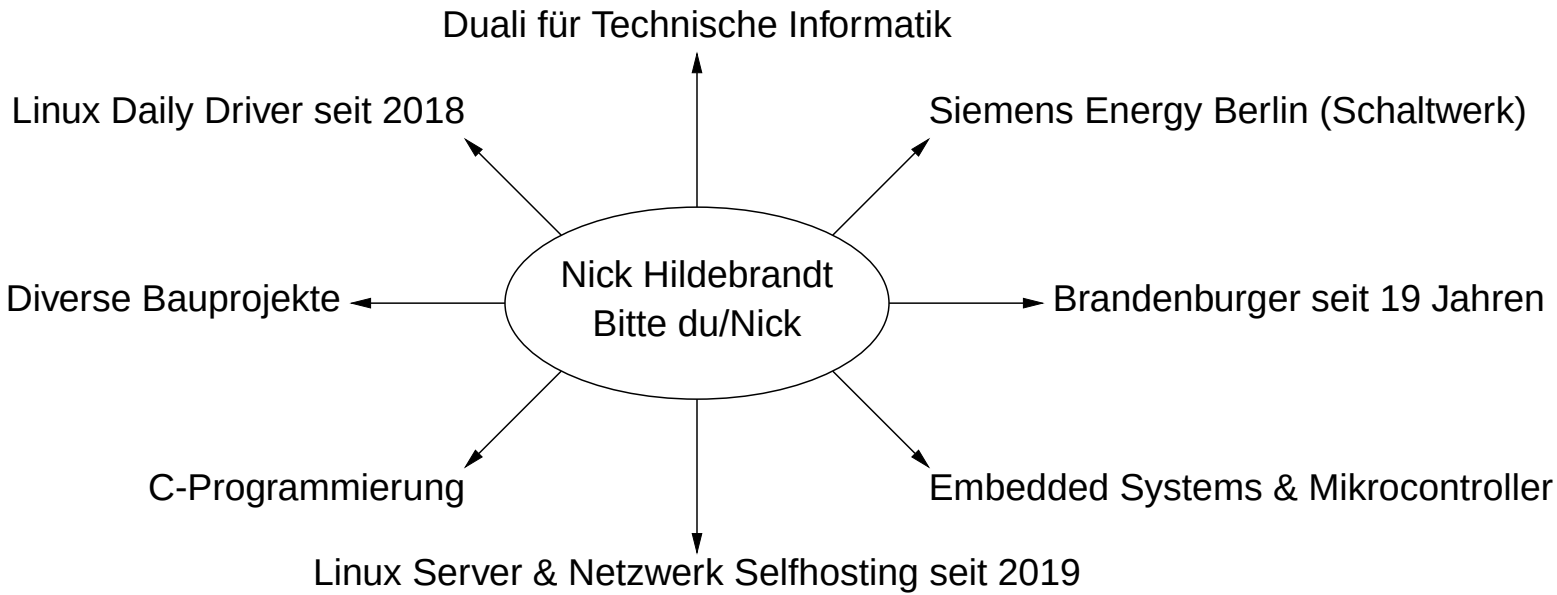
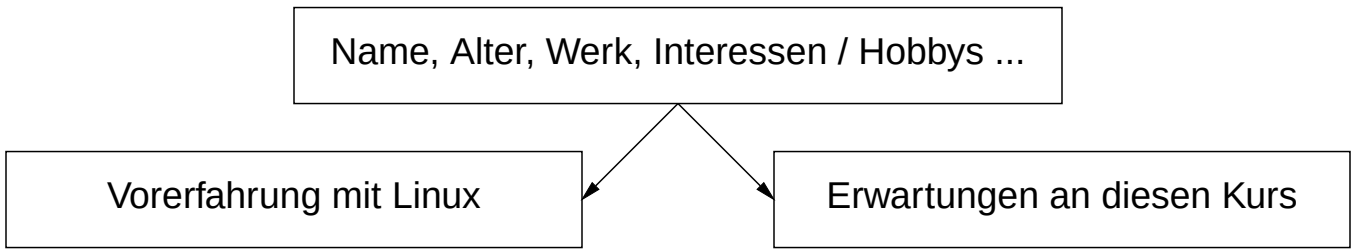
Linux Advanced

