Linux Kompakt



Vom Einstieg zur Systemadministration

Branislav Zeljak (BZ)

Version 0.1 - 10.2026

Versionshistorie

Version	Datum	Autor(en)	Änderungen
0.1	10.2026	BZ	Creation
		BZ	Basis: tuxcademy-Unterlagen (CC BY-SA 4.0)
0.1	10.2026	BZ	Final version

Inhaltsverzeichnis

Einleitung 1

I. Grundlagen 3

1. Einführung in Linux 4
- 1.1. Was ist Linux? [E] . . . 4
- 1.2. Die Geschichte von Linux [E] . . . 4
- 1.3. Freie Software, Open Source und die GPL [E] . . . 5
- 1.4. Der Kernel [F] . . . 5
- 1.5. Linux-Distributionen [E] . . . 6
- 1.6. Linux Standard und Verzeichnisstruktur [X] . . . 6

2. Erste Schritte in der Shell 7
- 2.1. Anmelden und abmelden [E] . . . 7
- 2.2. Aufbau von Kommandos [E] . . . 8
- 2.3. Arten von Kommandos [E] . . . 8
- 2.4. Komfort in der Shell [E] . . . 9
- 2.5. Hilfe finden [E] . . . 9
- 2.6. Administrative Rechte erlangen [F] . . . 10

3. Hilfe und Dokumentation 11
- 3.1. Manpages richtig lesen [E] . . . 11
- 3.2. Aufbau einer Manpage [E] . . . 12
- 3.3. Weitere Hilfequellen am Rechner [E] 12
- 3.4. Manpages durchsuchen und exportieren [F] . . . 13
- 3.5. Hilfe im Netz und in der Gemeinschaft [E] . . . 13
- 3.6. Gute Fragen stellen [E] . . . 14

4. Texteditoren 16
- 4.1. nano [E] . . . 16
- 4.2. vi und vim [E] . . . 16
  - 4.2.1. Tipps für den Alltag [F] . . . 17
- 4.3. Standard-Editor und emacs [X] . . . 18

5. vim vertieft 19
- 5.1. Das Denkmodell: Operator, Bewegung, Textobjekt [F] . . . 19
- 5.2. Register, Marken und Sprünge [F] . 20
- 5.3. Makros [F] . . . 20
- 5.4. Suchen und Ersetzen [F] . . . 21
- 5.5. Mehrere Dateien: Puffer, Fenster, Tabs [F] . . . 21
- 5.6. Die Konfigurationsdatei .vimrc [F] . . 22
- 5.7. Plugins und Erweiterungen [X] . . . 22

6. Das Dateisystem 24
- 6.1. Grundbegriffe [E] . . . 24
- 6.2. Dateitypen [E] . . . 24
- 6.3. Der Verzeichnisbaum nach FHS [E] . 25
- 6.4. Pseudo-Dateisysteme [F] . . . 25
- 6.5. Dateisysteme und Einhängen im Überblick [F] . . . 26
- 6.6. Wo liegt was auf Ubuntu, RHEL und openSUSE? [X] . . . 26

7. Dateien und Verzeichnisse verwalten 27
- 7.1. Pfade und Navigation [E] . . . 27
- 7.2. Verzeichnisse anlegen und löschen [E] 28
- 7.3. Dateien anlegen, kopieren, verschieben, löschen [E] . . . 28
- 7.4. Dateinamen-Muster (Globbing) [E] . 28
- 7.5. Links [F] . . . 29
- 7.6. Dateien suchen [F] . . . 29
- 7.7. Dateiinhalt anzeigen [E] . . . 29

8. Ein- und Ausgabe, Pipes und Filter 30
- 8.1. Standardkanäle [E] . . . 30
- 8.2. Pipes [E] . . . 31
- 8.3. Dateien lesen und zusammenfügen [E] . . . 31
- 8.4. Text sortieren und zusammenfassen [E] . . . 31
- 8.5. Spalten schneiden, ersetzen, verbinden [F] . . . 32
- 8.6. xargs und Kommandosubstitution [F] 32
- 8.7. Dateien vergleichen [F] . . . 32

9. Reguläre Ausdrücke, grep, sed und awk 33
- 9.1. Reguläre Ausdrücke [E] . . . 33
- 9.2. grep: Text durchsuchen [E] . . . 33
- 9.3. sed: der Stromeditor [F] . . . 34
- 9.4. awk: Spalten verarbeiten [F] . . . 34
- 9.5. Praxisbeispiele [X] . . . 35

10. Textverarbeitung in der Praxis 36
- 10.1. Filterketten entwickeln [F] . . . 36
- 10.2. awk als Programmiersprache [F] . . 37
- 10.3. sed für Änderungen in Dateien [F] . 38
- 10.4. Weitere Werkzeuge [F] . . . 38
- 10.5. JSON mit jq [X] . . . 39
- 10.6. Beispiel: Loganalyse [F] . . . 39

11. Die Shell im Detail 41
- 11.1. Shell-Varianten [E] . . . 41

11.2. Variablen und Umgebung [E] 41
11.3. Quoting: Sonderzeichen schützen [E] 42
11.4. Kommandos verknüpfen [E] 42
11.5. History, Alias und Vervollständigung [E] 42
11.6. Startdateien der bash [F] 43
11.7. Jobs: Vordergrund und Hintergrund [F] 43
11.8. Shell-Optionen und Erweiterungen [X] 43

12. Arbeiten im Terminal: tmux, Locales und Zeit 44
12.1. Terminal-Multiplexer tmux [E] 44
12.2. Locales und Spracheinstellungen [E] 45
12.3. Zeichensätze und Kodierung [F] . . . 46
12.4. Zeitzone und Uhrzeit [E] 46

13. Shellskripte 48
13.1. Das erste Skript [E] 48
13.2. Parameter und Variablen [E] 48
13.3. Verzweigungen [E] 49
13.4. Schleifen [E] 50
13.5. Funktionen [F] 50
13.6. Rückgabewerte und Fehlerbehandlung [F] 51
13.7. Beispiel: Sicherungsskript [F] 51
13.8. Debugging und gute Praxis [X] . . . 51

14. Shellskripte vertieft 52
14.1. Das Skriptgerüst [F] 52
14.2. Funktionen [F] 53
14.3. Arrays und assoziative Arrays [F] . . 53
14.4. Parameter-Expansion [F] 54
14.5. Optionen mit getopts [F] 54
14.6. Here-Documents und Here-Strings [F] 55
14.7. Fehlerbehandlung und trap [F] . . . 55
14.8. Debugging [F] 56
14.9. Eine praktische Beispielfunktion: Prozesse parallel ausführen [X] . . . 56

15. Archivieren und Komprimieren 58
15.1. tar [E] 58
15.2. gzip, bzip2 und xz [E] 59
15.3. zip und unzip [E] 59
15.4. Weitere Werkzeuge [F] 59

16. Versionsverwaltung mit Git 60
16.1. Warum Versionsverwaltung? [E] . . . 60
16.2. Erste Schritte [E] 60
16.3. Branches und Merge [E] 61
16.4. Remote-Repositories [E] 62
16.5. Konflikte lösen [F] 62
16.6. Änderungen rückgängig machen [F] 63
16.7. Tags, Suche und nützliche Hilfsmittel [F] 63
16.8. Typische Workflows [F] 63

17. Software aus Quellen und Programmierwerkzeuge 65
17.1. Warum aus Quellen bauen? [F] . . . 65
17.2. Die Werkzeugkette [F] 65
17.3. Der klassische Ablauf: configure, make, make install [F] 66
17.4. Makefiles [F] 67
17.5. Autotools, CMake und andere Bausysteme [X] 67
17.6. Pakete selbst bauen [X] 68
17.7. Werkzeuge zur Fehleranalyse [X] . . 68

II. Systemadministration 70

18. Systemadministration 71
18.1. Das root-Konto und Administratorrechte [E] 71
18.1.1. sudo einrichten [F] 71
18.2. Konfiguration und Systeminformationen [E] 72
18.3. Sprache, Zeichensatz und Zeit [F] . . 72
18.4. Kernelparameter und sysctl [X] . . . 73

19. Paketverwaltung 74
19.1. Grundlagen: Pakete und Repositories [E] 74
19.2. Alltägliche Befehle im Vergleich [E] . 75
19.3. Die Niederlevel-Werkzeuge dpkg und rpm [F] 76
19.4. Informationen über installierte Pakete [E] 76
19.5. Repositories konfigurieren [E] 77
19.6. Pakete herunterladen und inspizieren [E] 78
19.7. Abhängigkeitsprobleme lösen [F] . . 79
19.8. Weitere Paketsysteme: Snap, Flatpak, AppImage [F] 79
19.9. Aktualisierungsstrategien [F] 79
19.10. Eigene Pakete und Repositories [X] . 80

