Einführung in Linux für Lernende Informatik

Zielgruppe: Lernende Informatik EFZ in den Fachrichtungen Plattformentwicklung und Applikationsentwicklung.

Dauer: 6 bis 10 Lektionen, je nach Tiefe und Übungsanteil.

Ausgangslage

Linux ist in beiden Fachrichtungen relevant:

- Plattformentwicklung: Serverbetrieb, Benutzer- und Rechteverwaltung, Netzwerkdienste, Automatisierung, Monitoring und Troubleshooting.
- Applikationsentwicklung: Entwicklungsumgebungen, Git, Paketmanager, Shell, Container, Deployment und Betrieb von Anwendungen.

Diese Einführung soll nicht nur Befehle vermitteln, sondern ein grundlegendes Verständnis für das Arbeiten in einer Linux-Umgebung schaffen.

Lernziele

Die Lernenden können nach der Einführung:

- erklären, was Linux ist und wo es eingesetzt wird.
- sich in einer Linux-Shell orientieren.
- Dateien und Verzeichnisse sicher erstellen, verschieben, kopieren, suchen und löschen.
- Dateirechte und Besitzverhältnisse lesen und grundlegend anpassen.
- einfache Textdateien im Terminal bearbeiten.
- Programme mit einem Paketmanager installieren und entfernen.
- Prozesse anzeigen, starten und beenden.
- einfache Netzwerkdiagnosen durchführen.
- grundlegende Shell-Skripte lesen und schreiben.
- typische Unterschiede zwischen Windows und Linux benennen.
- Linux als Arbeitsumgebung für Entwicklung oder Betrieb einordnen.

Voraussetzungen

Empfohlen:

- Grundkenntnisse im Umgang mit einem Computer.
- Erste Erfahrung mit Dateisystemen und Pfaden.
- Optional: Grundkenntnisse in Git oder Programmierung.

Vorwissen für Docker

Docker baut stark auf Linux-Grundlagen auf. Die Lernenden müssen vor einer Docker-Einführung nicht Linux-Profis sein, sollten aber einige Konzepte praktisch angewendet haben. Dieses Vorwissen kann direkt in dieser Linux-Einführung aufgebaut werden.

Muss-Vorwissen

Die Lernenden sollten vor Docker sicher können:

- sich in einer Shell orientieren.
- Dateien und Verzeichnisse erstellen, anzeigen, kopieren, verschieben und löschen.
- absolute und relative Pfade unterscheiden.
- einfache Textdateien mit einem Editor wie `nano` oder VS Code bearbeiten.

- Befehle mit Optionen und Argumenten lesen.
- Paketlisten aktualisieren und Pakete installieren.
- Prozesse anzeigen und beenden.
- Umgebungsvariablen anzeigen und setzen.
- grundlegende Netzwerkbegriffe wie IP-Adresse, Port, DNS und `localhost` erklären.
- einfache Netzwerkdiagnosen mit `ping`, `ip`, `ss` oder `curl` durchführen.
- Git-Repositories klonen und Projektdateien im Terminal öffnen.

Besonders wichtige Begriffe für Docker

Diese Begriffe sollten vor oder während der Docker-Einführung nochmals gezielt aufgegriffen werden:

- Prozess: Ein Container startet normalerweise einen Hauptprozess.
- Dateisystem: Images und Container enthalten eigene Dateisysteme.
- Pfad: Volumes und Bind Mounts verbinden Host-Pfade mit Container-Pfaden.
- Port: Mit Portweiterleitungen wird ein Dienst im Container vom Host erreichbar.
- Netzwerk: Container können über Docker-Netzwerke miteinander kommunizieren.
- DNS-Name: In Docker Compose können Services über ihren Servicenamen erreicht werden.
- Umgebungsvariable: Viele Images werden über Environment Variables konfiguriert.
- Paketmanager: Ein `Dockerfile` installiert Abhängigkeiten oft mit `apt`, `npm`, `pip` oder ähnlichen Werkzeugen.
- Benutzerrechte: Container laufen mit einem Benutzer; Root im Container ist nicht gleichbedeutend mit sicherem Betrieb.
- Logausgabe: Container-Logs sind für Fehlersuche zentral.