Boot, Prozesse, Shell und Netzwerk

Ein bisschen was über mich



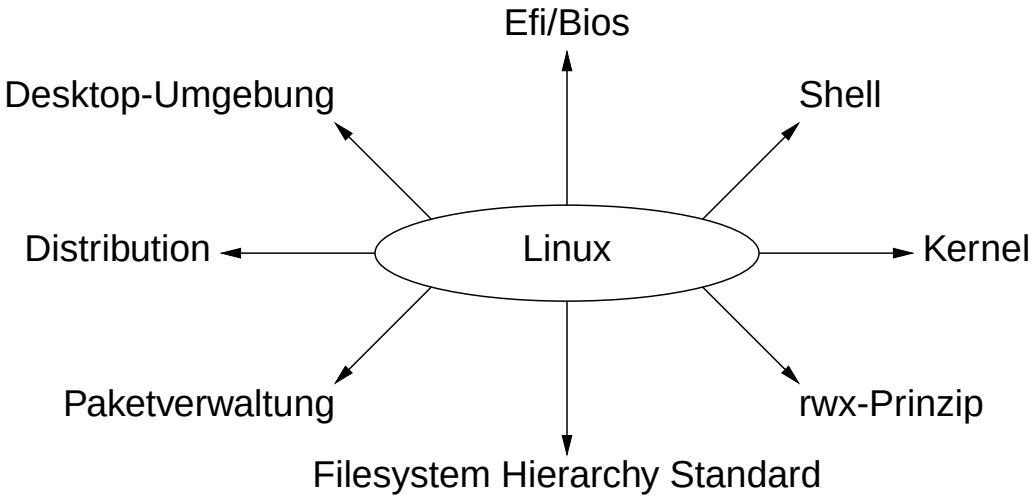
Etwas über euch



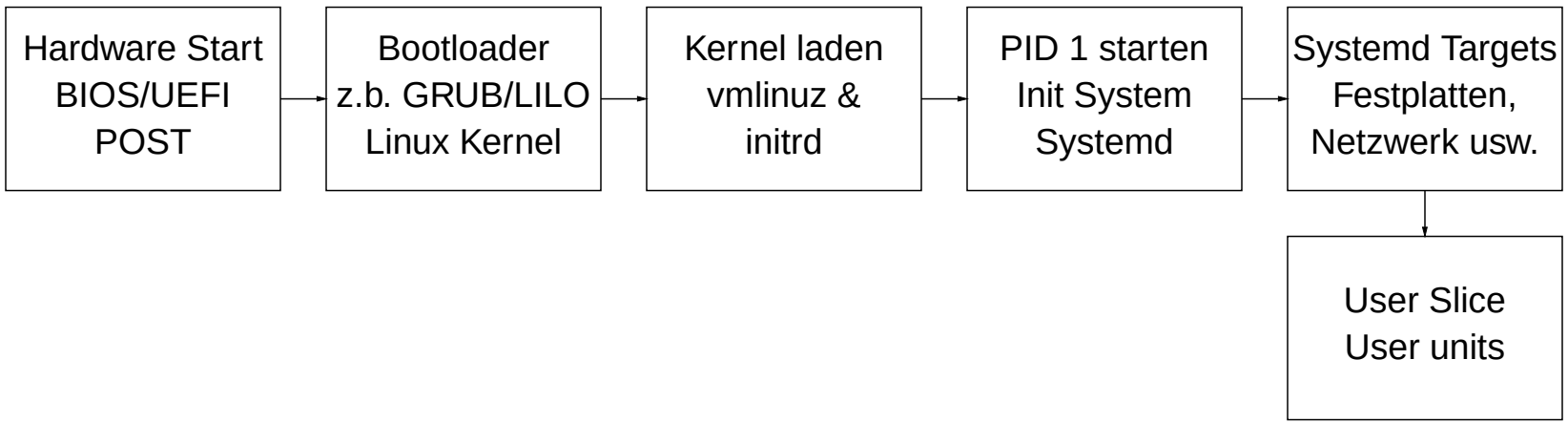
Was noch wichtig ist

- Bitte immer **SOFORT FRAGEN** wenn etwas unverständlich ist
- Nur die **Übungsaufgaben sind klausurrelevant**
- Alle Kursdaten sind online: **https://github.com/nickhildebrandt/Linux-Advanced**
- Meine E-Mail auch für die Zukunft: **nick.hildebrandt@siemens-energy.com**
- Bitte an das **Erstellen der Befehlsreferenz** für den Test denken

Was ist bekannt?



