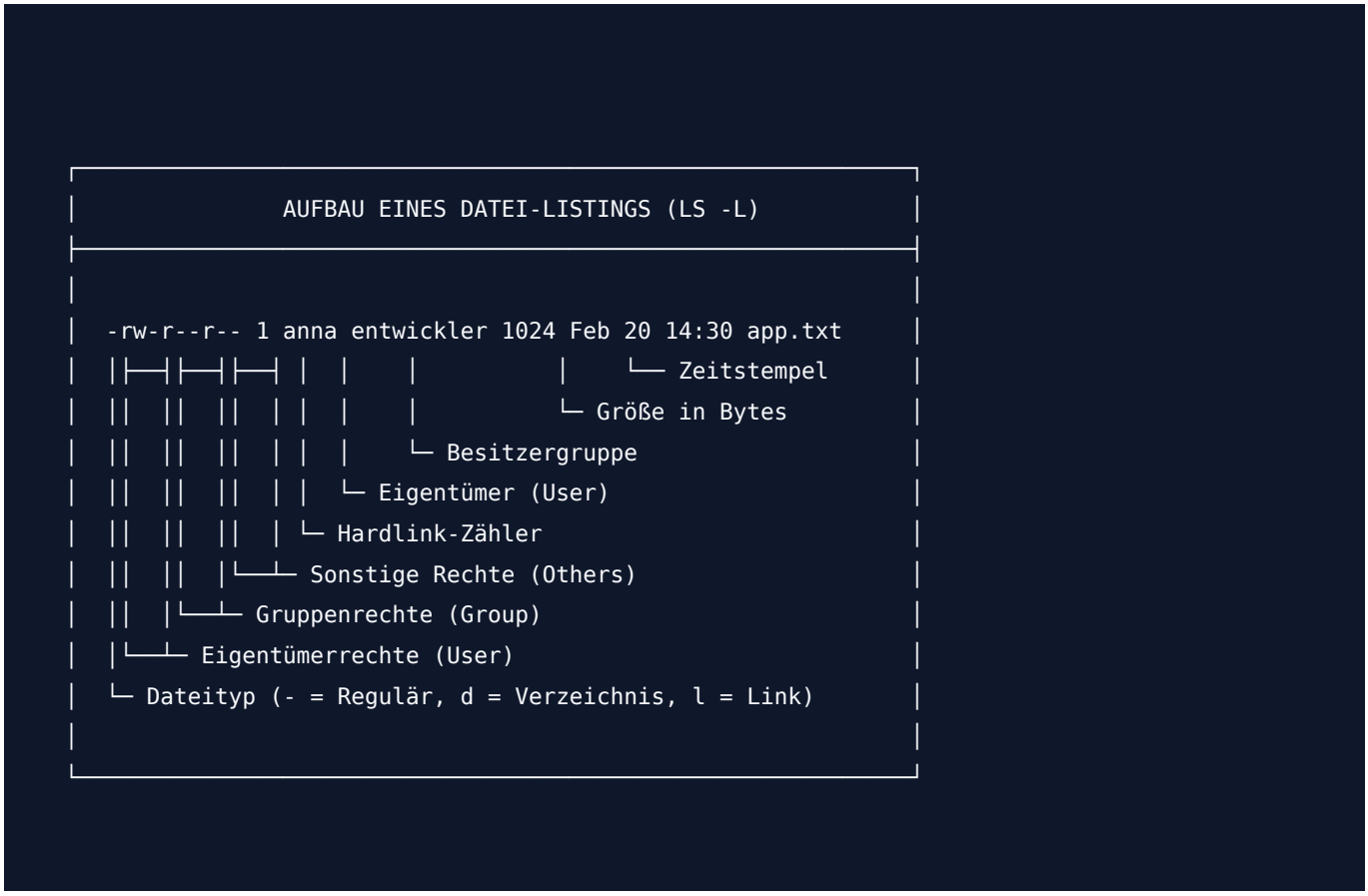
- Verwende aussagekräftige Gruppennamen
- Vermeide Leerzeichen und Sonderzeichen
- Verwende bevorzugt Kleinbuchstaben
- Reserviere IDs ab 1000 für benutzerdefinierte Gruppen
- Dokumentiere Zweck und Mitgliedschaften deiner Gruppen

Dateiberechtigungen im Detail

Das Linux-Berechtigungssystem verstehen

Es ist wichtig zu verstehen, dass jede Datei und jedes Verzeichnis in Linux drei Arten von Berechtigungen für drei verschiedene Kategorien hat.

