Linux Systemadministration Grundkurs

Detaillierter Seminar-Guide (18 UE)

Contents

1. Tag 1 – Die Bash und Lokale Benutzerverwaltung (09:00–16:00)

1.1. 09:15–10:00 Was ist eine Shell, was macht Bash?

Inhalt

Zu Beginn stelle ich den Teilnehmenden das *Unix-Prozessmodell* vor: Jeder gestartete Befehl erzeugt einen Kindprozess, dessen `PID` in der Variablen `$$` gespeichert ist, während `$PPID` auf den Elternprozess zeigt. Die laufende Shell ist also selbst „nur“ ein Prozess, der meist von einem Terminal-Emulator (`gnome-terminal`, `konsole` ...) oder von einem `login`-Dienst abstammt.

Darauf aufbauend erkläre ich, warum man zwischen **Login-Shells** und **Non-Login-Shells** unterscheidet. Eine Login-Shell erkennt man daran, dass `$0` mit einem führenden Minus (`-bash`) beginnt; sie liest zuerst `/etc/profile` und anschließend die erste gefundene Datei aus der Reihenfolge `~/.bash_profile`, `~/.bash_login` oder `~/.profile`. Startet man dagegen innerhalb einer laufenden Sitzung ein weiteres `bash`, erhält man eine interaktive Non-Login-Shell, die *nur* `~/.bashrc` einliest.

Ein zweiter Aspekt ist die Interaktivität: Setzt die Variable `$-` das Flag `i`, verhält sich die Shell interaktiv (Prompt, History, Readline-Editing); fehlt das Flag, handelt es sich um eine nicht-interaktive Instanz – typisch in Skripten oder bei Befehlen wie `bash -c '...'`.

Zum Abschluss dieser Theorieeinheit zeige ich live, wie unterschiedlich sich der Start einer Login-Shell, einer interaktiven Non-Login-Shell und einer nicht-interaktiven Shell anfühlt. Die Teilnehmenden beobachten dabei unmittelbar, welche Konfigurationsdateien gelesen werden und wann ein Prompt erscheint.

Übung – Schritt für Schritt zu eigenen Aliases

Ziel der Übung ist, dass jede*r Teilnehmende eine persönliche Datei `~/.bash_aliases` anlegt und den häufig genutzten Alias `ll` so erweitert, dass `ls` *alle* Dateien anzeigt und das Ergebnis farbig formatiert.

1. Prüfen, ob bereits eine Alias-Datei existiert. Im Terminal geben die Teilnehmenden zunächst

```
ls -l ~/.bash_aliases
```

ein. Falls der Befehl mit „No such file or directory“ endet, ist noch keine Datei vorhanden – genau das wollen wir ändern.

2. Datei anlegen und vorbereiten. Mit `touch` legen wir die Datei neu an und fügen eine Kommentarzeile hinzu, die ihren Zweck beschreibt:

```
touch ~/.bash_aliases
echo '# Eigene Shell-Aliases' >> ~/.
    bash_aliases
```

3. Alias `ll` definieren. Der erweiterte Alias soll folgende Optionen übernehmen:

- `-l` – lange Listenform (Rechte, Größe, Zeitstempel)
- `-h` – menschenlesbare Größen (K, M, G ...)
- `-a` – *alle* Dateien inklusive der „versteckten“ beginnen mit `.`
- `--color=auto` – Farbausgabe, aber nur, wenn die Ausgabe ein Terminal ist

Außerdem ist es praktisch, Verzeichnisse zuerst aufzulisten (`--group-directories-first`), weil man damit schneller navigiert. Alles zusammen fügen wir dauerhaft in die Alias-Datei ein:

```
echo "alias ll='ls -lh --group-directories-
    first -a --color=auto'" \
  >> ~/.bash_aliases
```

4. Alias sofort aktivieren. Eine neue Login- oder Non-Login-Shell würde die Änderung automatisch laden, aber wir wollen nicht neu starten. Darum lesen wir `~/.bashrc` (die diese Alias-Datei gewöhnlich am Ende einbindet) unmittelbar neu ein – `source` ist dafür das komfortable Kürzel:

```
source ~/.bashrc
```

5. Funktionstest. Mit

```
type -a ll   # zeigt, dass 'll' jetzt ein
    Alias ist
ll
```

prüfen die Teilnehmenden, ob die Definition greift und ob `ls` tatsächlich alle Dateien farbig listet. Variieren Sie die Befehle ruhig (`ll /etc`, `ll -R` ...), um den Nutzen der Optionen zu demonstrieren.

6. Bonus: weitere nützliche Aliases. Wer noch Zeit hat, kann beispielsweise

```
echo "alias grep='grep --color=auto'" >> ~/.bash_aliases
echo "alias ..='cd ..'"               >> ~/.bash_aliases
echo "alias ...='cd ../..'"           >> ~/.bash_aliases
source ~/.bashrc
```

hinzufügen und damit die Bedienung der Shell weiter beschleunigen.

Didaktischer Hinweis

Ermutigen Sie die Teilnehmenden, jede Änderung sofort zu testen. So erleben sie den direkten Zusammenhang zwischen Konfigurationsdatei und Shell-Verhalten. Zeigen Sie zum Schluss, wie man mittels `bash --noprofile --norc` eine *saubere* Shell ohne jede Konfiguration startet – ein wertvolles Werkzeug zur Fehlersuche, wenn ein Alias oder eine Funktion unerwartet stört.

1.2. 10:00–10:10 Kurzpause

1.3. 10:10–11:00 Shell-Mechanismen I

Inhalt

In dieser Einheit untersuchen wir vier Grundpfeiler jeder Shell-Arbeit: *Wildcards*, *Quoting*, *Umleitungen* und *Pipelines*. Alle Demo-Befehle lassen sich unverändert in jeder Bash (≥ 4.x) nachvollziehen.

1. Wildcards („Globbing“) Die Shell expandiert Platzhalter, *bevor* sie einen Befehl startet.

`*` ersetzt beliebig viele Zeichen (auch 0). `cp *.jpg ~/Bilder/` kopiert alle JPEGs.

`?` ersetzt genau ein Zeichen. `ls IMG_2025-05-0?.png` listet die ersten neun Tages-Snapshots.

`[ ]` Zeichenklassen; `[0-9]`, `[abc]` oder Negationen `[!A-Z]`. Beispiel: `rm photo_[12][0-9].raw` löscht Dateien `photo_10.raw ... photo_29.raw`.

Range-Grenzen prüfen. Durch Anführungszeichen *verhindern* wir die Expansion:

```
echo "*.log"        # gibt den Text *.log aus
```

2. Quoting – wann interpretiert die Shell Sonderzeichen?

- *Unquoted* (nackt) → Globbing, Parameter- und Kommandoersetzung aktiv.
- `"double quotes"` → Globbing *deaktiviert*, Variablen `$VAR` bleiben aktiv. Beispiel: `echo "Pfad: $PWD"`.
- `'single quotes'` → nichts wird ausgewertet; ideal für Regex oder JSON. Beispiel: `grep '^[0-9]\{3\}' daten.txt`.
- Backslash `\` → entwertet *nur das direkt folgende Zeichen*. Beispiel: `echo "Preis: 50\texteuro"`.

3. Umleitungen (Redirections)

`cmd > datei` stdout überschreibt Datei
`cmd >> datei` stdout wird angehängt
`cmd 2> datei` stderr in Datei, stdout bleibt Terminal
`cmd &> datei` stdout >und stderr zusammen
`cmd > datei 2>&1` klassische Form: erst stdout, dann stderr umlenken

Beispiel A. Fehler von `gcc` protokollieren, Ausgabe gleichzeitig sehen:

```
gcc main.c -o main 2> build.err | tee build.out
```

4. Pipes – Prozesse verbinden Mit `|` leiten wir den >ungepufferten> *stdout eines Kommandos in den stdin des nächsten.*

```
journalctl -b | less              # Systemd-Protokoll seitenweise
ps aux | grep '[h]ttpd'           % Prozessname filtern (grep selbst ausschliessen)
du -ah | sort -h | tail -n 20     # 20 groesste Dateien im Verzeichnisbaum
```

Übung – Hands-on

1. Alle `*.log` größer als 1 MB finden und komprimieren.

1. *Zunächst testen wir den Filter, >ohne> etwas zu verändern:*

```
find . -type f -name '*.log' -size +1M -print
```

2. Stimmt die Treffermenge, hängen wir eine Aktion an:

```
find . -type f -name '*.log' -size +1M -exec gzip {} \;
```

Erklärung: `find` ruft für jede gefundene Datei das Programm `gzip` auf. Die geschweiften Klammern stehen als Platzhalter für den Dateinamen, `\;` beendet den `-exec`-Block.

3. Variante für viele Dateien: Null-Terminator statt Leerzeichen, ein einziger `gzip`-Aufruf:

```
find . -type f -name '*.log' -size +1M -print0 | xargs -0 gzip
```

2. Pipeline-Analyse: `cat /etc/passwd | grep /bin/bash | wc -l`

1. `cat /etc/passwd` → *liefert die komplette Benutzerdatenbank Zeile für Zeile nach* `stdout`.
2. `grep /bin/bash` → *filtert nur Einträge, deren Login-Shell* `/bin/bash` *ist.*
3. `wc -l` → *zählt die verbleibenden Zeilen, also alle interaktiven Bash-Accounts.*

Diskussionsimpuls. *Fragen Sie die Gruppe, warum* `cat` *hier >(fast)<> überflüssig ist. Eine äquivalente, ressourcenschonendere Variante lautet:*

```
grep -c '/bin/bash' /etc/passwd
```

Dennoch ist die lange Form ein hervorragender Einstieg, um das Prinzip der Pipeline zu verstehen.

Tipps für das Seminar

- Demonstrieren Sie das Zusammenspiel von Wildcards und Quoting live, indem Sie temporäre Dateien anlegen (`touch file{1..3}.txt`) und nacheinander `ls file*`, `ls "file*"` und `ls file\*` ausführen.
- Nutzen Sie das Werkzeug `set -x` in einer Übungs-Shell, um zu zeigen, wie Bash >vor< *der Ausführung Platzhalter und Variablen expandiert.* `set +x` *schaltet den Trace-Modus wieder ab.*
- *Bei großen Klassen lohnt es sich, die Ergebnisse einer Übung via* `tmate` *oder Bildschirmfreigabe zentral anzuzeigen, damit alle denselben Output sehen.*

1.4. 11:00–11:15 Kaffee-Pause

1.5. 11:15–12:00 Shell-Mechanismen II

In dieser Einheit vertiefen wir den praktischen Umgang mit Variablen, Umgebungsvariablen und Kommando-Substitution in der Bourne-again shell (`bash`). Darüber hinaus lernen die Teilnehmenden typische Fallstricke wie einen falsch gesetzten `$PATH` oder Leerzeichen in Variablenwerten zu erkennen und zu beheben. Alle Beispiele lassen sich 1:1 im eigenen Terminal nachvollziehen.

1. Variablen und Umgebungsvariablen

Lokale Variablen Eine Variable wird in `bash` ohne Leerzeichen vor und nach dem Gleichheitszeichen angelegt:

```
NAME=alice
echo "$NAME"          # gibt "alice" aus
```

Exportieren in die Umgebung Nur exportierte Variablen stehen Kindprozessen (z. B. einem gestarteten Skript) zur Verfügung:

```
export NAME           # nachtraegliches Exportieren
env | grep ^NAME      # Kontrolle
```

Typische Stolperfalle 1 – Leerzeichen Wird ein Wert mit Leerzeichen ohne Anführungszeichen zugewiesen, schneidet `bash` ab dem ersten Leerzeichen ab:

```
GREETING=Hello World      # FALSCH, "World" wird als Befehl interpretiert!
GREETING="Hello World"    # RICHTIG
```

Typische Stolperfalle 2 – `$PATH` Ein vergessener Eintrag im Suchpfad führt dazu, dass eigene Skripte nicht gefunden werden. Prüfung:

```
echo "$PATH" | tr ':' '\n'
```

Fehlt ein Verzeichnis? Dann ergänzen:

```
export PATH="$PATH:$HOME/bin"
```

2. Kommando-Substitution

Mit `$( ... )` (empfohlen) oder dem älteren Backtick-Syntax `` `...` `` kann die Ausgabe eines Befehls in eine Variable übernommen oder direkt in einen anderen Befehl eingebettet werden.

```
NOW=$(date '+%F %T')
echo "Es ist jetzt $NOW"
```

3. Geführte Übung: `diskfree.sh`

Schritt 1 – Skript schreiben Wechseln Sie in Ihr persönliches Arbeitsverzeichnis (z. B. `~/scripts`) und erstellen Sie das Skript:

```
nano diskfree.sh
#----- Inhalt von diskfree.sh -----
#!/usr/bin/env bash
#
# Zeigt Datum und freie Plattenkapazitaet der Root-Partition an

NOW=$(date '+%F %T')
echo "Stand: $NOW"
df -h /
#----- Ende Datei -----
```

Schritt 2 – Skript ausführbar machen Das Skript ist zunächst nicht ausführbar. Das ändern wir mit dem `chmod`-Befehl:

```
chmod +x diskfree.sh
```

Schritt 3 – Skript in den `$PATH` legen Kopieren oder verschieben Sie das Skript in ein Verzeichnis, das bereits im `$PATH` liegt (z. B. `~/bin`). Falls `~/bin` noch nicht existiert, legen Sie es an und ergänzen Sie den Suchpfad wie oben gezeigt.

```
mkdir -p ~/bin
mv diskfree.sh ~/bin/
# temporaer fuer diese Shell
export PATH="$PATH:$HOME/bin"
```

Tipp: Damit der Zusatzpfad dauerhaft gilt, fügen Sie den `export`-Befehl am Ende Ihrer `~/.bashrc` ein.