Der Linux Bootvorgang



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

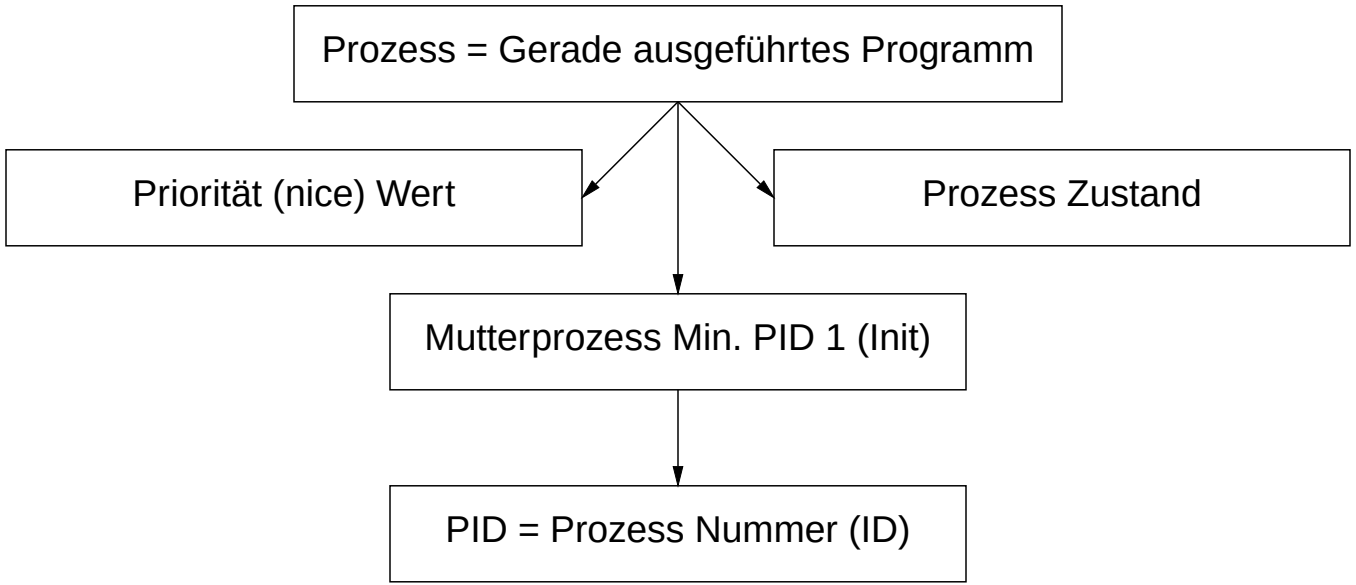
Runlevel und Systemd Targets

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?