Sinnvolle Reihenfolge

Bevor Docker eingeführt wird, sollten mindestens diese Linux-Module behandelt sein:

1. Shell-Grundlagen
2. Dateisystem und Pfade
3. Dateien anzeigen und bearbeiten
4. Paketverwaltung
5. Prozesse
6. Netzwerkgrundlagen
7. Entwicklungswerkzeuge mit Git und Umgebungsvariablen

Docker kann danach als nächster Schritt eingeführt werden. Für Applikationsentwickler ist besonders die Verbindung von App, Datenbank, Ports und Umgebungsvariablen wichtig. Für Plattformentwickler sind Prozesse, Netzwerke, Volumes, Logs und Betriebsaspekte besonders wichtig.

Mini-Check vor Docker

Die Lernenden sollten diese Aufgaben ohne Schritt-für-Schritt-Anleitung lösen können:

```
pwd
mkdir docker-vorwissen
cd docker-vorwissen
echo "Hallo Docker" > index.html
cat index.html
env | head
ip addr
ss -tulpen
curl http://example.com
git --version
```

Wenn diese Aufgaben verstanden werden, ist der Einstieg in Docker deutlich einfacher.

Vorgaben zur Arbeitsumgebung

Empfohlener Einstieg

Für den Beginn dieser Einführung wird WSL unter Windows empfohlen. Damit können die Lernenden schnell und mit wenig technischem Vorbereitungsaufwand mit Linux arbeiten. Die ersten Module konzentrieren sich auf Shell, Dateisystem, Dateien, Rechte, Paketverwaltung, Git und einfache Skripte. Für diese Themen ist WSL gut geeignet.

Die Debian-VM bleibt die Referenzumgebung für servernahe Themen. Sobald Dienste, SSH, Netzwerk, Firewall, Logs, systemd oder Betriebsszenarien behandelt werden, sollte auf eine echte Debian-VM gewechselt werden. Das ist besonders für Lernende in der Plattformentwicklung wichtig.

Kurzempfehlung:

- Start: WSL 2 mit Debian oder Ubuntu.
- Referenz: Debian Stable als VM.
- Vertiefung Plattformentwicklung: Debian-VM lokal oder auf Proxmox.
- Vertiefung Applikationsentwicklung: WSL mit Visual Studio Code, später optional VM für Deployment- und Serverübungen.

Empfohlene Linux-Distribution

Als fachliche Referenzdistribution wird Debian Stable empfohlen.

Empfehlung:

- Distribution: Debian Stable
- Variante für WSL: Debian oder Ubuntu
- Variante für VM: Debian ohne grafische Oberfläche, also als Serverinstallation
- Architektur: 64 Bit
- Benutzer: normaler Benutzer mit `sudo`-Rechten
- Paketmanager: `apt`
- Shell: `bash`

Warum Debian:

- Debian ist stabil, weit verbreitet und gut dokumentiert.
- Viele Serverumgebungen basieren direkt oder indirekt auf Debian.
- Die Lernenden arbeiten mit einer realistischen Serverumgebung.
- Die Befehle sind fast identisch zu Ubuntu, das in Entwicklung und Cloud ebenfalls häufig vorkommt.
- Debian eignet sich gut für Plattform- und Applikationsentwicklung.

Falls unter WSL Ubuntu statt Debian eingesetzt wird, kann die Einführung fast unverändert verwendet werden. Wichtig ist, dass die Lehrperson klar benennt, welche Umgebung für welche Aufgabe verwendet wird. Für die ersten Module sollte möglichst dieselbe WSL-Distribution verwendet werden, damit Befehle, Paketnamen und Fehlermeldungen vergleichbar bleiben.

Variante A: WSL unter Windows

WSL ist die einfachste Startvariante, wenn die Lernenden Windows-Notebooks verwenden und schnell mit Linux-Befehlen arbeiten sollen.

Empfehlung:

- WSL 2 verwenden.
- Debian oder Ubuntu aus dem Microsoft Store installieren.

- Für diese Einführung bevorzugt Debian verwenden, wenn es verfügbar ist.
- Visual Studio Code mit WSL-Erweiterung für Applikationsentwicklung einsetzen.