Vollständige Berechtigungsstruktur: Datei: `beispiel.txt`



Berechtigungsarten im Detail

Lesen (r = read):

Für Dateien bedeutet Leserecht:

- Dateiinhalt anzeigen mit cat, less, more
- Datei kopieren
- Datei öffnen und lesen
- Nach Text suchen mit grep
- Inhalt in Editor anzeigen (aber nicht speichern)

Praktische Beispiele:

```
# Datei lesen
cat beispiel.txt
less beispiel.txt
head -n 5 beispiel.txt

# Erste 5 Zeilen
tail -n 5 beispiel.txt

# Letzte 5 Zeilen
```

```
# Datei kopieren
cp beispiel.txt kopie.txt
```

```
# In Datei suchen
grep "Suchbegriff" beispiel.txt
```

```
# Was passiert ohne Leserecht?
chmod u-r beispiel.txt
cat beispiel.txt
```

Ausgabe: "Permission denied"

Für Verzeichnisse bedeutet Leserecht:

- Verzeichnisinhalt auflisten mit ls
- Dateien im Verzeichnis sehen
- Nach Dateien suchen
- Metadaten der Dateien sehen (Größe, Datum, etc.)

Praktische Beispiele:

```
ls -l verzeichnis/
ls -la verzeichnis/

# Zeigt auch versteckte Dateien
find verzeichnis/ -name "*.txt"
```

Was passiert ohne Leserecht?

```
chmod u-r verzeichnis/
ls verzeichnis/

# Ausgabe: "Permission denied"
```

Schreiben (w = write):

Für Dateien bedeutet Schreibrecht:

- Dateiinhalt ändern
- In Datei schreiben
- Datei löschen
- Datei umbenennen
- Datei verschieben
- Dateiattribute ändern
- Datei leeren

Praktische Beispiele:

```
# Datei erstellen/überschreiben
echo "Neuer Text" > beispiel.txt
```

```
# An Datei anhängen
echo "Weiterer Text" >> beispiel.txt
```

```
# Mit Editor bearbeiten
nano beispiel.txt
vim beispiel.txt
```

```
# Datei umbenennen/verschieben
mv beispiel.txt neu.txt
mv neu.txt /tmp/
```

```
# Datei löschen
rm beispiel.txt
```

```
# Was passiert ohne Schreibrecht?
chmod u-w beispiel.txt
echo "Test" > beispiel.txt

# Ausgabe: "Permission denied"
```

Für Verzeichnisse bedeutet Schreibrecht:

- Neue Dateien erstellen
- Bestehende Dateien löschen

- Dateien umbenennen
- Dateien verschieben
- Unterverzeichnisse erstellen
- Verzeichnisattribute ändern

Praktische Beispiele:

- mkdir verzeichnis/neuer_ordner
- touch verzeichnis/neue_datei.txt
- rm verzeichnis/alte_datei.txt
- mv verzeichnis/datei1.txt verzeichnis/datei2.txt

```
# Was passiert ohne Schreibrecht?
chmod u-w verzeichnis/
touch verzeichnis/test.txt

# Ausgabe: "Permission denied"
```

Ausführen (x = execute):

Für Dateien bedeutet Ausführrecht:

- Datei als Programm/Skript ausführen
- Binärdateien starten
- Shell-Skripte ausführen
- Programme ausführen

Praktische Beispiele:

```
# Shell-Skript erstellen
echo '#!/bin/bash' > skript.sh
echo 'echo "Hallo Welt"' >> skript.sh
chmod u+x skript.sh
./skript.sh
```

```
# Was passiert ohne Ausführrecht?
chmod u-x skript.sh
./skript.sh

# Ausgabe: "Permission denied"
```