Schritt 4 – Testen Testen wir das Skript, indem wir es einfach aufrufen. Die Ausgabe sollte in etwa so aussehen:

```
diskfree.sh
# Erwartete Ausgabe:
# Stand: 2025-05-03 11:30:12
# Dateisystem    Groesse Benutzt Verf. Verw% Eingeh\"angt auf
# /dev/sda2        50G     14G    34G   30% /
```

4. Zusammenfassung und Best Practices

- Nutze immer Anführungszeichen, wenn Variablen Leer- oder Sonderzeichen enthalten können.
- Setze eigene Skripte in ein separates, versionskontrolliertes Verzeichnis (z. B. `~/bin`) und füge dieses einmalig dem `$PATH` hinzu.
- Bevorzuge die moderne Kommando-Substitution `$( ... )`.
- Dokumentiere Skripte mit einem shebang (`#!/usr/bin/env bash`) und kurzen Kommentaren.

Nach dieser Session sind die Teilnehmer*innen in der Lage, kleine Hilfsskripte zu schreiben, sauber zu platzieren und typische Fehlerquellen beim Arbeiten mit Variablen zu vermeiden.

1.6. 12:00–13:00 Mittagspause

1.7. 13:00–13:45 Benutzer- und Gruppenkonzept

In dieser Lerneinheit untersuchen wir die Grundlage der Benutzer- und Rechtestruktur in GNU/Linux: `/etc/passwd`, `/etc/shadow`, numerische Benutzer- (UID) und Gruppen-IDs (GID), sekundäre Gruppen sowie die hinterlegten Passwort-Hashes. Alle Befehle lassen sich sofort im Kurs-System nachvollziehen.

1. Die Dateien `/etc/passwd` und `/etc/shadow`

`/etc/passwd` Speichert pro Zeile einen Benutzer-Account. Spalten (durch `:` getrennt):

1. **login-Name**
2. Passwort-Platzhalter (meist `x`, echtes Hash liegt in `/etc/sha`
3. **UID**
4. **GID**
5. Kommentar (GECOS)
6. Heimatverzeichnis
7. Loginshell

```
grep '^alice:' /etc/passwd
# alice:x:1001:1001:Alice Example,,,:/home/alice:/bin/bash
```

`/etc/shadow` Nur für `root` lesbar; enthält u. a. Passwort-Hashes und Alterungsinformationen.

```
sudo grep '^alice:' /etc/shadow
# alice:$6$VLx....:19500:0:99999:7:::
#         +----+---
#          SHA-512-Hash  ($6$)            <- Alterungsfelder
```

2. UID und GID

- **0** = `root` mit allen Rechten.
- 1-999 (Distribution-abhängig) = System- und Dienstkonten.
- Ab 1000 (Debian/Ubuntu) = reguläre Benutzerkonten.

Die primäre GID eines Users verweist meist auf eine gleichnamige Gruppe. Zusätzliche (sekundäre) Gruppen erlauben feingranulare Rechtevergabe ohne umständliche ACLs.

3. Passwort-Hashes verstehen

`$1$` MD5 (veraltet)
`$5$` SHA-256
`$6$` SHA-512 (Standard)

Hash-Wechsel (z. B. von MD5 → SHA-512) erfolgt über

```
sudo chage -d 0 alice   # erzwingt Passwortwechsel beim naechsten Login
```

4. Geführte Übung

Schritt 1 – Eigene Gruppen prüfen Überprüfen Sie, in welchen Gruppen Sie aktuell sind. Die Ausgabe zeigt die UID/GID und alle Gruppen an, in denen Sie Mitglied sind:

```
id          # zeigt UID/GID und Gruppen an
groups      # nur Gruppennamen
```

Erklären Sie, was primäre und sekundäre Gruppen in der Ausgabe bedeuten.

Schritt 2 – Gruppe `workshop` anlegen Die Gruppe `workshop` wird für die Teilnehmenden angelegt. Die ID ist egal, sie wird automatisch vergeben. Überprüfen Sie anschließend, ob die Gruppe erfolgreich angelegt wurde:

```
sudo groupadd workshop
getent group workshop   # Kontrolle
```

Schritt 3 – Teilnehmende hinzufügen Angenommen, die Anmeldungen heißen `alice`, `bob` und `carol`:

```
for user in alice bob carol; do
  sudo usermod -aG workshop "$user"
done
```

Hinweis: Das `-a` (append) ist entscheidend – ohne diese Option würden alle bisherigen Sekundärgruppen überschrieben.

Schritt 4 – Änderungen verifizieren Überprüfen Sie, ob die Gruppe `workshop` jetzt auch in der Ausgabe von `id` erscheint:

```
for user in alice bob carol; do
  id "$user" | grep workshop
done
```

Falls die zusätzliche Gruppe noch nicht erscheint, ist eine neue Anmeldung (oder `newgrp workshop`) erforderlich.

5. Best Practices

- Nie Hash-Algorithmen unter SHA-512 verwenden (`/etc/login.defs` kontrollieren).
- Systemkonten auf Shell `/usr/sbin/nologin` beschränken, falls kein Login nötig ist.
- Gruppen als logische Rechtecontainer einsetzen, nicht einzelne Benutzer in ACLs eintragen.
- Achten Sie auf konsistente UID/GID-Bereiche in gemischten Netzwerken (NFS, LDAP).

Nach dieser Session können die Teilnehmenden Benutzer- und Gruppendateien interpretieren, Passwort-Hashes erkennen, eigene Gruppen anlegen sowie Benutzer sicher zuordnen.

1.8. 13:45–14:00 Kurzpause

1.9. 14:00–14:45 Werkzeuge der Benutzerverwaltung

Dieser Abschnitt zeigt praxisnah, wie Sie mit den klassischen Werkzeugen `useradd`, `usermod`, `passwd`, `chage` und `gpasswd` Benutzer- und Gruppenkonten komfortabel anlegen, ändern und automatisiert verwalten. Die Befehle stammen aus dem `shadow-utils`-Paket und funktionieren auf allen verbreiteten Distributionen (Debian/Ubuntu, RHEL/Fedora, SUSE u. a.). Alle Beispiele lassen sich sofort auf dem Kurs-System als `root` oder via `sudo` nachvollziehen.

1. Konten anlegen – `useradd`

Grundsyntax Welche Optionen Sie verwenden, hängt von der Distribution ab. Hier die Standardsyntax für Debian/Ubuntu:

```
sudo useradd -m -s /bin/bash -c "Vollname"
    loginname
```

`-m` Erstellt das Heimatverzeichnis und kopiert das Skeleton aus `/etc/skel`.

`-s` Setzt die Login-Shell (Standard ist oft `/bin/sh` oder `/bin/bash`).

`-c` Kommentar (GECOS-Feld) – häufig Vor- und Nachname.

`-U` (optional) legt eine gleichnamige primäre Gruppe an.

`-G` (optional) Kommagetrennte Liste sekundärer Gruppen.

`-e` (optional) Ablaufdatum des Kontos (`YYYY-MM-DD`).

Voreinstellungen prüfen Die Standardwerte für `useradd` sind in `/etc/default/useradd` hinterlegt.

```
useradd -D          # liest /etc/default/
    useradd
```

Hier sehen Sie z. B. das Standard-Heimatverzeichnis, die Gruppe `users` usw. Passen Sie die Datei bei Bedarf an (z. B. anderes Skeleton-Verzeichnis).

2. Konten ändern – `usermod`

Typische Anwendungsfälle:

```
# Sekundaere Gruppe(n) ergaenzen
sudo usermod -aG docker,video alice

# Login-Shell wechseln
sudo usermod -s /usr/bin/zsh alice

sudo usermod -c "Alice Example (HR)" alice
```

Merke: **`-a`** (append) unbedingt mit **`-G`** kombinieren, sonst werden alte Gruppen überschrieben.

3. Passwörter verwalten – `passwd` und `chage`

- `passwd loginname` – interaktiver Passwortwechsel.
- `passwd -e loginname` – markiert das Passwort als abgelaufen → Benutzer muss bei der nächsten Anmeldung ein neues setzen.
- `chage` – Feingranulare Steuerung der Passwortalterung:

```
# Maximal 90 Tage gueltig, 7 Tage Vorwarnung
sudo chage -M 90 -W 7 alice

# Aktuelle Parameter anzeigen
sudo chage -l alice
```

`-M <n>` Maximale Gültigkeit in Tagen (0 = nie ablaufen)
`-m <n>` Minimale Tage zwischen Wechseln
`-W <n>` Tage Vorwarnfrist vor Ablauf

4. Gruppen gezielt pflegen – gpasswd

```
# alice zum Admin (Gruppenbesitzer) der Gruppe "project" machen
sudo gpasswd -A alice project

# bob und carol zu gewoehnlichen Mitgliedern hinzufuegen
sudo gpasswd -a bob project
sudo gpasswd -a carol project

# Mitgliederliste pruefen
getent group project
```

5. Skript-basierte Provisionierung

Gerade bei Schulungsrechnern oder in Dev/QA-Umgebungen ist es effizienter, Konten per Shell-Skript statt manuell anzulegen. Ein typischer Ablauf im Skript:

1. Prüfen, ob das Konto bereits existiert (`id` oder `getent passwd`).
2. Benutzer mit `useradd` erstellen.
3. Initiales Passwort setzen (`chpasswd` oder `passwd -stdin`).
4. Passwort sofort ablaufen lassen (`passwd -e` oder `chage -d 0`).
5. Optional: zusätzliche Gruppen, Ablaufdatum des Kontos usw.

6. Geführte Übung

Aufgabe Schreiben Sie `create_students.sh`, das die Benutzer `student1`, `student2`, `student3` mit einem temporären Passwort anlegt. Beim ersten `su` sollen sie gezwungen werden, ihr Passwort zu ändern.

Schritt 1 – Skript erstellen Wechseln Sie in Ihr persönliches Arbeitsverzeichnis (z. B. `~/scripts`) und erstellen Sie das Skript:

```
nano create_students.sh
#----- Datei: create_students.sh -----
#!/usr/bin/env bash
#
# Erstellt drei studentische Konten mit Ablaufpasswort.
# Aufruf: sudo ./create_students.sh

set -euo pipefail

students=(student1 student2 student3)
initial_pw='Start123!'          % Einfaches, aber sicheres Start-PW

for u in "${students[@]}"; do
  if id "$u" &>/dev/null; then
    echo "Konto $u existiert bereits - ueberspringe."
    continue
  fi

  # 1. Benutzer + primaere Gruppe erzeugen (-U) und Home anlegen (-m)
  useradd -m -U -s /bin/bash -c "Workshop-Teilnehmer $u" "$u"

  # 2. Startpasswort setzen (per stdin fuer skriptgesteuertes Arbeiten)
  echo "$u:$initial_pw" | chpasswd

  # 3. Passwort sofort ablaufen lassen, damit Benutzer es aendern muss
  passwd -e "$u"            # aequivalent: chage -d 0 "$u"

  echo "Konto $u angelegt."
done
#----- Ende Datei -----
```

Schritt 2 – Skript ausführbar machen und laufen lassen Das Skript ist zunächst nicht ausführbar. Das ändern wir mit dem `chmod`-Befehl:

```
chmod +x create_students.sh
sudo ./create_students.sh
```

Schritt 3 – Funktionstest Testen Sie die Konten, indem Sie sich als `student1` anmelden. Das Passwort ist `Start123!`. Nach dem ersten Login werden Sie aufgefordert, das Passwort zu ändern:

```
# als root:
su - student2
# Prompt:
# You are required to change your password immediately (administrator enforced)

# Nach erfolgreichem Wechsel
exit
```

7. Best Practices

- Halten Sie Provisionierungsskripte in Versionskontrolle (Git) – jede Kontenänderung ist nachvollziehbar.
- Verwenden Sie sichere, zufällig generierte Initialpasswörter (`openssl rand -base64 12`) und erzwingen Sie sofortigen Wechsel.
- Bevorzugen Sie aussagekräftige Kommentare (`-c`) und klar definierte Shells (`-s`) bei `useradd`.
- Dokumentieren Sie Parameteränderungen an `/etc/login.defs` oder `/etc/default/useradd`.
- Kombinieren Sie Gruppen (`gpasswd`) statt wildwachsende Zugriffslisten (ACLs) zu pflegen.