20. Benutzer und Gruppen 82
20.1. Grundlagen [E] 82
20.2. Die Benutzerdatenbanken [E] 83
20.3. Benutzerkonten verwalten [E] 83
20.3.1. Komfort unter Debian und Ubuntu [E] 84
20.4. Gruppen verwalten [E] 84
20.5. Gemeinsame Verzeichnisse [F] . . . 84

20.6. Passwortrichtlinien und Anmeldung [F] . . . 84
20.7. Zentrale Benutzerverwaltung [X] . . 85

21. PAM, zentrale Anmeldung und LDAP 86
21.1. Wie funktioniert die Anmeldung? [F] 86
21.2. PAM im Detail [F] . . . 87
21.3. Passwortrichtlinien und Sperren [F] 88
21.4. sudo im Detail [F] . . . 88
21.5. Zwei-Faktor-Anmeldung per TOTP [X] 89
21.6. LDAP-Verzeichnisdienste [X] . . . 89
21.7. SSSD: Anbindung an LDAP und Active Directory [X] . . . 90

22. Zugriffsrechte 92
22.1. Das Rechtekonzept [E] . . . 92
22.2. Rechte ändern: chmod [E] . . . 93
22.3. Standardrechte: umask [E] . . . 93
22.4. Besondere Rechte: SUID, SGID und Sticky-Bit [F] . . . 93
22.5. Access Control Lists (ACL) [F] . . . 94
22.6. Dateiattribute, Capabilities und Mandatory Access Control [X] . . . 95

23. SELinux, AppArmor und Auditing 96
23.1. Discretionary und Mandatory Access Control [X] . . . 96
23.2. SELinux [X] . . . 96
23.3. AppArmor [X] . . . 98
23.4. Das Audit-Framework [X] . . . 98
23.5. Härtung von Diensten [X] . . . 99

24. Prozessverwaltung 101
24.1. Was ist ein Prozess? [E] . . . 101
24.2. Prozesse anzeigen [E] . . . 101
24.2.1. Die Systemlast [F] . . . 102
24.3. Prozesse beenden: Signale [E] . . . 102
24.4. Prioritäten: nice und renice [F] . . . 103
24.5. Offene Dateien und Ressourcen [F] . 103
24.6. cgroups, OOM-Killer und Dienste im Hintergrund [X] . . . 103

25. Kernel, Module und Hardware 104
25.1. Aufgaben des Kernels [F] . . . 104
25.2. Kernelmodule [F] . . . 104
25.3. Kernelparameter zur Laufzeit: sysctl [F] . . . 105
25.4. Das initramfs [F] . . . 106
25.5. udev: Geräteverwaltung [F] . . . 106
25.6. Hardware-Informationen [E] . . . 107
25.7. Kernel installieren, wechseln und selbst bauen [X] . . . 107

26. Platten und Partitionen 110
26.1. Massenspeicher unter Linux [E] . . . 110
26.2. Partitionierung: MBR und GPT [E] . . 111
26.3. Partitionieren mit fdisk, gdisk und parted [E] . . . 111
26.4. Loop-Geräte als Übungsumgebung [X] 112

27. Software-RAID mit mdadm 113
27.1. Wozu RAID? [E] . . . 113
27.2. RAID-Level im Überblick [E] . . . 114
27.3. RAID mit mdadm anlegen [F] . . . 114
27.4. Status und Überwachung [F] . . . 115
27.5. Plattenausfall und Austausch [F] . . 116
27.6. Verbund stoppen und wieder zusammensetzen [F] . . . 117
27.7. Erweitern und umbauen [X] . . . 117
27.8. RAID mit LVM, LUKS und beim Systemstart [X] . . . 118
27.9. Grenzen und Planung [F] . . . 118

28. LVM und Verschlüsselung 121
28.1. Warum LVM? [F] . . . 121
28.2. LVM einrichten [F] . . . 122
28.3. Volumes vergrößern und verkleinern [F] . . . 122
28.4. Snapshots und Thin Provisioning [F] 123
28.5. RAID in LVM: Spiegelung und Parität [F] . . . 123
28.6. RAID-Volumes umwandeln, reparieren und prüfen [X] . . . 124
28.7. Verschlüsselung mit LUKS [F] . . . 125
28.8. SMART und Gesundheit der Platten [F] 125

29. Dateisysteme und Swap 127
29.1. Linux-Dateisysteme im Überblick [E] 127
29.2. Dateisysteme anlegen und prüfen [E] 127
29.3. Einhängen und Aushängen [E] . . . 128
29.4. Dauerhaft einhängen: /etc/fstab [E] 128
29.5. Swap [E] . . . 129
29.6. Belegung, Inodes und Quotas [F] . . 129
29.7. dd, Abbilder und Datenrettung [X] . 130

30. Moderne Dateisysteme: Btrfs und ZFS 131
30.1. Warum neue Dateisysteme? [F] . . . 131
30.2. Btrfs [F] . . . 132
30.3. ZFS [X] . . . 133
30.4. XFS und ext4: Pflege im Alltag [F] . . 134

31. Der Bootvorgang 136
31.1. Vom Einschalten zur Anmeldung [E] 136
31.2. Informationen zum Start [E] . . . 136
31.3. GRUB 2 [F] . . . 137

31.4. Probleme beim Start beheben [F] . . 137
31.4.1. Passwort von root zurücksetzen [X] 138
31.5. Herunterfahren und Neustart [E] . . 138

32. systemd **139**
32.1. Was ist systemd? [E] 139
32.2. Dienste mit systemctl steuern [E] . . 139
32.3. Unit-Typen und Dateien [F] 140
32.4. Eigene Service-Unit [F] 140
32.5. Absicherung und Ressourcenbegrenzung [X] 141
32.6. Weitere Werkzeuge [F] 141

33. Container-Grundlagen: Namespaces und cgroups **142**
33.1. Container sind Prozesse [X] 142
33.2. Namespaces [X] 143
33.3. cgroups: Ressourcen begrenzen [X] 144
33.4. chroot, systemd-nspawn und LXC [X] 144
33.5. Sicherheitsmechanismen [X] 145

34. Docker und Podman **147**
34.1. Konzepte [F] 147
34.2. Container starten und verwalten [F] 148
34.3. Images [F] 148
34.4. Eigene Images mit dem Dockerfile [F] 149
34.5. Volumes und Netzwerke [F] 150
34.6. Docker Compose [F] 150
34.7. Podman und rootless [F] 151
34.8. Aufräumen und Betrieb [X] 152

35. Virtualisierung mit KVM und libvirt **154**
35.1. Virtualisierung im Überblick [F] . . . 154
35.2. Virtuelle Maschinen anlegen [F] . . . 155
35.3. VMs steuern mit virsh [F] 155
35.4. Speicher für VMs [F] 156
35.5. Netzwerk für VMs [F] 157
35.6. Verwaltungsoberflächen [F] 157

36. Zeitgesteuerte Aufgaben **159**
36.1. Einmalige Aufgaben mit at [E] 159
36.2. Wiederkehrende Aufgaben mit cron [E] 159
36.3. systemd-Timer [F] 160

37. Protokollierung **162**
37.1. Wo finde ich Logs? [E] 162
37.2. journalctl: das Journal lesen [E] . . . 163
37.3. journald dauerhaft speichern und konfigurieren [F] 163
37.4. rsyslog [F] 164
37.5. Logrotation mit logrotate [F] 164
37.6. Logs auswerten [X] 165

III. Netzwerk **166**

38. TCP/IP-Grundlagen **167**
38.1. Schichtenmodell [E] 167
38.2. Adressen im lokalen Netz [E] 167
38.3. IPv4-Adressen und Netzmasken [E] . 168
38.4. Routing und Gateway [E] 168
38.5. TCP, UDP und Ports [E] 169
38.6. IPv6 [F] 169
38.7. Namensauflösung mit DNS [E] . . . 170
38.8. Nützliche Kennzahlen und Details [X] 170

39. Netzwerk konfigurieren **171**
39.1. Netzwerkstatus mit ip [E] 171
39.2. Werkzeuge zur Netzkonfiguration [E] 171
39.3. Hostname und Namensauflösung [E] 172
39.4. Netplan (Ubuntu) [F] 172
39.5. NetworkManager und nmcli [F] . . . 173
39.6. DHCP [F] 173
39.7. Firewall im Überblick [F] 173
39.8. Fehlersuche im Netzwerk [E] 174
39.8.1. Pakete mitschneiden mit tcpdump [X] 175
39.8.2. Routing und Weiterleitung aktivieren [X] 175