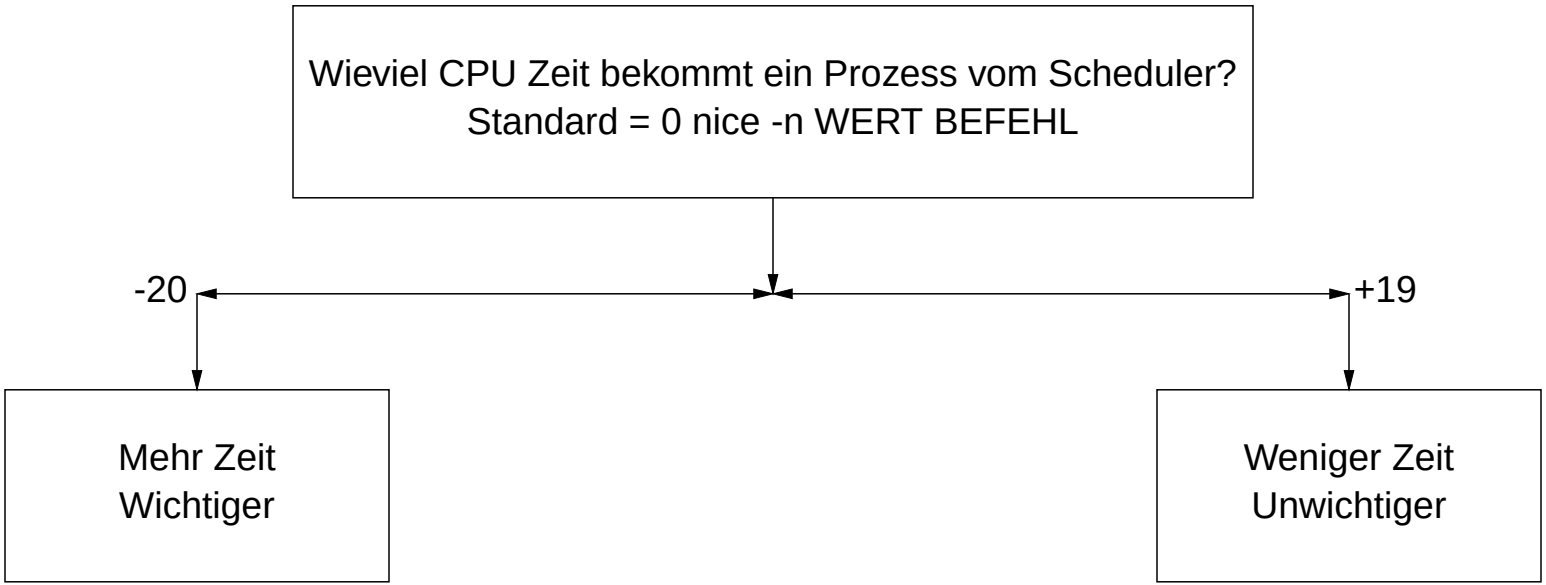
Prozess Zustände

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
```

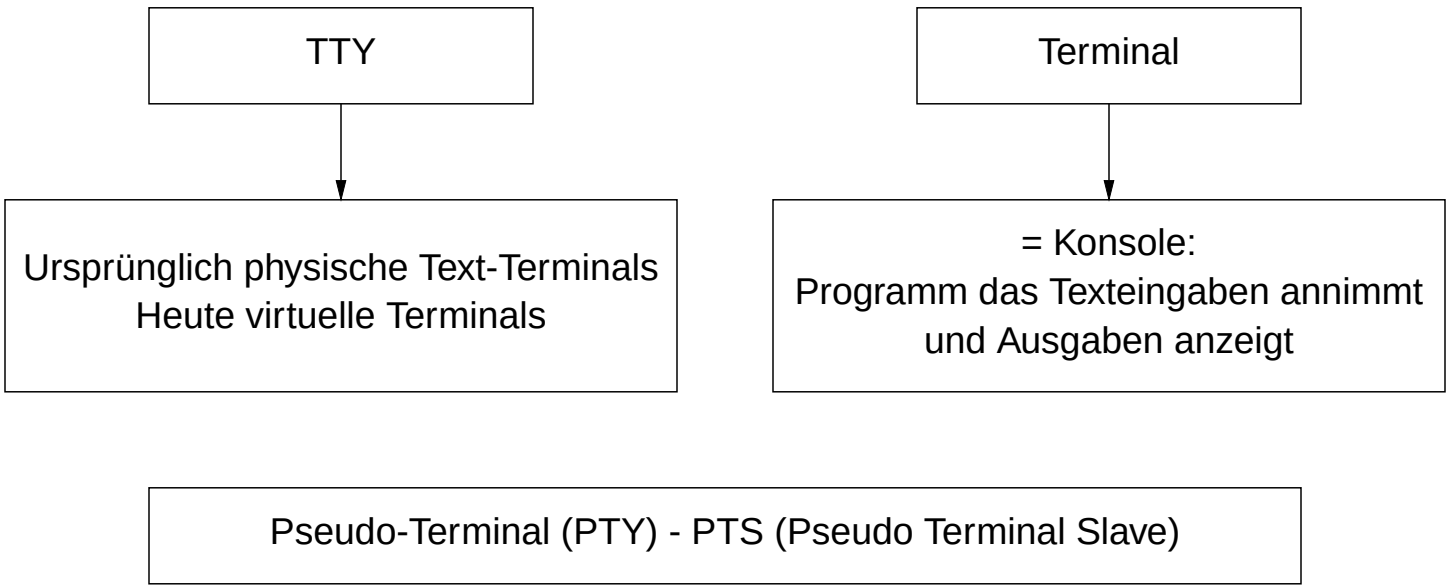
Ausgabe von ps -u BENUTZERNAME: Alle Benutzer Prozesse

```
PID  TTY TIME     CMD
1282 ?   00:03:50 pipewire
1286 ?   00:06:03 firefox
```

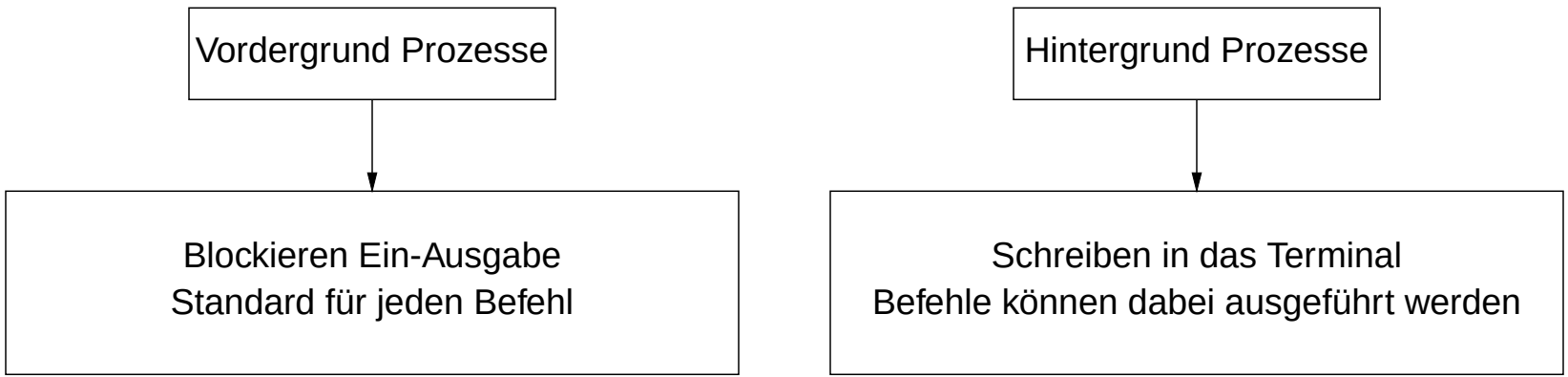
Ausgabe von ps -aux: Alle Prozesse