Nach dieser Session können die Teilnehmenden Benutzer automatisiert anlegen, Optionen sicher anpassen und Passwort- wie Gruppenverwaltung mit wenigen Shell-Zeilen robust abbilden.

1.10. 14:45–15:00 Kurzpause

1.11. 15:00–16:00 PAM-Architektur & Security-Demo

Pluggable Authentication Modules (PAM) bilden die flexible Schicht zwischen Programmen (z. B. `sshd`) und unterschiedlichsten Authentifizierungs- techniken. In dieser Einheit analysieren wir den Aufbau einer einzelnen PAM-Stack-Zeile, betrachten die häufig eingesetzten Module `pam_unix.so` und `pam_limits.so` und führen eine Login-Versuchsbegrenzung ein. Die anschließende Gruppen-Challenge setzt das Gelernte praktisch um.

1. Anatomie einer PAM-Zeile

Typ Control-Flag Modulname Optionen

Beispiel (`/etc/pam.d/sshd`):

```
auth required pam_unix.so
    try_first_pass nullok
account required pam_tally2.so deny=5 onerr=
    fail unlock_time=900
```

2. Wichtige Module

- **pam_unix.so** – klassisches Authentifizieren gegen `/etc/passwd` und `/etc/shadow`; unterstützt SHA-512-Hashes.
- **pam_limits.so** – liest Ressourcengrenzen aus `/etc/security/limits.conf` und setzt `ulimit`-Werte (Dateien, Prozesse u. a.).
- **pam_tally2.so** – zählt Fehlversuche pro Benutzer und kann Accounts nach n Fehlern sperren (*legacy*, aber noch weit verbreitet; Nachfolger auf einigen Distros: `pam_faillock.so`).

3. Demo – Fehlversuche begrenzen

Schritt für Schritt:

1. **Backup** anlegen `sudo cp /etc/pam.d/sshd /etc/pam.d/sshd.orig`
2. **Zeile einfügen** (direkt unter dem ersten `auth`-Eintrag):

```
auth required pam_tally2.so onerr=fail
    deny=5 unlock_time=900
```

3. **sshd** neu laden `sudo systemctl reload sshd`
4. **Testen**: fünfmal falsches Passwort eingeben → Konto gesperrt Status prüfen: `sudo pam_tally2 -user bob`

4. Demo – Ressourcengrenzen mit `pam_limits.so`

Syntax `/etc/security/limits.conf`

Domäne	*Typ*	*Item*	*Wert*	
@workshop	hard	nproc	100	(Prozesslimit)
@workshop	hard	nofile	4096	(Dateideskriptoren)
@workshop	soft	core	0	(kein Core-Dump)

Jede Änderung greift erst bei einer *neuen* Sitzung (SSH-Re-Login, `newgrp workshop` oder Reboot).

5. Gruppen-Challenge

Ziel A – SSH-Fehlversuche auf 5 begrenzen

1. Öffnen Sie `/etc/pam.d/sshd` mit `sudo nano`.
2. Fügen Sie die folgende Zeile hinzu (oder passen sie an, falls bereits vorhanden):

```
auth required pam_tally2.so deny=5 onerr
    =fail unlock_time=600
```

3. Speichern + `sshd` neu laden. `sudo systemctl reload sshd`
4. Testen Sie in der Kleingruppe durch bewusstes Falscheingeben der Passwörter und beobachten Sie `pam_tally2 -user <login>`.

Ziel B – Limits für Gruppe `workshop` setzen

1. Öffnen Sie `/etc/security/limits.conf`.
2. Fügen Sie (falls nicht vorhanden) ganz unten an:

```
@workshop hard nproc  120
@workshop hard nofile 2048
@workshop soft memlock 256000
```

3. Neue SSH-Sitzung starten → `ulimit -a` sollte die Werte zeigen.

6. Best Practices & Fehlersuche

- Verwenden Sie `pam_test` (Debian-Paket `libpam-test`) oder das Debug-Flag `debug` in Moduloptionen, um Fehlkonfigurationen aufzuspüren.
- Halten Sie immer eine funktionierende Konsolen-Root-Session offen, wenn Sie am PAM-Stack arbeiten – sonst droht Lock-Out!
- Notieren Sie Änderungen samt Zeitstempel in `/etc/pam.d/README.local`.
- Ziehen Sie `pam_faillock.so` in Erwägung, falls Ihre Distribution das ältere `pam_tally2.so` bereits abgekündigt hat.

Nach dieser Stunde verstehen die Teilnehmenden die Funktionsweise des PAM-Stacks, können Module gezielt einsetzen und Sicherheitsrichtlinien wie Fehlversuchs- und Ressourcenbegrenzungen selbständig implementieren.

2. Tag 2 – Storage, Boot & Dienste (09:00–16:00)

2.1. 09:00–09:45 Gerätedateien & Partitionierung

Dieser Abschnitt führt in das grundlegende Konzept der Gerätedateien unter Linux ein, erklärt die Unterschiede zwischen MBR- und GPT-Partitionstabellen und demonstriert den praktischen Umgang mit den Werkzeugen `lsblk`, `fdisk` (MBR) und `gdisk` (GPT). Alle Befehle lassen sich ohne Risiko auf einem temporären Loop-Device nachvollziehen.

1. Gerätedateien verstehen

- In `/dev` repräsentiert jede Datei ein Gerät (Block oder Zeichen). Beispiel: `/dev/sda`, `/dev/ttyS0`.
- **Major-Nummer**: verweist auf den Gerätetreiber **Minor-Nummer**: Instanz (bzw. Partition) dieses Treibers.
- Udev erzeugt Gerätedateien dynamisch; statische Einträge sind unüblich geworden.
- Anzeige per `ls -l /dev/sda` → `brw-rw-- 1 root dis` (b = Block, »8« = Major, »0« = Minor).

`lsblk` – Überblick über Blockgeräte

```
lsblk -o NAME,MAJ:MIN,SIZE,TYPE,MOUNTPOINT
# NAME   MAJ:MIN SIZE TYPE MOUNTPOINT
# sda      8:0    50G disk
# +--sda1   8:1    48G part /
# +--sda2   8:2     2G part [SWAP]
```

2. MBR vs. GPT – ein Vergleich

Einführung MBR (DOS-Label): 1983, IBM PC
GPT (GUID Partition Table): 2000er, Teil von UEFI

Max. Partitionszahl MBR (DOS-Label): 4 primäre (oder 3 + erweiterte)
GPT (GUID Partition Table): 128 (standardmäßig)

Max. Datenträgergröße MBR (DOS-Label): 2 TiB (32-Bit LBA)
GPT (GUID Partition Table): 9,4 Zettabyte (64-Bit LBA)

Redundanz MBR (DOS-Label): Keine – Single Boot Record
GPT (GUID Partition Table): Primäre und Backup-Tabelle

Prüfsumme MBR (DOS-Label): Nein
GPT (GUID Partition Table): CRC32 über Header und Tabelle

Boot-Code MBR (DOS-Label): 440 Byte im MBR
GPT (GUID Partition Table): Separater EFI System Partition

Kompatibilität MBR (DOS-Label): Alle BIOS-Systeme
GPT (GUID Partition Table): Benötigt UEFI oder BIOS-Hybrid-Boot

3. Partitionieren mit `fdisk` (MBR)

Interaktive Sitzung (Auszug) Was Sie hier sehen, ist ein Auszug aus einer interaktiven Sitzung mit `fdisk`:

```
sudo fdisk /dev/sdb
Command (m for help): n       # neue Partition
Partition type
   p   primary (0 primary, 0 extended, 4 free)
Select (default p): p
Partition number (1-4, default 1): 1
First sector (2048-...), default 2048: <RETURN>
Last sector, +sectors or +size{K,M,G}: +1G
...
Command (m for help): t       # Typ \"andern
Hex code (type L to list): 83 # Linux
Command (m for help): w       # schreiben
```

Nicht-interaktiv (Skript-Modus)

```
printf 'n\np\n1\n\n+1G\nt\n83\nw\n' | sudo fdisk /dev/sdb
```

4. Partitionieren mit `gdisk` (GPT)

```
sudo gdisk /dev/sdb
Command (? for help): n       # neue Partition
Partition number (1-128, default 1): 1
First sector (34-...), default 2048: <RETURN>
Last sector, +sectors or +size{K,M,G,T,P}: +512M
Current type is 'Linux filesystem'
...
Command (? for help): w       # schreiben
```

Vorteil: Einfacher Wechsel des Partitions-GUID-Typs durch Menübefehl `t` → Eingabe des Hex-Codes, z. B. `EF00` für EFI System-Partition.

5. Geführte Übung – 100 MB Loop-Device mit drei Partitionen

Schritt 0 – saubere Umgebung schaffen Eine Loop-Device-Übung erfordert eine saubere Umgebung. Wechseln Sie in Ihr persönliches Arbeitsverzeichnis (z. B. `~/lab`) und löschen Sie alte Images:

```
cd ~/lab                % persoenlicher Spielplatz
rm -f disk.img          # alte Images loeschen
sudo losetup -D         # alle Loop-Geraete freigeben (optional)
```

Schritt 1 – Image-Datei erzeugen Um eine Loop-Device-Übung zu machen, benötigen wir eine Datei, die wie ein Blockgerät aussieht. Wir erzeugen eine 100 MiB große Datei mit Nullen:

```
dd if=/dev/zero of=disk.img bs=1M count=100 status=progress
# Ergebnis: 100 MiB Null-gefuelltes File
```

Schritt 2 - Loop-Gerät verbinden Ein Loop-Device ist ein virtuelles Blockgerät, das eine Datei als Blockgerät behandelt. Wir verwenden `losetup`, um die Datei `disk.img` mit einem Loop-Device zu verbinden. Das `-find`-Flag sucht ein freies Loop-Device und gibt den Namen zurück. Das `-show`-Flag zeigt den Namen des Loop-Devices an: `/dev/loopX`. Das Ergebnis wird in `LOOP` gespeichert, damit wir es später verwenden können:

```
LOOP=$(sudo losetup --find --show disk.img)
echo "$LOOP"            # z.B. /dev/loop7
```

Schritt 3 – Partitionstabelle wählen Für die Übung nutzen wir GPT (mehr Partitionen, saubereres Layout):

```
sudo parted -s "$LOOP" mklabel gpt
```

Schritt 4 – Drei Partitionen anlegen Wir erstellen drei Partitionen mit den folgenden Eigenschaften:

- 20 MiB ext4 (Primär)
- 30 MiB xfs (Primär)
- Rest (49 MiB) linux-swap (Primär)

```
# 1. 20 MiB ext4
sudo parted -s "$LOOP" mkpart primary ext4 1 MiB 21MiB

# 2. 30 MiB xfs
sudo parted -s "$LOOP" mkpart primary xfs 21 MiB 51MiB

# 3. Rest (~ 49 MiB) linux-swap
sudo parted -s "$LOOP" mkpart primary linux-swap 51MiB 100%

# Kontrolle
sudo partprobe "$LOOP"
lsblk "$LOOP"
```

Schritt 5 – Dateisysteme initialisieren Das Loop-Device ist jetzt partitioniert. Wir formatieren die Partitionen mit den gewünschten Dateisystemen. Die Swap-Partition wird mit `mkswap` initialisiert. Die Partitionen sind `p1`, `p2` und `p3`:

```
sudo mkfs.ext4 "${LOOP}p1"
sudo mkfs.xfs  "${LOOP}p2"
sudo mkswap    "${LOOP}p3"
```

Schritt 6 – Mount-Test Der Test ist einfach: Wir mounten die Partitionen in zwei Verzeichnisse und prüfen mit `df`, ob alles korrekt ist. Die Mount-Punkte sind `/mnt/loop1` und `/mnt/loop2`. Die Swap-Partition wird nicht gemountet:

```
sudo mkdir -p /mnt/loop{1,2}
sudo mount "${LOOP}p1" /mnt/loop1
sudo mount "${LOOP}p2" /mnt/loop2
df -h | grep loop
```

Danach unmounten und Loop-Gerät wieder lösen:

```
sudo umount /mnt/loop1 /mnt/loop2
sudo losetup -d "$LOOP"
```

6. Tipps & Fehlerbehebung

- Bei Fehlermeldung `device is busy` hilft `sudo losetup -a` → belegte Loop-Geräte auflisten.
- `partprobe` oder `blockdev -rereadpt` sorgt dafür, dass der Kernel neue Partitionstabellen sofort erkennt.
- GPT-Header lassen sich mit `gdisk -l` auf Konsistenz prüfen; beschädigte Backups können rekonstruiert werden (`g` → `e`).
- Für automatisierte Workflows bevorzugen Sie `parted -s` (*script mode*) oder `sfdisk` für reine MBR-Disketten.