Für Verzeichnisse bedeutet Ausführrecht:

- In das Verzeichnis wechseln (cd)
- Auf Dateien im Verzeichnis zugreifen
- Verzeichnis durchqueren
- Unterverzeichnisse aufrufen
- Dateien im Verzeichnis ausführen (wenn zusätzlich ausführbar)

Praktische Beispiele:

- cd verzeichnis/
- ls -l verzeichnis/datei.txt
- ./verzeichnis/programm

```
# Was passiert ohne Ausführrecht?
chmod u-x verzeichnis/
cd verzeichnis/

# Ausgabe: "Permission denied"
```

Wichtige Kombinationen:

Häufig verwendete Berechtigungskombinationen:

```
chmod 755 verzeichnis/

# drwxr-xr-x
```

Bedeutung

- Besitzer kann alles (`rwx`)
- Gruppe kann lesen und ausführen (`r-x`)
- Andere können lesen und ausführen (`r-x`)

Typisch für: Programm-Verzeichnisse, öffentliche Ordner

```
chmod 644 datei.txt

# -rw-r--r--
```

Bedeutung

- Besitzer kann lesen & schreiben (`rw-`)
- Gruppe kann nur lesen (`r--`)
- Andere kann nur lesen (`r--`)

Typisch für: Programm-Verzeichnisse, öffentliche Ordner

```
chmod 600 private.key

# -rw-------
```

Bedeutung:

- Besitzer kann lesen und schreiben (`rw-`)
- Gruppe hat keine Rechte (`-`)
- Andere haben keine Rechte (`-`)

Typisch für: Sensitive Dateien, SSH-Keys

Erweiterte Berechtigungen (SUID, SGID, Sticky Bit)

In Linux gibt es neben den grundlegenden Berechtigungen (`rwx`) auch spezielle Berechtigungen, die für bestimmte Sicherheits- und Verwaltungsaufgaben wichtig sind.

Spezielle Berechtigungen visualisiert:

```
┌──────────────────────────────────────────────────┐
│              SPEZIALBERECHTIGUNGEN IN LINUX      │
├──────────────────────────────────────────────────┘
```



SUID (Set User ID) im Detail

Was ist SUID?

- Programm läuft mit den Rechten des Dateibesitzers
- Wird durch `s` in den Besitzer-Ausführrechten angezeigt
- Numerischer Wert: 4000

Praktische Beispiele:

```
# SUID anzeigen
ls -l /usr/bin/passwd

# Ausgabe: -rwsr-xr-x root root /usr/bin/passwd
# Beachte das `s` im Besitzer-Ausführrechten
```

Warum ist das wichtig?

- Normaler Benutzer kann sein eigenes Passwort ändern

- Programm läuft während der Ausführung temporär mit den Rechten des Dateieigentümers (Root)
- Ermöglicht kontrollierten Zugriff auf sensible Ressourcen wie `/etc/shadow`

SUID setzen

```
# Symbolische Methode
chmod u+s programm

# Numerische Methode (Oktal 4000)
chmod 4755 programm
```

Was passiert im Hintergrund?

- 1. Benutzer startet Programm
- 2. Der Kernel setzt die effektive User-ID (EUID) auf den Dateieigentümer (z. B. Root)
- 3. Programm kann Aktionen mit höheren Rechten ausführen
- 4. Nach Beendigung des Prozesses erlöschen die erhöhten Rechte

Klassische Anwendungsfälle:

```
# Passwortänderung (benötigt Schreibzugriff auf /etc/shadow)
ls -l /usr/bin/passwd
# Ausgabe: -rwsr-xr-x 1 root root ... /usr/bin/passwd

# Temporäre administrative Rechteausweitung
ls -l /usr/bin/sudo
# Ausgabe: -rwsr-xr-x 1 root root ... /usr/bin/sudo
```

> **Sicherheitsentwicklung – Capabilities und run0 statt SUID:** Da SUID-Binaries im Falle von Programmfehlern ein erhebliches Sicherheitsrisiko (Privilege Escalation) bergen, setzen moderne Distributionen zunehmend auf risikoärmere Alternativen:
>
> - **Linux Capabilities:** Netzwerkwerkzeuge wie `/usr/bin/ping` benötigen heute kein SUID mehr; sie nutzen stattdessen gezielte Kernel-Rechte wie `cap_net_raw+p` oder unprivilegierte ICMP-Sockets (`net.ipv4.ping_group_range`).
> - **run0 (systemd 256+):** Als moderne Alternative zu `sudo` startet `run0` Befehle in einer isolierten, transienten systemd-Service-Unit unter Polkit-Autorisierung – vollständig ohne SUID-Binärdatei.