```
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]
```

Unterschied Terminal - TTY



Vorder- und Hintergrund Prozesse



Hintergrund Prozesse erstellen

Neue Prozesse:
BEFEHL **&**

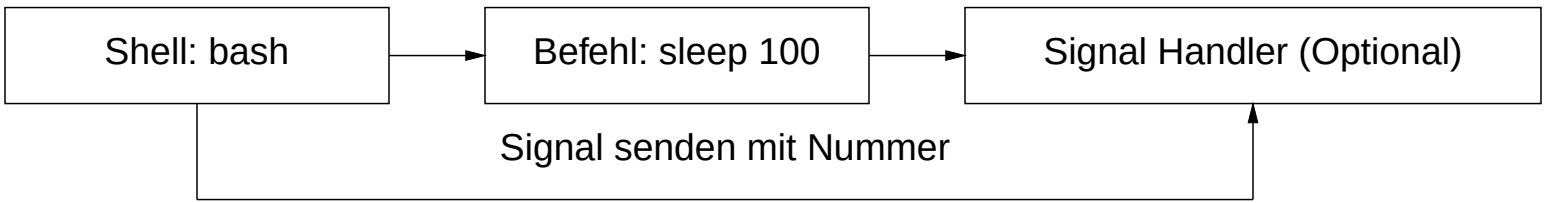
Aktive Prozesse:
1. Schlafen legen mit **STRG + Z**
2. Jobs anzeigen mit jobs
3. **bg JOB_ID** - Hintergrund bzw. **fg JOB_ID** - Hintergrund

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)

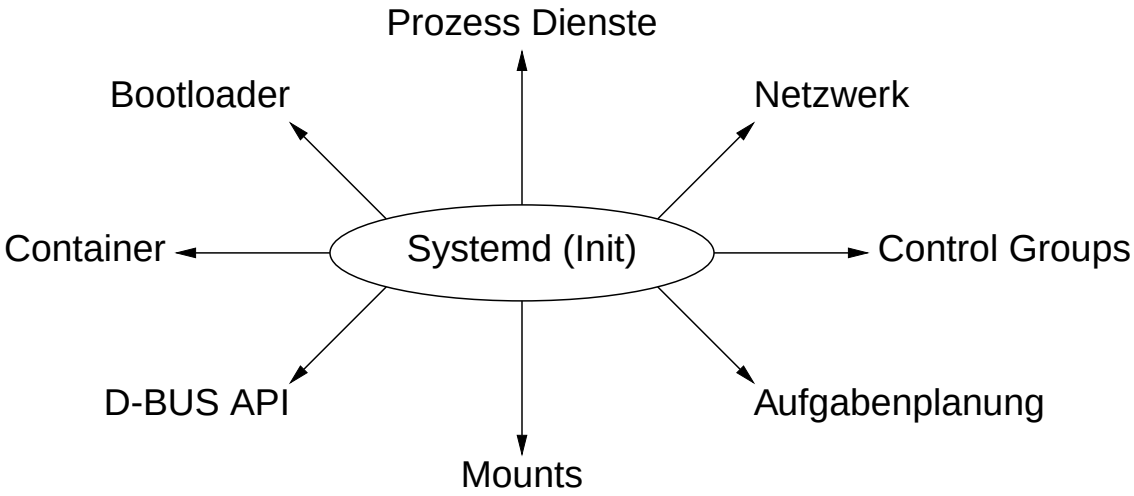
Nummer	Name	Beschreibung
2	**SIGINT**	Unterbrechung (z.B. mit Strg+C), Programm kann sauber beenden
15	**SIGTERM**	Standardmäßiges Beenden, erlaubt sauberes beenden
9	**SIGKILL**	Erzwingt sofortiges Beenden, keine Bereinigung möglich

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

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

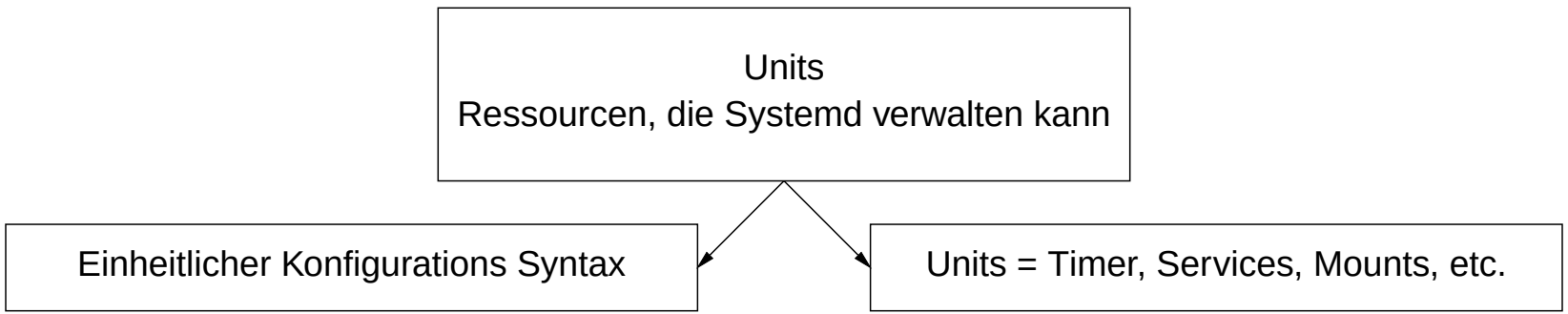
Übungsaufgaben – Prozesse unter Linux

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?

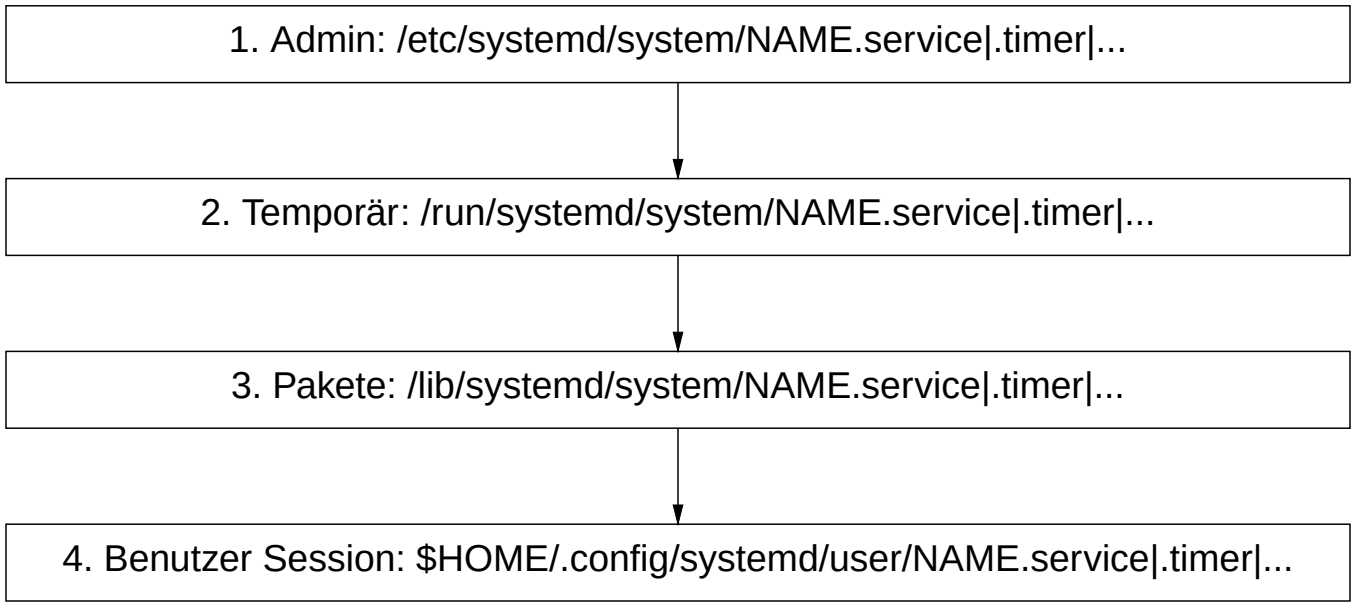


Beispiel: Services

