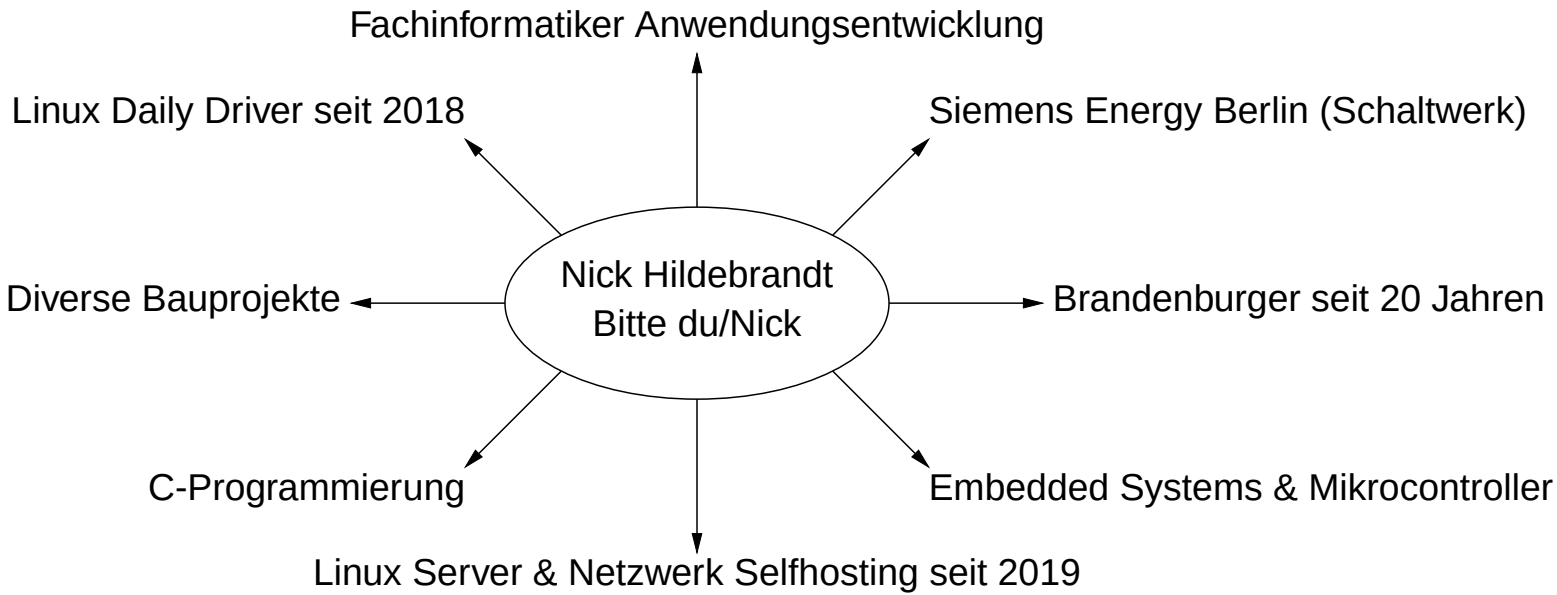
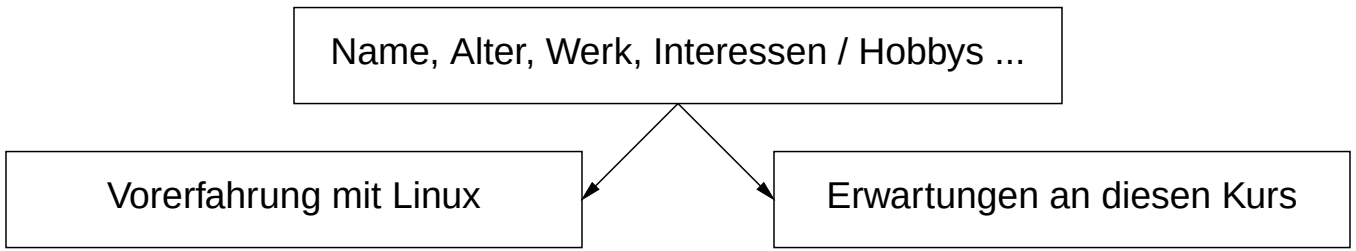
Linux Grundlagen und Projekt Kurs

Distributionen, Desktop, Installation, Kernel, Prozesse, Shell Scripting,
Systemd, Logging, SSH, Netzwerk

Ein bisschen was über mich



Etwas über euch



Was noch wichtig ist

- Bitte immer **SOFORT FRAGEN** wenn etwas unverständlich ist
- Wenn der Kurs zu eintönig, langweilig oder zu schwer ist, **SOFORT MELDEN**
- Diese Präsentation wird euch über GitHub zur Verfügung gestellt - bitte aktuell halten!
- Donnerstag kleiner Test (ca 20-30 min.) mit Note - Multiple Choice und Freitext Fragen.
- Mitarbeit Spielt für die Bewertung auch eine Rolle
- Anfangs und Endzeiten sowie Pausen und Mittagszeiten

Wochenüberblick

Als kleines Praxisprojekt kam die Idee auf, nach der Theorie am Nachmittag, einen Minecraft-Server in 2er- bis 3er-Teams aufzusetzen und zu konfigurieren.

Wochentag	**Vormittag**	**Nachmittag**
Montag	Organisatorisches • Einstieg in GNU/Linux Installation der VM • Shellgrundlagen	Vorstellung Minecraft-Projekt Installation der Server-VM
Dienstag	Benutzer & Berechtigungen • Softwareinstallation Prozesse, Systemd & Timer	Java installieren • Minecraft benutzer Minecraft-Server installieren
Mittwoch	Shell-Scripting • Logging SSH Grundlagen	Systemd Service Systemd Timer Backup Skript
Donnerstag	Minecraft-Server SSH konfigurieren & testen Testvorbereitung	Test schreiben Feedback

Einführung - Was ist Linux? - Rechercheaufgabe

1. Was ist Linux und was ist GNU?
2. Wie entstand GNU und wie entstand Linux?
3. Was haben GNU und Linux miteinander zu tun?
4. Was sind Distributionen und welche gibt es?
5. Was sind Desktopumgebungen und welche gibt es?
6. Welche UI-Toolkits nutzen die verschiedenen Desktopumgebungen?
7. Was ist ein Display-Server?
8. Wie unterscheiden sich Xorg und Wayland?

Linux interaktiv Lernen

Was ist Linux?

- Linux ist der **Kern (Kernel)** eines Betriebssystems
- Er steuert die **Hardware** (CPU, Speicher, Geräte)
- Linux selbst ist **kein vollständiges Betriebssystem**
- 1991 von Linus Torvalds entwickelt (später auch Git), als freie Alternative zu Minix (Unix)
- Mit einer E-Mail stellte er sein Projekt vor und staterte die Weltweite gemeinsame Entwicklung
- Andere Programme + Linux = komplettes **Betriebssystem**
- Wird benutzt für **PCs, Server, Smartphones (Android) und Embedded-Geräte**

Was ist Unix?

- Ursprünglich in den 1970er Jahren bei **AT&T Bell Labs** entwickelt
- Von **Ken Thompson** und **Dennis Ritchie**, die auch die **C-Programmiersprache** schufen
- **Komplettes Betriebssystem** (monolithisch), nicht nur ein Kernel
- Vor allem genutzt auf Großrechnern und Servern
- Unix selbst gibt es heute nicht mehr von AT&T
  Teile des Quellcodes wurden veröffentlicht → Grundlage für freie Systeme wie **BSD**
  Daneben existieren noch kommerzielle Varianten wie **AIX, HP-UX, Solaris**

Linux vs. Unix

Linux	**Unix**
Open-Source (GPL) In C entwickelt (Kernel + GNU-Tools) Kernel + Zusatzsoftware nötig	Proprietär, kommerziell In C entwickelt (originale Unix-Tools) Komplettes Betriebssystem (monolithisch)

*Linux ist kein Unix, sondern **Unix-like**:*
Es hat den Systemaufbau und die Unix-Philosophie übernommen,
wurde aber unabhängig als freier Kernel neu entwickelt.

Was ist GNU?

- **GNU** = „**G**NU's **N**ot **U**nix“ (ein Wortspiel)
- Eine **Sammlung freier Programme** (Compiler, Tools, Shells usw.)
- Ziel: Ein freies Unix-kompatibles Betriebssystem schaffen
- Alle Teile waren da – nur der Kernel fehlte
- Mit dem **Linux-Kernel** zusammen ergibt sich ein komplettes Betriebssystem: **GNU/Linux**
- Wichtig: Ohne GNU-Programme könnte man Linux alleine kaum benutzen

GNU, Freie Software und Open Source

- **Richard Stallman** gründete GNU und die **Free Software Foundation (FSF)** aus Frustration, weil er am MIT einen fehlerhaften Druckertreiber nicht reparieren durfte – der Quellcode war proprietär und nicht zugänglich.
- Die FSF hatte das Ziel, **freie Software** zu schaffen und diese unter der **GNU General Public License (GPL)** zu veröffentlichen.
- **Diese Lizenz garantiert vier Rechte**:
  1. Freiheit, Software auszuführen
  2. Freiheit, den Quellcode zu verstehen
  3. Freiheit, Software zu verändern
  4. Freiheit, Software weiterzugeben
- Mit **Open Source** ist die Quelloffenheit gemeint (ohne die ethische Komponente der Nutzerrechte).

Was ist eine Software Lizenz?

Große Open-Source-Projekte

Linux Kernel	Apache HTTP Server	PostgreSQL
Alle ~7 Min. Verbesserungen Release alle ~6 Monate > 15.000 Entwickler weltweit	Seit 1995 Meistgenutzter Webserver Apache Software Foundation	Ursprung 1986 in Berkeley Stabilität & Funktionsvielfalt Populäre Datenbank

- Jeder kann den Code einsehen und Fehler / Sicherheitslücken beheben (kollektive Kontrolle)
- Schnellere Entwicklung durch Zusammenarbeit und Entkopplung von einzelnen Unternehmen
- Die Existenz der Software hängt nicht vom Überleben eines Unternehmens ab

Wie Linux entwickelt wird – Beispiel für kollaborative Entwicklung

Linux Distributionen im Vergleich

	Debian	**Arch Linux**	**Red Hat Enterprise Linux**
Paketformat:	.deb	Paketformat: PKGBUILD	Paketformat: .rpm
Paketmanager:	APT	Paketmanager: pacman	Paketmanager: dnf / yum
Release-Zyklus:	Stable Releases	Rolling Release	Enterprise Releases (LTS)
Nutzungsmodell:	Frei & kostenlos	Frei & kostenlos	Bezahlte Enterprise-Version
Forks:	Ubuntu, Linux Mint	Manjaro (auf Arch-Basis)	CentOS, Fedora

***Distributionen unterscheiden sich vor allem im Release-Zyklus**,*
*dem **Paketmanager** und den **Nutzungsbedingungen** (frei oder bezahlt).*

Anwendungsbereiche von Linux

Server
Web-, Datenbank-
Cloud-Systeme

Desktop
Laptops
Workstations

Smartphones
Android

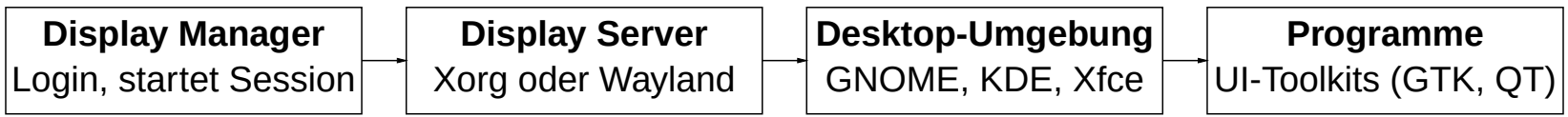
Embedded-Systeme
Smart-Home
IoT-Geräte

Supercomputer
Top-500 Supercomputer

Industrie & Automotive
Roboter
Steuerungen

Warum ausgerechnet Linux?

Aufbau der Desktopumgebungen unter Linux



- **Display Manager**: Login-Bildschirm, startet den Display-Server und die Desktop-Sitzung
- **Display Server**: Vermittelt zwischen Programmen und Hardware (Xorg = älter, Wayland = moderner, sicherer)
- **GUI-Toolkits**: Bibliotheken wie GTK oder Qt, die grafische Elemente (Fenster, Buttons, Menüs) bereitstellen

Wayland ist eine modernere reine Client Alternative zu X.Org, arbeitet besser mit 3D-Grafikkarten zusammen und verhindert, dass Programme Eingaben anderer ausspähen – schränkt aber Funktionen wie Bildschirmaufnahmen ein.