SGID (Set Group ID)

Was ist SGID?

- **Bei Verzeichnissen:** Neue Dateien erben automatisch die Gruppe des übergeordneten Verzeichnisses
- **Bei Programmen:** Programm läuft mit den Rechten der zugewiesenen Gruppe
- Wird durch `s` in den Gruppen-Ausführrechten angezeigt
- Hat den numerischen Wert 2000



SGID setzen

```
chmod g+s /shared/projekt/

# oder numerisch
chmod 2775 /shared/projekt/
```

Was passiert dann?

- 1. Du erstellst eine neue Datei im Verzeichnis
- 2. Die Datei erhält automatisch die Gruppe des Verzeichnisses
- 3. Alle Teammitglieder können zusammenarbeiten

Beispiel für ein Projektverzeichnis:

`/shared/projekt/` (Gruppe: entwickler)

- ── `dokumente/` (erbt Gruppe: entwickler)
- ├── `konzept.txt` (erbt Gruppe: entwickler)
- └── `planung.pdf` (erbt Gruppe: entwickler)
- ── `source/` (erbt Gruppe: entwickler)

```
  └── `main.cpp`         (erbt Gruppe: entwickler)
```

Häufige Anwendungsfälle:

- Gemeinsame Projektverzeichnisse
- Web-Server-Dokumentenverzeichnisse
- Geteilte Entwicklungsordner

- Gruppen-Dokumentensammlungen

Tipps für die Praxis:

- Kombiniere SGID mit passenden Gruppenrechten
- Stelle sicher, dass die Gruppe die richtigen Mitglieder hat
- Überprüfe die Vererbung nach dem Einrichten

Überprüfen der Einstellungen:

```
# Verzeichnis erstellen und SGID setzen
sudo mkdir -p /shared/projekt
sudo chown root:entwickler /shared/projekt
sudo chmod 2775 /shared/projekt

# Berechtigungen prüfen (beachte das 's' bei den Gruppenrechten)
ls -ld /shared/projekt
# Ausgabe: drwxrwsr-x 2 root entwickler 4096 ... /shared/projekt

# Neue Datei anlegen
touch /shared/projekt/test.txt

# Gruppe prüfen (hat automatisch 'entwickler' geerbt)
ls -l /shared/projekt/test.txt
# Ausgabe: -rw-r--r-- 1 anna entwickler 0 ... /shared/projekt/test.txt
```

Sticky Bit

Was ist das Sticky Bit?

Ein Sicherheitsmerkmal, das hauptsächlich für Verzeichnisse verwendet wird:

- Nur der Besitzer darf seine eigenen Dateien löschen oder umbenennen
- Andere Benutzer können keine fremden Dateien löschen
- Wird durch `t` in den anderen-Ausführrechten angezeigt
- Hat den numerischen Wert 1000

Visualisierung:



Sticky Bit setzen

```
chmod +t /shared/public/

# oder
chmod 1777 /shared/public/
```

Beispiel-Szenario:

`/shared/public/` (Sticky Bit gesetzt)

- ── `anna/` (Besitzer: anna)
- └── `projekt.txt` Nur anna kann dies löschen
- ── `bob/` (Besitzer: bob)
- └── `notizen.txt` Nur bob kann dies löschen
- ── `shared.txt` Nur der Ersteller kann dies löschen

Was passiert im Hintergrund?