Installation unter Windows:

```
wsl --install -d Debian
```

Falls Debian nicht verfügbar ist:

```
wsl --list --online
wsl --install -d Ubuntu
```

Nach der Installation in WSL:

```
sudo apt update
sudo apt upgrade
sudo apt install curl wget nano tree htop dnsutils git
```

Vorteile:

- sehr schnell eingerichtet
- keine separate VM-Verwaltung notwendig
- ideal für Shell, Git, Scripting und Entwicklung
- gute Integration mit Visual Studio Code
- geringe Einstiegshürde für die erste Lektion

Grenzen:

- nicht alle Server- und Netzwerkaspekte sind identisch zu einer echten VM
- systemd, Dienste, Netzwerk und Firewall können sich anders verhalten
- weniger geeignet für realistische Plattform- und Betriebsübungen

Empfohlene Nutzung:

- erste Lektionen für alle Lernenden
- Applikationsentwicklung mit Git, Skripten und lokalen Entwicklungswerkzeugen
- Wiederholung und individülles Üben ausserhalb des Unterrichts

Variante B: Debian als virtülle Maschine

Diese Variante ist die beste Wahl, wenn die Lernenden eine möglichst realistische Linux-Serverumgebung kennenlernen sollen.

Einsatzmöglichkeiten:

- lokal auf dem Notebook oder PC mit VirtualBox, VMware Workstation oder Hyper-V
- zentral auf einem vorhandenen Proxmox-Server

Empfohlene VM-Ressourcen:

Ressource	Empfehlung
CPU	1 bis 2 vCPU
RAM	2 GB
Festplatte	20 GB
Netzwerk	NAT für lokale VM, Bridge bei Proxmox
Betriebssystem	Debian Stable ohne GUI

Vorteile:

- echtes Linux-System mit eigenem Bootprozess
- geeignet für Dienste wie SSH, Webserver, Firewall, Logs und Benutzerverwaltung

- gute Vorbereitung auf Server- und Cloudumgebungen
- ideal für Plattformentwicklung

Nachteile:

- etwas mehr Vorbereitungsaufwand
- VM muss installiert, gepflegt und bei Problemen repariert werden
- lokale Virtualisierung kann je nach Gerät eingeschränkt sein

Empfohlene Nutzung:

- Plattformentwicklung: ab den servernahen Modulen bevorzugt Debian-VM verwenden.
- Applikationsentwicklung: Debian-VM verwenden, wenn Serverbetrieb, Deployment oder SSH geübt werden soll.
- Gemischte Klassen: mit WSL starten und für ausgewählte Serverübungen auf VM wechseln.

Variante C: Debian-VM auf Proxmox

Diese Variante ist besonders sinnvoll, wenn bereits ein Proxmox-Server vorhanden ist.

Empfohlener Ablauf:

1. Debian-ISO auf Proxmox hochladen.
2. VM-Vorlage oder einzelne VM erstellen.
3. Pro Lernende oder Lerngruppe eine eigene VM bereitstellen.
4. Netzwerk per Bridge anbinden, damit SSH-Zugriff möglich ist.
5. Einen normalen Benutzer mit `sudo`-Rechten erstellen.
6. SSH-Zugriff testen.

Empfohlene Basiskonfiguration:

```
sudo apt update
sudo apt upgrade
sudo apt install sudo openssh-server curl wget nano tree htop dnsutils
sudo systemctl enable ssh
sudo systemctl start ssh
```

Vorteile:

- zentrale Verwaltung durch die Lehrperson
- Snapshots vor riskanten Übungen möglich
- einheitliche Umgebung für alle Lernenden
- sehr gut für Server-, Netzwerk- und Dienstübungen

Hinweise:

- Vor grösseren Übungen einen Snapshot erstellen.
- VM-Namen klar strukturieren, zum Beispiel `linux-lernende-01`.
- IP-Adressen oder DNS-Namen dokumentieren.
- Zugriff per SSH von den Clients testen.