40. Netzwerkdienste **176**
40.1. SSH – sicher auf entfernten Rechnern arbeiten [E] 176
40.2. Anmeldung mit Schlüsseln [E] 176
40.3. Dateien übertragen: scp und rsync [E] 177
40.4. Weitere Dienste im Überblick [F] . . 177
40.5. Grundschablone: einen Dienst in Betrieb nehmen [F] 178

41. Netzwerkwerkzeuge in der Praxis **179**
41.1. Vorgehen: Schicht für Schicht [E] . . 179
41.2. Schnittstellen, Adressen und Routen: ip [E] 179
41.3. Erreichbarkeit und Weg: ping, traceroute, mtr [E] 180
41.4. Sockets und Ports: ss, lsof, nc [E] . . 180
41.5. HTTP testen mit curl [F] 181
41.6. Ports und Dienste erkunden: nmap [F] 181
41.7. Mitschneiden mit tcpdump und Wireshark [F] 181
41.8. Bandbreite und Qualität messen: iperf3 [X] 182

42. SSH im Detail **184**
42.1. Grundlagen [E] 184
42.2. Schlüsselbasierte Anmeldung [E] . . 184
42.3. Der Client: `~/.ssh/config` [F] 185

42.4. Dateien übertragen: sftp und sshfs [F] 186
42.5. Der Server: sshd und Härtung [F] . . 186
42.6. Tunnel und Portweiterleitung [F] . . 187
42.7. Zertifikate, Mehrfaktor und Automatisierung [X] 187

43. Firewall: nftables, iptables, ufw und firewalld 189
43.1. Wozu eine Firewall? [E] 189
43.2. Der Weg eines Pakets durch Netfilter [F] 189
43.3. nftables [F] 190
43.4. NAT und Portweiterleitung [F] 192
43.5. iptables (Altsystem) [F] 193
43.6. Frontends: ufw und firewalld [F] . . 193
43.7. Protokollierung, Tests und Härtung [X] 194

44. DNS-Server: BIND, Unbound und dnsmasq 196
44.1. Wie DNS funktioniert [E] 196
44.2. Abfragen mit dig, host und resolvectl [E] 197
44.3. Unbound als Cache-Resolver [F] . . . 197
44.4. dnsmasq für kleine Netze [F] 198
44.5. Autoritativer Server mit BIND 9 [X] . 198
44.6. Systematische Fehlersuche [F] 200

45. DHCP und Zeitsynchronisation 202
45.1. DHCP: Adressen automatisch vergeben [E] 202
45.2. DHCP-Server mit dnsmasq [F] 203
45.3. DHCP-Server mit Kea [F] 203
45.4. Warum die Uhrzeit wichtig ist [E] . . 204
45.5. chrony und systemd-timesyncd [F] . 204
45.6. Zeitserver im eigenen Netz und Virtualisierung [X] 205

46. Webserver: nginx und Apache 207
46.1. HTTP in Kürze [E] 207
46.2. nginx installieren und konfigurieren [E] 207
46.3. Apache HTTP Server [F] 208
46.4. HTTPS und Let's Encrypt [F] 209
46.5. Reverse-Proxy [F] 210
46.6. Zugriffsschutz, Protokolle und Leistung [X] 210

47. Proxy und Lastverteilung 213
47.1. Proxy-Arten [F] 213
47.2. HAProxy als Load Balancer [F] 213
47.3. TCP-Lastverteilung [F] 215
47.4. Ausfallsicherheit des Load Balancers [F] 215
47.5. Squid als Forward-Proxy [F] 215

48. VPN: WireGuard und OpenVPN 217
48.1. Wozu ein VPN? [E] 217
48.2. WireGuard: Grundprinzip [F] 218
48.3. Beispiel: Roadwarrior und Server [F] 218
48.4. Site-to-Site mit WireGuard [F] 219
48.5. OpenVPN im Überblick [F] 219
48.6. Fehlersuche und Sicherheit [X] . . . 220

49. Dateidienste: NFS und Samba 222
49.1. Überblick [E] 222
49.2. NFS: Server [F] 222
49.3. NFS: Client [F] 223
49.4. Samba: Freigaben für Windows [F] . 223
49.5. Samba-Freigaben unter Linux einhängen [F] 224
49.6. Active Directory, ACLs und Sicherheit [X] 225

50. E-Mail-Server: Postfix und Dovecot 227
50.1. Der Weg einer E-Mail [E] 227
50.2. Ein SMTP-Dialog von Hand [E] 228
50.3. Postfix einrichten [F] 228
50.4. Warteschlange und Fehlersuche [F] . 229
50.5. Dovecot: IMAP-Zugriff [F] 229
50.6. SPF, DKIM, DMARC und Betrieb [X] . 230

51. Datenbanken: MariaDB und PostgreSQL 233
51.1. Relationale Datenbanken und SQL [E] 233
51.2. MariaDB [F] 234
51.3. PostgreSQL [F] 234
51.4. Sicherung und Wiederherstellung [F] 235
51.5. Betrieb, Leistung und Replikation [X] 235

IV. Sicherheit und Betrieb 238

52. Sicherheit und Härtung 239
52.1. Grundprinzipien [E] 239
52.2. Updates einspielen [E] 239
52.3. Angriffsfläche prüfen [E] 240
52.4. Benutzer und Anmeldung absichern [F] 240
52.5. Brute-Force-Schutz mit fail2ban [F] . 240
52.6. Zwingende Zugriffskontrolle: SELinux und AppArmor [X] 241
52.7. Weitere Werkzeuge [X] 241

53. Datensicherung 242
53.1. Warum und was sichern? [E] 242
53.2. Sicherungsarten [E] 242
53.3. Sichern mit tar und rsync [E] 243
53.4. Skript und Zeitsteuerung [F] 243
53.5. Spezialwerkzeuge [F] 243

53.6. Konsistente Sicherungen und Abbilder [X] . . . 244

54. Leistung und Fehlersuche 245
54.1. Systemlast auf einen Blick [E] . . . 245
54.2. Systematisches Vorgehen [E] . . . 245
54.3. Leistungsanalyse [F] . . . 246
54.4. Offene Dateien und Verbindungen [F] 246
54.4.1. Ressourcenlimits [F] . . . 247
54.5. Systemaufrufe und Kernel-Werkzeuge [X] . . . 247

55. Fehlersuche an Praxisszenarien 248
55.1. Methodik [E] . . . 248
55.2. Szenario 1: Das System bootet nicht [F] . . . 249
55.3. Szenario 2: Die Platte ist voll [F] . . . 249
55.4. Szenario 3: Ein Dienst startet nicht [F] 250
55.5. Szenario 4: Netzwerk nicht erreichbar [F] . . . 250
55.6. Szenario 5: Hohe Last [F] . . . 251
55.7. Szenario 6: Speicher knapp und OOM-Killer [F] . . . 251
55.8. Szenario 7: Anmeldung nicht möglich [F] . . . 252
55.9. Szenario 8: Zertifikat abgelaufen und Weiteres [X] . . . 252

56. Automatisierung mit Ansible 254
56.1. Warum Automatisierung? [E] . . . 254
56.2. Installation, Inventar und Ad-hoc-Befehle [E] . . . 254
56.3. Playbooks [F] . . . 255
56.4. Variablen, Bedingungen, Schleifen und Templates [F] . . . 257
56.5. Rollen [F] . . . 258
56.6. Vault, Tests und Betrieb [X] . . . 259

57. Monitoring und Alarmierung 261
57.1. Wozu Monitoring? [E] . . . 261
57.2. Prometheus und node_exporter [F] 262
57.3. Alarmregeln und Alertmanager [F] . 263
57.4. Grafana [F] . . . 264
57.5. Klassisch: Nagios, Icinga und Zabbix [F] . . . 264
57.6. Zentrale Logs und Betrieb [X] . . . 265

58. TLS, Zertifikate und eigene PKI 267
58.1. Grundlagen der Kryptografie [E] . . 267
58.2. TLS und X.509-Zertifikate [E] . . . 268
58.3. Schlüssel, Zertifikatsanforderung und selbstsigniertes Zertifikat [F] . . 269
58.4. Eigene Zertifizierungsstelle [F] . . . 269
58.5. Let's Encrypt und ACME [F] . . . 270
58.6. Betrieb: Konfiguration, Überwachung und Fehlersuche [X] . . . 270
58.7. Dateien verschlüsseln und signieren mit GnuPG [F] . . . 271

59. Performance-Tuning 273
59.1. Grundsätze [F] . . . 273
59.2. CPU und Prozesse [F] . . . 274
59.3. Speicher [F] . . . 274
59.4. Datenträger und Dateisysteme [F] . 275
59.5. Netzwerk [F] . . . 276
59.6. Werkzeuge zur Gesamtbetrachtung [X] . . . 276