Nach dieser Einheit können die Teilnehmenden Blockgeräte identifizieren, den Unterschied zwischen MBR und GPT erklären, Partitionen mit `fdisk`/`gdisk` erstellen und ein komplettes Testszenario auf einem Loop-Device sicher durchspielen.

2.2. 09:45–10:00 Kurzpause

2.3. 10:00–10:45 Dateisysteme & Mounts

In dieser Einheit befassen wir uns mit modernen Linux-Dateisystemen (`ext4`, `xfs`, `btrfs`), erklären das Prinzip des Journaling und üben den vollständigen Lebenszyklus „Partition → Dateisystem → Mount → `/etc/fstab`“. Alle Schritte lassen sich direkt am Kurs-Rechner nachvollziehen.

1. Dateisysteme im Vergleich

	ext4	xfs	btrfs
Einführung	2008 (Linux 2.6.28)	1994 (SGI IRIX)	2009 (Linux 2.6.29)
Stärken	Universell, stabil, Fehler-Resistenz	Hohe Parallel-I/O, riesige Dateien	Snapshots, Checksums, RAID 0/1/10/5/6
Max. Dateigröße	16 TiB	8 EiB	16 EiB
Journaling	Ja (Ordered, Writeback, Data=Journal)	Ja (metadata)	Copy-on-Write (CoW)
Online-Resize	**Ja**	**Ja**	**Ja** (auch Shrink[1])
fsck-Zeit	Linear	Parallel	

Journaling-Prinzip: Metadaten-Änderungen werden zuerst in ein Journal geschrieben; bei Absturz kann das Dateisystem die Transaktionen nachholen oder verwerfen → signifikant geringere Wiederherstellungs-Zeiten gegenüber klassischem `ext2`.

2. Dateisystem erzeugen – `mkfs`

```
# ext4 (Standard)
sudo mkfs.ext4 /dev/sdb1

# xfs (Blocksize 4096, Label DATA)
sudo mkfs.xfs -f -b size=4096 -L DATA /dev/sdb2

# btrfs (mit Kompression zstd)
sudo mkfs.btrfs -f -d single -m single -L MEDIA -O compress=zstd /dev/sdb3
```

3. Einhängen – `mount`

Temporär (bis zum Reboot):

```
sudo mkdir -p /data
sudo mount /dev/sdb1 /data
mount | grep /data
```

Sofortiger Zugriff auf Label oder UUID:

```
sudo blkid /dev/sdb1          # zeigt UUID="..." TYPE="ext4"
sudo mount UUID=1234-ABCD /data
```

4. Dauerhaft – `/etc/fstab`

Aufbau *Quelle Ziel Typ Optionen Dump Pass*

```
UUID=1234-ABCD  /data  ext4  defaults,noatime  0  2
```

Dump 0 = kein `dump`, 1 = Backup-Kandidat.

Pass Reihenfolge für `fsck`. 1 = Root-FS, 2 = alle weiteren.

Häufige Mount-Optionen	
`defaults`	rw, suid, dev, exec, auto, nouser, async
`noatime`	Kein Zugriffszeit-Update → schnellere R
`discard`	TRIM für SSD/NVMe aktiviert
`compress=zstd`	(btrfs) automatische COW-Kompression

5. Geführte Übung – Partition `/data` erstellen und testen

Ausgangslage Freier Datenträger `/dev/sdc` mit 5 GiB Kapazität.

Schritt 1 – Partition anlegen (GPT) Hier erstellen wir eine primäre Partition mit dem Typ `ext4` und formatieren sie mit dem Label `DATA`. Die Partitionstabelle ist GPT.

```
sudo parted -s /dev/sdc mklabel gpt
sudo parted -s /dev/sdc mkpart primary ext4 0% 100%
sudo partprobe /dev/sdc              # Kernel-Refresh
```

Schritt 2 – Dateisystem erstellen Um die Partition zu formatieren, verwenden wir `mkfs.ext4` mit dem Label `DATA`. Das `-L`-Flag vergibt den Namen, der später in `/etc/fstab` verwendet wird:

```
sudo mkfs.ext4 -L DATA /dev/sdc1
```

Schritt 3 – Temporär mounten und testen Das Dateisystem wird in `/data` eingehängt. Wir testen den Zugriff mit `ls` und `touch`. Danach wird das Dateisystem wieder unmountet:

```
sudo mkdir -p /data
sudo mount /dev/sdc1 /data
touch /data/testfile
ls -al /data
sudo umount /data
```

Schritt 4 – `/etc/fstab` eintragen Das Dateisystem soll beim Booten automatisch eingehängt werden. Wir tragen es in die `/etc/fstab` ein. Das `blkid`-Kommando ermittelt die UUID der Partition und gibt sie in der richtigen Form aus. Die UUID wird dann in die `/etc/fstab` geschrieben. Die `defaults`-Optionen sind Standardwerte, die für die meisten Anwendungen geeignet sind. Die `noatime`-Option sorgt dafür, dass der Zugriff auf Dateien nicht zeitlich aktualisiert wird, was die Leistung verbessert:

```
UUID=$(blkid -s UUID -o value /dev/sdc1)
echo "UUID=${UUID} /data ext4 defaults,noatime 0 2" \
  | sudo tee -a /etc/fstab
```

Schritt 5 – Reboot-Test Um sicherzustellen, dass alles funktioniert, starten wir den Rechner neu und prüfen, ob das Dateisystem automatisch eingehängt wird. Wir verwenden `findmnt`, um zu überprüfen, ob das Dateisystem korrekt eingehängt ist. Das `findmnt`-Kommando zeigt alle eingehängten Dateisysteme an:

```
sudo reboot                  # VM/Rechner neu starten
# Nach dem Login:
findmnt /data                % sollte eingehangen sein
```

Fehler? Mit `sudo journalctl -b` suchen → Einträge von `systemd-mount` oder `fsck` geben Aufschluss (falsche UUID, fehlendes xfs-Kernel-Modul etc.).

6. Troubleshooting & Best Practices

- Nutzen Sie `UUID` oder `PARTUUID` statt Gerätenamen (`/dev/sdX`) – letztere ändern sich bei hinzukommenden USB-Disks.
- Aktivieren Sie Journaling-Optionen passend zum Workload (`data=writeback` kann Meta-Only-Journal-Leistung erhöhen, aber bei Stromausfall Daten verlieren).
- `btrfs scrub` regelmäßig ausführen, um stille Bit-Rot-Fehler zu erkennen.
- Für hochparallele Workloads (Log-Aggregation) bietet `xfs` oft bessere Performance als `ext4`.
- Testen Sie `fstab`-Einträge ohne Reboot: `sudo mount -a -v` – listet Fehler sofort.

Nach dieser Einheit können die Teilnehmenden Dateisysteme fachgerecht anlegen, temporär wie dauerhaft einhängen, Journaling-Eigenschaften bewerten und Boot-sichere Einträge in `/etc/fstab` erstellen.

2.4. 10:45–11:00 Kaffee-Pause

2.5. 11:00–12:00 Boot-Vorgang (UEFI → GRUB → Kernel → systemd) (UEFI to GRUB to Kernel to systemd)

Diese Einheit zerlegt den Linux-Boot-Prozess in vier klar erkennbare Phasen: **UEFI-Firmware**, **GRUB (bootloader)**, **Kernel + initrd** und **systemd** als User-Space Init. Wir lernen, den Ablauf mit `systemd-analyze` zu visualisieren und typische Boot-Parameter über `/etc/default/grub` zu verändern. Am Ende klonen wir interaktiv einen GRUB-Eintrag, entfernen den Parameter `quiet` und prüfen die Änderung per Reboot.

1. Vier Phasen im Detail

1. **UEFI-Firmware**
   - Liest NVRAM-Variable `BootOrder`, sucht EFI-Binary (meist `\EFI\ubuntu\shimx64.efi` oder `grubx64.efi`) auf der **EFI System Partition (ESP)**.
   - Secure Boot validiert Microsoft/Distribution-Signatur.
2. **GRUB (Stage 2)**
   - Lädt `/boot/grub/grub.cfg` oder `grub.cfg` im ESP-Pfad `/EFI/<distro>/`.
   - Menü zeigt Kernellisten; Default wird nach `GRUB_TIMEOUT` automatisch gestartet.
3. **Kernel + initrd**
   - Kernel entpackt initrd (initramfs), initialisiert Treiber (Block, LVM, mdraid, Crypto).
   - Übergabe der Root-Partition durch Kernel-Parameter `root=` oder via initramfs-Hooks.
4. **systemd (PID 1)**
   - Liest `/usr/lib/systemd/system/default.target`, aktiviert Socket/Service-Units parallel.
   - Abschluss gemessen von `systemd-analyze`.

2. Boot-Dauer analysieren

Gesamtübersicht

```
systemd-analyze
# Startup finished in 2.642s (firmware) +
    1.104s (loader) +
#                   2.997s (kernel) + 9.873s
    (userspace) = 16.618s
```

Kritische Kette

- `systemd-analyze critical-chain` zeigt die Abhängigkeiten zwischen den Boot-Einheiten.
- `systemd-analyze plot` erzeugt eine SVG-Grafik mit detaillierten Zeitangaben.

Hinweis: Die Boot-Dauer ist von der Hardware abhängig. Die Übung sollte auf einem physischen Rechner durchgeführt werden, um die Boot-Dauer zu messen. Die VM-Umgebung kann die Boot-Dauer erheblich beeinflussen.

```
systemd-analyze critical-chain
```

Visuelle Zeitleiste

- `systemd-analyze plot` erzeugt eine SVG-Grafik mit detaillierten Zeitangaben.
- Die Grafik zeigt die Boot-Dauer in Millisekunden und die Abhängigkeiten zwischen den Einheiten.
- Die Grafik kann mit einem Webbrowser geöffnet werden.
- Die Grafik wird im aktuellen Verzeichnis als `boot.svg` gespeichert.

Hinweis: Die Grafik kann je nach Systemkonfiguration und Hardware variieren. Die Boot-Dauer kann auch von der Anzahl der installierten Dienste abhängen.

```
systemd-analyze plot > boot.svg
xdg-open boot.svg
```

3. GRUB anpassen – `/etc/default/grub`

```
sudo nano /etc/default/grub
# Wichtige Variablen:
GRUB_TIMEOUT=5          # Sekunden bis Autostart
GRUB_DEFAULT=0          # Index oder "saved"
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash"
```

- `GRUB_CMDLINE_LINUX_DEFAULT`: Kernel-Parameter nur für Standard-Eintrag.
- `GRUB_CMDLINE_LINUX`: gilt für *alle* Einträge.
- Nach Änderungen immer `sudo update-grub` (*Debian/Ubuntu*) bzw. `sudo grub2-mkconfig -o /boot/grub2/grub.cfg` (RHEL/Fedora) ausführen.

4. Geführte Übung – GRUB-Eintrag klonen ohne `quiet`

Ziel Einen alternativen Menüpunkt erzeugen, der beim Boot alle Kernel-Meldungen anzeigen lässt (nützlich für Troubleshooting).

Schritt 1 – UUID der Root-Partition ermitteln Die UUID der Root-Partition wird benötigt, um den GRUB-Eintrag zu erstellen.

```
ROOT_UUID=$(blkid -s UUID -o value /dev/sda2)
echo "$ROOT_UUID"
```

Schritt 2 – Benutzerdefinierten Eintrag in `/etc/grub.d/40_custom` Der Eintrag wird in `/etc/grub.d/40_custom` erstellt.

```
sudo nano /etc/grub.d/40_custom
#--- Dateianhang ---
menuentry 'Linux (verbose)' {
    insmod gzio
    insmod part_gpt
    insmod ext2
    linux /vmlinuz root=UUID=${ROOT_UUID} ro
    initrd /initrd.img
}
#--- Ende ---
```

- `quiet splash` wird *nicht* gesetzt → Kernel loggt Boot-Meldungen auf Konsole.
- `ro` hält Root-FS anfangs read-only; systemd remountet später rw.

Schritt 3 – Konfiguration regenerieren Um die Änderungen zu aktivieren, muss GRUB neu konfiguriert werden.

- `update-grub` (Debian/Ubuntu) oder `grub2-mkconfig -o /boot/grub2/grub.cfg` (RHEL/Fedora).
- `grub-mkconfig` generiert die Konfiguration aus den Skripten in `/etc/grub.d/` und der Datei `/etc/default/grub`.

```
sudo chmod +x /etc/grub.d/40_custom
sudo update-grub
```

Schritt 4 – Default temporär testen Dies ist der einfachste Weg, um den neuen Eintrag zu testen.