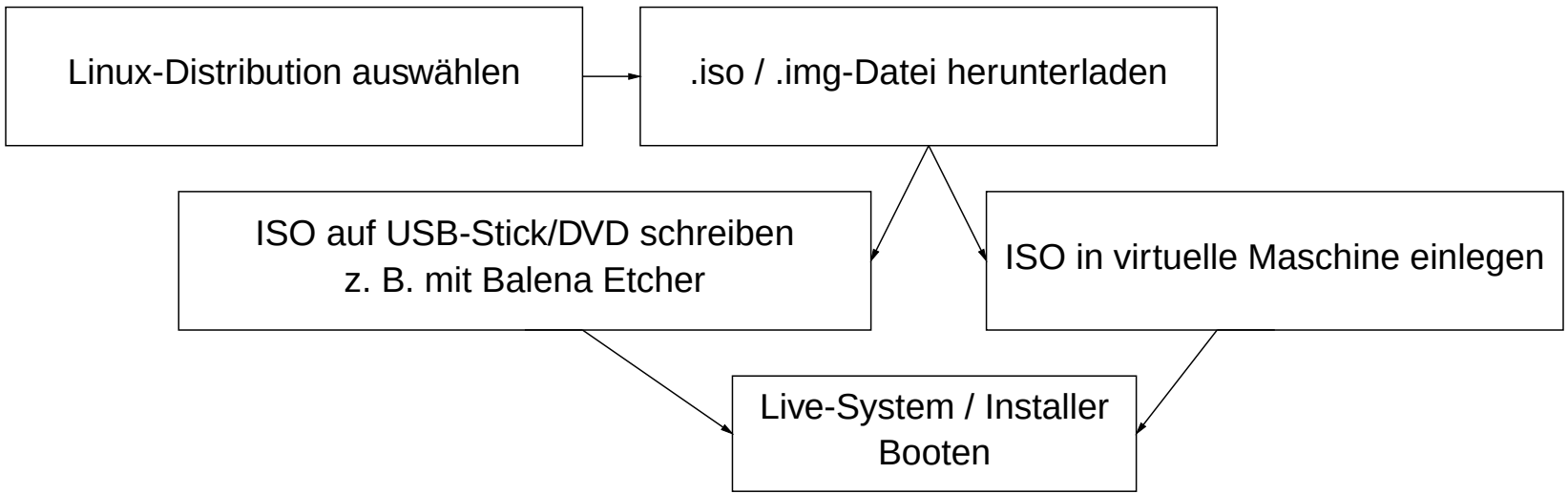
Linux-Desktops im Vergleich

	GNOME (seit 1999)	**KDE Plasma (seit 1996)**	**Xfce (seit 1996)**
Toolkit:	GTK 4	Qt 5/6	GTK 2/3
Display-Server:	Wayland	X11 (Wayland im Aufbau)	X11
Charakteristik:	Minimalistisch	Sehr anpassbar	Ressourcenschonend

Alle Desktops lassen sich unter jeder Distribution installieren.
Einige Distributionen sind jedoch auf einen bestimmten Desktop oder eine bestimmte Nutzererfahrung optimiert.

Darüber hinaus existieren viele weitere Desktops und sogenannte Window-Manager,
die es ermöglichen, eigene Arbeitsumgebungen zu gestalten oder für Embedded-Systeme zu optimieren.

Linux installieren – Übersicht



Linux unter Windows ausführen durch Virtualisierung

Virtualisierung bedeutet, dass ein Hypervisor eine komplette Rechnerumgebung nachbildet. So kann ein Betriebssystem wie Linux innerhalb von Windows laufen. Ein Beispiel für einen Hypervisor ist VirtualBox.

1. VirtualBox von der offiziellen Webseite herunterladen und installieren
2. Neue virtuelle Maschine anlegen und Name + Typ **Linux - Debian** auswählen
3. Arbeitsspeicher **(RAM) auf 4096MB**
4. Virtuelle **Festplattengröße auf 30GB**
5. ISO-Datei von Debian herunterladen und in die VM einlegen
6. **Virtuelle Maschine starten** und Linux-Installer durchlaufen

Debian installieren

1. Debian-ISO herunterladen – **netinst**, um automatisch immer die aktuellste Version zu installieren
2. Vom Medium starten und **Graphical install** auswählen - Sprache **Deutsch** wählen
3. Netzwerk automatisch konfigurieren - Hostname setzen **(Namen der Maschine)**
4. **Kein Root-Passwort** (Felder leer lassen) - **Name / Passwort für den Benutzer** vergeben – dieser ist dann **gleichzeitig Administrator**
5. Partitionierung: **Geführt** – **gesamte Festplatte verwenden** - **Änderungen auf die Festplatte schreiben**
6. Paketmanager für Deutschland (deb.debian.org) konfigurieren - **GNOME** und **Standard-Systemw-erkzeuge** auswählen (Bei einem Server **nur Standard-Systemwerkzeuge**)
7. **Installation des Bootloaders** (GRUB) bestätigen - danach **Neustarten**

Treiber unter Linux

Unter Linux liefert der Kernel die meisten Treiber bereits mit und lädt beim Start automatisch die passenden Module. Fehlen Treiber, können zusätzliche Kernel-Module – auch proprietäre – installiert werden.

Partitionsschema unter Linux

Eine Partition teilt eine Festplatte in Abschnitte, die für verschiedene Zwecke genutzt und mit unterschiedlichen Dateisystemen formatiert werden können.

<table>
<tr><td>EFI (FAT32)
Notwendig für UEFI-Systeme
Enthält den Bootloader
Bei BIOS-Boot Optional (ersten 512 Byte)</td><td>Swap
Auslagerungsspeicher für Rammangel
Auch als Swap-Datei (dynamisch - ZRAM)
Nicht Essentiell (SSD)</td></tr>
<tr><td colspan="2">Daten (ext4, Btrfs, ZFS)
Enthält System & Benutzerdaten
Btrfs/ZFS ermöglichen Datenintegrität & mehrere Festplatten
FHS-Ordner können eigene Partitionen sein</td></tr>
</table>

Filesystem Hierarchy Standard (FHS)

Der Filesystem Hierarchy Standard (FHS) stammt aus der Unix-Welt und legt fest, welche Verzeichnisse ein Betriebssystem besitzen muss und wie sie strukturiert sind.

- Unter Linux werden Pfade mit **/ (Forward Slash)** geschrieben - bei Windows **\ (Backslash)**
- Es gibt **keine Laufwerksbuchstaben** Geräte werden in einen Pfad eingebunden
- Linux, macOS BSD haben eigene Varianten vom FHS entwickelt und dieses erweitert

FHS-Ordner Funktionen

/bin	Wichtige Benutzerprogramme (essenzielle Befehle)
/sbin	Systemprogramme für Administration
/etc	Systemkonfigurationsdateien
/proc /dev /sys	Virtuelle Dateisysteme des Kernels mit Geräten und Prozessen
/var	Veränderliche Daten (Logs, Spools, Caches)
/tmp	Temporäre Dateien
/usr	Anwendungsprogramme und Bibliotheken
/lib	Wichtige Systembibliotheken
/home	Persönliche Verzeichnisse der Benutzer
/boot	Kernel, Bootloader und Startdateien
/mnt /media	Mountpoints für z.B. Festplatten
/opt	Optionale Zusatzsoftware
/root	Home-Verzeichnis des Administrators (root)

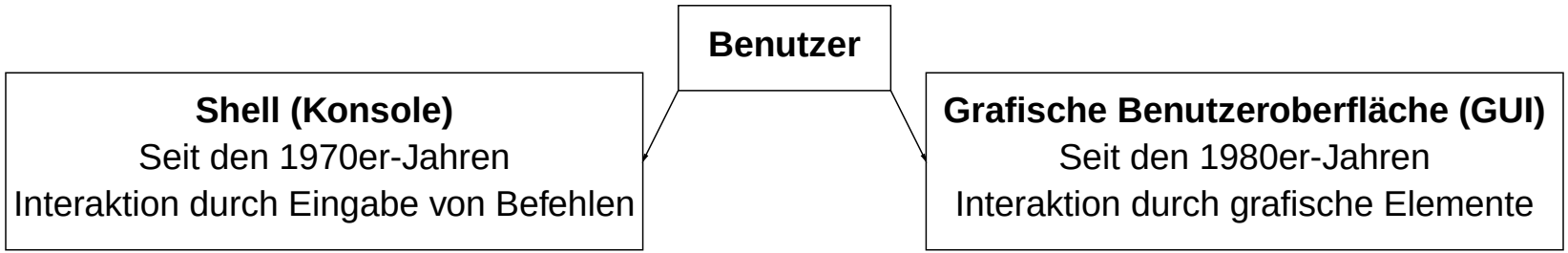
Alles ist eine Datei

Unter Linux (Unix) gilt die Philosophie: ***Alles ist eine Datei***

- **/dev** – Geräte (Festplatten, Terminals, USB-Sticks, Drucker)
- **/sys** – Kernel- und Geräteeinstellungen (z. B. Energieverwaltung, Treiber-Infos)
- **/proc** – Informationen über Prozesse und den Kernel (virtuelles Dateisystem)
- **/etc** – Konfigurationsdateien für Dienste und das System

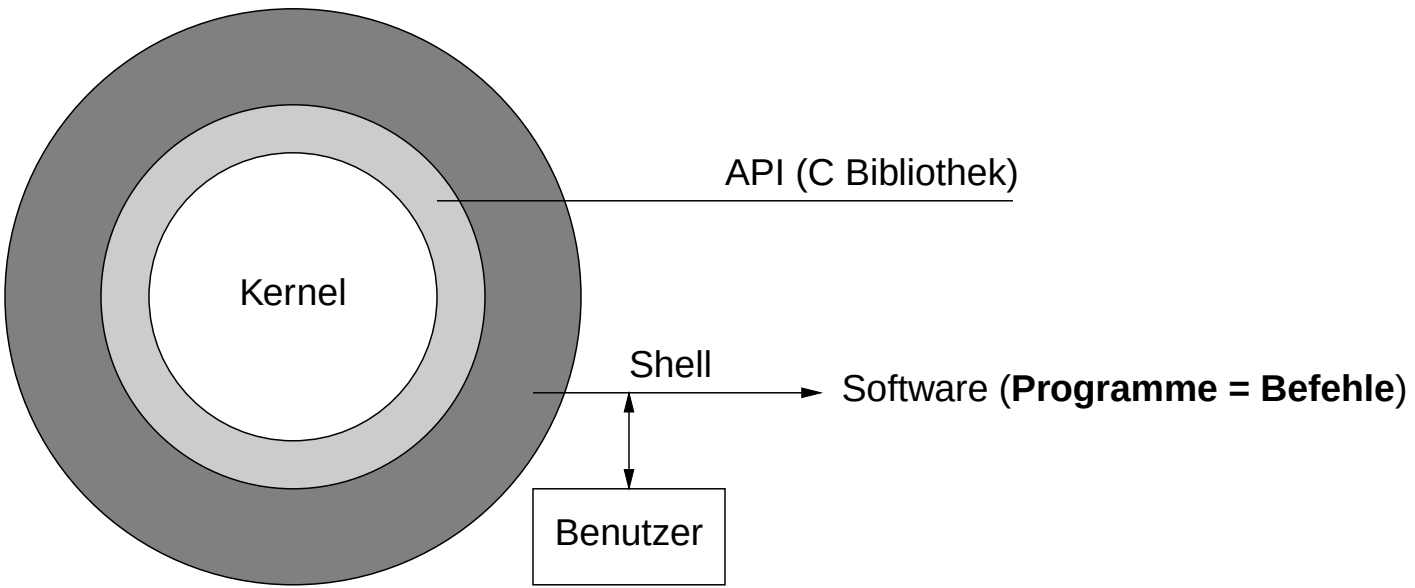
Vorteil: *Alles kann mit denselben Werkzeugen über das Dateisystem gelesen, geschrieben und bearbeitet werden.*

Interaktion mit dem Computer



Linux kann sowohl mit grafischen als auch mit rein textbasierten Oberflächen genutzt werden. Auf Servern wird aus Sicherheits- und Stabilitätsgründen meist kein Desktop installiert, sondern ausschließlich die Shell verwendet.

Was ist eine Shell?



Sh, Bash und Zsh

Eigenschaft	**sh**	**bash**	**zsh**
Standard auf	Unix, BSD, POSIX	Linux, älteres macOS	macOS, Linux, BSD
POSIX-kompatibel	Ja	Teilweise	Nein
sh-kompatibel	Ja	Ja (größtenteils)	Teilweise
Skripting	Grundlegend	Erweiterte Funktionen	Noch mächtiger
Autovervollständigung	Einfach	Besser	Sehr fortgeschritten
Globbing	Basis (‘*‘, ‘?‘)	Erweitert (‘**‘, ‘@( )‘)	Sehr mächtig (‘<1-100>‘)

Sh wurde durch POSIX standardisiert und z. B. in dash implementiert, während Bash, zsh und fish mehr Funktionen bieten, aber nicht dem Standard folgen.

Bash und sh sind der Standard auf Servern, Embedded- und Netzwerkgeräten.

Was ist eigentlich POSIX?

***P**ortable **O**perating **S**ystem **I**nterface*
*X = Uni**x***

POSIX ist ein Standard für die Umsetzung eines Unix- oder unixähnlichen Betriebssystems, der von der IEEE (Institute of Electrical and Electronics Engineers) seit 1988 entwickelt wird; er spezifiziert z. B. Shell-Syntax, Dateisystem- und Prozessverwaltung, um die Portabilität von Software zwischen verschiedenen Unix-Systemen zu gewährleisten.