60. Hochverfügbarkeit und Cluster 279
60.1. Verfügbarkeit und Redundanz [E] . . 279
60.2. Einfache Lösung: virtuelle IP mit Keepalived [F] . . . 280
60.3. Cluster mit Pacemaker und Corosync [X] . . . 281
60.4. Replikation: DRBD und Datenbanken [X] . . . 282
60.5. Planung, Tests und Grenzen [F] . . . 282

61. Cloud, cloud-init und Infrastruktur als Code 284
61.1. Cloud-Grundlagen [F] . . . 284
61.2. Images und Erstkonfiguration mit cloud-init [F] . . . 285
61.3. Infrastruktur als Code mit Terraform [F] . . . 286
61.4. Kubernetes im Überblick [X] . . . 287
61.5. Kosten, Sicherheit und Betrieb [F] . 288

62. Skriptprojekte aus der Praxis 291
62.1. Strukturvorlage für robuste Skripte [F] 291
62.2. Projekt 1: Verzeichnis sichern mit Rotation [F] . . . 292
62.3. Projekt 2: Platten- und Dienstwächter [F] . . . 293
62.4. Projekt 3: Aufräumen und Benutzeranlage [F] . . . 294
62.5. Projekt 4: Logdateien auswerten [F] 294
62.6. Qualität: ShellCheck, Tests und Zeitsteuerung [X] . . . 295

63. Zertifizierungen und Lernpfade 297
63.1. Warum zertifizieren? [E] . . . 297
63.2. Zuordnung der Prüfungsthemen zu den Kapiteln [F] . . . 298
63.3. Lernstrategie und Übungsumgebung [F] . . . 298
63.4. Prüfungsvorbereitung [F] . . . 299

63.5. Weiterführende Themen und Karrierepfade [X] 299

A. Befehlsübersicht 301

B. Übungen mit Lösungen 303

C. Projektaufgaben mit Lösungswegen 309

D. Lernkontrolle 314

E. Nachschlagetabellen 317

F. Glossar 321

G. Quellen und Lizenz 325

H. Abkürzungen 326

Linux Kompakt

Dieser Workshop führt in kompakter Form vom ersten Login auf der Kommandozeile bis zur Administration eines Linux-Servers. Er richtet sich an **ambitionierte Einsteiger**: Vorkenntnisse in einem anderen Betriebssystem werden vorausgesetzt, Linux-Kenntnisse nicht. Die Inhalte orientieren sich an den Themen der Linux-Essentials- und LPIC-1-Prüfungen und sind auf die gängigen Distributionen **Ubuntu LTS**, **Debian**, **Red Hat Enterprise Linux** (und Derivate wie Rocky Linux) sowie **openSUSE** abgestimmt.

Aufbau

- **Teil Grundlagen**: Hilfesystem, Shell und Terminal, Editoren (`vi`/`vim`), Dateisystem, Dateien, Filter, reguläre Ausdrücke, `sed`, `awk`, Textverarbeitung in der Praxis, Shellskripte, Archive, Git und Build-Werkzeuge.
- **Teil Systemadministration**: Pakete, Benutzer, PAM und LDAP, Rechte, SELinux, Prozesse, Kernel, Platten, RAID und LVM, Dateisysteme (auch Btrfs und ZFS), Bootvorgang, `systemd`, Container, Docker, Virtualisierung, Zeitsteuerung und Protokollierung.
- **Teil Netzwerk**: TCP/IP-Grundlagen, Netzkonfiguration, Netzwerkwerkzeuge, SSH, Firewall, DNS-, DHCP-, NTP-, Web-, Proxy-, Mail-, Samba-/NFS- und Datenbankserver sowie VPN.
- **Teil Sicherheit und Betrieb**: Härtung, Datensicherung, Fehlersuche und Praxisszenarien, Ansible, Monitoring, TLS und PKI, Performance-Tuning, Hochverfügbarkeit, Cloud und Infrastruktur als Code, Skriptprojekte sowie Zertifizierungen.
- **Anhang**: Befehlsübersicht, Übungen und Projektaufgaben mit Lösungen, Lernkontrolle, Nachschlagetabellen, Glossar, Lizenz und Quellen, Abkürzungsverzeichnis.
- **Kapitelkopf**: Jedes Kapitel beginnt mit einer Infobox, die Inhalt, Stufe, Lernziele und Vorkenntnisse nennt.

Niveau-Kennzeichnung

Jeder Abschnitt trägt eine Kennzeichnung, die im Inhaltsverzeichnis und unter der Überschrift erscheint. So lässt sich der Stoff je nach Vorwissen auswählen:

Kürzel	Niveau	Bedeutung
[E]	**Einsteiger**	Grundlagen, die jeder Linux-Anwender kennen sollte.
[F]	**Fortgeschritten**	Alltagswissen für Administratoren; setzt die Einsteiger-Abschnitte voraus.
[X]	**Experte**	Vertiefung, Sonderfälle und Hintergründe; beim ersten Durchgang überspringbar.

Empfohlene Lesereihenfolge: zuerst alle Abschnitte [E], dann [F], danach [X].

Schreibweisen

- Ein Prompt mit `$` bedeutet: als normaler Benutzer ausführen. Ein Prompt mit `#` bedeutet: als `root`, in der Praxis meist mit vorangestelltem `sudo`.
- Ausgaben stammen von Ubuntu LTS und können je nach Distribution und Version leicht abweichen. Wo Distributionen sich unterscheiden, sind beide Varianten genannt.
- Platzhalter stehen in spitzen Klammern, zum Beispiel `<datei>`. Optionale Teile stehen in eckigen Klammern.

- Beispiele mit Netzadressen verwenden die für Dokumentation reservierten Bereiche (`192.0.2.0/24`, `example.org`).

> Achtung
> Viele Befehle in diesem Dokument verändern das System (Partitionierung, Benutzer, Rechte). Alle Übungen gehören in eine **virtuelle Maschine** oder auf ein Testsystem, niemals zuerst auf einen Produktivserver. Vor jeder Änderung an wichtigen Konfigurationsdateien eine Sicherheitskopie anlegen.

Quellen

Der Workshop ist eine Zusammenfassung und Überarbeitung der freien Schulungsunterlagen des tuxcademy-Projekts (*Linux-Grundlagen*, *Linux für Fortgeschrittene*, *Linux Essentials*, *Linux kompakt*). Details zu Lizenz und Quellen stehen im Anhang.
Weiterführend: `https://www.tuxcademy.org/` `https://distrowatch.com/`

Teil I.

Grundlagen

1. Einführung in Linux

Inhalt: Geschichte von Unix und Linux, freie Software und Lizenzen, Aufbau eines Linux-Systems, Distributionen und ihre Familien, erste Orientierung auf dem Desktop.

Stufe: [E] Einsteiger

Lernziele: Nach diesem Kapitel können Sie

- die Begriffe Kernel, Distribution und freie Software zu erklären,
- die wichtigsten Distributionsfamilien zu unterscheiden,
- eine passende Distribution für einen Einsatzzweck auszuwählen.

Vorkenntnisse: Keine. Ein Rechner oder eine virtuelle Maschine zum Ausprobieren ist hilfreich.

1.1. Was ist Linux?

Niveau: Einsteiger

Streng genommen ist **Linux** nur der **Kernel**, also der Teil des Betriebssystems, der Hardware, Speicher, Prozesse und Dateisysteme verwaltet. Ein nutzbares System entsteht erst durch die Kombination des Kernels mit den GNU-Werkzeugen (Shell, Compiler, Systemprogramme), einem Init-System, Bibliotheken, Server- und Desktop-Software sowie einer Paketverwaltung. Dieses Gesamtpaket heißt **Distribution**.

Linux läuft auf nahezu jeder Hardware: auf Servern und in der Cloud, auf Smartphones (Android basiert auf dem Linux-Kernel), in Routern, Fernsehern, Autos und auf allen der 500 schnellsten Supercomputer.

1.2. Die Geschichte von Linux

Niveau: Einsteiger

Jahr	Ereignis
1969	Ken Thompson und Dennis Ritchie entwickeln bei den Bell Labs das Betriebssystem **Unix**. Der Name ist ein Wortspiel auf *Multics*.
1983	Richard Stallman startet das **GNU-Projekt**: ein vollständig freies, Unix-kompatibles System. Es fehlt noch ein Kernel.
1991	Linus Torvalds, Student in Helsinki, veröffentlicht die erste Version seines Kernels **Linux** und bittet im Usenet um Rückmeldungen.
1992	Der Kernel wird unter die GPL gestellt. Erste Distributionen entstehen (1993: Debian und Slackware).
1994–96	Kernel 1.0 (1994) und 2.0 (1996); Red Hat Linux und SUSE Linux etablieren sich als kommerzielle Distributionen.
2003	Kernel 2.6 mit vielen Verbesserungen für Server und Desktop.
2011	Kernel 3.0. Seitdem erscheint etwa alle 9–10 Wochen eine neue Version. Kernel 4.0 (2015), 5.0 (2019) und 6.0 (2022) markieren nur Zählsprünge.