- `grub-reboot` → Boot-Eintrag für den nächsten Reboot auswählen (temporär).
- `grub-set-default` → Boot-Eintrag dauerhaft setzen.

```
sudo grub-reboot 'Linux (verbose)'   % valid for the next reboot
sudo reboot
```

Der Boot-Bildschirm zeigt nun detaillierte Kernel-Meldungen (`[ OK ] Started ...`). Bei Erfolg können Sie den Menüpunkt dauerhaft per `GRUB_DEFAULT="Linux (verbose)"` aktivieren.

5. Troubleshooting & Sicherheit

- Notfall: Im GRUB-Menü `e` für Edit → temporäre Kernel-Optionen ergänzen, z. B. `systemd.unit=rescue.target`.
- EFI-Binary-Pfad prüfen: `efibootmgr -v` listet alle Boot-Einträge mit GUID.
- Bei Secure Boot ± Custom Kernels: Signieren mit `sbsign` oder Secure Boot im BIOS deaktivieren.
- `systemd-journal-remote` kann frühe Boot-Logs erfassen, falls `quiet` nicht entfernt werden darf.

Nach dieser Session verstehen die Teilnehmenden den kompletten Startvorgang, können Boot-Dauer analysieren, GRUB-Parameter anpassen und eigene Menüeinträge sicher verwalten.

2.6. 12:00–13:00 Mittagspause

2.7. 13:00–13:45 Services starten & stoppen

In dieser Einheit vertiefen wir das Arbeiten mit `systemd`. Wir beleuchten die Grundlagen von `systemctl`, erklären den Aufbau von Unit-Dateien, zeigen den Zusammenhang mit Targets (Start-Runleveln) und werfen einen Blick auf die unterliegende cgroups-Integration. Als Praxisübung erstellen wir einen Systemd-Timer `motd.timer`, der zwei Minuten nach jedem Boot eine dynamische »Message of the Day« erzeugt.

1. `systemctl` – wichtigste Befehle

`systemctl status <unit>`	Zustand, Main-PID, let
`systemctl start|stop <unit>`	Sofortiger Start / Stopp
`systemctl enable|disable <unit>`	Auto-Start beim Boot a
`systemctl restart <unit>`	Hartes Neustarten (stop
`systemctl reload <unit>`	Konfiguration neu einle
`systemctl list-units -type=service`	Alle laufenden Services

Beispiel

```
systemctl start sshd.service
systemctl enable sshd.service
systemctl is-enabled sshd              # =>
    enabled
```

2. Anatomy einer Unit-Datei

```
[Unit]
Description=Example HTTP Server
After=network.target

[Service]
Type=simple
User=www-data
ExecStart=/usr/local/bin/httpd -f /etc/httpd
    .conf
Restart=on-failure

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

- **Unit-Sektion** – Meta-Info, Abhängigkeiten (`Requires`, `After`).
- **Service-Sektion** – Startbefehl, Benutzerkontext, Restart-Politik.
- **Install-Sektion** – Verknüpft Unit mit Target(s), damit `systemctl enable` einen Symlink anlegt.

3. Targets = moderne Runlevel

`graphical.target`	GUI + multi-user Service Layer
`multi-user.target`	Netzwerk-/Daemon-Ebene (»Runle
`rescue.target`	Single-User-Mode mit Netzwerk aus
`emergency.target`	Minimal-Shell, keine Mounts

Default-Target anzeigen / wechseln

```
systemctl get-default
sudo systemctl set-default multi-user.target
```

4. cgroups v2 – Ressourcenkontrolle

- Jede Unit läuft in einer eigenen cgroup unter `/sys/fs/cgroup/`.
- Ressourcenlimits via Unit-Optionen: `MemoryMax=512M`, `CPUWeight=200`, `IOReadBandwidthMax=` ...
- `systemctl status -no-pager <unit>` zeigt cgroup-Pfad und Statistiken (CPU -, Memory-Usage).

5. Geführte Übung – `motd.timer`

Ziel Alle zwei Minuten nach jedem Boot soll eine frische MOTD-Datei unter `/etc/motd` erscheinen.

Schritt 1 – Skript anlegen Hier erstellen wir ein einfaches Skript, das die MOTD-Datei aktualisiert. Das Skript wird in `/usr/local/bin` abgelegt und mit `chmod` ausführbar gemacht. Das Skript gibt den aktuellen Zeitstempel aus und schreibt ihn in die MOTD-Datei. Die MOTD-Datei wird mit `tee` erstellt, damit wir die Ausgabe auch sehen können:

```
sudo mkdir -p /usr/local/bin
sudo nano /usr/local/bin/update-motd.sh
#--- Datei: update-motd.sh ---
#!/usr/bin/env bash
echo "Willkommen! $(date)" | tee /etc/motd
#--- Ende Datei ---
sudo chmod +x /usr/local/bin/update-motd.sh
```

Schritt 2 – Service-Unit schreiben Die Service-Unit wird in `/etc/systemd/system/motd.service` abgelegt. Die Unit wird mit `systemctl` gestartet, um die MOTD-Datei zu aktualisieren.

```
sudo nano /etc/systemd/system/motd.service
[Unit]
Description=Generate MOTD

[Service]
Type=oneshot
ExecStart=/usr/local/bin/update-motd.sh
```

Schritt 3 – Timer-Unit schreiben Ein Timer wird in `/etc/systemd/system/motd.timer` abgelegt. Der Timer wird so konfiguriert, dass er 2 Minuten nach dem Booten ausgeführt wird. Die Option `Persistent=true` sorgt dafür, dass der Timer auch nach verpassten Boot-Zyklen ausgeführt wird.

```
sudo nano /etc/systemd/system/motd.timer
[Unit]
Description=Run MOTD generator 2min after
    boot

[Timer]
OnBootSec=2min
Unit=motd.service
Persistent=true    # nach verpasstem Boot
    ausfuehren

[Install]
WantedBy=timers.target
```

Schritt 4 – Aktivieren + Starten Aktivieren Sie den Timer, damit er beim Booten automatisch gestartet wird. Die Option `-now` sorgt dafür, dass der Timer sofort gestartet wird.

```
sudo systemctl daemon-reload
sudo systemctl enable --now motd.timer
systemctl list-timers motd.timer
```

Das Timer-Listing zeigt `NEXT` und `LAST`-Ausführung. Nach 2 Minuten prüfen:

```
cat /etc/motd
sudo journalctl -u motd.service --since -5m
```

Schritt 5 – Reboot-Test Um zu überprüfen, ob der Timer auch nach einem Reboot funktioniert, starten Sie den Rechner neu. Nach dem Reboot wird der Timer automatisch gestart.

- `systemctl list-timers` zeigt alle aktiven Timer.
- `systemctl status motd.timer` zeigt den Status des Timers.
- `systemctl status motd.service` zeigt den Status des MOTD-Services.

```
sudo reboot
# Einloggen nach Reboot:
systemctl list-timers motd.timer    # NEXT
    in ~2min
```

6. Best Practices

- Verwenden Sie `Type=exec` für kurze Prozesse, `Type=forking` für klassische Daemons.
- Halten Sie Service-Logs zentral: `journalctl -f -u <unit>`.
- Nutzen Sie `systemd-analyze blame`, um langsame Services zu identifizieren.
- Timer sind die systemd-Alternative zu `cron` – präziser und mit Abhängigkeitsmodell.
- Ressourcenlimits direkt in der Unit statt via `ulimit` setzen.

Nach dieser Session beherrschen die Teilnehmenden das Starten, Stoppen und Aktivieren von Services mit `systemctl`, verstehen den Aufbau von Unit-Dateien inkl. cgroup-Optionen und können wiederkehrende Aufgaben über systemd-Timer zuverlässig automatisieren.

2.8. 13:45–14:00 Kurzpause

2.9. 14:00–14:45 Datensicherung & Notfallsystem

In dieser Einheit schaffen wir ein solides Fundament für *Backup & Recovery* unter Linux. Wir vergleichen klassische Werkzeuge wie `tar` und `rsync`, geben einen Überblick über Snapshot-Techniken (LVM, Btrfs & ZFS) und zeigen, wie ein Rescue-ISO als letztes Rettungs-System eingesetzt wird. Am Ende sichern wir ein komplettes Home-Verzeichnis mit `tar`, lagern es datumsbasiert in `/backup/YYYY-MM-DD/` aus und verifizieren die Rückspielbarkeit.

1. `tar` – das Schweizer Taschenmesser

Nutzen von `tar` für Backups:

```
sudo tar --xattrs --acls -czvf home-$(date
    +%F).tar.gz /home/alice
```

2. `rsync` – inkrementell und remote

```
rsync -aAXHv --delete /home/  /mnt/usb-disk/
    backup/home/
# -a  archive (preserves almost everything)
# -A  ACLs   -X  xattrs
# -H  hardlinks   -v verbose
```

Snapshots mit `-link-dest`

- `-link-dest` → inkrementelle Backups mit Hardlinks.
- `rsync` vergleicht Quell- und Zielverzeichnis und erstellt Hardlinks für unveränderte Dateien.
- Beispiel: `rsync -a -link-dest=/backup/prev /home/ /backup/$(date +%F)/`

```
rsync -a --link-dest=/backup/prev /home/ /
    backup/$(date +%F)/
```

Hardlinks minimieren Speicherbedarf – nur geänderte Blöcke belegen Platz.

3. Snapshots kurz erklärt

- `lvcreate -s -n snap -L 2G VG/LV` → Copy-on-Write, schnell, benötigt Speicher für Delta.
- `btrfs subvolume snapshot /data /snap/$(date +%F)` → instantan, send/receive für Off-Site.
- `zfs snapshot pool/dataset@daily` → Integrität via Checksums, replizierbar mit `zfs send`.

4. Rescue-ISO

- **SystemRescue**, **Ubuntu Live**, **GRML** – alle liefern aktuelle Kernel, LVM & Btrfs-Tools, chroot-Fähigkeit.
- Typische Tasks: `cryptsetup luksOpen`, `mount /dev/mapper/root /mnt`, `grub-install`, Passwort-reset, fsck ohne gemountetes FS.
- **Best Practice**: Boot-Test einmal *vor* dem Ausfall durchführen (»fire drill«), um Treiber- oder Secure-Boot-Hürden zu erkennen.

5. Geführte Übung – Home sichern und Rücksichern

Schritt 1 – Zielverzeichnis vorbereiten Unser Zielverzeichnis ist `/backup/YYYY-MM-DD`. Wir verwenden `date` im Format `YYYY-MM-DD` und erstellen das Verzeichnis mit `mkdir`. Das `-p`-Flag sorgt dafür, dass auch alle übergeordneten Verzeichnisse angelegt werden, falls sie noch nicht existieren:

```
TODAY=$(date +%F)
sudo mkdir -p /backup/$TODAY
```

Schritt 2 – Archiv erstellen Das Home-Verzeichnis wird mit `tar` gesichert. Das `-p`-Flag sorgt dafür, dass die Berechtigungen der Dateien und Verzeichnisse im Archiv erhalten bleiben.

```
sudo tar --xattrs --acls -cpf /backup/$TODAY
    /home_full.tar /home
# optional komprimieren:
sudo gzip /backup/$TODAY/home_full.tar
```

Schritt 3 – Integrität verifizieren Integrationsprüfung des Archivs mit `gzip` → `gzip -t` prüft die Integrität des Archivs. Das `-t`-Flag sorgt dafür, dass nur die Integrität des Archivs überprüft wird, ohne es zu entpacken. Ein Rückgabewert von 0 bedeutet, dass die Integrität des Archivs in Ordnung ist:

```
gzip -t /backup/$TODAY/home_full.tar.gz   #
    return code 0 = OK
```

Schritt 4 – Test-Restore Ein Test-Restore wird durchgeführt, um sicherzustellen, dass das Archiv korrekt erstellt wurde und die Daten wiederhergestellt werden können. Wir entpacken das Archiv in ein temporäres Verzeichnis und vergleichen die Verzeichnisbäume mit `diff`. Das `head`-Kommando zeigt nur die ersten 10 Unterschiede an, um die Ausgabe zu begrenzen. Wenn keine Unterschiede vorhanden sind, wird keine Ausgabe angezeigt:

```
sudo mkdir /tmp/restore
sudo tar -xpf /backup/$TODAY/home_full.tar.
    gz -C /tmp/restore
diff -r /home/ /tmp/restore/home/ | head
```

Keine Ausgabe → Verzeichnisbäume sind identisch.

Schritt 5 – Bereinigungsstrategie Wieso Backup & Recovery?

- **Backup**: Datenverlust durch Hardwaredefekt, versehentliches Löschen, Malware, Ransomware.
- **Recovery**: Systemausfall durch Hardwaredefekt, Softwarefehler, Konfigurationsfehler.