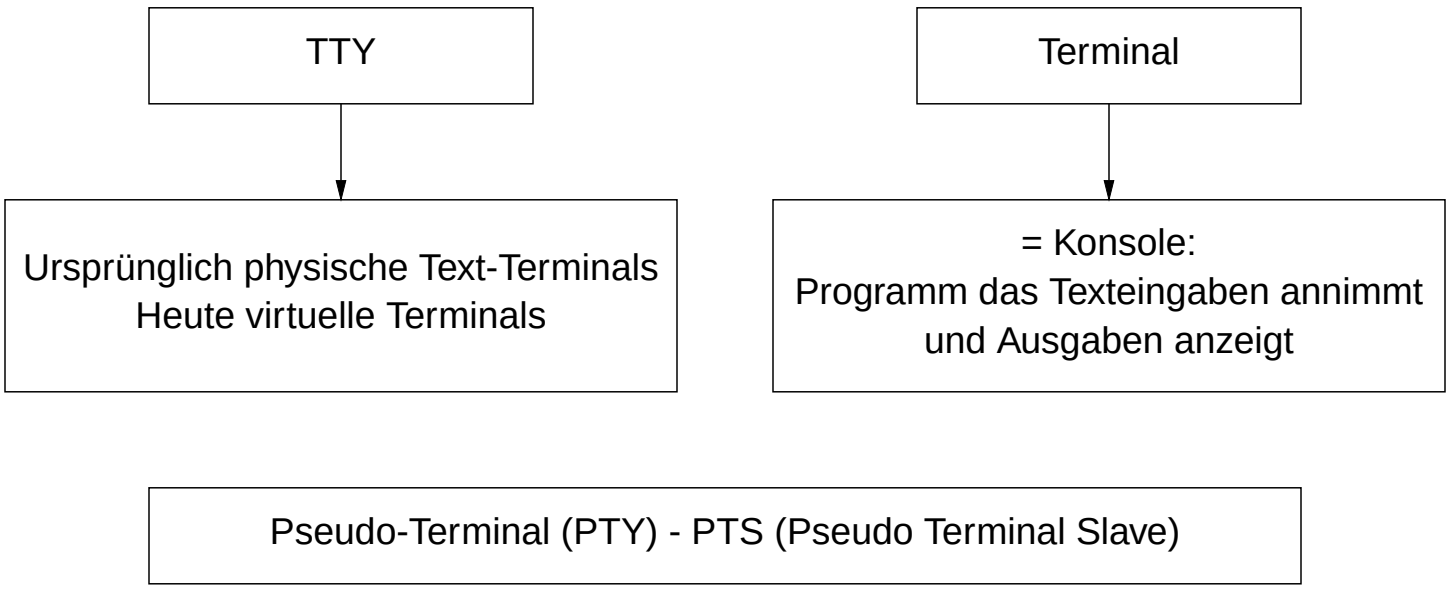
Grafische Terminal-Emulatoren

Ein grafischer Terminal-Emulator ist ein Programm mit Fensteroberfläche, das ein klassisches Text-Terminal nachbildet, in dem man Befehle eingeben und deren Ausgaben direkt angezeigt bekommen kann.

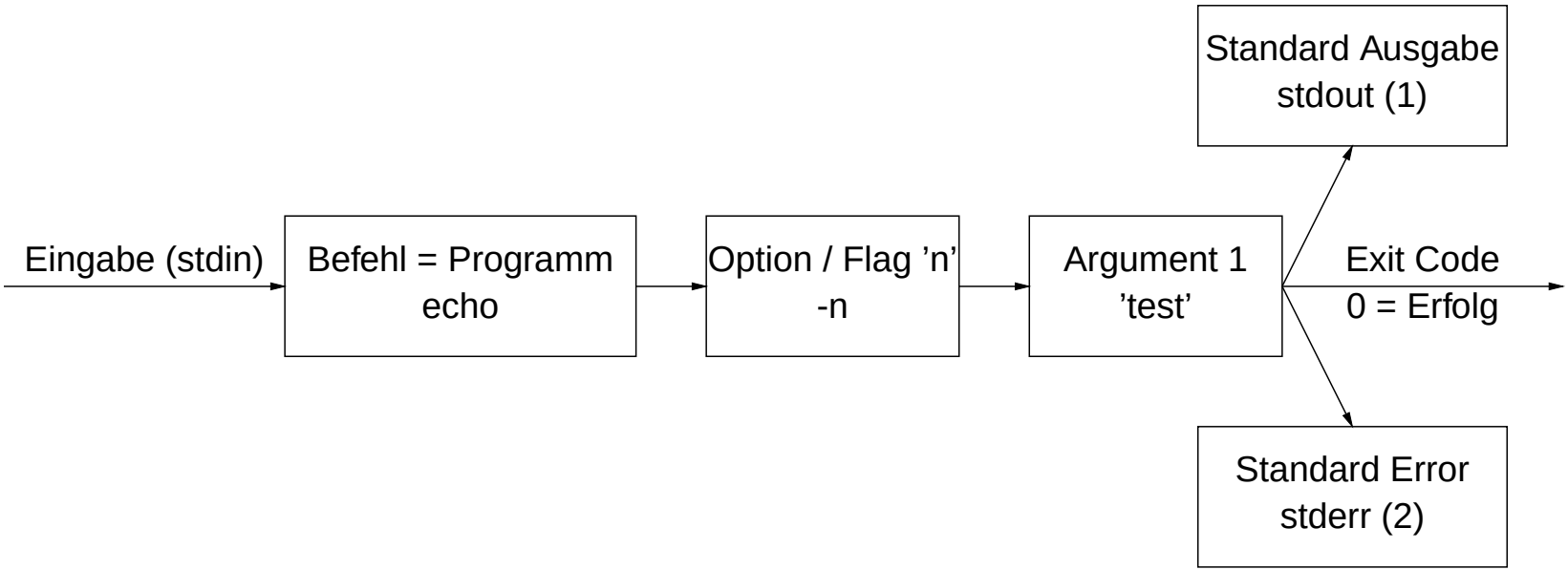
- **GNOME Terminal** (Standard unter GNOME 3)
- **GNOME Console** (moderne, einfache Alternative)
- **Konsole** (Terminal-Emulator der KDE-Plasma-Umgebung)

*Unter GNOME startet man das Terminal über **Aktivitäten** - **Terminal** (Strg + Alt + T).*

Unterschied Terminal - TTY



Ein Befehl im Detail



Pfade - Wo Befehle ausgeführt werden

*Ein **Pfad** beschreibt den Ort einer Datei oder eines Verzeichnisses im Dateisystem. Dabei gibt es:*

- **Absolute Pfade**: beginnen immer mit **/**, z. B. **/home/nick/Dokumente**
- **Relative Pfade**: beziehen sich auf das aktuelle Verzeichnis, z. B. **../Bilder**

```
# Aktuelles Verzeichnis anzeigen
pwd

# Beispielausgabe:
/home/nick/Dokumente
```

ls - anzeigen, was sich in einem Ordner befindet

*Mit **ls** lässt sich der Inhalt eines Verzeichnisses anzeigen.*
*Mit der Option **-a** werden auch versteckte Dateien (beginnen mit Punkt) angezeigt.*

```
# Aktuelles Verzeichnis (relativer Pfad)
ls

# Absoluter Pfad
ls /home/nick/Dokumente

# Mit versteckten Dateien
ls -a
```

cd – in ein anderes Verzeichnis wechseln

*Mit **cd** kann man zwischen Verzeichnissen wechseln.*
Es lassen sich sowohl relative als auch absolute Pfade verwenden.

```
# Relativer Pfad (ein Verzeichnis nach oben)
cd ..

# Absoluter Pfad
cd /home/nick/Dokumente

# Zurück ins Home-Verzeichnis
cd ~
```

cp – Dateien und Ordner kopieren

*Mit **cp** werden Dateien und Verzeichnisse kopiert.*
*Für Ordner muss die Option **-r** (rekursiv) verwendet werden.*

```
# Datei mit relativem Pfad kopieren
cp datei.txt ../backup/

# Datei mit absolutem Pfad kopieren
cp /home/nick/test.txt /home/nick/Dokumente/

# Ganzen Ordner kopieren (rekursiv)
cp -r /home/nick/Bilder /home/nick/Backup/
```

mv – Dateien und Ordner verschieben/umbenennen

*Mit **mv** können Dateien und Verzeichnisse verschoben oder umbenannt werden.*
Es ist kein rekursives Flag erforderlich.

```
# Datei verschieben (relativ)
mv datei.txt ../backup/

# Datei verschieben (absolut)
mv /home/nick/test.txt /home/nick/Dokumente/

# Ordner verschieben oder umbenennen
mv /home/nick/Bilder /home/nick/Backup/
```

rm – Dateien und Ordner löschen

*Mit **rm** werden Dateien gelöscht.*
*Für ganze Ordner und deren Inhalte wird die Option **-r** (rekursiv) benötigt.*

```
# Einzelne Datei löschen
rm datei.txt

# Ordner rekursiv löschen
rm -r alter_ordner
```

ln – Verknüpfungen anlegen

*Mit **ln** lassen sich Verknüpfungen auf Dateien erstellen.*
*Es gibt **harte Links** (Zeigen auf die Ziel Inode Nummer)*
*und **symbolische Links** (zeigen auf den Ziel Pfad).*

```
# Harter Link (zweiter Name für dieselbe Datei)
ln original.txt kopie.txt

# Symbolischer Link (wie eine Abkürzung)
ln -s /home/nick/original.txt link.txt

# Symbolischen Link löschen
rm link.txt
```

Inode-Nummern

Jede Datei im Linux-Dateisystem hat eine eindeutige
***Inode-Nummer** welche Metadaten wie Besitzer, Rechte, Größe*
und Speicherort enthält. Mehrere Dateinamen können auf dieselbe Inode zeigen.

```
# Mit -i zeigt ls die Inode-Nummern an
ls -i

# Beispielausgabe:
123456 datei.txt
123456 kopie.txt   <-- harter Link (gleiche Inode)
```

cat, less, head, tail – Ausgeben von Dateien

*Mit **cat, less, head** und **tail** können Inhalte von Dateien angezeigt werden.*

```
# cat – gesamte Datei ausgeben
cat datei.txt

# less – gesamte Datei mit scrolling ausgeben
less datei.txt

# head – erste Zeilen anzeigen / tail – letzte Zeilen anzeigen
head datei.txt
head -n 5 datei.txt
tail datei.txt
tail -f /var/log/syslog   # "live" folgen
```

grep – Text in Dateien suchen

grep *durchsucht Dateien nach Textmustern (Strings oder regulären Ausdrücken).*
So kann man gezielt Zeilen finden, ohne die gesamte Datei öffnen zu müssen.

```
# In einer einzelnen Datei nach “Fehler” suchen
grep “Fehler” system.log

# Groß-/Kleinschreibung ignorieren (-i)
grep -i “warnung” messages.log

# Rekursiv in allen Dateien ab aktuellem Verzeichnis suchen (-r)
grep -r “TODO”

# Zeilennummern mit ausgeben: -n
# Nur Dateinamen ausgeben: -l
```

find – Dateien im Dateisystem suchen

find *durchsucht Verzeichnisse nach Dateien oder Ordnern*
und erlaubt die Filterung nach Typ, Name, Größe, Datum und vielem mehr.

```
# Alle Dateien mit bestimmtem Namen finden
find /home/nick -name “test.txt”

# Suche unabhängig von Groß-/Kleinschreibung
find /home/nick -iname “bild.jpg”

# Nur Verzeichnisse (d = directory) anzeigen
find /etc -type d

# Nur reguläre Dateien (f = file) anzeigen
find /var/log -type f
```

Hilfe bei Kommandos erhalten

Unter Linux gibt es mehrere Möglichkeiten, sich Hilfe zu einem Befehl anzeigen zu lassen.
Viele Programme bieten eine ***-h****- oder* ***--help****-Option.*
Detaillierte Informationen stehen in den ***Manpages*** *zur Verfügung.*

```
# Kurzinfo zum Befehl
ls --help
cp -h

# Ausführliche Dokumentation (Manpage)
man ls
man cp
```

Dateien bearbeiten mit Nano

Nano *ist ein einfacher Texteditor für die Shell, leicht verständlich und für Einsteiger geeignet.*

```
nano datei.txt # Datei öffnen/bearbeiten

# Bewegen in der Datei: Pfeiltasten
# Speichern: Strg + O
# Beenden: Strg + X
# Text suchen Strg + W
```

Dateien bearbeiten mit Vim

Vim *ist ein komplexer Editor mit vielen Funktionen, der durch Plugins zu einer vollwertigen Entwicklungsumgebung erweitert werden kann.*

Vim in 100 Sekunden

```
# Datei öffnen/bearbeiten
vim datei.txt

# Bewegung im Text
 h, j, k, l  (links, runter, hoch, rechts) oder  Pfeiltasten

# In den Insert-Modus wechseln um Text einzugeben
 i  - ( ESC ) zum verlassen des Insert-Modus

# Speichern / Beenden
:write   oder   :w⇒ :quit oder :q
```

Übungsaufgabe 1: Kommandozeilen-Basics & Editoren

1. Lege in deinem Home-Verzeichnis einen neuen Ordner ‘uebung‘ an und wechsle in diesen Ordner
2. Erstelle darin eine neue Datei ‘test.txt‘ und schreibe mit einem Editor einen kurzen Text hinein
3. Zeige dein aktuelles Arbeitsverzeichnis an und überprüfe den Inhalt des Ordners
4. Kopiere die Datei ‘test.txt‘ in eine neue Datei ‘kopie.txt‘ im selben Ordner
5. Verschiebe die Datei ‘kopie.txt‘ in ein neu erstelltes Unterverzeichnis ‘backup‘
6. Lösche anschließend den gesamten Ordner ‘backup‘
7. Lege einen Symbolischen Link von einer Dateiebene über deinem aktuellen Verzeichnis auf ‘test.txt‘ an
8. Zeige die ersten 2 und letzten 10 Zeilen von ‘test.txt‘, sowie einmal die gesamte Datei an
9. Suche in ‘test.txt‘ nach einem bestimmten Wort, das du vorher hineingeschrieben hast

Praxisprojekt Minecraft-Server

1. Lege eine neue virtuelle Maschine in VirtualBox an und installiere ein aktuelles Debian-System – bewusst **ohne grafische Benutzeroberfläche**, um den Fokus auf die Shell zu legen.
2. Richte in VirtualBox die notwendigen Portfreigaben ein: • **Port 22/TCP** für den SSH-Zugang • **Port 25565/TCP** für den Minecraft-Server
3. Mache dich mit der Bedienung der Kommandozeile und den ersten Linux-Grundbefehlen vertraut – dies wird später für die Administration des Servers wichtig.