1.3. Freie Software, Open Source und die GPL

Niveau: Einsteiger

Freie Software erlaubt vier Freiheiten: das Programm **auszuführen**, den Quelltext zu **studieren und zu ändern**, Kopien **weiterzugeben** und **veränderte Versionen zu veröffentlichen**. Der Begriff *Open Source* betont stärker den offenen Entwicklungsprozess und ist im Alltag meist gleichbedeutend.

Lizenz	Kerneigenschaft
GPL	*Copyleft*: Abgeleitete Werke, die weitergegeben werden, müssen ebenfalls unter der GPL stehen. Gilt für den Kernel (Version 2) und viele GNU-Programme.
LGPL	Wie GPL, erlaubt aber das Binden der Bibliothek in proprietäre Programme.
BSD / MIT	Sehr freizügig. Verlangt im Wesentlichen nur die Nennung der Urheber.
Apache 2.0	Freizügig, mit ausdrücklicher Patentklausel.

> **Wichtig**
>
> Frei bedeutet nicht kostenlos. Freie Software darf verkauft werden (zum Beispiel als Support-Abonnement bei Red Hat oder SUSE), der Quelltext muss aber zugänglich bleiben.

1.4. Der Kernel

Niveau: Fortgeschritten

Der Kernel bildet die Schnittstelle zwischen Programmen und Hardware. Seine wichtigsten Aufgaben:

- **Prozessverwaltung**: Zuteilung der CPU an Prozesse (Scheduler).
- **Speicherverwaltung**: virtueller Speicher, Auslagerung (Swap), Caches.
- **Gerätetreiber**: Zugriff auf Platten, Netzwerkkarten, USB usw. Viele Treiber sind ladbare **Module**.
- **Dateisysteme** und **Netzwerkstack**.
- **Systemaufrufe** (*system calls*) als Schnittstelle für Anwendungsprogramme.

```
$ uname -r                  # laufende Kernelversion
6.8.0-45-generic
$ lsmod | head -3           # geladene Module
$ modinfo ext4              # Informationen zu einem Modul
# modprobe <modul>          # Modul laden (inkl. Abhaengigkeiten)
# modprobe -r <modul>       # Modul entladen
```

Linux unterscheidet **Kernelraum** (privilegiert) und **Benutzerraum** (eingeschränkt). Ein Programm kann Hardware nur über Systemaufrufe ansprechen. Dadurch kann ein fehlerhaftes Anwenderprogramm das System nicht ohne Weiteres zum Absturz bringen.

1.5. Linux-Distributionen

Niveau: Einsteiger

Eine Distribution wählt Kernel, Software, Installationsprogramm, Paketverwaltung und Veröffentlichungszyklus aus. Die wichtigsten Vertreter:

Distribution	Familie	Pakete	Merkmale
Debian	Debian	`.deb`, `apt`	Sehr stabil, community-getragen, Basis vieler Ableger.
Ubuntu (LTS)	Debian	`.deb`, `apt`	Alle zwei Jahre LTS mit fünf Jahren Standard-Support; sehr verbreitet in Cloud und Desktop.
RHEL	Red Hat	`.rpm`, `dnf`	Kommerziell, lange Support-Zyklen, Standard im Unternehmensumfeld.
Rocky / Alma	Red Hat	`.rpm`, `dnf`	Kostenlose, RHEL-kompatible Nachbauten.
Fedora	Red Hat	`.rpm`, `dnf`	Aktuelle Software, Vorläufer von RHEL.
openSUSE / SLES	SUSE	`.rpm`, `zypper`	Werkzeug `YaST`; Leap/SLES stabil, Tumbleweed rollierend.
Arch Linux	Arch	`pacman`	Rollierend, minimalistisch, für Fortgeschrittene.

Die Unterschiede betreffen vor allem Paketverwaltung, Konfigurationswerkzeuge und Pfade. Die Grundlagen (Shell, Dateisystem, Rechte, `systemd`) sind überall gleich. Eine Übersicht bietet `https://distrowatch.com/`.

1.6. Linux Standard und Verzeichnisstruktur

Niveau: Experte

Um Anwendungen distributionsübergreifend lauffähig zu halten, definieren der **Filesystem Hierarchy Standard** (FHS) und früher die *Linux Standard Base* (LSB) einheitliche Pfade. Der FHS wird in Kapitel 6 behandelt.

```
$ cat /etc/os-release       # Distribution und Version
$ hostnamectl               # Host, Kernel, Virtualisierung
$ lsb_release -a            # nur wenn das Paket lsb-release installiert ist
```

2. Erste Schritte in der Shell

Inhalt: Anmelden, Terminal und Shell, Aufbau eines Befehls, Hilfe mit `man`, Tastenkürzel und Verlauf, Ausgaben lesen.

Stufe: [E] Einsteiger

Lernziele: Nach diesem Kapitel können Sie

- ein Terminal zu öffnen und Befehle mit Optionen und Argumenten einzugeben,
- die Hilfesysteme `man`, `--help` und `info` zu nutzen,
- mit Verlauf, Tab-Vervollständigung und Tastenkürzeln effizient zu arbeiten.

Vorkenntnisse: Kapitel 1 oder gleichwertiges Grundwissen.

2.1. Anmelden und abmelden

Niveau: Einsteiger

Linux ist ein Mehrbenutzersystem: Wer arbeiten will, muss sich mit Benutzername und Passwort anmelden. Das geschieht entweder an einem grafischen Anmeldebildschirm, an einer textbasierten Konsole oder über das Netzwerk mit SSH. Auf einem Desktop öffnet man danach ein **Terminal**-Programm. Es startet eine **Shell**, die Kommandos entgegennimmt.
Die Eingabeaufforderung (*Prompt*) sieht typischerweise so aus:

```
tux@kanban:~$           # normaler Benutzer 'tux' auf Rechner 'kanban', Verzeichnis ~ (Heimat
    )
root@kanban:~#          # Administrator
```

Zeichen am Ende: `$` für normale Benutzer, `#` für `root`. Mit `exit` oder `Strg+D` beendet man die Sitzung. Mit `passwd` ändert man das eigene Passwort.

```
$ whoami                # aktueller Benutzername
tux
$ id                    # Benutzer-ID, Gruppen
uid=1000(tux) gid=1000(tux) groups=1000(tux),27(sudo)
$ hostname              # Rechnername
$ passwd                # Passwort aendern
```

> **Wichtig**
>
> Linux unterscheidet **Groß- und Kleinschreibung**: `Datei`, `datei` und `DATEI` sind drei verschiedene Dateien, und `LS` ist kein Kommando. Passworteingaben erscheinen nicht auf dem Bildschirm, auch nicht als Sternchen.

2.2. Aufbau von Kommandos

Niveau: Einsteiger

Ein Kommando besteht aus dem Befehlsnamen, optionalen **Optionen** und **Argumenten**, getrennt durch Leerzeichen:

```
$ ls -l /etc            # Befehl: ls, Option: -l, Argument: /etc
$ ls -a -l -h           # Optionen einzeln
$ ls -alh               # Kurzoptionen zusammengefasst
$ ls --all --human-readable --format=long   # Langoptionen (GNU-Stil)
```

Kurzoptionen beginnen mit einem Minus, Langoptionen (GNU) mit zwei. Das Ende der Optionen markiert `--`; so lassen sich Dateien behandeln, deren Name mit einem Minus beginnt (`rm -- -datei`).

Wichtige einfache Kommandos

Kommando	Wirkung
`date`	Datum und Uhrzeit anzeigen (`date +%F` liefert z. B. 2026-10-14)
`cal`	Kalender des aktuellen Monats
`echo text`	Text ausgeben
`uptime`	Laufzeit und Last des Rechners
`uname -a`	Kernel- und Systeminformationen
`who`, `w`	Angemeldete Benutzer
`clear`	Bildschirm leeren (auch `Strg+L`)
`history`	Zuletzt eingegebene Kommandos

2.3. Arten von Kommandos

Niveau: Einsteiger

Ein Befehl kann ein **Shell-Builtin** (in der Shell eingebaut, z. B. `cd`, `echo`), ein **externes Programm** (eine Datei, z. B. `/bin/ls`), ein **Alias** oder eine **Funktion** sein. Wo die Shell nach externen Programmen sucht, steuert die Variable `PATH`.

```
$ type cd ls echo       # wie wird der Befehl aufgeloest?
cd is a shell builtin
ls is aliased to `ls --color=auto'
echo is a shell builtin
$ which ls              # Pfad des Programms
/usr/bin/ls
$ echo $PATH            # Suchpfad, Verzeichnisse durch ':' getrennt
/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin
```

Programme im aktuellen Verzeichnis werden *nicht* automatisch gefunden. Sie müssen mit `./programm` aufgerufen werden. Das ist eine Sicherheitsmaßnahme.