- 1. Ein Benutzer erstellt eine Datei im Verzeichnis
- 2. Die Datei behält ihren ursprünglichen Besitzer
- 3. Nur der Besitzer (und root) können die Datei löschen
- 4. Andere Benutzer können die Datei nicht löschen, auch wenn sie Schreibrechte haben

Praktischer Test

Als Benutzer `anna` eine Datei im gemeinsamen Verzeichnis erstellen:

```
touch /shared/public/anna.txt
ls -l /shared/public/anna.txt
# Ausgabe: -rw-r--r-- 1 anna anna ... /shared/public/anna.txt
```

Als anderer Benutzer `bob` versuchen, Annas Datei zu löschen (wird abgewiesen):

```
rm /shared/public/anna.txt
```

> Erwartete Fehlerausgabe: `rm: cannot remove '/shared/public/anna.txt': Operation not permitted`

Häufige Anwendungsfälle:

- `/tmp` Verzeichnis (temporäre Dateien für alle Benutzer)
- `/var/tmp` Verzeichnis (persistente temporäre Dateien)
- Öffentliche Upload- und Drop-Verzeichnisse
- Austauschordner für Teams

Sicherheitsaspekte:

- Verhindert versehentliches oder böswilliges Löschen fremder Dateien
- Ermöglicht gemeinsame Nutzung ohne Datenverlust
- Wichtig für Multi-User-Systeme und Shared-Hosting-Umgebungen

Best Practices:

- Verwende das Sticky Bit für öffentlich beschreibbare Verzeichnisse (`chmod 1777`)
- Kombiniere es mit sinnvollen umask-Vorgaben
- Bereinige alte temporäre Dateien automatisiert über `systemd-tmpfiles`

Überprüfen der Einstellungen:

```
# Verzeichnis anlegen und Sticky Bit setzen
sudo mkdir -p /shared/public
sudo chmod 1777 /shared/public

# Rechte prüfen (beachte das 't' bei den Rechten der Sonstigen)
ls -ld /shared/public
# Ausgabe: drwxrwxrwt 2 root root 4096 ... /shared/public
```

Zusammenfassende Übungen

Übung 1: Grundlegende Systemkonfiguration

In dieser praktischen Übung wirst du die gelernten Konzepte der Systemarchitektur und Rechteverwaltung anwenden.

Szenario:

Du bist als neuer Linux-Administrator eingestellt worden und sollst ein Entwicklungssystem für ein Team einrichten. Dabei musst du die Systemarchitektur verstehen und korrekte

Berechtigungen setzen.

Anforderungen

1. Systemverständnis

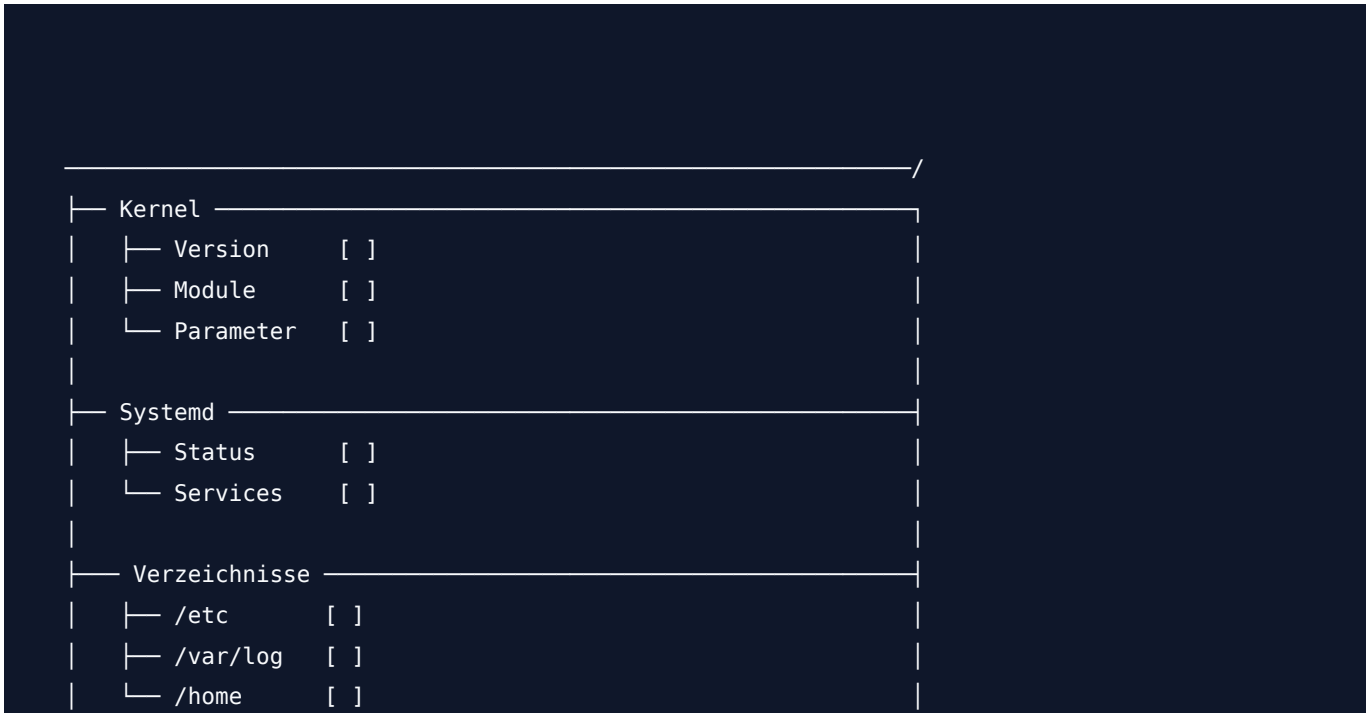
- Kernel-Informationen auslesen
- Systemd-Status überprüfen
- Wichtige Systemverzeichnisse identifizieren