Administratorberechtigungen unter Linux

Wie unter Windows sind für systemverändernde Aktionen unter Linux höhere Rechte nötig.
*Dafür gibt es den **Root−Nutzer**, der volle Systemrechte hat. Oft ist das Root-Konto deaktiviert.*
*Stattdessen erhalten normale Benutzer per **sudo** oder **doas** temporär Root-Rechte.*

```
# Ein einzelner Befehl als Root ausführen
sudo BEFEHL z.B sudo rm -r /ordner

# Eine Interaktive Root-Shell starten
sudo -i # Mit exit wieder verlassen
sudo -u BENUTZER # Befehle als anderer Benutzer ausführen
```

***Wichtig:** Damit ein Benutzer sudo verwenden kann, muss er in der **sudo** Gruppe sein.*
*Bei einigen Distributionen wird stattdessen die Gruppe **wheel** verwendet.*

Benutzer & Gruppen – Einführung

*Linux ist ein **Mehrbenutzersystem**: Es können mehrere Benutzer gleichzeitig am System arbeiten*
*Benutzer können reale **Personen oder Dienste** (z. B. Webserver) sein. Benutzer werden in **Gruppen** organisiert. Ein Benutzer kann in **mehreren Gruppen** sein.*
Über Gruppen werden Berechtigungen auf Geräte, Dateien und Befehle vergeben.

Benutzer	**Gruppe**
Benutzername (login)	Gruppenname
Benutzer-ID (UID)	Gruppen-ID (GID)
Passwort-Hash	Mitgliederliste (Benutzer)
Passwort-Aging (min/max/warn/expire)	-
Primäre Gruppen-ID (GID)	-
Home-Verzeichnis	-
Login-Shell	-
Kommentar	Kommentar

Benutzer & Gruppen – Informationen anzeigen

```
# Aktuellen Benutzer anzeigen
whoami

# UID, GID und alle Gruppenmitgliedschaften
id

# Nur Gruppenmitgliedschaften
groups

# Details zu einem spezifischen Eintrag
getent passwd BENUTZERNAME
getent group  GRUPPENNAME
```

*Benutzer- und Gruppendaten stehen klassisch in **/etc/passwd** und **/etc/group***

Benutzer anlegen – interaktiv (adduser)

adduser *(Debian/Ubuntu) ist ein Assistent welcher das Home-Verzeichnis anlegt, Shell und Passwort setzt und eine gleichnamige Primärgruppe erstellt.*

```
# @root
sudo adduser alice

# Folgt interaktiver Dialog (Passwort, Name, etc.)
# Danach kann man sich als alice anmelden
su alice
```

Benutzer anlegen – nicht-interaktiv (useradd)

***useradd** ist das **low-level**-Tool (universell verfügbar)*
was es ermöglicht, alle Einstellungen zu setzen.

```
# @root
# -m: Home anlegen, -s: Login-Shell setzen,
# -c: Kommentar, -U: Primärgruppe anlegen
sudo useradd -m -s /bin/bash -c "Bob Beispiel" -U bob

# Passwort für bob setzen
sudo passwd bob
```

*Standardwerte kommen aus **/etc/login.defs** und **/etc/default/useradd***

Dienst-/Service-Benutzer anlegen

Dienstkonten haben i. d. R. ***keine Login-Shell*** *und*
kein Home. *Sie dienen Prozessen (Diensten), um Rechte sauber zu trennen.*

```
# @root
# --system: UID im Systembereich <1000,
# --shell /usr/sbin/nologin: keine Anmeldung möglich
sudo useradd --system --no-create-home \
     --shell /usr/sbin/nologin neuer_dienst

# Alternativ mit eigener Gruppe:
sudo useradd --system --user-group --no-create-home \
     --shell /usr/sbin/nologin appuser
```

Service-Benutzer werden oft in ***systemd-Units*** *via* ***User=***
und ***Group=*** *gesetzt und nach dem Service benannt.*

Benutzer löschen

Beim Löschen entscheiden: Home-Verzeichnis und Mailspool mit entfernen?
Achtung bei gemeinsam genutzten Gruppen/Verzeichnissen!

```
# @root
# Komfortbefehl (Debian/Ubuntu):
sudo deluser alice
sudo deluser --remove-home alice   # inkl. /home/alice

# Generisch:
sudo userdel bob
sudo userdel -r bob                # inkl. HOME und Mailspool
```

Gruppen verwalten

```
# @root
# Gruppe anlegen
sudo groupadd developers

# Benutzer zu Zusatzgruppei sudo hinzufügen ⇒ Benutzer zum Admin machen
sudo usermod -aG sudo alice

# Benutzer aus Gruppe entfernen
sudo gpasswd -d alice developers

# Gruppe löschen
sudo groupdel developers
```

*-**aG** ist entscheidend: Ohne -**a** werden bestehende Zusatzgruppen überschrieben!*

Dateiberechtigungen unter Linux

Unter Linux gibt es drei grundlegende Berechtigungen:
Lesen, Schreiben *und* ***Ausführen*** *Diese gelten jeweils für:*
einen ***Benutzer****, eine* ***Gruppe*** *und alle* ***Anderen***

Lesen (r)	Dateiinhalt ansehen / Ordner auflisten
Schreiben (w)	Datei verändern / Dateien im Ordner hinzufügen oder löschen
Ausführen (x)	Programm starten / Ordner öffnen

Berechtigungen anzeigen lassen

*Mit **ls -l** lassen sich die Berechtigungen, Eigentümer und Gruppen einer Datei anzeigen.*

```
ls -l

# Beispielausgabe:
-rwxr-xr--  1 nick users   1234 Mär  5 10:00 skript.sh
```

- rwx: Der Besitzer **nick** darf die Datei lesen, schreiben und ausführen.
- r-x: Mitglieder der Gruppe **users** dürfen die Datei lesen und ausführen, aber nicht schreiben.
- r--: Alle anderen dürfen die Datei nur lesen.

Eigentümer und Gruppe ändern

*Mit **chown** (change owner) und **chgrp** kann man den Besitzer und die Gruppe einer Datei ändern.*

```
# Besitzer ändern
chown benutzer datei.txt

# Besitzer und Gruppe ändern
chown benutzer:gruppe datei.txt

# Nur Gruppe ändern
chgrp gruppe datei.txt

# Rekursiv auf Verzeichnisse anwenden
chown -R benutzer:gruppe ordner/
```

Berechtigungen setzen

*Mit **chmod** können Berechtigungen gesetzt werden.*
Dies ist möglich mit Buchstaben oder Zahlen (Oktalnotation).

```
# Mit Buchstaben
chmod u+x datei.sh   # Benutzer darf ausführen
chmod g-w datei.txt  # Gruppe darf nicht mehr schreiben
chmod o+r datei.txt  # Andere dürfen lesen

# Mit Zahlen (r=4, w=2, x=1)
chmod 755 datei.sh   # rwx für Benutzer, rx für Gruppe/Andere
chmod 644 datei.txt  # rw für Benutzer, r für Gruppe/Andere
```

Achtung: Ordner müssen immer ausführbar sein!

Oktalnotation von Berechtigungen

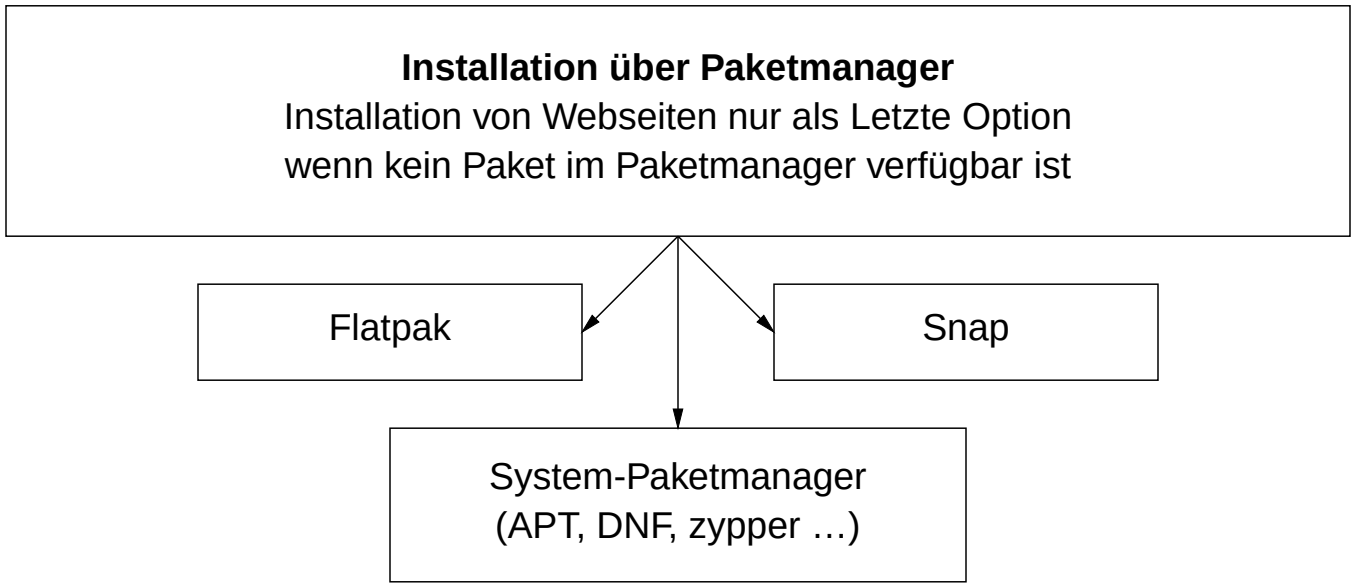
*Die Oktalnotation ist eine **Alternative zur Buchstabendarstellung** (r, w, x) und fasst die Rechte für Benutzer, Gruppe und Andere inform von **3 Oktalzahlen** (0-7) zusammen.*

Oktalwert	Symbolische Darstellung
0	---
1	--x
2	-w-
3	-wx
4	r--
5	r-x
6	rw-
7	rwx

Übungsaufgabe 2: Benutzer, Gruppen & Berechtigungen

1. Arbeite als Administrator und lege einen neuen Benutzer an
2. Erstelle eine neue Gruppe und füge den Benutzer dieser Gruppe hinzu
3. Lege einen gemeinsamen Ordner an, in dem nur Mitglieder der Gruppe lesen und schreiben dürfen
4. Erstelle einen zweiten Benutzer der die daten im Ordner nur Lesen kann
5. Entziehe allen anderen Benutzern (außer Besitzer und Gruppe) die Rechte auf diesen Ordner
6. Prüfe mit verschiedenen Benutzern, ob die Berechtigungen korrekt greifen

Softwareinstallation unter Linux



Paketmanager unter Linux – APT

APT *ist der Standard-Paketmanager für Debian, Ubuntu und viele Derivate. Es verwaltet Abhängigkeiten automatisch, aktualisiert* **DEB-Pakete** *und kann sowohl über Kommandozeile als auch grafische Oberflächen genutzt werden. Die Software stammt aus den* **Paketquellen**, *die regelmäßig aktualisiert werden müssen.*

```
# Paketlisten aktualisieren
sudo apt update

# Installierte Pakete aktualisieren
sudo apt upgrade

# Erweiterte Aktualisierung (neue Abhängigkeiten, Kernel usw.)
sudo apt full-upgrade
# alias: sudo apt dist-upgrade
```

Software installieren mit APT

```
# Paketlisten aktualisieren
sudo apt update

# Paket installieren
sudo apt install PAKETNAME

# Nach Paketen suchen
sudo apt search SUCHBEGRIFF
```

Automatische Updates & Neustart von Diensten

Mit **unattended-upgrades** *können Sicherheits- und Paketupdates automatisch im Hintergrund eingespielt werden.*

Das Tool **needrestart** *prüft nach einem Update, welche Dienste oder Bibliotheken neu gestartet werden müssen, damit die Änderungen wirksam werden.*

```
# Automatische Updates aktivieren
sudo apt install unattended-upgrades

# Neustart relevanter Dienste nach Updates
sudo apt install needrestart
```