2.4. Komfort in der Shell

Niveau: Einsteiger

Tastenkombination	Wirkung
`Tab`	Namen von Befehlen und Dateien vervollständigen; doppelt gedrückt: Auswahl anzeigen
↑, ↓	Durch die History blättern
`Strg+R`	Rückwärts in der History suchen
`Strg+A` / `Strg+E`	Zum Zeilenanfang / Zeilenende
`Strg+C`	Laufendes Programm abbrechen
`Strg+D`	Eingabeende (*EOF*); in einer leeren Zeile beendet es die Shell
`Strg+Z`	Programm anhalten (siehe Jobs, Kapitel 11)

2.5. Hilfe finden

Niveau: Einsteiger

Linux bringt seine Dokumentation mit. Wichtigstes Werkzeug ist die **Handbuchseite** (*man page*).

```
$ man ls                # Handbuchseite; q beendet, / sucht, n naechster Treffer
$ man 5 passwd          # Seite aus Abschnitt 5 (Dateiformate)
$ man -k kopieren       # nach Stichwort suchen (wie: apropos kopieren)
$ whatis ls             # Kurzbeschreibung in einer Zeile
$ ls --help             # kompakte Optionsuebersicht
$ info coreutils        # ausfuehrliche GNU-Dokumentation
$ ls /usr/share/doc     # Zusatzdokumentation der Pakete
```

Abschnitt	Inhalt
1	Benutzerkommandos
2	Systemaufrufe des Kernels
3	Bibliotheksfunktionen
4	Gerätedateien und Treiber
5	Dateiformate und Konfigurationsdateien (z. B. `passwd`, `fstab`)
6	Spiele
7	Sonstiges, Konventionen (z. B. `regex`, `systemd.unit`)
8	Administrationskommandos (nur für `root`)

Eine Handbuchseite folgt einem festen Aufbau: `NAME`, `SYNOPSIS` (Aufrufsyntax), `DESCRIPTION`, `OPTIONS`, `EXAMPLES`, `FILES`, `SEE ALSO`. In der `SYNOPSIS` stehen optionale Teile in eckigen Klammern, Wiederholbares mit drei Punkten.

> **Tipp**
>
> Zuerst `--help`, dann `man`. Bei Fehlermeldungen den *genauen* Wortlaut in eine Suchmaschine eingeben. Die Foren der Distributionen und `https://wiki.archlinux.org/` sind auch für andere Distributionen eine ausgezeichnete Quelle.

2.6. Administrative Rechte erlangen

Niveau: Fortgeschritten

Systemverwaltung erfordert die Rechte des Benutzers `root`. Man arbeitet aber nicht dauerhaft als `root`, sondern hebt gezielt die Rechte einzelner Befehle an:

```
$ sudo apt update           # einzelnen Befehl als root (Passwort des EIGENEN Benutzers)
$ sudo -i                   # interaktive root-Shell
$ su -                      # Wechsel zu root (Passwort von root, falls gesetzt)
$ su - anna                 # Wechsel zu anderem Benutzer, mit dessen Umgebung
```

Ubuntu sperrt das `root`-Passwort und nutzt ausschließlich `sudo`; die Mitgliedschaft in der Gruppe `sudo` (Debian/Ubuntu) bzw. `wheel` (RHEL, openSUSE) erlaubt die Nutzung. Details in Kapitel 18.

3. Hilfe und Dokumentation

Inhalt: Manpages und ihre Abschnitte, `info`, `apropos`, Dokumentation unter `/usr/share/doc`, Hilfe im Netz, Fragen richtig stellen.
Stufe: [E] Einsteiger bis [F] Fortgeschritten
Lernziele: Nach diesem Kapitel können Sie

- Manpages gezielt zu finden, zu lesen und zu durchsuchen,
- weitere Dokumentationsquellen zu nutzen,
- Probleme so zu beschreiben, dass andere helfen können.

Vorkenntnisse: Kapitel 2.

3.1. Manpages richtig lesen

Niveau: Einsteiger

Die **Manpages** (*manual pages*) sind die Referenz zu fast jedem Programm, jeder Konfigurationsdatei und jedem Systemaufruf. Sie sind in **Abschnitte** gegliedert:

Nr.	Inhalt	Beispiel
1	Benutzerbefehle	`ls(1)`
2	Systemaufrufe des Kernels	`open(2)`
3	Bibliotheksfunktionen (C)	`printf(3)`
4	Gerätedateien	`null(4)`
5	Dateiformate und Konfigurationsdateien	`passwd(5)`
6	Spiele	`fortune(6)`
7	Verschiedenes, Konzepte	`signal(7)`
8	Administrationsbefehle (meist `root`)	`mount(8)`

Die Schreibweise `passwd(5)` nennt den Abschnitt. Das ist wichtig, weil es zum gleichen Namen mehrere Seiten geben kann: `passwd(1)` beschreibt den Befehl, `passwd(5)` die Datei `/etc/passwd`.

```
$ man ls                        # erste passende Seite
$ man 5 passwd                  # bestimmten Abschnitt waehlen
$ man -k passwd                 # nach Stichwort suchen (wie apropos)
$ apropos "copy file"           # Suche in den Kurzbeschreibungen
$ whatis ls                     # einzeilige Beschreibung
$ man -f passwd                 # alle Seiten mit diesem Namen
$ man -a passwd                 # alle nacheinander anzeigen
$ man -w ls                     # Pfad der Manpage-Datei
$ man -k . | wc -l              # wie viele Seiten gibt es?
```

Beim ersten Aufruf nach der Installation hilft `sudo mandb` (Index aktualisieren). Wenn `man -k` nichts findet, fehlt meist dieser Index.

Navigation im Pager

Manpages werden mit `less` angezeigt:

Taste	Wirkung
`Leertaste`, `b`	Seite vor / zurück
`j`, `k` bzw. Pfeiltasten	Zeile vor / zurück
`g`, `G`	Anfang / Ende
`/muster`, `n`, `N`	Vorwärts suchen, nächster / vorheriger Treffer
`?muster`	Rückwärts suchen
`h`	Hilfe des Pagers
`q`	Beenden

3.2. Aufbau einer Manpage

Niveau: Einsteiger

1. **NAME**: Name und Kurzbeschreibung.
2. **SYNOPSIS**: Aufrufsyntax. Eckige Klammern `[ ]` markieren Optionales, senkrechte Striche Alternativen, `...` Wiederholungen, Fettdruck Wörtliches, Kursivschrift Platzhalter.
3. **DESCRIPTION**: Ausführliche Beschreibung, bei Programmen mit allen Optionen.
4. **OPTIONS**, **ENVIRONMENT**, **FILES**: Optionen, beeinflussende Variablen, verwendete Dateien.
5. **EXIT STATUS**, **EXAMPLES**: Rückgabewerte und Beispiele (nicht immer vorhanden).
6. **SEE ALSO**: Verwandte Seiten – oft der beste Weg zu weiterführender Information.

> **Tipp**
> Beginnen Sie bei langen Manpages mit `SYNOPSIS` und `EXAMPLES`. Wer eine bestimmte Option sucht, tippt im Pager `/^ *-r`, um Zeilen zu finden, die mit der Option beginnen. Ein Aufruf wie `man ssh_config` (Konfiguration) statt `man ssh` (Programm) ist oft der entscheidende Unterschied.

3.3. Weitere Hilfequellen am Rechner

Niveau: Einsteiger

```
$ ls --help | less                        # kurze Optionsuebersicht (fast jedes GNU-Programm)
$ info coreutils                          # ausfuehrliche GNU-Handbuecher
                                            (Navigation: n, p, u, Enter, q)
$ type -a ls                              # Alias, Builtin oder Programm?
$ help cd                                 # Hilfe zu Shell-Builtins
$ ls /usr/share/doc/bash                  # Paketdokumentation: README, Changelogs, Beispiele
$ dpkg -L openssh-server | grep -i doc    # Debian: Dateien eines Pakets
$ rpm -qd openssh-server                  # RHEL: Dokumentationsdateien
$ tldr tar                                # praxisnahe Kurzbeispiele (Paket tldr)
$ cheat tar                               # aehnlich (Paket cheat)
```

Viele Programme liefern Beispielkonfigurationen mit: `/usr/share/doc/<Paket>/examples/` oder Dateien mit Endung `.sample`/`.example` im Konfigurationsverzeichnis. Das ist oft der schnellste Weg zu einer lauffähigen Konfiguration.