2. Berechtigungsstruktur

- Entwicklergruppe erstellen
- Projektverzeichnisse anlegen
- Korrekte Berechtigungen setzen
- Erweiterte Berechtigungen implementieren

Überprüfungsstruktur:

System-Überprüfung:



```
└──────────────────────────────────────────────────────────┘
```

Berechtigungs-Überprüfung:

```
┌──────────────────────────────────────────────────────────┐
│              PROJEKTVERZEICHNIS-RECHTESTRUKTUR           │
├──────────────────────────────────────────────────────────┤
│                                                          │
│   /srv/projekt/                                          │
│   ├── src/                drwxrwsr-x   root:entwickler   │
│   │   └── [Dateien erben Gruppe 'entwickler' automatisch]│
│   │                                                      │
│   ├── docs/               drwxrwsr-x   root:entwickler   │
│   │   └── [Dokumentation für das gesamte Entwicklerteam] │
│   │                                                      │
│   └── shared/             drwxrwxrwt   root:root         │
│       └── [Sticky Bit: Löschen nur durch Dateieigentümer]│
│                                                          │
└──────────────────────────────────────────────────────────┘
```

Mögliche Lösung für Aufgabe 1:

```
#!/bin/bash
# 1. Systemüberprüfung
echo "=== Systemüberprüfung ==="
uname -a
systemctl status
ls -la /etc /var/log /home

# 2. Berechtigungsstruktur erstellen
echo "=== Berechtigungsstruktur erstellen ==="
```

```
# Gruppe erstellen
sudo groupadd dev

# Verzeichnisstruktur
sudo mkdir -p /proj/{src,docs,shared}

# Grundberechtigungen
sudo chown -R root:dev /proj/src /proj/docs
sudo chown root:root /proj/shared

# Erweiterte Berechtigungen
sudo chmod 2775 /proj/src /proj/docs

# SGID
sudo chmod 1777 /proj/shared

# Sticky Bit
# 3. Überprüfung
echo "=== Überprüfung ==="
ls -la /proj/
getent group dev
```

Übung 2: Systemarchitektur und Berechtigungen

In dieser praktischen Übung wirst du die grundlegenden Konzepte der Linux-Systemarchitektur und Berechtigungsverwaltung anwenden.