Moderne Paketformate – Flatpak & Snap

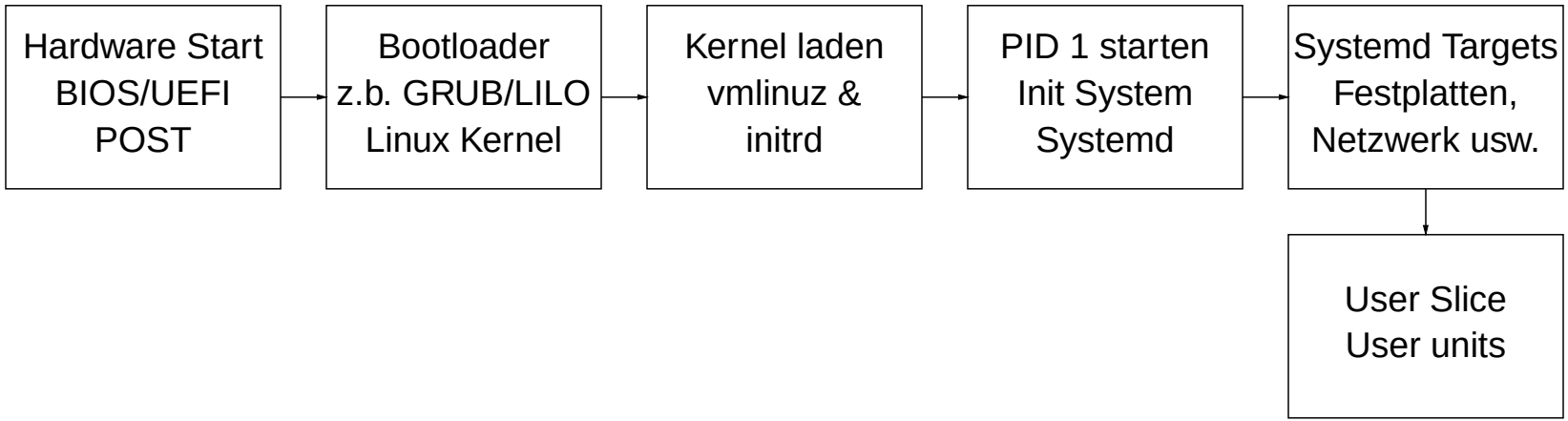
Moderne Paketformate bündeln alle Abhängigkeiten mit in einem Paket.
Das erleichtert die Installation verschiedener Versionen und macht die Pakete auf allen Linux-Distributionen nutzbar.

```
# Flatpak installieren (Debian/Ubuntu)
sudo apt install flatpak

# Flatpak-Standard-Repository (Flathub) hinzufügen
flatpak remote-add --if-not-exists flathub \
https://flathub.org/repo/flathub.flatpakrepo

# Paket über Flatpak installieren
flatpak install flathub org.mozilla.firefox
```

Übungsaufgabe 3: Paketverwaltung mit APT & Flatpak

1. Aktualisiere die Paketquellen
2. Aktualisiere alle installierten Pakete und das System
3. Suche nach einem Programm deiner Wahl (z. B. gedit)
4. Installiere ein Programm (z. B. htop)
5. Konfiguration Flatpak und installiere eine Anwendung aus Flathub (z. B. Firefox)
6. Richte automatische Updates mit unattended-upgrades ein
7. Sorge dafür, dass nach Updates automatisch Dienste neu gestartet werden

Der Linux Bootvorgang



Das Init System startet für alle Funktionen (WLAN, Display...) das richtige Programm - ***einen Prozess***

Systemd Slice *= Kapselung von Ressourcen und Berechtigungen →* ***Container***

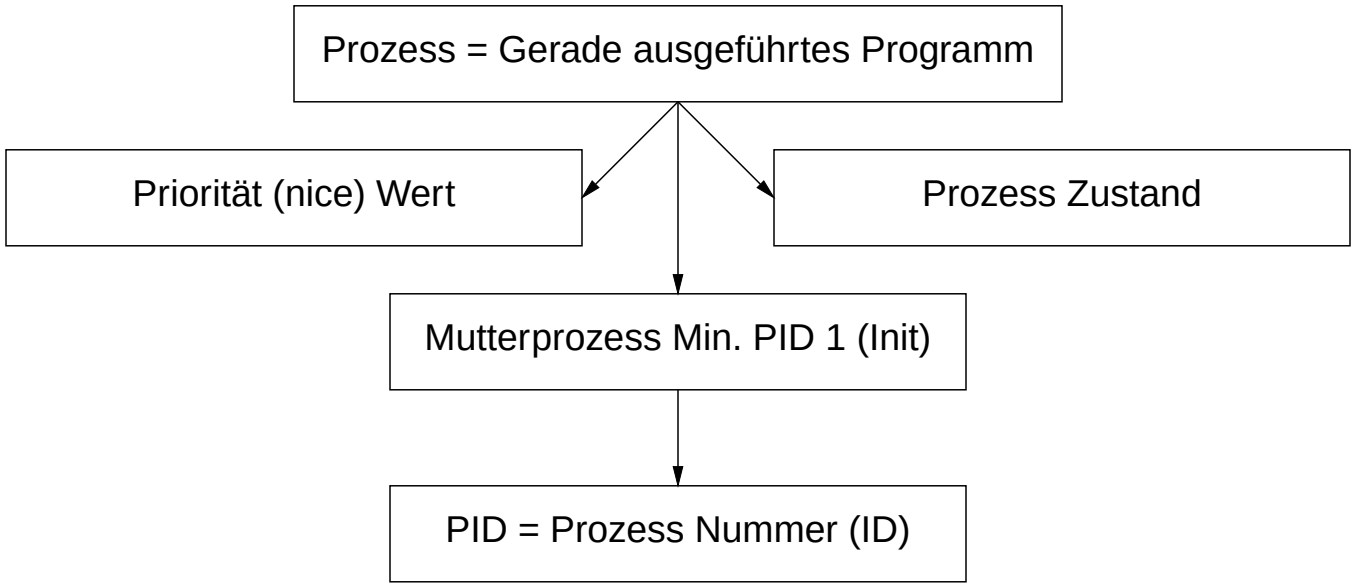
Runlevel und Systemd Targets

Runlevel	Modus	Systemd Target
0	Herunterfahren	poweroff.target
1	Einzelbenutzermodus ohne Netzwerk und GUI	rescue.target
2	Mehrbenutzerbetrieb ohne Netzwerk und GUI	wie rescue.target
3	Mehrbenutzerbetrieb ohne GUI	multi-user.target
4	Nicht definiert	Nicht definiert
5	Mehrbenutzerbetrieb mit GUI	graphical.target
6	Neustart	reboot.target

*Runlevel wurden von **Systemd Targets** abgelöst*

*init und runnlevel geht noch: **systemctl isolate und systemctl get-default** ist bevorzugt*

Was ist eigentlich ein Prozess?



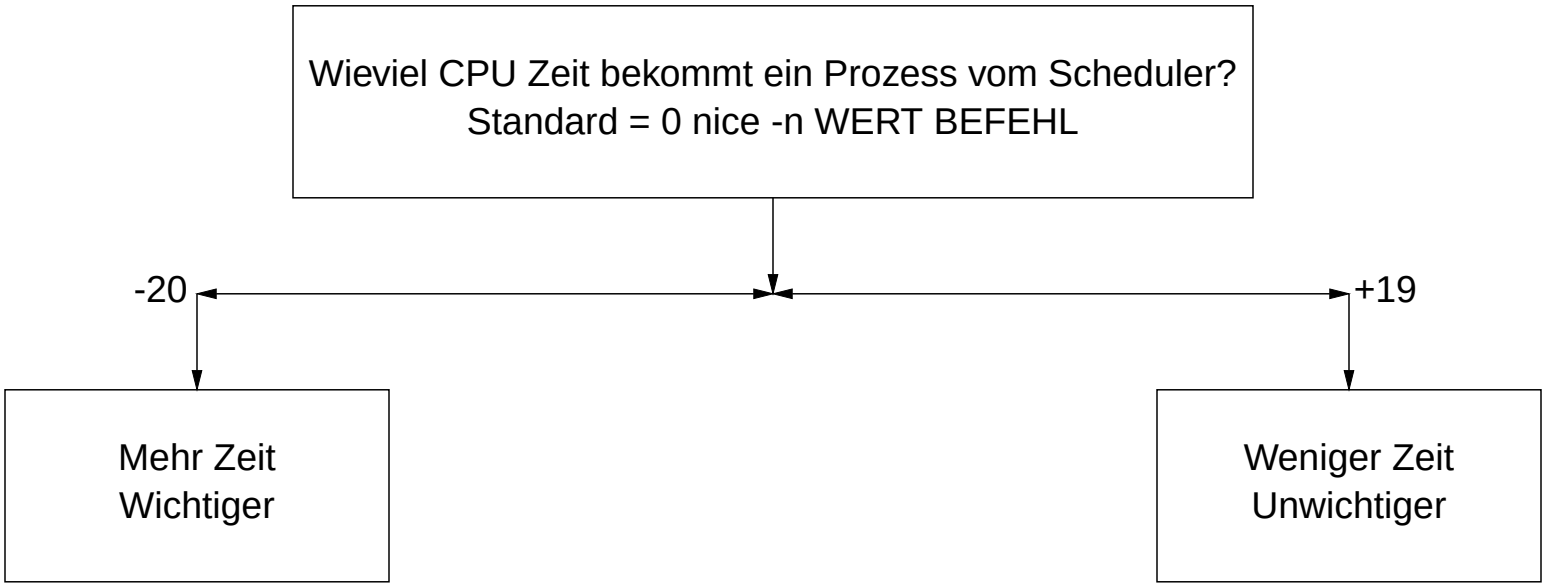
Prozess Zustände

Status	Name	Zustand
R	**Running**	Prozess läuft und führt Anweisungen aus
S	**Schlafend**	Prozess wartet auf das Eintreten eines Ereignisses, z.B. auf Benutzereingaben
T	**Gestoppt**	Prozess wurde durch ein Signal gestoppt und führt keine Anweisungen aus
Z	**Zombie**	Prozess hat die Ausführung abgeschlossen, wurde nicht von der Mutter getrennt

*Der Zustand kann in htop in der Spalte **S** abgelesen werden*

*Mann kann durch das: **Senden von Signalen** den Zustand verändern*

Prozess Prioritäten (Nice-Wert)



Prozesse anzeigen - top/htop

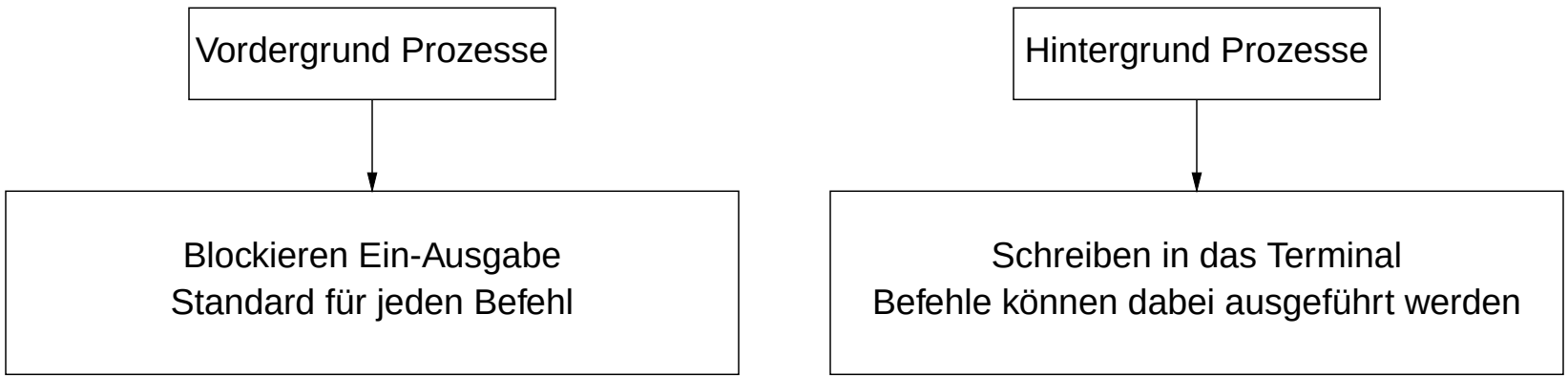
Ausgabe von top:

```
PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM    TIME+   COMMAND
001 root      20   0  167404  11976   9176 S   0.0   0.1   0:00.39 systemd
006 root      20   0    3504    364    132 S   0.0   0.0   0:02.73 init
088 root      20   0    6608   2616   2372 S   0.0   0.0   0:00.02 cron
139 root      20   0    5496   1036    944 S   0.0   0.0   0:00.00 agetty
140 root      20   0    5872   1000    912 S   0.0   0.0   0:00.00 agetty
156 nick      20   0  168144   2908      0 S   0.0   0.0   0:00.00 (sd-pam)
161 nick      20   0    7196   3428   3136 S   0.0   0.0   0:00.00 bash
```

Prozesse anzeigen - ps

```
Ausgabe von ps: Aktuelles TTY bzw. Terminal Was ist der Unterschied?
PID   TTY      TIME     CMD
83888 pts/0    00:00:00 bash
84079 pts/0    00:00:00 ps

Ausgabe von ps -u BENUTZERNAME: Alle Benutzer Prozesse
PID  TTY TIME     CMD
1282 ?   00:03:50 pipewire
1286 ?   00:06:03 firefox

Ausgabe von ps -aux: Alle Prozesse
USER    PID %CPU %MEM VSZ     RSS  TTY      STAT START   TIME COMMAND
root    1   0.0  0.0  167780  9016 ?        Ss   Mär09   0:02 /sbin/init
root    2   0.0  0.0  0       0    ?        S    Mär09   0:00 [kthreadd]
```

Vorder- und Hintergrund Prozesse



Hintergrund Prozesse erstellen

Neue Prozesse:
BEFEHL **&**

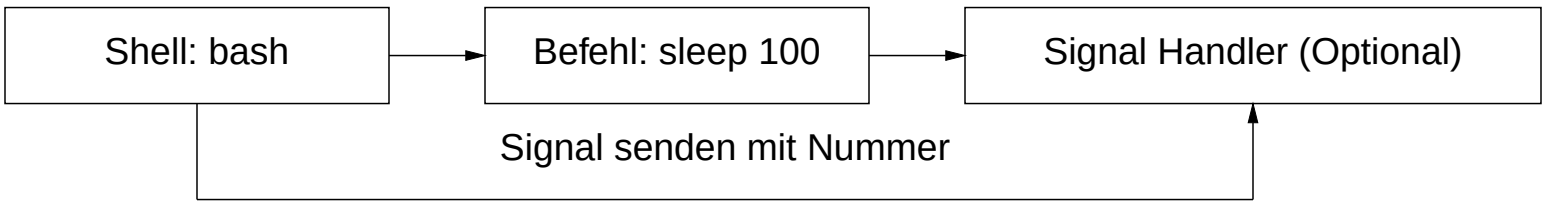
Aktive Prozesse:
1. Schlafen legen mit **STRG + Z**
2. Jobs anzeigen mit jobs
3. **bg JOB_ID** - Hintergrund bzw. **fg JOB_ID** - Hintergrund

Beispielausgabe von jobs:
[1]- Angehalten sleep 100
[2]+ Angehalten sleep 5055

PID ermitteln

```
Manuell nach der Prozess ID (PID) Suchen:
ps -u BENUTZERNAME

Alle Prozesse mit einem bestimmten Namen anzeigen:
pgrep NAME

Ausgabe von pgrep chromium:
314473
375796
440524
440756
```

Prozesse und Signale



Unter Linux besitzt jedes Signal eine eindeutige Nummer. Programme können Signale anhand dieser Nummern abfangen und entsprechend handeln, um beispielsweise beim Beenden noch offene Daten zu speichern und das Programm sauber zu beenden.