```
[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: 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

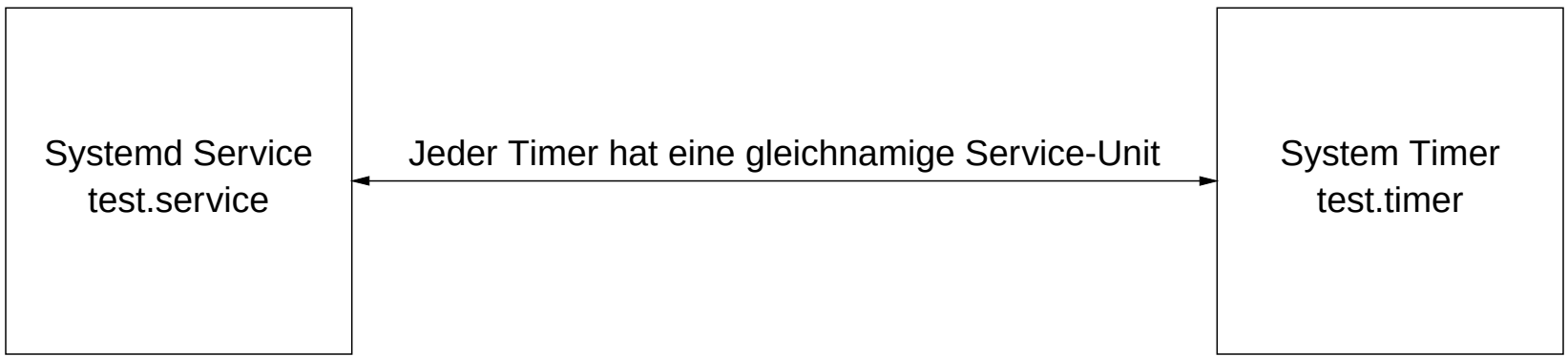
Mit Cron oder Timer Units kannst du Befehle zu einem bestimmten Zeitpunkt automatisiert ausführen