```
find /backup -maxdepth 1 -type d -mtime +30
    -exec sudo rm -rf {} \;
# loescht Backups aelter als 30 Tage
```

6. Best Practices

- **3-2-1-Regel**: Drei Kopien, zwei Medien, eine davon Off-Site.
- Backups regelmäßig *lesen* – ein ungeprüftes Backup ist kein Backup.
- Verwenden Sie `-listed-incremental` für tägliche Deltas, Full-Backup wöchentlich, Aufbewahrung via `-remove-files`.
- Bewahren Sie Rescue-ISO & Passwort (LUKS, GPG) offline (USB-Stick + Safe) auf.
- Verschlüsseln Sie externe Backups (`rsync → sshfs` oder `tar | gpg -symmetric`).

Nach dieser Session beherrschen die Teilnehmenden den sicheren Einsatz von `tar` und `rsync`, verstehen Snapshot-Konzepte und können im Notfall mit einer Rescue-ISO das System sowie wichtige Daten zuverlässig wiederherstellen.

2.10. 14:45–15:00 Kurzpause

2.11. 15:00–16:00 Fehlerbehebung am *initramfs*

In dieser Session trainieren wir den Umgang mit frühen Boot-Fehlern, die bereits *vor* dem Wechsel auf das Root-Dateisystem auftreten. Wir provozieren gezielt eine Kernel-Panic, indem wir im GRUB-Eintrag eine falsche Root-UUID angeben. Anschließend werten wir die vorherige fehlgeschlagene Boot-Sequenz mit `journalctl -b -1` aus und zeigen, wie man das System direkt im GRUB-Prompt repariert.

1. Theorie – was passiert im `initramfs`?

1. GRUB lädt Kernel (`vmlinuz`) und initial Ramdisk (`initrd`, auch *initramfs*).
2. Das initramfs-Skript sucht das Root-Dateisystem (meist über `root=UUID=<...>`).
3. Scheitert dies → »/init« bricht ab, startet eine **BusyBox-Shell** oder ruft `panic`: Kernel-Panic mit Call-Trace auf der Konsole.
4. Nach erfolgreichem Mount wird `pivot_root` / (`switch_root`) ausgeführt und die Kontrolle an `systemd` übergeben.

Typische Ursachen

- Falsche UUID/PARTUUID, Gerät wurde umbenannt (`/dev/sda` → `/dev/vda`).
- Fehlendes Modul im initramfs (RAID, LUKS, NVMe-Treiber).
- Beschädigte initrd-Datei.

2. Kernel-Panic simulieren

Schritt 1 – Aktuellen GRUB-Menüeintrag editieren

1. Beim GRUB-Screen die gewünschte Linux-Zeile markieren, `e` drücken.
2. In der `linux`-Zeile die gültige `root=UUID=xxxxxxxx-xxxx-...` durch einen Fantasie-Wert ersetzen.
3. Mit `Ctrl+x` (oder `F10`) booten.

Erwartung Nach kurzem Laden endet der Boot-Prozess im `BusyBox (initramfs)`-Prompt oder zeigt eine Kernel-Panic.

3. Analyse nach Neustart

```
sudo journalctl -b -1 -p err..alert
# zeigt nur Fehlermeldungen des VORHERIGEN Boots
```

Filtert man zusätzlich nach dem Schlüsselwort `ROOT`, findet man meist Meldungen wie:

```
kernel: VFS: Cannot open root device "UUID=deadbeef-..."
dracut-initqueue: Warning: Could not find root UUID=deadbeef-...
```

4. Troubleshooting-Challenge

Aufgabe A – Fehlerhafte UUID erkennen

1. Reboot → GRUB-Menü → `c` (Command-Line).
2. Alle erkannten Dateisysteme auflisten:
   `grub> ls`.
3. Richtiges Gerät (z. B. `(hd0,gpt2)`) mounten:
   `grub> set root=(hd0,gpt2)`
   `grub> ls /` → sollte `vmlinuz`, `initrd.img` enthalten.
4. Tatsächliche UUID anzeigen:
   `grub> cat /etc/fstab` oder
   `grub> search --no-floppy --set=root --fs-uuid <TAB>`.

Aufgabe B – Korrektur im GRUB-Prompt

1. Ausgangspunkt: GRUB-Shell (`grub>`).
2. Kernel + initrd mit korrekter Root-Angabe neu laden:

```
linux /vmlinuz root=UUID=<korrekte-uuid> ro
initrd /initrd.img
boot
```

3. System bootet erfolgreich in den Multi-User-Mode.

5. Dauerhafte Reparatur

```
sudo blkid | grep /dev/sda2      # oder passendes Root-Device
sudo nano /etc/default/grub      % GRUB_CMDLINE_LINUX_DEFAULT aendern
sudo update-grub                 # oder grub2-mkconfig -o /boot/grub2/grub.cfg
```

6. Best Practices

- Halten Sie ein Live-Rescue-Medium bereit (SystemRescue, GRML) – falls GRUB selbst beschädigt ist.
- Erstellen Sie nach Kernel-/initramfs-Updates ein automatisches Backup der alten initrd (`/boot/backup-initrd/`).
- Aktivieren Sie `GRUB_TERMINAL_OUTPUT="gfxterm serial"` für Remote-Konsole via IPMI.
- Nutzen Sie `dracut -H -f` bzw. `update-initramfs -u` nach Hardware-Änderungen.

Nach dieser Session können die Teilnehmenden kritische Boot-Fehler reproduzieren, mithilfe von journalctl analysieren und noch im GRUB-Prompt die nötigen Korrekturen vornehmen, um das System ohne Neuinstallation wieder hochzufahren.

3. Tag 3 – Paketmanagement, Kernel & Systemsicherheit (09:00–16:00)

3.1. 09:00–09:45 Paketverwaltung (DEB- & RPM-Welt)

Die Linux-Distributionsfamilien teilen sich traditionell in zwei große Ökosysteme: die **DEB-Welt** (Debian, Ubuntu u. a.) und die **RPM-Welt** (Fedora, RHEL, openSUSE). Beide verwenden binäre Paketdateien (`.deb` bzw. `.rpm`), die Metadaten, Abhängigkeiten und Signaturen enthalten. Wir vergleichen die Architektur von `apt` / `dpkg` mit `dnf` (Nachfolger von `yum`) und demonstrieren den Umgang mit Repositories, GPG-Signaturen und Paketversionen.

1. Architektur der DEB-Werkzeuge

`dpkg` Low-Level-Backend. Installiert, entfernt und konfiguriert einzelne `.deb`-Dateien (`dpkg -i foo.deb`, `dpkg -status foo`).

`apt` / `apt-get` High-Level Resolver. Spricht mit den Repository-Servern, löst Abhängigkeiten, prüft Signaturen und ruft abschließend `dpkg` auf.

`apt-cache` Lokales Metadaten-Recherche-Tool (`apt policy <pkg>`, `apt rdepends <pkg>`).

Repository-Definition Unten sehen Sie ein Beispiel für eine Repository-Definition in `/etc/apt/sources.list.d/ubuntu.list`.

```
# /etc/apt/sources.list.d/ubuntu.list
deb [arch=amd64 signed-by=/usr/share/
    keyrings/ubuntu.gpg] \
    http://archive.ubuntu.com/ubuntu jammy
        main universe
```

- `signed-by=` verweist auf einen GPG-Public-Keyring (ersetzt das veraltete `apt-key`).
- Mehrere Komponenten (main, universe, multiverse) können Leerraum-getrennt in einer Zeile stehen.

2. Signaturen und Vertrauenskette

1. Maintainer signiert Release-Datei → SHA256 über Paketindex.
2. `apt` verifiziert Release.gpg mit Distri-Key (liegt in `/usr/share/keyrings/`).
3. Paket-Hashes (*InRelease* → `Packages.gz`) sichern Integrität jeder `.deb`.

Manueller Key-Import Um einen GPG-Key zu importieren, verwenden Sie den `curl`-Befehl, um den Key von einem Server herunterzuladen, und speichern Sie ihn in `/usr/share/keyrings/`.

```
curl -fsSL https://example.com/key.gpg | \
  sudo tee /usr/share/keyrings/example.gpg
      >/dev/null
```

3. RPM-Welt mit `dnf`

`rpm` Low-Level, analog zu `dpkg`.

`dnf` Dependency-Resolver. Besitzt internes Python-API, unterstützt Delta-RPMs und Modularity-Streams.

`dnf history` Transaktionslog mit Undo-Funktion.

```
sudo dnf install htop
sudo dnf history info last
sudo dnf downgrade htop-3.0.5-1.fc40
```

Vergleich auf einen Blick

Funktion	DEB-Welt	RPM-Welt
Low-Level Tool	`dpkg`	`rpm`
Resolver + Repo Mgr	`apt`	`dnf`
Rollback-Möglichkeit	`apt-get downgrade`	`dnf history undo`
Delta-Pakete	(experimentell)	**Ja**
Modular Streams	(PPA, Backports)	**Ja** (`module`)

4. Paketversionen und Pinning

```
# Alle verfuegbaren Versionen anzeigen
apt list -a htop

# PPA-Version priorisieren (Pin-Datei)
echo "Package: *
Pin: origin ppa.launchpad.net
Pin-Priority: 700" | sudo tee /etc/apt/
    preferences.d/ppa.pref
```

`Pin-Priority` > 500 gewinnt gegenüber gleichnamigen Paketen aus den offiziellen Repos.

5. Geführte Übung – PPA einbinden und Downgrade testen

Ziel

- Fremd-Repository (PPA) hinzufügen.

- `htop` installieren (neue Version).
- Anschließend gezielt auf eine ältere Version downgraden.

Schritt 1 – PPA einbinden Um ein PPA hinzuzufügen, verwenden wir den `add-apt-repository`-Befehl. Das PPA wird in der Datei `/etc/apt/sources.list.d/` abgelegt.

```
sudo apt install software-properties-common
sudo add-apt-repository ppa:maxiberta/htop-next
sudo apt update
```

Schritt 2 – Paket installieren Um ein Paket zu installieren, verwenden wir den `apt install`-Befehl. Das PPA wird in der Datei `/etc/apt/sources.list.d/` abgelegt.

```
sudo apt install htop
htop --version       # z.B. 3.3.0~git2025...
```

Schritt 3 – Verfügbare Versionen prüfen Wir müssen die Versionen des Pakets `htop` prüfen, um sicherzugellen, ob die Downgrade-Version verfügbar ist.

```
apt list -a htop | column -t
# htop 3.3.0~git2025   ppa
# htop 3.2.2-1ubuntu1  jammy-updates
# htop 3.0.5-2         jammy
```

Schritt 4 – Downgrade Bei einem Downgrade verwenden wir den `apt install`-Befehl mit der gewünschten Version.

```
sudo apt install htop=3.2.2-1ubuntu1
htop --version        # bestaetigt aeltere Version
```

Alternative: Paket sperren Um zu verhindern, dass ein Paket aktualisiert wird, können wir `apt-mark hold` verwenden. Das Paket wird dann nicht mehr aktualisiert, auch wenn eine neuere Version verfügbar ist.

```
sudo apt-mark hold htop          # haelt Version fest
apt-mark showhold
```

6. Fehlerbehebung

- `apt -fix-broken install` repariert unaufgelöste Abhängigkeiten.
- `dpkg -configure -a` setzt unterbrochene Post-Install-Skripte fort.
- `dnf clean metadata` entfernt veraltete Repo-Cachen.
- GPG-Fehler? – Keyring prüfen (`apt-key list`, `rpm -import ...`).

7. Best Practices

- Für produktive Systeme nur signierte, vertrauenswürdige Repos verwenden; PPAs zurückhalten oder in Staging testen.
- Regelmäßig `apt upgrade -with-new-pkgs` bzw. `dnf upgrade` – Security-Patches kommen meist täglich.
- Logs aufbewahren (`/var/log/apt/history.log`, `dnf history`) → Nachvollziehbarkeit bei Regressionen.
- Snapshot-basierte Rollbacks (Btrfs-/LVM-Snapshot) vor großen Upgrades erstellen.

Nach dieser Einheit beherrschen die Teilnehmenden das Hinzufügen neuer Repositories, die sichere Installation und das Downgraden von Paketen sowie wesentliche Unterschiede zwischen der DEB- und der RPM-Welt.

3.2. 09:45–10:00 Kurzpause

3.3. 10:00–10:45 Software-Installation aus Quellcode

Nicht jede gewünschte Software liegt in einem Distributions-Repository oder passend zur eigenen Version vor. In solchen Fällen hilft der klassische Dreischritt `./configure && make && make install`. Damit das Ergebnis trotzdem paketkonform bleibt, nutzen wir **checkinstall**, das aus den installierten Dateien automatisch ein `.deb`- (oder `.rpm`-) Paket baut. Diese Einheit erklärt die notwendigen Build-Abhängigkeiten, Autotools-Workflow, Konfigurationsoptionen und ein sauberes Rollback.

1. Build-Abhängigkeiten ermitteln

- Debian/Ubuntu: `sudo apt build-dep <binary-pkg>` (installiert alle *Build-Depends* aus `debian/control`).
- Fedora/RHEL: `sudo dnf builddep <src-rpm>` oder `sudo dnf install @development-tools`.
- Fehlende Bibliotheken erkennt man zur Not am `configure: error: ... library not found`.

Essenzielle Pakete `build-essential` (DEB) bzw. `gcc gcc-c++ make autoconf automake libtool pkg-config` – ohne sie bricht jeder Compile ab.

2. Autotools-Workflow im Detail

1. `./configure`
   - Prüft Toolchain + Header, schreibt `Makefile`.
   - Wichtige Flags:
     `-prefix=/usr/local` (Install-Pfad)
     `CFLAGS="-O2 -march=native"` (Optimierung)
     `-disable-static` (keine `*.a`-Libs).

2. `make -j$(nproc)`
   - Kompiliert; paralleler Build beschleunigt deutlich.
   - `make check` führt Unittests aus (falls vorhanden).
3. `make install`
   - Kopiert Binär- & Support-Dateien in `$prefix` (z. B. `/usr/local` oder via `DESTDIR=/tmp/pkgroot`).
   - Ohne Paketmanager schwer rückgängig → daher *checkinstall*.

3. `checkinstall` – vom »make install« zum Paket