Entscheidungsempfehlung

Situation	Empfehlung
Erste Lektionen für alle Lernenden	WSL mit Debian oder Ubuntu
Fokus Applikationsentwicklung	WSL mit Debian oder Ubuntu
Fokus Plattformentwicklung	Start mit WSL, danach Debian-VM auf Proxmox oder lokal

Situation	Empfehlung
Gemischte Klasse	WSL als Einstieg, Debian-VM als Referenz für Serverübungen
Wenig Vorbereitungszeit	WSL
Realistische Serverübungen	Debian-VM lokal oder auf Proxmox
Zentrale Kontrolle und Snapshots	Proxmox

Pragmatische Empfehlung für diesen Kurs:

1. Mit WSL starten, damit alle Lernenden schnell arbeitsfähig sind.
2. Debian Stable als Referenzsystem für die Kursunterlagen definieren.
3. Für Applikationsentwicklung WSL mit Visual Studio Code als Hauptumgebung verwenden.
4. Für Plattformentwicklung ab den servernahen Themen auf Debian-VM wechseln.
5. Wenn Proxmox vorhanden ist, pro Lernende oder Lerngruppe eine Debian-VM bereitstellen.
6. Bei Aufgaben zu Diensten, SSH, Netzwerk und Logs klar markieren, ob sie in WSL oder nur in einer VM sinnvoll funktionieren.

Minimaler Start für die erste Lektion

Vor der ersten Lektion sollte jede lernende Person Zugriff auf eine der folgenden Umgebungen haben:

- bevorzugt WSL mit Debian/Ubuntu und funktionierendem Terminal
- alternativ Debian-VM mit Benutzerkonto und `sudo`-Rechten

Kurzer Funktionstest:

```
whoami
pwd
ls
sudo apt update
```

Wenn diese Befehle funktionieren, ist die Umgebung für die ersten Module bereit.

Optional für Applikationsentwicklung:

- Visual Studio Code installieren.
- Erweiterung `WSL` oder `Remote - SSH` installieren.
- Git installieren und Zugriff auf ein Git-Repository testen.

Modul 1: Was ist Linux?

Inhalte

- Betriebssystem, Kernel, Distribution.
- Unterschied zwischen Linux, Ubuntu, Debian, Red Hat, Arch usw.
- Typische Einsatzgebiete: Server, Cloud, Container, Embedded, Entwicklung.
- Open Source und Paketverwaltung.
- Unterschied CLI und GUI.

Begriffe

- Kernel
- Distribution
- Shell
- Terminal
- Paketmanager

- Root
- Benutzer
- Prozess
- Dienst

Übung

Die Lernenden recherchieren und beantworten kurz:

1. Wo begegnet Linux im Alltag oder Beruf?
2. Warum wird Linux häufig auf Servern eingesetzt?
3. Was ist der Unterschied zwischen Linux und Ubuntu?

Modul 2: Erste Schritte in der Shell

Inhalte

- Terminal öffnen.
- Prompt verstehen.
- Befehlsstruktur: Befehl, Optionen, Argumente.
- Hilfe anzeigen mit `man`, `--help` und `-h`.
- Wichtige Tastenkombinationen.

Befehle

```
whoami
pwd
ls
cd
clear
history
man ls
```

Übung

1. Aktüllen Benutzer anzeigen.
2. Aktülles Verzeichnis anzeigen.
3. Inhalt des Home-Verzeichnisses anzeigen.
4. In ein anderes Verzeichnis wechseln.
5. Hilfe zum Befehl `ls` anzeigen.

Bezug zu Docker

Docker wird im Alltag fast immer über die Kommandozeile bedient. Die Lernenden müssen Befehle wie `docker run`, `docker ps`, `docker logs` oder `docker compose up` lesen können. Besonders wichtig ist das Verständnis von Befehlsstruktur, Optionen und Argumenten.

Modul 3: Dateisystem und Navigation

Inhalte

- Linux-Dateisystemhierarchie.
- Absolute und relative Pfade.
- Wichtige Verzeichnisse: `/home`, `/etc`, `/var`, `/tmp`, `/usr`, `/bin`, `/opt`.
- Versteckte Dateien.

- Gross- und Kleinschreibung.

Befehle

```
ls -la
cd /etc
cd ~
tree
find
```

Falls `tree` nicht installiert ist:

```
sudo apt update
sudo apt install tree
```

Übung

1. In das Home-Verzeichnis wechseln.
2. Versteckte Dateien anzeigen.
3. Den Unterschied zwischen `.` und `..` erklären.
4. Eine Datei namens `Test.txt` und eine Datei namens `test.txt` erstellen und vergleichen.