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?

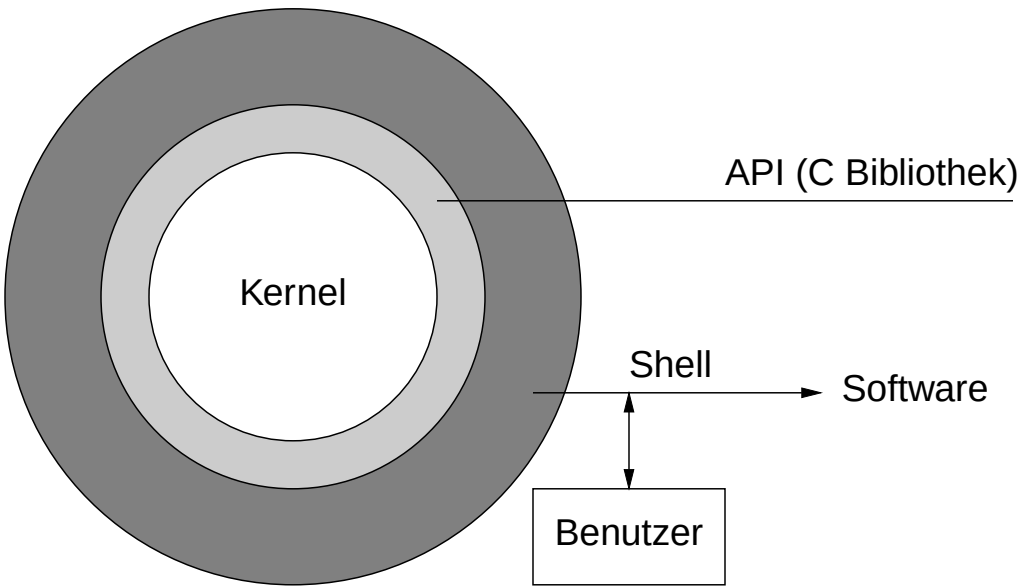
```
Alle Timer anzeigen
# @root
systemctl list-timers

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

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

Übungsaufgabe: 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
```

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.

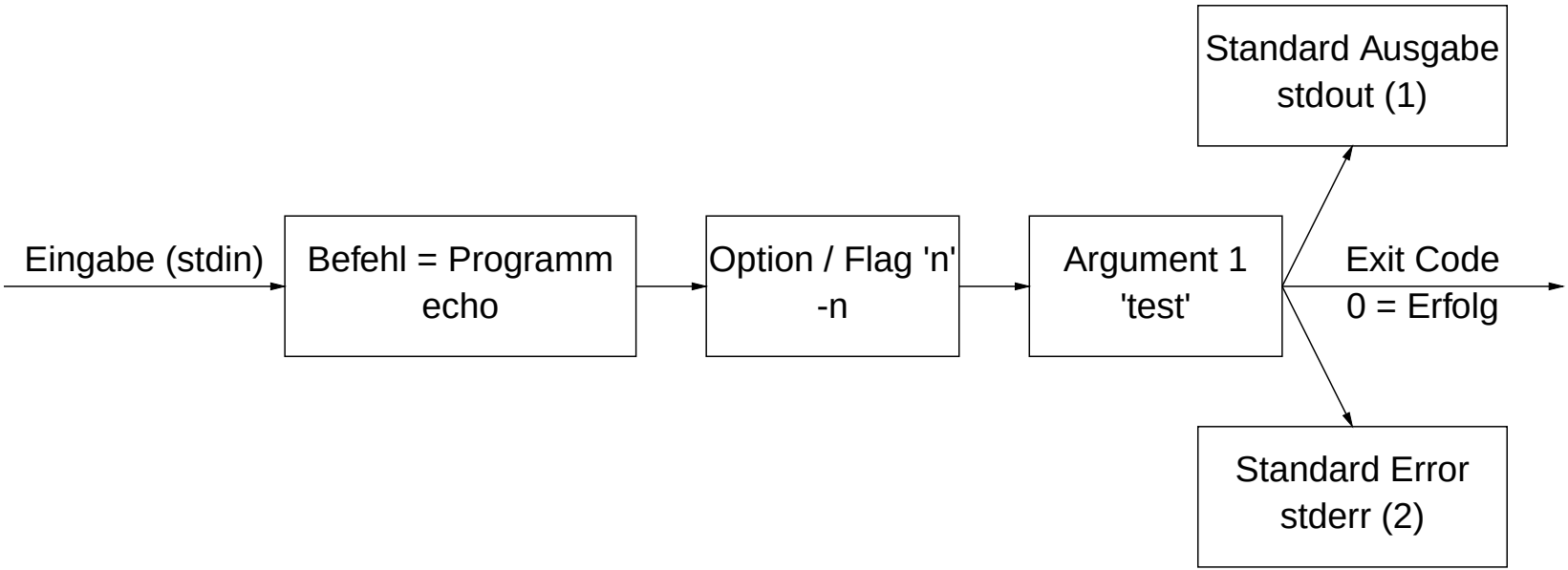
Bash und sh sind der Standard auf Servern, Embedded- und Netzwerkgeräten.