Prozesse beenden (killen)

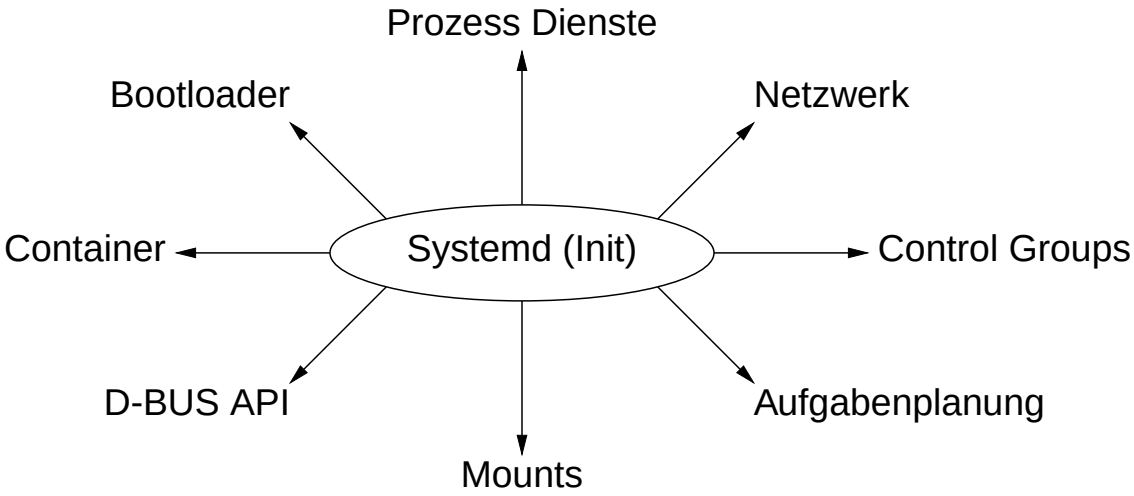
Nummer	Name	Beschreibung
2	**SIGINT**	Unterbrechung (z.B. mit Strg+C), Programm kann sauber beenden
15	**SIGTERM**	Standardmäßiges Beenden, erlaubt sauberes beenden
9	**SIGKILL**	Erzwingt sofortiges Beenden, keine Bereinigung möglich

```
Einen Prozess mit seiner Prozess ID (PID) beenden:
kill -SIGNAL_NUMMER PID

Alle Prozesse mit einem bestimmten Namen beenden:
pkill SIGNAL_NUMMER NAME
```

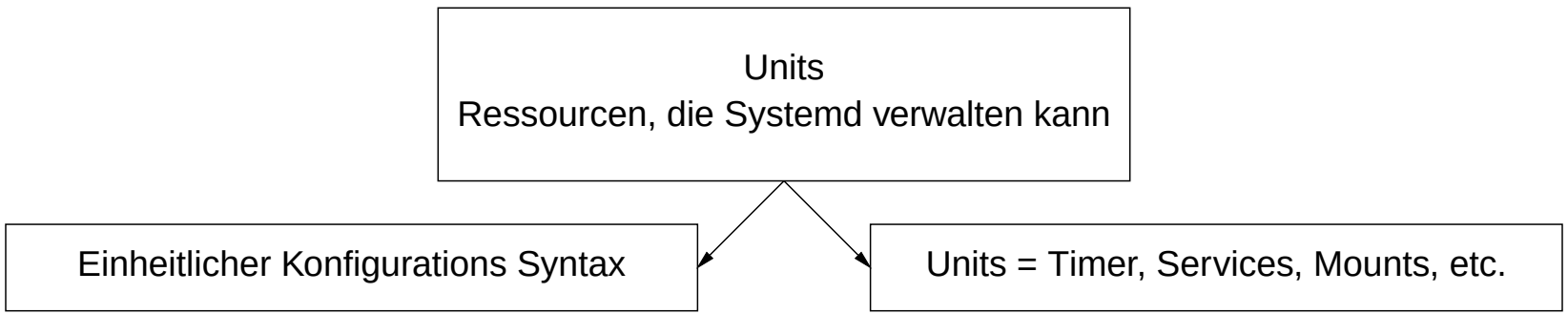
Übungsaufgaben 4 – Prozesse unter Linux

1. Zeige alle laufenden Prozesse mit dem Befehl top an
2. Finde die PID deiner Shell mit dem Befehl pgrep bash heraus
3. Starte den Befehl sleep 60 als Hintergrundprozess
4. Starte den Befehl sleep 30 mit niedriger Priorität mittels nice -n 10 sleep 30
5. Starte sleep 100, pausiere ihn mit STRG+Z und setze ihn dann mit bg im Hintergrund fort
6. Erstelle mit sleep 120 & einen Hintergrundprozess und beende ihn kontrolliert mittels kill -15 PID
7. Starte sleep 300 & und erzwinge mit kill -9 PID ein sofortiges Ende dieses Prozesses

Systemd: Prozess- und Applikationsinfrastruktur API



Wie funktioniert Systemd?



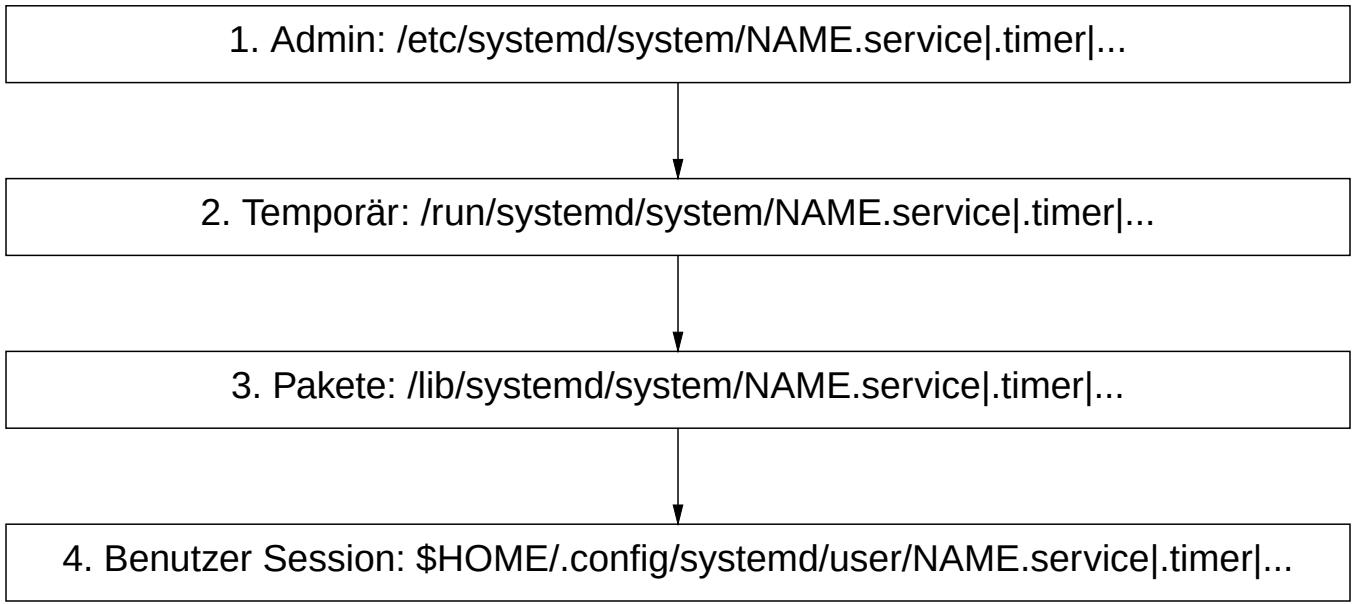
The Tragedy of systemd

Beispiel: Services

```
[Unit]
Description=Mein einfacher Service
After=network.target

[Service]
Type=simple
Restart=on-failure

# Benutzer und Gruppe, unter denen der Prozess läuft
User=nick
Group=nick

# Start- und Stop-Kommandos definieren
ExecStart=/usr/bin/mein_programm --option wert
ExecStop=/usr/bin/mein_programm --stop
```

```
# Systemverzeichnisse sind schreibgeschützt (/usr, /etc, /boot)
ProtectSystem=strict # oder ReadWritePaths=RW-ORDNER

# Eigenes /tmp, /dev und Netzwerk
PrivateTmp=yes
PrivateDevices=yes
PrivateNetwork=yes

# Kein sudo oder doas
NoNewPrivileges=yes

#CPU und RAM begrenzen
MemoryLimit=500M
CPUQuota=50%

[Install]
WantedBy=multi-user.target
```

Wo liegen die Konfigurationsdateien?



Wie interagiere ich mit einem Service?

```
Wenn eine Unit Konfigurationsdatei verändert wurde: Systemd Neustarten
# @root
systemctl daemon-reload

Service aktivieren
# @root
systemctl enable NAME.service

Service starten
# @root
systemctl start NAME.service

Status anzeigen
systemctl status NAME.service
```

Übungsaufgabe 5: Eigener Systemd Service

Erstelle und starte mithilfe der Vorlage in /etc/systemd/system/test.service einen Service für den Befehl sleep infinity

```
[Unit]
Description=Mein einfacher Service

[Service]
Type=simple
ExecStart=/usr/bin/sleep infinity

[Install]
WantedBy=multi-user.target
```

Tipp: Du kannst Dateien mit z.B. nano /etc/systemd/system/test.service bearbeiten

Aufgabenplanung

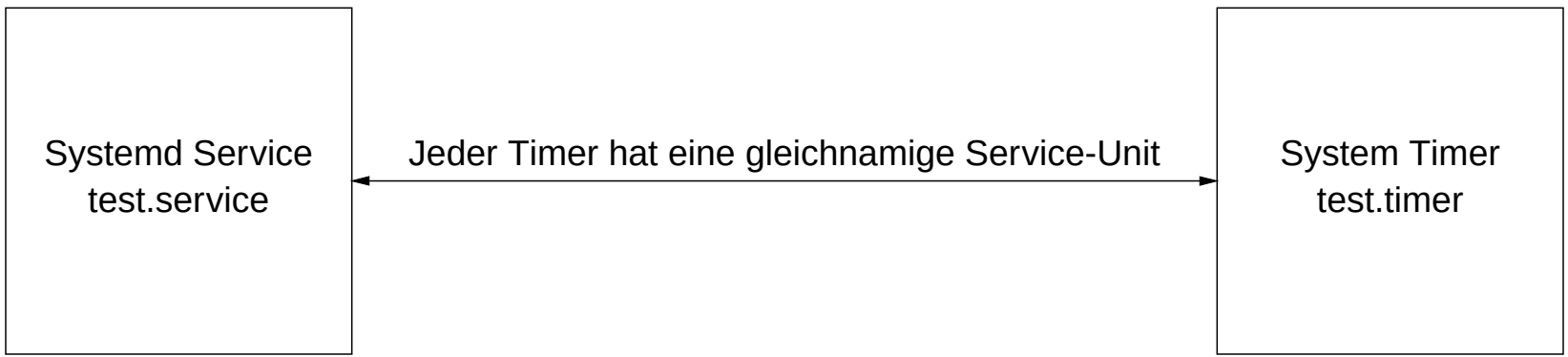
Mit Cron oder Timer Units kannst du Befehle zu einem bestimmten Zeitpunkt automatisiert ausführen