```
sudo checkinstall --install=no --pkgname=
    wget-custom \
  --pkgversion=1.24.5 --pkgrelease=1
# erzeugt wget-custom_1.24.5-1_amd64.deb
```

- Interceptet `make install`, protokolliert alle geschriebenen Dateien → erstellt Paket.
- `-install=no`: nur bauen, nicht sofort einspielen (nützlich für Tests oder mehrere Architekturen).
- Unterstützt DEB, RPM, Slackware (`-type=`) – wird vom jeweiligen Backend abhängig.

4. Geführte Übung – `wget` selbst bauen und säubern

Ziel Quellcode von `wget` (neueste Release) kompilieren, als DEB-Paket registrieren und anschließend restlos entfernen.

Schritt 1 – Voraussetzungen Welche Pakete sind nötig? → `apt build-dep wget`

- `build-essential` → Compiler, Make, Autotools.
- `libssl-dev` → OpenSSL-Entwicklungspaket.
- `libpcre2-dev` → PCRE2-Entwicklungspaket.
- `zlib1g-dev` → Zlib-Entwicklungspaket.

```
sudo apt update
sudo apt install build-essential
    checkinstall \
                libssl-dev libpcre2-dev
                    zlib1g-dev
```

Schritt 2 – Quellcode laden Um `wget` zu kompilieren, laden wir den Quellcode von der GNU-Website herunter. Wir verwenden die aktuelle Version 1.24.5.

```
wget https://ftp.gnu.org/gnu/wget/wget
    -1.24.5.tar.gz
tar xzf wget-1.24.5.tar.gz
cd wget-1.24.5
```

Schritt 3 – Konfigurieren & Kompilieren Hier verwenden wir die `configure`-Optionen, um `wget` mit OpenSSL zu kompilieren. Das `-prefix`-Flag gibt an, wo die Binärdateien installiert werden sollen. `make` kompiliert den Quellcode und `make check` führt Unittests aus (optional).

```
./configure --prefix=/usr/local --with-ssl=
    openssl
make -j$(nproc)
make check        # optional
```

Schritt 4 – Paket mit `checkinstall` erzeugen Das `checkinstall`-Skript wird aufgerufen, um ein DEB-Paket zu erstellen.

```
sudo checkinstall --pkgname=wget-custom \
                  --pkgversion=1.24.5 \
                  --provides=wget --backup=
                     no
# Fragen im Dialog:
#  - description, category (use "utils")
#  - license, group, etc.
```

Schritt 5 – Paket installieren + testen Nun installieren wir das Paket und testen die Installation.

```
sudo apt install ./wget-custom_1.24.5-1
    _amd64.deb
wget --version | head -n1        # zeigt
    1.24.5
```

Schritt 6 – Sauber entfernen Als Nächstes entfernen wir das Paket mit `apt` → `checkinstall` hat alle Dateien registriert. Das `autoremove`-Flag entfernt alle Abhängigkeiten, die nicht mehr benötigt werden.

```
sudo apt remove wget-custom
sudo apt autoremove             # evtl.
    verwaiste -dev Pakete
```

5. Fehlerbehebung

- `configure: error: ... library not found` → passende `libfoo-dev` installieren.
- »C compiler cannot create executables« → `build-essential` oder `gcc` fehlt.
- `make: No rule to make target ...` → eventuell `./autogen.sh` (GTK- oder Git-Tree) nötig.
- Konflikt mit Distributions-Paket: `sudo apt-mark hold wget` vor Installation des Custom-Pakets oder `Provides: wget` in `checkinstall`.

6. Best Practices

- Bauen Sie immer in einem separaten Arbeitsverzeichnis (`$HOME/src/`) – keine Root-Dateien in $HOME.
- Nutzen Sie `DESTDIR` für Test-Installationen (`make DESTDIR=/tmp/pkgroot install`) ohne Root-Rechte.

- Halten Sie Build-Skripte (`build.sh`) in Git → reproduzierbare Builds sind Gold wert.
- Für komplexe Projekte langfristig auf `dpkg-buildpackage` (DEB) oder `rpmbuild` umsteigen; `checkinstall` ist ideal für Einmal-Builds.

Nach dieser Session verstehen die Teilnehmenden den vollständigen Quellcode-Build-Prozess, können Abhängigkeiten klären, selbst kompilierte Software als Paket registrieren und bei Bedarf rückstandsfrei entfernen.

3.4. 10:45–11:00 Kaffee-Pause

3.5. 11:00–12:00 Kernel-Grundlagen

Der Linux-Kernel bildet die unterste Schicht des Systems – er verwaltet Speicher, Prozesse, I/O und Geräte. In dieser Einheit erhalten die Teilnehmenden einen praxisorientierten Einstieg in die Themen Versionsschema, Kernel-Module, Konfigurationswerkzeuge (`make menuconfig`), dynamisches Laden mit `modprobe` sowie Fehlerdiagnose per `dmesg`. Zum Abschluss schreiben wir ein Mini-Modul `hello.ko`, laden es und verifizieren dessen Log-Ausgabe.

1. Versionsschema verstehen

<Major>.<Minor>.<Patch>[-rcN][-stable]

- **6.9.0-rc7**: Release-Candidate der nächsten Hauptversion.
- **6.8.10**: Stable-Patch (Regression + Security-Fixes).
- **Long Term Support (LTS)**: Wird ≥ 2 Jahre gepflegt (z. B. 6.6.y).

Aktuelle Version anzeigen:

```
uname -r          # Kernel-Release
uname -v          # Build-Datum, Compiler
```

2. Kernel-Module (.ko)

`lsmod`	Liste aller geladenen Module
`modinfo foo.ko`	Abhängigkeiten, Lizenz, Author
`modprobe foo`	Modul + benötigte Deps laden
`rmmod foo`	Entladen (wenn nicht benutzt)

Autoload via udev Gerät → `/lib/modules/$(uname -r)/modules.alias` → modprobe.

3. Kernel konfigurieren – `make menuconfig`

1. `apt source linux` oder `git clone torvalds/linux`.
2. `cp /boot/config-$(uname -r) .config` als Basis.
3. `make menuconfig`: **M**=Module, **Y**=fest einkompiliert, **N**=aus.
4. `make -j$(nproc)` + `make modules_install`; `make install` legt `vmlinuz`, `initrd` an.

Tipp: Für neue Hardware nur benötigte Treiber einkompilieren, um die initrd klein und den Boot schneller zu halten.

4. Live-Management mit `modprobe`

```
# Modul + Parameter laden
sudo modprobe i915 enable_psr=0

# Blacklist dauerhaft (Boot)
echo "blacklist nouveau" | sudo tee /etc/
    modprobe.d/blacklist-gpu.conf
```

Parameter in `/sys/module/<name>/parameters/` einsehbar.

5. Kernel-Logs per `dmesg`

```
dmesg -H | less          # farbig, suchbar
dmesg --follow           # Live-Stream (like
    tail -f)
dmesg -T | grep -i usb # Zeitstempel +
    Filter
```

Fehlende Firmware oder Device-Probleme erscheinen hier häufig als erste Hinweise.

6. Geführte Übung – Modul `hello.ko`

Ziel Eigenes Modul schreiben, kompilieren, laden, entladen und Log-Nachrichten prüfen.

Schritt 1 – Verzeichnis anlegen Wichtig nicht vergessen das Verzeichnis zu wechseln, da sonst die Module nicht korrekt geladen werden können.

```
mkdir -p ~/kernel-lab/hello
cd ~/kernel-lab/hello
```

Schritt 2 – Quelltext `hello.c` Der Quelltext ist sehr einfach gehalten und besteht nur aus den minimalen Funktionen, die benötigt werden, um ein Modul zu erstellen.

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Workshop");
MODULE_DESCRIPTION("Simple Hello Module");

static int __init hello_init(void)
{
    pr_info("hello: Modul geladen!\n");
    return 0;
}
static void __exit hello_exit(void)
{
    pr_info("hello: Modul entladen.\n");
}
module_init(hello_init);
module_exit(hello_exit);
```

Schritt 3 – `Makefile` Die Makefile-Konfiguration ist sehr einfach gehalten und enthält nur die minimalen Optionen, die benötigt werden, um ein Modul zu erstellen.

```
obj-m := hello.o
KDIR  := /lib/modules/$(shell uname -r)/build
PWD   := $(shell pwd)

all:
    $(MAKE) -C $(KDIR) M=$(PWD) modules
clean:
    $(MAKE) -C $(KDIR) M=$(PWD) clean
```

Schritt 4 – Kompilieren Nun kompilieren wir das Modul mit dem `make`-Befehl.

```
sudo apt install build-essential linux-headers-$(uname -r)
make        # erzeugt hello.ko
```

Schritt 5 – Laden und testen Nun können wir das Modul laden und testen.

```
sudo insmod hello.ko
dmesg | tail -n3      # "hello: Modul geladen!"
lsmod | grep hello    # bestaetigt den Ladezustand

sudo rmmod hello
dmesg | tail -n3      # "Modul entladen"
```

Schritt 6 – Fehlerbehebung

- `insmod: ERROR: ... invalid module format` → Headers passen nicht zur laufenden Kernel-Version.
- `unknown symbol` → fehlende Abhängigkeit, im `Makefile` gegebenenfalls `hello-objs := foo.o` ergänzen oder `modprobe` statt `insmod` (zieht Deps).

7. Best Practices

- Entwickeln Sie Module auf einem `KVM/QEMU`-Gast — Abstürze killen nicht den Host.
- Symbol-Verweise via `EXPORT_SYMBOL_GPL()` beschränken proprietäre Module; Distributionen blocken oft non-GPL.
- `modprobe.d`-Dateien in Versionskontrolle (`/etc/git`) — Nachvollziehbarkeit.
- Nutzen Sie `make LOCALVERSION=-custom` — eigener Suffix im `uname -r` verhindert Mix-up mit Distributions-Kernel.

Nach dieser Session verstehen die Teilnehmenden das Kernel-Versionsschema, den Lebenszyklus eines Moduls, können Kernel-Konfigurationen anpassen und einfache Treiber kontrolliert kompilieren, laden und wieder entfernen.

3.6. 12:00–13:00 Mittagspause

3.7. 13:00–13:45 Systemsicherheit mit *sudo*

`sudo` ermöglicht es, einzelne Befehle mit erhöhten Rechten auszuführen, ohne ein vollständiges `root`-Kennwort preiszugeben. Der zentrale Konfigurationspunkt ist die Datei `/etc/sudoers` (bzw. Fragmente in `/etc/sudoers.d/`). Wir erarbeiten die Syntax, setzen das *Least-Privilege*-Prinzip konkret um und zeigen, wie man Befehle gezielt frei- oder ausschließt. Zum Schluss testen wir die Audit-Logs.

1. `sudoers` – Aufbau & Syntax

```
# <User|%Group|User_Alias>  <Host_Alias> = (RunAs_Alias) <Options>  <Cmnd_Alias | Einzelbefehl>
alice           ALL           = (root)      NOPASSWD:             /usr/bin/apt-get update
```

Host_Alias → FQDN oder `ALL`. Praktisch bei Cluster-Deployments.

RunAs_Alias → z. B. `(www-data)` anstelle von `(root)`.

Cmnd_Alias → Liste erlaubter Programme, optional mit Arg-Filtern.

Optionen → `NOPASSWD`, `NOEXEC`, `SETENV` ...

Visudo statt Nano `visudo` sperrt die Datei exklusiv und führt einen Syntax-Check durch → korruptes `sudoers` sperrt Sie sonst aus.

2. Least-Privilege in der Praxis

- **Minimaler Befehlsumfang**: lieber `/bin/systemctl restart apache2.service` als `/bin/systemctl * apache2.service`.
- **NOEXEC** verhindert das Nachladen weiterer Binaries via `LD_PRELOAD`.
- **secure_path** in `sudoers` setzt einen festen Suchpfad → Schützt vor Path-Hijacking.
- **env_reset** löscht unsichere Umgebungsvariablen.