Bezug zu Docker

Docker arbeitet mit Pfaden auf zwei Ebenen: auf dem Host und im Container. Für Volumes und Bind Mounts müssen die Lernenden verstehen, welcher Pfad zum eigenen System gehört und welcher Pfad im Container liegt. Auch der Build-Kontext bei `docker build .` basiert auf dem aktuellen Verzeichnis.

Modul 4: Dateien und Verzeichnisse bearbeiten

Inhalte

- Dateien erstellen, anzeigen, kopieren, verschieben und löschen.
- Verzeichnisse erstellen und entfernen.
- Textausgabe und Umleitungen.
- Unterschied zwischen > und >>.

Befehle

```
touch
mkdir
cp
mv
rm
rmdir
cat
less
head
tail
echo
```

Übung

```
mkdir linux-übung
cd linux-übung
```

```
touch notizen.txt
echo "Meine erste Linux-Datei" > notizen.txt
cat notizen.txt
cp notizen.txt kopie.txt
mv kopie.txt backup.txt
ls -l
```

Zusatz:

1. Drei Unterordner `docs`, `scripts` und `logs` erstellen.
2. Eine Datei in jeden Ordner legen.
3. Eine Datei umbenennen.
4. Eine Datei sicher löschen.

Bezug zu Docker

Beim Erstellen eigener Images werden Dateien aus dem Projektverzeichnis in ein Image kopiert. Befehle wie `COPY`, `ADD` und `.dockerignore` setzen voraus, dass die Lernenden Dateien, Ordner und relative Pfade sicher verstehen. Auch Compose-Projekte bestehen aus mehreren Textdateien und Verzeichnissen.

Modul 5: Textdateien bearbeiten

Inhalte

- Konfigurationsdateien als einfache Textdateien.
- Editoren im Terminal.
- Minimaler Umgang mit `nano`.
- Optional: Einstieg in `vim`.

Befehle

```
nano datei.txt
cat datei.txt
less datei.txt
grep Suchbegriff datei.txt
```

Übung

1. Datei `profil.txt` erstellen.
2. Name, Lehrberuf und Fachrichtung eintragen.
3. Datei speichern und im Terminal anzeigen.
4. Mit `grep` nach der Fachrichtung suchen.

Bezug zu Docker

Docker-Konfiguration ist textbasiert. Die wichtigsten Dateien sind `Dockerfile`, `compose.yaml`, `.env` und oft Konfigurationsdateien der Anwendung. Die Lernenden sollten solche Dateien lesen, gezielt ändern und Fehlermeldungen auf Zeilennummern beziehen können.

Modul 6: Benutzer, Gruppen und Rechte

Inhalte

- Benutzer und Gruppen.
- Besitzer, Gruppe, Andere.

- Lesen, Schreiben, Ausführen.
- Rechteanzeige mit `ls -l`.
- Grundlagen von `chmod`, `chown` und `sudo`.
- Warum man nicht dafürhaft als root arbeitet.

Befehle

```
ls -l
id
groups
chmod
chown
sudo
```

Beispiel

```
echo "echo Hallo Linux" > script.sh
ls -l script.sh
chmod +x script.sh
./script.sh
```

Übung

1. Eine Datei erstellen.
2. Rechte mit `ls -l` analysieren.
3. Eine Skriptdatei ausführbar machen.
4. Erklären, was `rwxr-xr--` bedeutet.

Bezug zu Docker

Container haben eigene Benutzer- und Rechtekontexte. Bei Volumes können Dateirechte sichtbar werden, wenn ein Container Dateien schreibt und der Host sie danach lesen oder bearbeiten soll. Für Dockerfiles ist wichtig, warum Anwendungen nicht unnötig als `root` laufen sollten.

Modul 7: Paketverwaltung

Inhalte

- Warum Software unter Linux meist über Paketmanager installiert wird.
- Paketqüllen.
- Aktualisieren, Suchen, Installieren, Entfernen.
- Unterschied zwischen `apt update` und `apt upgrade`.

Befehle für Debian/Ubuntu

```
sudo apt update
sudo apt upgrade
apt search htop
sudo apt install htop
htop
sudo apt remove htop
```

Übung

1. Paketliste aktualisieren.
2. Ein Paket suchen.
3. `htop` installieren.
4. Laufende Prozesse mit `htop` betrachten.
5. `htop` wieder entfernen.