3.4. Manpages durchsuchen und exportieren

Niveau: Fortgeschritten

```
$ man --pager='less -p "^OPTIONS"' tar   # direkt zu einem Abschnitt springen
$ man ls | col -b | grep -A2 -- '--all'  # Formatierung entfernen, Option suchen
$ man -t ls > ls.ps                      # als PostScript ausgeben
$ man -t ls | ps2pdf - ls.pdf            # als PDF (Paket ghostscript)
$ MANWIDTH=80 man ls | cat               # feste Breite, ohne Pager
$ man -l ./meinprogramm.1                # lokale Manpage-Datei ansehen
```

Eine eigene Manpage ist eine Textdatei im `roff`-Format:

```
.TH MEINTOOL 1 "Maerz 2025" "1.0" "Benutzerhandbuch"
.SH NAME
meintool \- sichert Dateien
.SH SYNOPSIS
.B meintool
[\fB\-v\fR] \fIZIEL\fR
.SH DESCRIPTION
.B meintool
kopiert Dateien nach \fIZIEL\fR.
.SH OPTIONS
.TP
.B \-v
Ausfuehrliche Ausgabe.
```

Zum Testen dient `man ./meintool.1`. Installiert wird die Seite unter `/usr/local/share/man/man1/`.

3.5. Hilfe im Netz und in der Gemeinschaft

Niveau: Einsteiger

Quelle	Wofür?
Wikis der Distributionen	Anleitungen und Hintergründe: ArchWiki (auch für andere Distributionen sehr gut), Ubuntu Wiki, Debian Wiki, Red-Hat-Dokumentation
Foren und Q&A-Seiten	Konkrete Probleme: Stack Exchange (Unix & Linux, Server Fault), Ubuntu Forums, Linux-Foren
Mailinglisten und Chats	Projektspezifische Hilfe, Entwicklerkontakt
Projektseiten	Offizielle Handbücher, Release Notes, Bugtracker
kernel.org (Linux Foundation)	Kernel-Dokumentation

Hinweis

Anleitungen aus dem Netz nicht blind übernehmen. Vor dem Ausführen prüfen, was ein Befehl tut (`man`), insbesondere bei `sudo`, `rm`, `dd`, `chmod -R` und bei Einzeilern, die Downloads direkt an `sudo sh` weiterleiten. Auf das Alter der Quelle und die Distribution achten.

3.6. Gute Fragen stellen

Niveau: Einsteiger

Eine gute Problembeschreibung spart allen Beteiligten Zeit:

1. **Umgebung**: Distribution, Version, Kernel (`cat /etc/os-release`, `uname -r`), Hardware/VM.
2. **Ziel**: Was soll erreicht werden – nicht nur, was nicht klappt.
3. **Schritte**: Welche Befehle wurden genau eingegeben?
4. **Fehlermeldung**: Vollständig und als Text (kein Foto), samt Relevantem aus `journalctl` oder Logdateien.
5. **Bisherige Versuche**: Was wurde schon ausprobiert und mit welchem Ergebnis?
6. **Minimalbeispiel**: Das Problem auf das Wesentliche reduzieren.

```
$ cat /etc/os-release | head -3        # Distribution
$ uname -a                             # Kernel und Architektur
$ lsb_release -a 2>/dev/null           # falls installiert
$ journalctl -xe --no-pager | tail -50 # aktuelle Meldungen
```

Passwörter, Schlüssel und interne Hostnamen vor dem Veröffentlichen von Logs entfernen.

Übungen und Kontrollfragen

1. Welche Manpage-Abschnitte gibt es zu `printf`, und wie ruft man die des Shell-Befehls und die der C-Funktion auf?

   Lösung: `man -f printf` zeigt `printf(1)` (Befehl) und `printf(3)` (C-Bibliothek). Aufruf: `man 1 printf` bzw. `man 3 printf`.

2. Sie suchen einen Befehl zum Umbenennen vieler Dateien, kennen aber den Namen nicht. Wie gehen Sie vor?

   Lösung: `apropos rename` oder `man -k rename`; dann die gefundenen Seiten (`rename(1)`, `mv(1)`) ansehen.

3. Was bedeutet `[-v] ZIEL ...` in einer Synopsis?

   Lösung: `-v` ist optional, `ZIEL` ist ein Platzhalter, die Punkte stehen für weitere Argumente derselben Art.

4. Wo finden Sie Beispielkonfigurationen eines installierten Pakets?

   Lösung: Unter `/usr/share/doc/<Paket>/` (Debian oft `examples`); Dateien eines Pakets zeigen `dpkg -L` bzw. `rpm -ql`.

5. Warum soll eine Fehlermeldung als Text und nicht als Foto gepostet werden?

   Lösung: Text ist durchsuchbar, lässt sich kopieren und wird von Suchmaschinen gefunden; Fotos sind schlecht lesbar und verlieren Details.

4. Texteditoren

Inhalt: Bearbeiten von Textdateien im Terminal mit `nano` und `vim`: Modi, Bewegen, Ändern, Suchen, Ersetzen, Konfiguration.

Stufe: [E] Einsteiger bis [F] Fortgeschritten

Lernziele: Nach diesem Kapitel können Sie

- Konfigurationsdateien mit `nano` sicher zu bearbeiten,
- in `vim` zwischen Modi zu wechseln, zu navigieren, zu ändern und zu speichern,
- Suchen und Ersetzen sowie einfache Makros und Einstellungen zu nutzen.

Vorkenntnisse: Grundlegende Shell-Bedienung (Kapitel 2).

Konfigurationsdateien und Skripte sind Textdateien. Ein Administrator muss daher mindestens einen Editor im Terminal beherrschen, denn auf Servern gibt es keine grafische Oberfläche.

4.1. nano

Niveau: Einsteiger

`nano` ist der einfachste Editor im Terminal. Alle wichtigen Befehle stehen am unteren Bildschirmrand (`^` bedeutet `Strg`).

```
$ nano datei.txt          # Datei oeffnen oder neu anlegen
$ sudo nano /etc/hosts    # Systemdatei bearbeiten
$ nano +25 datei.txt      # an Zeile 25 beginnen
```

Tasten	Wirkung
`Strg+O`, `Enter`	Speichern (*Write Out*)
`Strg+X`	Beenden (fragt bei Änderungen nach dem Speichern)
`Strg+W`	Suchen
`Strg+K` / `Strg+U`	Zeile ausschneiden / einfügen
`Strg+_`	Zu einer Zeilennummer springen
`Alt+U`	Rückgängig
`Strg+G`	Hilfe

4.2. vi und vim

Niveau: Einsteiger

Der Editor `vi` (meist als `vim` installiert) ist auf *jedem* Unix- und Linux-System vorhanden, auch im Rettungsmodus. Er wirkt zunächst ungewohnt, weil er **Modi** kennt:

Modus	Zweck
Normalmodus	Beim Start aktiv. Tasten sind *Befehle* (bewegen, löschen, kopieren). Rückkehr jederzeit mit `Esc`.
Einfügemodus	Text wird eingegeben. Aktivierung mit `i`, `a`, `o` u. a.
Kommandozeilenmodus	Beginnt mit `:` im Normalmodus. Speichern, Beenden, Suchen/Ersetzen.

Überlebens-Set

Datei öffnen: `vi datei`. Zum Schreiben `i` drücken. Mit `Esc` zurück in den Normalmodus. Dann `:wq` + `Enter` zum Speichern und Beenden oder `:q!` zum Beenden *ohne* Speichern.

Wichtige Befehle im Normalmodus

Befehl	**Wirkung**
`h j k l` (oder Pfeiltasten)	Links, unten, oben, rechts
`w` / `b`	Wortweise vor / zurück
`0` / `$`	Zeilenanfang / Zeilenende
`gg` / `G`	Dateianfang / Dateiende; `25G` springt zu Zeile 25
`i` / `a`	Einfügen vor / nach dem Cursor
`I` / `A`	Einfügen am Zeilenanfang / -ende
`o` / `O`	Neue Zeile unter / über der aktuellen öffnen
`x`	Zeichen löschen
`dd` / `dw`	Zeile / Wort löschen (wird zwischengespeichert)
`yy` / `p`	Zeile kopieren / einfügen (`P`: davor)
`u` / `Strg+R`	Rückgängig / wiederherstellen
`.`	Letzte Änderung wiederholen
`3dd`, `5yy`	Zahl vor einem Befehl wiederholt ihn
`/muster`, `n`, `N`	Vorwärts suchen, nächster / vorheriger Treffer

Kommandozeilenmodus

```
:w                  speichern
:w neu.txt          unter neuem Namen speichern
:q                  beenden (nur ohne Aenderungen)
:q!                 beenden und Aenderungen verwerfen
:wq   oder  ZZ      speichern und beenden
:set number         Zeilennummern anzeigen
:%s/alt/neu/g       in der ganzen Datei 'alt' durch 'neu' ersetzen
:%s/alt/neu/gc      dto., mit Rueckfrage
:10,20d             Zeilen 10 bis 20 loeschen
:!ls                externes Kommando ausfuehren
:r datei            Inhalt einer anderen Datei einlesen
```

4.2.1. Tipps für den Alltag [F]

- `vimtutor` startet eine interaktive Einführung von etwa 30 Minuten.
- `v` (zeichenweise), `V` (zeilenweise) und `Strg+V` (blockweise) starten den **visuellen Modus** zum Markieren; danach `d`, `y` oder `>` anwenden.
- `ZZ` und `:x` speichern nur bei Änderungen und beenden.
- Mit `:e!` lädt man die zuletzt gespeicherte Fassung neu.
- Dauerhafte Einstellungen stehen in `~/.vimrc`, zum Beispiel `set number`, `syntax on`, `set expandtab`.