3. Paket-Black- / Whitelist

Oft sollen Benutzer nur *einzelne* Paketoperationen ausführen:

```
Cmnd_Alias PKG_WHITELIST = /usr/bin/apt-get update,\
                           /usr/bin/apt-get install htop,\
                           /usr/bin/apt-get install lynx
bob  ALL = (root) PKG_WHITELIST
```

Für eine echte *Black*-List (»alles außer«) legt man stattdessen eine White-List an und verbietet das generische `apt-get install *` vollständig.

4. Geführte Übung (Troubleshooting)

Ziel Benutzer `webop` darf ausschließlich `systemctl restart apache2` ausführen.

1. Benutzer anlegen:

```
sudo adduser --disabled-password webop
```

2. `visudo -file=/etc/sudoers.d/webop` öffnen und eintragen:

```
Cmnd_Alias APACHERESTART = /bin/
    systemctl restart apache2.service
webop       ALL = (root) APACHERESTART
```

3. Rechte testen:

```
su - webop
sudo -l                          # zeigt
    exakt eine Zeile
sudo systemctl restart apache2
sudo systemctl stop apache2    # => "not
    allowed to execute"
```

4. Audit-Log prüfen (Debian/Ubuntu):

```
sudo tail /var/log/auth.log | grep sudo
# ... webop : TTY=pts/1 ; COMMAND=/bin/
    systemctl stop apache2.service ...
```

Auf RHEL/Fedora lautet die Datei `/var/log/secure`.

5. Häufige Stolperfallen

- Wildcards zu großzügig gesetzt: `/bin/systemctl * apache2.service` lässt auch `start` & `stop` zu.
- Aliase vergessen → Redundanz / Copy-Paste-Fehler.
- `NOPASSWD` inflationär: nur für Automationsaccounts!
- Keine Timeouts: `timestamp_timeout=0` erzwingt Passworteingabe bei jedem Befehl (empfohlen für sensible Server).

6. Best Practices

- Konfigurationsfragmente in `/etc/sudoers.d/` nach *Rollen* benennen (`webops`, `dbadmin`).
- Großen Roll-Out: `ansible sudoers_module` oder `cfengine` verwenden.
- Zweistufiges Review: Pull-Request → CI-Linter (`visudo -c`) → Merge → Deployment.
- Audit-Daten nach SIEM forwarden (`rsyslog & remote TLS`, `journald forward`).

Nach dieser Session können die Teilnehmenden präzise `sudoers`-Regeln erstellen, das Least-Privilege-Prinzip durchsetzen und unzulässige Befehle zuverlässig protokollieren.

3.8. 13:45–14:00 Kurzpause

3.9. 14:00–14:45 Wartung & Monitoring

Regelmäßige Wartung und frühzeitige Erkennung von Problemen sind entscheidend für einen stabilen Serverbetrieb. In dieser Einheit lernen wir, Logs effizient auszuwerten (`journalctl`), Rotationsrichtlinien anzupassen, Aufgaben zeitgesteuert mit `cron` oder ad-hoc mit `at` auszuführen und einfache Health-Checks zu etablieren. Zum Abschluss richten wir einen wöchentlichen Cron-Job ein, der `apt update && apt -y upgrade` ausführt und das Ergebnis per Mail versendet.

1. Protokolle auswerten – `journalctl`

- **Zeitraum**: `journalctl --since "2025-05-01 00:00" ...`
- **Dienstfilter**: `journalctl -u ssh.service --grep "Failed`
- **Persönliche Persistenz**: `/var/log/journal/` legt system-weite, binäre Logs ab; Größe begrenzen mit `SystemMaxUse=1G` in `/etc/systemd/journald.conf`.
- **Aufräumen**: `journalctl --vacuum-time=30d`.

2. Log-Rotation

Klassische Text-Logs (`/var/log/*.log`) rotieren via `logrotate` (Tägliche Ausführung durch `/etc/cron.daily/logrotate`). Konfigurationsbeispiel:

```
#/etc/logrotate.d/nginx
/var/log/nginx/*.log {
    weekly
    rotate 8
    compress
    delaycompress
    missingok
    notifempty
    postrotate
        systemctl kill -s SIGUSR1 nginx
    endscript
}
```

Tipp Prüfen Sie Rotationsregeln mit `logrotate --debug /etc/logrotate.conf`.

3. Aufgabenplanung – `cron` und `at`

`cron` Periodische Jobs; Syntax `min hour dom mon dow user cmd`. System-Crontabs liegen in `/etc/cron.*` oder fragmentiert unter `/etc/cron.d/`.

`at` Einmalige Ausführung zu festem Zeitpunkt.

```
# Einmalig morgen 01:30
echo "tar czf /backup/etc-$(date +%F).tar.gz
    /etc" | at 01:30 tomorrow
atq    # Queue ansehen
atrm 5 # Job-ID 5 loeschen
```

4. Health-Checks

- **systemd Watchdogs**: `WatchdogSec=60s` → Dienst muss »kick« senden.
- **Nagios/Icinga Plugins**: Skripte mit Exit-Codes 0 = OK, 1 = Warn, 2 = Crit.
- **Node Exporter + Prometheus Alertmanager**: Zeitreihenbasiertes Monitoring, Redundanz via HA-pairs.
- **Self-Healing Hooks**: `OnFailure=restart.service` in Unit-Datei.

5. Geführte Übung – Wöchentlicher Update-Cron-Job

Voraussetzung Lokaler MTA wie `postfix --relay = none` oder `msmtp`/`sSMTP`; Paket `mailutils` installiert.

Schritt 1 – Root-Crontab öffnen Um die Root-Crontab zu bearbeiten, verwenden wir `sudo crontab -e`.

```
sudo crontab -e
```

Schritt 2 – Job eintragen Den Job tragen wir in die Crontab ein.

- `0 3 * * 1` → Montag, 03:00 Uhr.
- `apt update` → Paketlisten aktualisieren.
- `apt -y upgrade` → Pakete aktualisieren.
- `2>&1` → Fehler an Standardausgabe leiten.
- `mail -s "Weekly updates"` → Mail mit Betreff.

```
# Min (0)  Hour (3)  - Day  - Month  -
    Weekday (Mo)
  0          3          *      *        1
          \
  apt update && apt -y upgrade 2>&1 | \
      mail -s "Weekly updates $(hostname)"
          admin@example.com
```

Erläuterung

- Ausführung montags 03:00 Uhr Serverzeit.
- `2>&1` leitet Fehler (2) an Standardausgabe (1) → vollständige Mail.
- Mails landen im Posteingang von `admin@example.com`; lokal → `/var/mail/admin`.

Schritt 3 – Testlauf forcieren Testen sollten wir dies auch.

```
sudo run-parts --report /etc/cron.weekly
# oder
sudo bash -c 'apt update && apt -y upgrade'
    | \
  mail -s "TEST updates" admin@example.com
```

Schritt 4 – Prüfung

1. `mail` öffnen → Nachricht vorhanden?
2. `grep CRON /var/log/syslog | tail` zeigt erfolgreiche Job-Execution.

6. Best Practices

- Für kritische Jobs separate Benutzer + minimalberechtigte `sudoers`-Einträge (`NOPASSWD:/usr/bin/apt-get`).
- `MAILFROM` in `/etc/ssmtp/ssmtp.conf` setzen oder `mail -r`.
- Cron-Umgebung bewusst: Pfad = `PATH=/usr/sbin:/usr/bin:/sbin:/bin`; explizite Pfade in Skripten verwenden.
- `systemd timer` statt `cron` für komplexe Abhängigkeiten (`After=network-online.target`).
- Kombinieren Sie `journalctl -f` mit `multitail` oder `lnav` für Live-Monitoring.

Nach dieser Session können die Teilnehmenden Logdateien interpretieren, Rotation implementieren, periodische Wartungsaufgaben per Cron oder At planen und den Erfolg automatisiert per E-Mail überwachen.

3.10. 14:45–15:00 Kurzpause

3.11. 15:00–16:00 Abschlussprojekt “Frischinstallation”

Zum Abschluss bündeln wir alle bisherigen Themen in einem kompletten Systemaufbau: Eine **10 GB große virtuelle Disk** wird manuell partitioniert, mit einem Minimal-Paketsatz bespielt, der Bootloader konfiguriert und anschließend sicherheitsgehärtet. Gearbeitet wird in Zweier-Teams, jede Gruppe präsentiert das Ergebnis per Screen-Share.

1. Projektziel

Ein schlankes, reproduzierbares Linux-Grundsystem, das

- im UEFI-Modus startet,
- eine klare Partitionierung aufweist (**EFI, / root, swap**),
- nur essenzielle Pakete enthält (»Netinst«-Ansatz)
- und direkt nach dem Boot elementare Hardening-Maßnahmen aktiviert.

2. Vorbereitungen

1. **Virtuelle Maschine** anlegen (QEMU/KVM oder VirtualBox, 1 vCPU, 1 GiB RAM, 10 GB VirtIO-Disk).
2. **Install-ISO** (aktuelles Debian netinst oder Ubuntu Server $<1GB$) einbinden.
3. **Firmware** → OVMF/EDK II für UEFI einstellen, Secure-Boot optional aktivieren.

3. Manuelle Partitionierung

Mount-Punkt	Typ	Größe
ESP (`/boot/efi`)	FAT32, GPT-Typ EF00	512 MiB
`/` (root)	ext4 oder xfs	8,5 GiB
swap	linux-swap	1 GiB

Hinweis: Die ESP muss **Boot + ESP**-Flag tragen; alle Partitionen auf 1 MiB-Alignment achten (Parted → `mkpart ... 1MiB`).

4. Minimal-Paketsatz

- **Basis**: `openssh-server`, `sudo`, `curl`, `vim-tiny`, `ca-certificates`.
- **Abwählen**: *standard-system-utilities*, *popularity-contest*, *games-commons*.
- Debian: Task-Auswahl `dialog` → nur »SSH server« markieren.

Netzinst trick

- **Debian**: `d-i pkgsel/include string` → `openssh-server sudo vim-tiny`.
- **Ubuntu**: `d-i pkgsel/standard boolean false` → `pkgsel/include string openssh-server sudo vim-tiny`.

Hinweis: Die `pkgsel`-Zeilen sind in der `preseed.cfg` zu setzen, die Sie im `netinst`-Verzeichnis ablegen.

```
# Preseed-Minimal
d-i pkgsel/include string openssh-server
    sudo vim-tiny
d-i pkgsel/standard   boolean false
```

5. Bootloader einrichten (GRUB-UEFI)

6. Security-Hardening (Baseline)

- **Unprivilegierte Benutzer anlegen** (`adduser alice`, `usermod -aG sudo alice`).
- **SSH**: Passwort-Login deaktivieren, starke Kex/Algorithms:

```
Match all
  PasswordAuthentication no
  KexAlgorithms curve25519-sha256@libssh
      .org
```

- **Firewall**: `ufw default deny incoming`, `ufw allow 22/tcp`, `ufw enable`.
- **Automatic Updates**: `unattended-upgrades` aktivieren, Mail-Report an Admin.
- **sudoers**: Timestamp-Timeout 0, `requiretty` (Option in `/etc/sudoers.d/hardening`):

```
Defaults timestamp_timeout=0
Defaults !visiblepw
```

7. Snapshot erstellen (Rollback-Punkt)

1. Virtuelle Maschine herunterfahren.
2. Im Hypervisor GUI »Create Snapshot → baseline«.
3. Beschreibung: `Minimal 10 GB UEFI Hardening`.

8. Team-Präsentation

Jede Gruppe zeigt in 3 - 4 Minuten:

- Partitionstabelle (`lsblk -f`).
- Boot-Zeit (`systemd-analyze`).
- Auszug aus `/etc/fstab` + `/etc/default/grub`.
- Nachweis der Hardening-Schritte (`sudo -l`, `ufw status`, `ssh -v`).

9. Diskussionspunkte

1. Reichen 1 GiB Swap → Hibernation vs. Server-Use-Case?
2. `xfs` vs. `ext4` für Root – Vor- / Nachteile.
3. Secure Boot aktivieren – Aufwand vs. Benefit?
4. Automatisierte Builds (PXE, Cloud-Init) für größere Umgebungen.

10. Ergebnis

Nach erfolgreichem Durchlauf haben die Teilnehmenden:

- ein voll funktionsfähiges, gehärtetes System in ~45 min erstellt,
- alle Kernkomponenten (Partitioning, GRUB, Paketwahl, Security) praktisch angewandt,
- einen Snapshot als Rollback-Punkt (Grundlage für weiteres Experimentieren).

Damit endet das Kursprogramm – das erworbene Wissen deckt den gesamten Lebenszyklus eines Linux-Servers ab, von der ersten Partition bis zum produktionsreife Hardening.