Bezug zu Docker

Viele Dockerfiles installieren Programme und Bibliotheken mit Paketmanagern. Befehle wie `apt update` und `apt install` tauchen deshalb direkt in Dockerfiles auf. Die Lernenden sollten verstehen, dass ein Image dadurch reproduzierbar aufgebaut wird und dass installierte Pakete Teil des Images werden.

Modul 8: Prozesse und Dienste

Inhalte

- Was ist ein Prozess?
- Prozesse anzeigen.
- Prozesse beenden.
- Hintergrundprozesse.
- Dienste mit `systemctl`.

Befehle

```
ps aux
top
htop
kill
systemctl status
systemctl list-units --type=service
```

Übung

1. Aktive Prozesse anzeigen.
2. Einen Prozess suchen.
3. Einen einfachen Hintergrundprozess starten:

```
sleep 300 &
ps aux | grep sleep
kill <PID>
```

4. Status eines Dienstes anzeigen, zum Beispiel:

```
systemctl status ssh
```

Bezug zu Docker

Ein Container ist eng mit Prozessen verbunden: Meist läuft darin genau ein Hauptprozess. Wenn dieser Prozess endet, stoppt auch der Container. Befehle wie `docker ps`, `docker logs` und `docker stop` sind deshalb leichter verständlich, wenn Prozesse, Hintergrundprozesse und Dienststatus bekannt sind.

Modul 9: Netzwerkgrundlagen unter Linux

Inhalte

- IP-Adresse anzeigen.
- Erreichbarkeit testen.
- DNS-Auflösung prüfen.
- Ports und Verbindungen.
- SSH als wichtiges Werkzeug.

Befehle

```
ip addr
ping
nslookup
ss -tulpen
ssh
scp
```

Falls `nslookup` fehlt:

```
sudo apt install dnsutils
```

Übung

1. Eigene IP-Adresse anzeigen.
2. Einen Host anpingen.
3. DNS-Auflösung für `example.com` prüfen.
4. Offene Ports anzeigen.
5. Diskutieren: Warum ist SSH für Plattform- und Applikationsentwicklung wichtig?

Bezug zu Docker

Docker verwendet eigene Netzwerke für Container. Ports müssen bewusst vom Container auf den Host veröffentlicht werden, zum Beispiel `8080:80`. In Docker Compose erreichen sich Services über DNS-Namen wie `postgres` oder `web`. Darum sind `localhost`, Ports, DNS und offene Verbindungen zentrale Grundlagen.

Modul 10: Shell-Scripting

Inhalte

- Shell-Skripte als Automatisierung.
- Shebang.
- Variablen.
- Einfache Bedingungen.
- Ausführrechte.

Beispiel

```
#!/bin/bash

echo "Hallo $USER"
echo "Aktülles Verzeichnis: $(pwd)"
echo "Heute ist: $(date)"
```

Übung

1. Datei `info.sh` erstellen.

2. Beispielskript einfügen.
3. Datei ausführbar machen.
4. Skript starten.

```
chmod +x info.sh
./info.sh
```

Zusatz für Fortgeschrittene

Ein Skript schreiben, das:

- einen Ordnernamen als Argument annimmt.
- prüft, ob der Ordner existiert.
- falls nicht, den Ordner erstellt.
- eine Logdatei mit Datum anlegt.

Bezug zu Docker

Shell-Skripte werden häufig für lokale Entwicklungsabläufe, Build-Schritte oder Einstiegspunkte in Containern verwendet. Auch einfache Automatisierung rund um `docker compose up`, Tests, Logs und Aufräumen basiert oft auf Shell-Befehlen.

Vertiefung für Plattformentwicklung

Themen

- Benutzer und Gruppen verwalten.
- SSH-Server konfigurieren.
- Dienste installieren und mit `systemctl` verwalten.
- Logs in `/var/log` analysieren.
- Firewall-Grundlagen mit `ufw`.
- Cronjobs.
- Speicherplatz und Dateisysteme.

Befehle

```
sudo adduser lernende
sudo usermod -aG sudo lernende
systemctl status ssh
journalctl -xe
df -h
du -sh *
crontab -e
sudo ufw status
```

Projektidee

Die Lernenden richten eine kleine Linux-VM als Server ein:

1. SSH aktivieren.
2. Neün Benutzer erstellen.
3. Webserver installieren.
4. Beispielseite bereitstellen.
5. Dienststatus prüfen.
6. Logs analysieren.