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.

Ein Befehl im Detail



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: 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.

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: 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

Übungsaufgabe: Eigenes Bash Skript

Wähle eine der folgenden Aufgaben oder eine eigene Idee:

1. Überlege dir ein Skript, das regelmäßig (z.B. alle 10 Sekunden oder via Cron) deinen Rechner überwacht: z.B. wie viele Prozesse du hast, was gerade besonders viel CPU verbraucht, oder wie viele Tabs Firefox offen hat. Lass dir die Infos cool anzeigen – vielleicht mit Farben oder Emojis. Vielleicht schreibst du sie auch in ein Logfile.
2. Schreib ein interaktives Bash-Textadventure mit Verzweigungen (read, case, if). Am Anfang wird ein „Prozessname“ genannt (z.B. Schattenprozess X), den man sich merken muss. Im Laufe der Geschichte triffst du Entscheidungen (links, rechts, kämpfen, fliehen ...), und am Ende musst du den richtigen Namen eingeben, um den Endboss zu besiegen.
3. Du bist Linux-Admin und dein Rechner wird von Chaosprozessen überrannt. Bau dir ein Bash-Skript + systemd-Service, das beim Booten oder einmal täglich automatisch alte Prozesse killt, unnötige Logdateien löscht, oder dich warnt, wenn etwas „unordentlich“ ist.

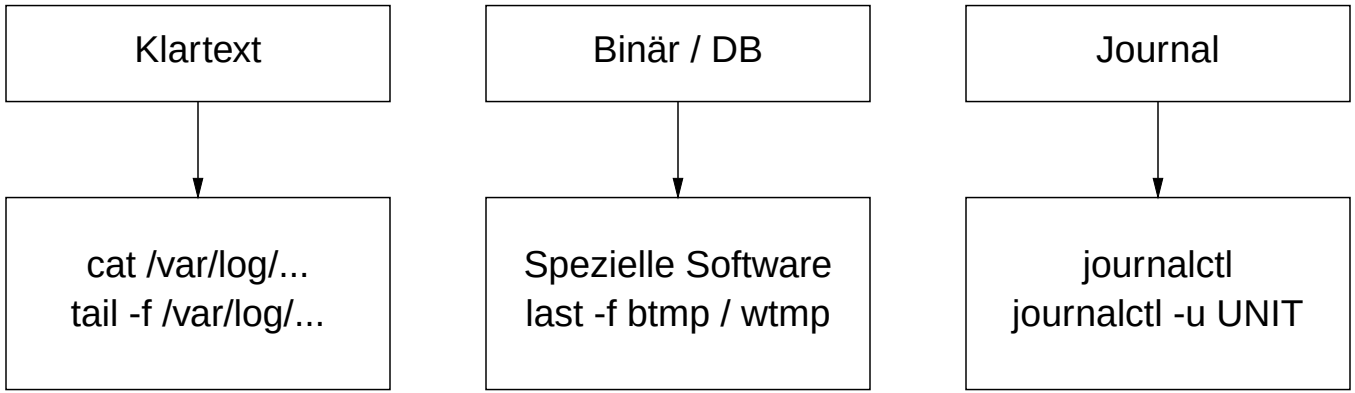
Bearbeitungshinweis

- **Googeln und KI** sind ausdrücklich erwünscht
- Ihr könnt mir jederzeit alle **Fragen stellen** – ich helfe gerne
- Bringt **eigene Ideen** ein und probiert viel aus
- Das Skript sollte später um **Logging und Netzwerkbefehle** erweitert werden können
- Die **nächsten Nachmittage** stehen dafür zur Verfügung

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.