Systemd Timer Units	**Crontab**
Steuerung über *.timer und *.service Dateien	Steuerung über /etc/crontab oder crontab -e
OnCalendar= für Zeitsteuerung	*/5 * * * * Syntax für Zeitsteuerung
Minimale Genauigkeit: 1 Sekunde	Minimale Genauigkeit: 1 Minute
Abhängigkeiten über Units möglich	Keine direkten Abhängigkeiten möglich
Logging erfolgt über journald	Logging meist in separaten Dateien oder per Mail
Aktivierung: systemctl enable/start	Automatisch nach Änderung der Crontab aktiv
Status sichtbar mit systemctl list-timers	Keine einfache Statusübersicht vorhanden

Crontab erstellen

```
crontab -e # Als Benutzer unter dem der Crontab läuft
Select an editor. To change later, run 'select-editor'.
  1. /bin/nano        <---- Am besten nano

[Minute] [Stunde] [Tag(Monat)] [Monat] [Wochentag] BEFEHL # * = Jede

# Alle zwei Minuten für 10 Sekunden schlafen
2 * * * * sleep 10

@yearly     Einmal pro Jahr (0 0 1 1 *)
@monthly    Einmal pro Monat (0 0 1 * *)
@weekly     Einmal pro Woche (0 0 * * 0)
@daily      Einmal pro Tag (0 0 * * *)
@hourly     Einmal pro Stunde (0 * * * *)
```

Systemd Timer Units



Systemd Timer Service erstellen

```
# @root
nano /etc/systemd/system/test.service
[Unit]
Description=test ausgeben

[Service]
Type=oneshot
ExecStart=/usr/bin/echo "test"
```

Systemd Timer Unit erstellen

```
# @root
nano /etc/systemd/system/test.timer
[Unit]
Description=Starte den Systemd Service test.service alle 10 Minuten

[Timer]
# OnCalendar=daily # Jeden Tag um 00:00 Uhr
# OnCalendar=hourly # Jede volle Stunde
# OnCalendar=2025-12-24 18:00:00 # Einmalig am 24. Dezember 2025 um 18:00 Uhr
# OnCalendar=Fri 13:00:00 # Jeden Freitag um 13:00 Uhr
OnCalendar=*:0/10:*
Persistent=true

[Install]
WantedBy=timers.target
```

Wie interagiere ich mit einem Timer?

```
Alle Timer anzeigen
# @root
systemctl list-timers

Wenn eine Unit Konfigurationsdatei verändert wurde: Systemd Neustarten
# @root
systemctl daemon-reload

Timer aktivieren
@root
systemctl enable NAME.timer
systemctl start NAME.timer
```

Übungsaufgabe 6: Aufgabenplanung

1. Erstelle mit dem Befehl 'crontab -e' eine Crontab für den Root-Benutzer, die jede Stunde den Befehl 'echo "Hallo Welt"' ausführt.
2. Erstelle einen systemd-Service in der Datei '/etc/systemd/system/test.service', der den Befehl 'echo "Hallo Welt"' als oneshot ausführt. Erstelle anschließend die gleichnamige Timer-Unit in der Datei '/etc/systemd/system/test.timer' und sorge dafür, dass der Timer jede Stunde ausgeführt wird. Aktiviere danach nur den Timer und lasse dir anschließend alle aktiven Timer auf dem System anzeigen.

```
# Wichtige Befehle @root
crontab -e
systemctl list-timers
systemctl daemon-reload
systemctl enable NAME.timer
systemctl eingabe NAME.timer
```

Praxisprojekt Minecraft-Server

1. Aktualisiere das gesamte System mit den offiziellen Paketquellen, um Sicherheit und Stabilität sicherzustellen.
2. Installiere eine geeignete Java-Version (**headless**), da der Server keine grafische Oberfläche benötigt.
3. Lege einen dedizierten **Minecraft-Service-Benutzer** an, unter dem der Server später betrieben wird.
4. Suche im Dateisystem einen geeigneten Speicherort für die Minecraft-Server-Dateien, und lege dort ein neues Verzeichnis mit den entsprechenden Zugriffsrechten an.
5. Recherchiere im Internet nach passender Minecraft-Server-Software, z. B. den offiziellen Mojang-Server oder PaperMC (für Plugin-Unterstützung). Achte auf die **Kompatibilität mit der installierten Java-Version**
6. Teste und starte den Minecraft-Server zunächst manuell, um sicherzustellen, dass er korrekt funktioniert.

Was ist ein Shell Skript?

Ein Shell-Skript ist eine Aneinanderreihung von Shell-Befehlen in einer Textdatei, die von der Shell interpretiert ein neues Programm ergibt.

```
#!/bin/bash <-- Shebang = Welche Shell
# Datei: test.sh <-- Kommentar
echo "Hallo Welt" <-- Befehl
```

Shell Skripte werden zur Automatisierung und Beschreibung von z.B. Server Konfiguration genutzt

Bevor wir das Skript ausführen (‘./NAME‘), muss es mit ‘chmod +x NAME‘ ausführbar gemacht werden

Variablen

```
#!/bin/bash

# Zeichenkette (String = Standard)
text="Hallo Bash"
echo "Text: ${text}"

# Ganze Zahl (Integer) => (declare -i)
zahl=42
echo "Text: ${zahl}"

# Array (declare -a)
arr=("Apfel" "Birne" "Kirsche")
echo "Array: ${arr[0]}, ${arr[1]}, ${arr[2]}"
```

Ausgabe von Befehlen in Variablen

```
#!/bin/bash

# String zuweisen
datum=$(echo "Inhalt")
echo "Datum: ${datum}"

# Array (Pro Zeile / Tab)
dateien=($(ls))
echo "Datei: ${dateien[0]}"
```

Sichere Interpolation von Variablen und Befehlen

Ohne Anführungszeichen werden Variablen mit Leerzeichen in mehrere Teile
Variablen in geschweiften Klammern schützen zusätzlich vor falscher
Interpretation bei zusammengesetzten Namen wie ${user}name

```
#!/bin/bash

# Bei Befehlen
echo "$(hostname)"

# Bei Variablen (statt $name)
name="Max Mustermann"
echo "${Max Mustermann}"
```

Umgebungsvariablen

Umgebungsvariablen enthalten Informationen über den aktuellen Zustand des Systems und der Benutzerumgebung.

Variable	Bedeutung
${PATH}	Suchpfad für ausführbare Programme
${HOME}	Pfad zum Home-Verzeichnis des aktuellen Benutzers
${USER}	Aktuell angemeldeter Benutzername
${SHELL}	Pfad zur verwendeten Login-Shell (z.B. /bin/bash)
${PWD}	Aktuelles Arbeitsverzeichnis
${OLDPWD}	Vorheriges Arbeitsverzeichnis (z.B. nach ‘cd -‘)
${LANG}	Standardsprache und Zeichensatz (z.B. en_US.UTF-8)
${LC_ALL}	Überschreibt alle LC_* Sprach-/Regionseinstellungen
${DISPLAY}	X11 Display-Nummer (z.B. :0 für grafische Sitzungen)
${XDG_SESSION_TYPE}	Typ der aktuellen Sitzung (z.B. tty, x11, wayland)

Nutzereingaben: stdin

```
#!/bin/bash
echo "Name?:"
read eingabe
echo "Sie sind: ${eingabe}"

# Aufruf
# Über eine Pipe
echo "Hallo Welt" | skript.sh <-- Name der skript Datei
# oder einfach über die Tastatur antworten + ENTER
```

Eine Unix-Pipe (|) leitet die Ausgabe eines Befehls als Eingabe (stdin) an einen anderen Befehl weiter.

Ausgaben: stdout und stderr

```
#!/bin/bash

# Normale Info Ausgabe
echo "Test"

# Fehler ausgeben
# 1 (stdout) = Ausgabe von echo nehmen und nach 2 (sterr) umleiten
echo "Error!!!" 1>&2
```

Die Trennung von stdout und stderr erleichtert Fehleranalyse, Logging und gezielte Umleitungen

Ausgaben an Dateien

```
#!/bin/bash

# Dateiinhalt überschreiben
echo "Test" > datei

# Dateiinhalt behalten und Text hinzufügen
echo "Test2" >> datei
```

Das Umleiten von Ausgaben in Dateien ist nützlich für Logging, da es Prozesse dokumentiert und die Fehlersuche erleichtert.

Das Nichts - /dev/null

/dev/null - Void des Kernels – alles, was hier landet, verschwindet

```
# Ausgabe verstecken
echo "Ich bin nicht zu sehen" > /dev/null

# Fehler nicht ausgeben
find / -name datei 2>/dev/null
```

Übungsaufgabe 7: Benutzer-Log

Schreibe ein Bash-Skript, das:

1. Den Benutzer nach seinem Namen fragt und die Eingabe in einer Variablen speichert
2. Das aktuelle Datum und die Uhrzeit speichert
3. Diese Informationen in eine Datei schreibt, und den Benutzer freundlich begrüßt

```
$ ./benutzerlog.sh
Wie heißt du?
> Alex
Hallo Alex! Deine Anmeldung wurde gespeichert.

$ cat benutzerlog.txt
Alex hat sich am 2025-03-20 um 14:35:12 angemeldet.
```

If Kontrollstruktur und Fehlerauswertung

```
#!/bin/bash

# Sleep ohne Zeit verursacht einen Fehler
sleep

# Den Rückgabewert kann mann über ${?} auslesen
echo ${?} # = 1 (Fehler) 0 wäre OK

# Mit if überprüfen
if $(sleep); then
    echo “Alles Okay”
else
    echo “NEIN!”
fi
```

Im Fehlerfall Skript abbrechen

Standardmäßig führt ein Bash-Skript alle Befehle bis zum Ende aus, auch wenn einer davon mit einem Fehler (Statuscode ≠ 0) endet. Um das zu verhindern, kann am Anfang des Skripts folgende Option gesetzt werden

```
#!/bin/bash

# Skript abbrechen, wenn ein Befehl fehlschlägt (Exit-Code ≠ 0)
set -e

# Fehler, wenn auf eine nicht gesetzte Variable zugegriffen wird
set -u

# Fehler, wenn ein Befehl in einer Pipeline fehlschlägt
set -o pipefail
```

If Kontrollstruktur Vergleiche

```
if [[ ${?} == 1 ]]; then
    echo “Error”
fi
```

Vergleich	Boolisch	Arithmetisch
Gleich	$a == $b	$a -eq $b
Ungleich	$a != $b	$a -ne $b
Größer als	$a > $b	$a -gt $b
Größer oder gleich	$a >= $b	$a -ge $b
Kleiner als	$a < $b	$a -lt $b
Kleiner oder gleich	$a <= $b	$a -le $b
Logisches UND	$a -gt 0 && $b -gt 0	$a -gt 0 -a $b -gt 0
Logisches ODER	$a -gt 0 || $b -gt 0	$a -gt 0 -o $b -gt 0

Switch-Case für Argumente

```
${0} = Programm Name => test.sh
${1} = Ersten Argument usw.
${*} = Alle Argumente
${#} = Anzahl der Argumente

case ${1} in
    hallo)
        echo “Hallo”
        ;;
    *)
        echo “Wenn nichts zutrifft”
        ;;
esac
```

While Schleife

Solange die Bedingung der while-Schleife true ist, werden die Befehle in ihr ausgeführt.

```
while [[ ${1} ]]; do # Eine Variable ist true wenn sie nicht leer ist
    echo ${1}
    shift # shift verschiebt die Positionsparameter ${1}, ${2}, … nach links
          # ${1} => ${2} usw.
done
```

For Schleife

For-Schleifen dienen dazu, über Listen (z.B. von Dateien) zu iterieren oder eine Aktion mehrfach auszuführen.

```
# ls * - Alle Dateien im aktuellen Ordner ausgeben
for DATEI in "$(ls *)"; do
    echo "Datei gefunden: ${DATEI}"
done
```

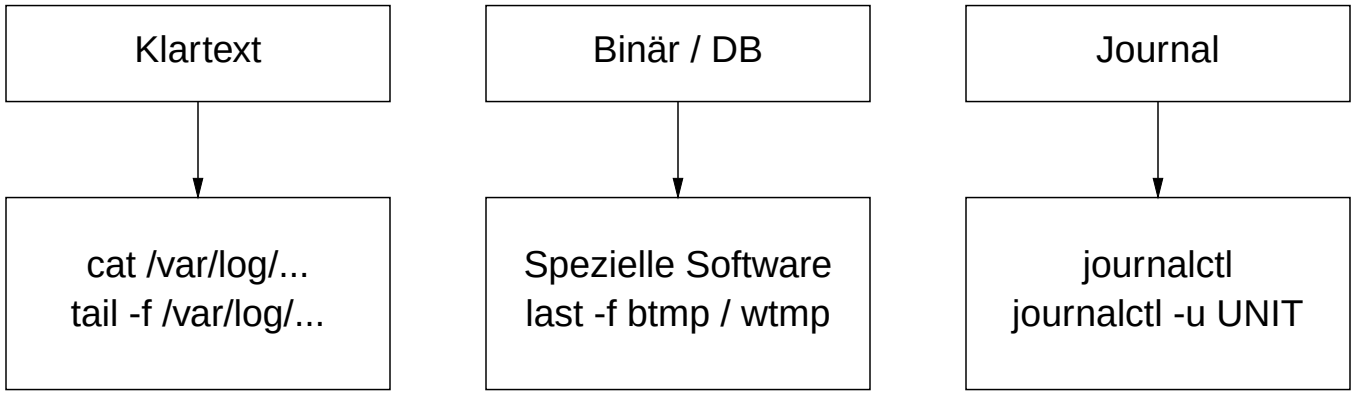
Übungsaufgabe 8: Interaktives Skript mit Argumenten

1. Brich das Skript bei Fehlern, nicht gesetzten Variablen oder Pipe-Fehlern ab
2. Frage den Benutzer nach seinem Namen mit read
3. Speichere das aktuelle Datum, die Uhrzeit und den Hostnamen in Variablen
4. Begrüße den Benutzer mit einer Nachricht wie „Hallo NAME auf HOST!“
5. Schreibe Name, Datum, Uhrzeit und Hostname in eine Logdatei
6. Gib bei Benutzernamen azubi zusätzlich Umgebungsvariablen wie $HOME und $SHELL aus
7. Erstelle ein Array mit drei zufälligen Hinweisen und gib sie mit einer for-Schleife aus
8. Verarbeite alle übergebenen Argumente ($1, $2, …) mit einer while-Schleife und shift
9. Nutze case, um bei Argumenten wie --debug oder --help passende Ausgaben zu machen

Logging

Logging ist das automatische Aufzeichnen von Ereignissen, Abläufen oder Fehlern in einem System, um deren Verhalten nachvollziehen und analysieren zu können.