Beispiel mit Nginx:

```
sudo apt update
sudo apt install nginx
systemctl status nginx
curl http://localhost
```

Bezug zu Docker

Für Plattformentwicklung ist Docker eine weitere Betriebsform für Dienste. Die Linux-Grundlagen bleiben relevant: Ports müssen erreichbar sein, Logs müssen gelesen werden, Speicher muss geplant werden und Dienste müssen überwacht werden. Container ersetzen diese Betriebskenntnisse nicht, sondern setzen sie voraus.

Vertiefung für Applikationsentwicklung

Themen

- Entwicklungsumgebung unter Linux.
- Git und SSH-Keys.
- Programmiersprachen installieren.
- Abhängigkeiten verwalten.
- Umgebungsvariablen.
- Anwendungen starten und Logs lesen.
- Grundlagen von Docker oder Containern.

Befehle

```
git --version
ssh-keygen
ssh -T git@github.com
node --version
python3 --version
pip --version
env
export APP_ENV=development
```

Projektidee

Die Lernenden starten eine kleine Webanwendung unter Linux:

1. Git-Repository klonen.
2. Abhängigkeiten installieren.
3. Anwendung starten.
4. Port prüfen.
5. Logausgabe interpretieren.
6. Anwendung stoppen.

Beispiel mit Python:

```
mkdir webdemo
cd webdemo
echo "Hello from Linux" > index.html
python3 -m http.server 8000
```

In einem zweiten Terminal:

```
curl http://localhost:8000
```

Bezug zu Docker

Für Applikationsentwicklung ist Docker vor allem eine reproduzierbare Entwicklungsumgebung. Eine Anwendung, ihre Datenbank und Zusatzdienste können mit Docker Compose gemeinsam gestartet werden. Die hier geübten Themen Git, Abhängigkeiten, Umgebungsvariablen, Ports und Logs werden dabei direkt wiederverwendet.

Gemeinsame Abschlussaufgabe

Die Lernenden erstellen eine kurze Dokumentation mit folgenden Punkten:

1. Welche Linux-Distribution wurde verwendet?
2. Welche Befehle wurden gelernt?
3. Welche Aufgabe war einfach?
4. Welche Aufgabe war schwierig?
5. Wo wird Linux in der eigenen Fachrichtung eingesetzt?
6. Ein eigenes kleines Shell-Skript mit Erklärung.

Bewertungsvorschlag

Kriterium	Gewichtung
Orientierung im Dateisystem	20 Prozent
Sicherer Umgang mit Dateien und Rechten	20 Prozent
Verständnis von Prozessen, Paketen und Netzwerk	20 Prozent
Qualität der praktischen Aufgabe	25 Prozent
Dokumentation und Reflexion	15 Prozent

Kurzer Befehls-Spickzettel

Zweck	Befehl
Aktülles Verzeichnis anzeigen	`pwd`
Dateien anzeigen	`ls -la`
Verzeichnis wechseln	`cd pfad`
Datei erstellen	`touch datei.txt`
Ordner erstellen	`mkdir ordner`
Datei kopieren	`cp qülle ziel`
Datei verschieben	`mv qülle ziel`
Datei löschen	`rm datei`
Datei anzeigen	`cat datei`
Datei durchsuchen	`grep suchbegriff datei`
Hilfe anzeigen	`man befehl`
Rechte anzeigen	`ls -l`
Rechte ändern	`chmod +x script.sh`
Paketliste aktualisieren	`sudo apt update`
Paket installieren	`sudo apt install paket`
Prozesse anzeigen	`ps aux`
Netzwerk anzeigen	`ip addr`
Verbindung testen	`ping ziel`

Hinweise für die Durchführung

- Befehle zürst gemeinsam demonstrieren, danach selbstständig üben lassen.
- Fehler bewusst zulassen und als Troubleshooting-Anlass verwenden.
- Bei Löschbefehlen wie `rm` klare Sicherheitsregeln vereinbaren.
- Lernende sollen Befehle nicht nur kopieren, sondern kurz erklären können.
- Plattformlernende sollten früh mit SSH, Diensten und Logs arbeiten.
- Applikationslernende sollten früh mit Git, Entwicklungsservern und Umgebungsvariablen arbeiten.