Szenario:

Als neuer Linux-Administrator sollst du:

- Die Systemarchitektur analysieren
- Ein Verzeichnissystem mit korrekten Berechtigungen aufsetzen
- Erweiterte Berechtigungen implementieren

Teil 1: Systemanalyse

Systemkomponenten-Checkliste:



Teil 2: Berechtigungskonfiguration

Projektstruktur:

`/projekt/`

- ── `docs/` [2775]
- ├── `intern/` [2770]
- └── `public/` [2775]
- ──────── `scripts/` [2771]
- ──────── `shared/` [1777]

Berechtigungsmatrix:

- ──────────────┬──────────┬──────────┬──────────┐
- Verzeichnis│ Owner │ Group │ Others │
- ──────────────┼──────────┼──────────┼──────────┤
- docs │ rwx │ rwx │ r-x │
- intern │ rwx │ rwx │ --- │
- public │ rwx │ rwx │ r-x │

- scripts | rwx | rwx | --x |
- shared | rwx | rwx | rwx |
- ———————————|———————|———————|———————|

Mögliche Lösung:

```
#!/bin/bash
# Teil 1: Systemanalyse
echo "=== Systemanalyse ==="

# Hardware
echo "CPU Info:"
lscpu | grep "Model name"
echo "RAM Status:"
free -h
echo "Festplatten:"
df -h

# Kernel
echo "Kernel Version:"
uname -r
echo "Geladene Module:"
lsmod | head -n 5

# Dateisystem
echo "Wichtige Verzeichnisse:"
ls -l / | grep -E "^d"

# Teil 2: Berechtigungskonfiguration
echo "=== Berechtigungskonfiguration ==="

# Projektstruktur erstellen
sudo mkdir -p /projekt/{docs/{intern,public},scripts,shared}

# Gruppe erstellen
sudo groupadd projekt

# Berechtigungen setzen
```

```
sudo chown -R root:projekt /projekt
sudo chmod 2775 /projekt/docs
sudo chmod 2770 /projekt/docs/intern
sudo chmod 2775 /projekt/docs/public
sudo chmod 2771 /projekt/scripts
sudo chmod 1777 /projekt/shared

# Überprüfung
echo "=== Überprüfung ==="
ls -la /projekt/
```

Überprüfung:

Berechtigungstest-Matrix:

```
┌──────────────────────────────────────────────────────┐
│ Test 1: Systemanalyse                                │
├──────────────────┬──────────┬────────────────────────┤
│ Komponente       │ Status   │ Check                  │
├──────────────────┼──────────┼────────────────────────┤
│ Hardware-Info    │ [ ]      │ [ ]                    │
│ Kernel-Details   │ [ ]      │ [ ]                    │
│ Dateisystem      │ [ ]      │ [ ]                    │
└──────────────────┴──────────┴────────────────────────┘

┌──────────────────────────────────────────────────────┐
│ Test 2: Berechtigungen                               │
├──────────────────┬──────────┬────────────────────────┤
│ Verzeichnis      │ SGID     │ Perm                   │
├──────────────────┼──────────┼────────────────────────┤
│ /projekt/docs    │ [ ]      │ [ ]                    │
│ intern           │ [ ]      │ [ ]                    │
│ public           │ [ ]      │ [ ]                    │
│ scripts          │ [ ]      │ [ ]                    │
│ shared           │ [ ]      │ [ ]                    │
└──────────────────┴──────────┴────────────────────────┘
```

Befehlsreferenz (Cheatsheet)

Für den schnellen Zugriff bei der täglichen Systemarbeit fasst die folgende Referenz die elementaren Grundkommandos der Linux-Administration zusammen:

Kernel-Informationen abrufen Zeigt Kernel-Version, Architektur und Hostname

$ `uname -a`

Geladene Kernel-Module auflisten Zeigt alle aktuell im Kernel aktiven Module

$ `lsmod`

Kernel-Modul laden Lädt ein Kernel-Modul inklusive aller Abhängigkeiten

$ `sudo modprobe <modul>`

Kernel-Modul sicher entladen Entlädt ein nicht mehr benötigtes Modul aus dem Kernel

$ `sudo modprobe -r <modul>`

Systemd-Dienststatus abfragen Prüft Ausführungsstatus und Logs einer systemd-Unit

$ `systemctl status <unit>`

Boot-Geschwindigkeit analysieren Zeigt Startdauer der einzelnen Dienste beim Bootvorgang