Log Dateien befinden sich in /log



Die wichtigsten Logdateien

Logdatei (mit Pfad)	Zweck
/var/log/syslog	Allgemeine Systemmeldungen (Dienste, Kernel, Cron, Netzwerk)
/var/log/auth.log	Authentifizierungsversuche, sudo, ssh-Logins
/var/log/wtmp	Binärlog: Anmeldungen, Neustarts, Runlevel-Änderungen
/var/log/btmp	Binärlog: Fehlgeschlagene Loginversuche
/var/log/lastlog	Binärlog: Letzte erfolgreiche Anmeldung jedes Nutzers
/var/log/faillog	Binärlog: Fehlgeschlagene Logins pro Benutzer
/var/log/apt/history.log	Chronik von Paketinstallationen über apt
/var/log/apt/term.log	Terminalausgaben von apt

Viele Logdateien enthalten ähnliche Informationen. Je nach Quelle – etwa Kernel, Hardware oder Systemdienste – unterscheiden sie sich vor allem im Detailgrad und Kontext der protokollierten Ereignisse.

Logs automatisch Löschen

Wenn wir Logs nicht automatisch rotieren (also alte Logs löschen oder archivieren),
wird unser System irgendwann aufgrund von zu wenig Speicher nichtmehr funktionieren

logrotate = Service, der alte Logs rotiert

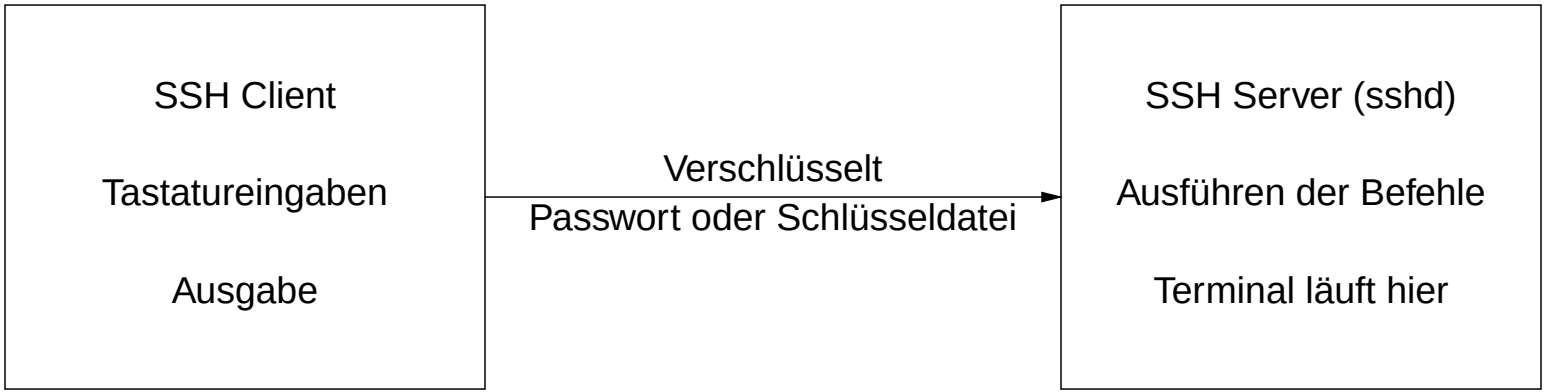
```
# @root - Alle Befehle
apt install logrotate # (systemctl status logrotate) = sollte laufen

nano /etc/logrotate.conf # (systemctl restart logrotate)
# rotate log files weekly
weekly
# keep 4 weeks worth of backlogs
rotate 4
# create new (empty) log files after rotating old ones
create
```

Erste Hilfe bei Fehlern unter Linux

1. Zeige aktuelle Systemfehler mit journalctl -xe oder journalctl -u SERVICE an
2. Durchsuche Logdateien mit grep -r -f error warning /var/log nach Fehlern
3. Starte Programme mit --debug oder -vv, um mehr Details zu sehen.
4. Prüfe Dienste mit systemctl status dienstname.
5. Zeige Dateiberechtigungen mit ls -l /pfad/zur/datei an.
6. Teste die Netzwerkverbindung mit ping -c 4 1.1.1.1.
7. Suche Hardwarefehler mit dmesg | grep -i error.

SSH - Die Idee

SSH steht für Secure Shell und ermöglicht die sichere, verschlüsselte Fernsteuerung und Datenübertragung zwischen Computern über ein Netzwerk



SSH Server installieren

Das Paket openssh-server bringt nur den SSH-Server mit. Möchte man auch auf andere Computer per SSH zugreifen, muss zusätzlich das Paket openssh-client installiert werden.

```
# @root

apt install openssh-server

systemctl enable sshd
systemctl start sshd

systemctl status sshd
```

SSH Server konfigurieren

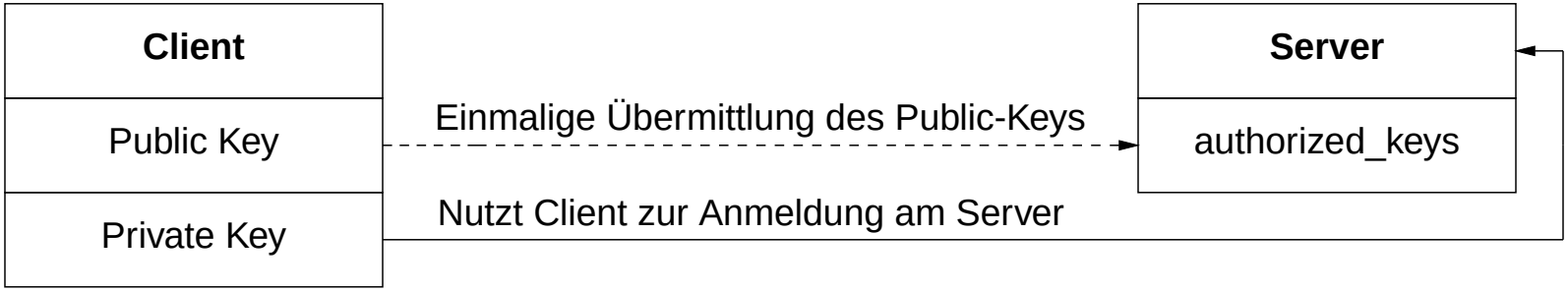
```
# @root
nano /etc/ssh/sshd_config # (systemctl restart sshd)
Port 22
ListenAddress 0.0.0.0

# Authentication:
PermitRootLogin prohibit-password
StrictModes yes
MaxAuthTries 3
MaxSessions 2
PubkeyAuthentication yes

PermitTunnel no
X11Forwarding no
```

SSH Verbindung aufbauen und Dateien kopieren

```
# Mit dem SSH Server Terminal verbinden
ssh ZIEL_BENUTZER@ZIEL_HOST

# Dateien vom Client zum Server kopieren
scp DATEI ZIEL_BENUTZER@ZIEL_HOST:/PFAD

# Ordner vom Client zum Server kopieren
scp -r DATEI ZIEL_BENUTZER@ZIEL_HOST:/PFAD

# Dateien vom Server zum Client kopieren
scp ZIEL_BENUTZER@ZIEL_HOST:/PFAD SPEICHERORT

# Ordner vom Server zum Client kopieren
scp -r ZIEL_BENUTZER@ZIEL_HOST:/PFAD SPEICHERORT
```

SSH-Anmeldung mit Schlüsseldateien – Konzept

*SSH-Schlüssel verwenden ein asymmetrisches Schlüsselpaar, bestehend aus: **Private Key** (geheim, bleibt auf dem Client) und **Public Key** (wird auf den Ziel-Server kopiert).*
So ist eineautomatische sichere Anmeldung ohne Passwort möglich.



*Wichtig: Der **Private Key** darf niemals weitergegeben werden.*
Der Public Key hingegen kann bedenkenlos auf beliebig vielen Servern hinterlegt werden.

SSH-Schlüssel erzeugen & auf Server kopieren

```
# 1) Auf dem Client: Schlüsselpaar erzeugen (ed25519 empfohlen)
ssh-keygen -t ed25519
# Folgen: Eingabepfad (z.B. /home/nick/.ssh/id_ed25519) und optional Passwort

# 2) Public Key auf den Server kopieren (einfachste Methode)
ssh-copy-id -i ~/.ssh/id_ed25519.pub BENUTZER@SERVER

# 4) Verbindung per Schlüssel testen (mit -i Pfad zum Private Key)
ssh BENUTZER@SERVER
```

Hinweis: Das setzten eines Passworts für den Private Key erhöht die Sicherheit, macht aber die automatische Anmeldung ohne Benutzereingabe unmöglich.

Übungsaufgabe 9: SSH

1. Installiere über apt das Paket openssh-server und stelle sicher, dass der SSH-Dienst über systemd gestartet wird und läuft
2. Konfiguriere den SSH-Server so, dass Root-Verbindungen erlaubt sind und maximal 3 fehlgeschlagene Anmeldeversuche sowie höchstens 2 gleichzeitige Verbindungen zugelassen werden.
3. Schließe dich mit einem Partner zusammen und versucht, euch gegenseitig per SSH auf dem jeweils anderen Computer anzumelden und Befehle auszuführen.
4. Erstellt jeweils ein SSH-Schlüsselpaar mit ssh-keygen und kopiert den Public Key eures Partners mit ssh-copy-id auf euren Computer.
5. Testet die Anmeldung per SSH-Schlüssel ohne Passwortabfrage.
6. Kopiert anschließend Dateien und Ordner per scp von einem Computer auf den anderen

Praxisprojekt Minecraft-Server

1. Erstelle eine eigene **systemd-Service-Unit** für den Minecraft-Server. Der Service soll unter dem Minecraft-Benutzer laufen und nur die nötigen Zugriffsrechte auf den Server-Ordner haben.
2. Achte darauf, dass der Dienst im Fehlerfall automatisch neu startet und mit dem System hochfährt.
3. Überprüfe die Rechtevergabe: Nur der Minecraft-Benutzer darf im Installationsordner schreiben, andere Benutzer haben keinen Zugriff.
4. Teste, ob der Dienst korrekt startet und im Hintergrund stabil läuft.
5. Erstelle ein Backup-Skript, das den gesamten Minecraft-Ordner sichert, und richte einen **systemd-Timer** ein, um regelmäßige Backups automatisiert durchzuführen.

Praxisprojekt Minecraft-Server

1. Richte den **SSH-Zugang** für den Server ein. Deaktiviere die Root-Anmeldung und erlaubt ausschließlich die Anmeldung eines Administrationsbenutzers per **SSH-Schlüssel**
2. Überprüfe die Konfiguration des Minecraft-Servers und passe bei Bedarf Parameter in der **server.properties**-Datei an (z. B. Servername, Whitelist, maximale Spielerzahl).
3. Starte den Server und teste, ob ein Beitritt über einen Minecraft-Client möglich ist.
4. Gehe offene Fragen im Team durch und reflektiere gemeinsam, welche Schritte noch optimiert werden können.

Testrelevante Themen

1. Linux, GNU und Unix – Herkunft, Unterschiede, Open Source
2. Desktop-Umgebungen, Display-Server, Distributionen
3. Filesystem Hierarchy Standard (FHS)
4. Shell und grundlegende Befehle: ls, cd, cp, mkdir, touch, echo, mv, rm, ln, cut, grep, nano, vim
5. Berechtigungen unter Linux: sudo, Benutzer & Gruppen anlegen/löschen
6. Dateiberechtigungen mit chmod und chown setzen
7. Prozesse: Was ist ein Prozess, Zustände, Nice-Wert, Vorder-/Hintergrundprozesse, PIDs, Signale
8. Systemd: Grundlagen, Aufbau eines Services, Timer vs. Cron
9. Umgebungsvariablen, stdout/stderr, Umleiten und Pipen
10. SSH-Konfiguration inkl. Schlüsseldateien

Feedback & Auswertung

1. Was hat euch in dieser Woche besonders gut gefallen?
2. Gab es Themen, die ihr noch ausführlicher behandelt haben wollt?
3. Welche Inhalte waren für euch am wichtigsten oder spannendsten?
4. Was hat euch weniger gefallen oder war schwer verständlich?
5. Was nehmt ihr aus dem Kurs für eure weitere Arbeit mit Linux mit?