$ `systemd-analyze blame`

Dateisysteme & Speicherplatz prüfen Listet Partitionen, Typen und freien Speicherplatz auf

$ `df -Th`

Blockgeräte visualisieren Zeigt Festplatten, Partitionen, UUIDs und Mountpunkte

$ `lsblk -f`

Partition manuell mounten Hängt ein Dateisystem an einem definierten Pfad ein

$ `sudo mount /dev/sdX /mnt`

Fstab-Einträge testen Prüft alle Dateisystemeinträge der /etc/fstab ohne Neustart

$ `sudo mount -a`

Neuen Benutzer erstellen Erstellt ein Benutzerkonto mit Home-Verzeichnis und Bash

$ `sudo useradd -m -s /bin/bash <user>`

Benutzer-Passwort setzen Ändert das Kennwort eines Benutzers

$ `sudo passwd <user>`

Benutzer zu Gruppe hinzufügen Fügt einen Benutzer sicher zu einer sekundären Gruppe hinzu

$ `sudo usermod -aG <gruppe> <user>`

Standardrechte setzen (Dateien) Eigentümer voll, Gruppe und Sonstige lesend

$ `chmod 644 <datei>`

Ausführrechte setzen (Skripte & Ordner) Eigentümer voll, Gruppe und Sonstige lesend/ausführend

$ `chmod 755 <datei_oder_ordner>`

SGID-Bit für Ordner aktivieren Erzwingt automatische Gruppenvererbung für neue Dateien

$ `chmod 2775 <ordner>`

Sticky Bit für Ordner aktivieren Löschschutz für gemeinsame Ordner (nur Ersteller darf löschen)

$ `chmod 1777 <ordner>`

Besitzverhältnisse rekursiv ändern Ändert Benutzer- und Gruppenbesitz für Verzeichnisbäume

$ `sudo chown -R <user>:<gruppe> <pfad>`

Standard-Berechtigungsmaske anzeigen Zeigt die aktive umask für neu erstellte Dateien

```
$ umask
```

Weiterführende Ressourcen

Die folgenden offiziellen Standards, Dokumentationen und Vertiefungsartikel unterstützen dich beim fundierten Ausbau deiner Administrations-Kenntnisse:

Filesystem Hierarchy Standard (FHS) 3.0 Offizielle FHS-Spezifikation der Linux Foundation

systemd System & Service Manager Offizielle Dokumentation und Referenz zum systemd-Ökosystem

Linux Kernel Modul-Dokumentation Offizielle Kernel.org-Dokumentation zur Linux-Modulverwaltung

chmod & Dateiberechtigungen Leitfaden Fundamentale Vertiefung zu Berechtigungen, Bitmasken und Oktalwerten

Befehlszeilenprozessor in Linux Architektur von Shell, TTY, Pipes und I/O-Streams

Bash Grundkurs #1: Erstes Skript Fundamentaler Einstieg in die Bash-Skriptprogrammierung

Fazit

Mit dem Verständnis der 5-Schichten-Systemarchitektur, der Navigation im FHS-Verzeichnisbaum, der persistenten Speicherverwaltung über `/etc/fstab` sowie der präzisen Beherrschung von Standard- und Sonderberechtigungen (SUID, SGID, Sticky Bit) hast du das unverrückbare Fundament für deine Laufbahn als Linux-Administrator gelegt.

> **Praxis-Tipp:** Nutze bei gemeinsamen Team- und Projektordnern konsequent das **SGID-Bit** (`chmod 2775`), damit neu erstellte Dateien automatisch der gemeinsamen Gruppe gehören und Teammitglieder ohne manuelle Rechteanpassungen kollaborativ arbeiten können.

Im nächsten Modul unseres Administrationskurses widmen wir uns der tiefgreifenden Benutzerverwaltung, PAM-Authentifizierung und Zugriffskontrolle: **Weiter geht es mit:** Linux Administration #2: Fortgeschrittene Benutzerverwaltung

Zur Kursübersicht: Alle Linux-Administrations-Artikel & Module

Quelle: https://admindocs.de/de/linux-administration/linux-administration-grundlagen-der-linux-administration