4.3. Standard-Editor und emacs

Niveau: Experte

Viele Programme (`crontab -e`, `visudo`, `git commit`) starten den in den Variablen `VISUAL` oder `EDITOR` hinterlegten Editor. Unter Debian und Ubuntu wählt `sudo update-alternatives --config editor` den Systemstandard, `select-editor` den des Benutzers.

```
$ export EDITOR=nano       # nur fuer diese Sitzung
$ echo 'export EDITOR=nano' >> ~/.bashrc    # dauerhaft
```

`emacs` ist ein weiterer, sehr mächtiger Editor, der als programmierbare Umgebung verstanden werden kann (Mail, Kalender, Entwicklung). Grundbefehle: `Strg+X Strg+F` (Datei öffnen), `Strg+X Strg+S` (speichern), `Strg+X Strg+C` (beenden), `Strg+G` (Aktion abbrechen).

5. vim vertieft

Inhalt: Registers, Makros, Marken, Fenster und Tabs, Suchen und Ersetzen mit Mustern, `.vimrc`, nützliche Plugins.

Stufe: [F] Fortgeschritten bis [X] Experte

Lernziele: Nach diesem Kapitel können Sie

- effizient mit Textobjekten, Registern und Makros zu arbeiten,
- mehrere Dateien in Fenstern und Tabs zu bearbeiten,
- `vim` über die Datei `.vimrc` an eigene Bedürfnisse anzupassen.

Vorkenntnisse: Kapitel 4 (Editoren).

5.1. Das Denkmodell: Operator, Bewegung, Textobjekt

Niveau: Fortgeschritten

Der eigentliche Vorteil von `vim` liegt in der Kombinierbarkeit: Ein **Operator** (was) wird mit einer **Bewegung** oder einem **Textobjekt** (worauf) kombiniert, optional mit einem Zähler.

Operator	Bedeutung
`d`	löschen (delete, schneidet in ein Register aus)
`c`	ändern (change = löschen + Einfügemodus)
`y`	kopieren (yank)
`>`, `<`	ein- bzw. ausrücken
`gU`, `gu`	Groß- bzw. Kleinschreibung
`=`	automatisch formatieren/einrücken
`!`	durch einen externen Filter schicken

Bewegung / Objekt	Bedeutung
`w`, `b`, `e`	Wortanfang vor, Wortanfang zurück, Wortende
`0`, `^`, `$`	Zeilenanfang, erstes Zeichen, Zeilenende
`gg`, `G`, `{`, `}`	Dateianfang, Dateiende, Absatz zurück/vor
`f,`, `t,`	bis zum nächsten Komma (einschließlich / davor)
`iw`, `aw`	Wort (inner / mit Leerraum)
`i(`, `a(`	Inhalt einer Klammer (ohne / mit Klammern)
`i[`, `i{`, `i<`	dto. für eckige, geschweifte und spitze Klammern
`ip`, `ap`	Absatz
`is`, `as`	Satz

Beispiele: `dw` löscht ein Wort, `d2w` zwei Wörter, `ci(` ändert den Inhalt der Klammer, in der der Cursor steht, `yap` kopiert den Absatz, `gUiw` macht das Wort groß, `dd` löscht die Zeile (Verdopplung = ganze Zeile), `3dd` drei Zeilen. Der Punkt `.` wiederholt die letzte Änderung – die wohl wichtigste Taste überhaupt.

> **Tipp**
>
> Mit `ct"` wird bis vor das nächste Anführungszeichen geändert, mit `ci"` der Inhalt zwischen zwei Anführungszeichen. Wer `ciw` + Text + Esc mit `w` und `.` kombiniert, ersetzt mehrere gleichartige Wörter ohne Makro.

5.2. Register, Marken und Sprünge

Niveau: Fortgeschritten

Register sind benannte Zwischenablagen. Man spricht sie mit `"` und einem Namen an.

Register	Inhalt
`""`	Standardregister (letztes Löschen/Kopieren)
`"0`	zuletzt Kopiertes (`y`), wird von `d` nicht überschrieben
`"a` … `"z`	Benannte Register; Großbuchstaben hängen an
`"+`, `"*`	Systemzwischenablage (Clipboard)
`"_`	Schwarzes Loch: löscht, ohne Register zu ändern
`".`, `":`, `"/`	Zuletzt eingefügter Text, letzter Befehl, letztes Suchmuster
`"%`	Aktueller Dateiname

```
"ayy        Zeile in Register a kopieren
"Ayy        Zeile an Register a anhaengen
"ap         Register a einfuegen
"+yy        in die Systemzwischenablage kopieren
:registers  alle Register anzeigen
Ctrl-r a    im Einfuege- oder Kommandomodus Register a einfuegen
```

Marken merken sich Positionen: `ma` setzt Marke `a`, `'a` springt in deren Zeile, `` `a `` (Backtick) genau an die Position. Großbuchstaben (`mA`) gelten dateiübergreifend. Hilfreiche Sprünge: `Ctrl-o`/`Ctrl-i` (Sprungliste zurück/vor), `` `` `` (zurück zur letzten Position), `gi` (zur letzten Einfügeposition), `%` (zur passenden Klammer), `*` und `#` (nächstes/vorheriges Vorkommen des Worts unter dem Cursor).

5.3. Makros

Niveau: Fortgeschritten

Ein Makro ist eine aufgezeichnete Tastenfolge in einem Register:

```
qa          Aufzeichnung in Register a starten
 ...        Tasten eingeben (z. B. I// <Esc> j)
q           Aufzeichnung beenden
@a          Makro einmal abspielen
@@          zuletzt gespieltes Makro wiederholen
20@a        20-mal abspielen
:%normal @a Makro auf jede Zeile anwenden
```

Beispiel: In einer Liste von Zeilen `name=wert` soll aus jeder Zeile `export NAME="wert"` werden. Makro: `qa0iexport <Esc>f=r"A"<Esc>jq`, anschließend `100@a`. Ein Makro sollte stets mit einer Bewegung enden, die es auf die nächste Zeile bringt, damit die Wiederholung funktioniert.

5.4. Suchen und Ersetzen

Niveau: Fortgeschritten

```
:s/alt/neu/          erste Ersetzung in der Zeile
:s/alt/neu/g         alle Ersetzungen in der Zeile
:%s/alt/neu/g        ganze Datei
:%s/alt/neu/gc       mit Rueckfrage (y/n/a/q/l)
:5,12s/alt/neu/g     Zeilen 5 bis 12
:'<,'>s/alt/neu/g    markierter Bereich (Visual-Modus)
:%s/\s\+$//e         Leerraum am Zeilenende entfernen
:%s/\(\w\+\)@\(\w\+\)/\2 at \1/g   Gruppen mit \1, \2
:%s/\v(\w+)@(\w+)/\2 at \1/g       "very magic": einfachere Syntax
:g/muster/d          alle Zeilen mit Muster loeschen
:v/muster/d          alle Zeilen ohne Muster loeschen
:g/^$/d              leere Zeilen loeschen
:g/muster/normal A;  an jede passende Zeile ein Semikolon haengen
```

`:g` (global) führt für alle Zeilen, die zum Muster passen, einen Ex-Befehl aus – eines der mächtigsten Werkzeuge. Mit `:set hlsearch incsearch ignorecase smartcase` wird die Suche komfortabler: Treffer werden hervorgehoben, die Suche beginnt schon während des Tippens, und Großschreibung zählt nur, wenn man sie im Muster verwendet. `:noh` schaltet die Hervorhebung ab.

5.5. Mehrere Dateien: Puffer, Fenster, Tabs

Niveau: Fortgeschritten

Befehl	Wirkung
`:e datei`	Datei in neuem Puffer öffnen
`:ls`, `:b 3`, `:bn`, `:bp`	Puffer anzeigen, wechseln
`:sp datei`, `:vsp datei`	Horizontal / vertikal teilen
`Ctrl-w h/j/k/l`	Zwischen Fenstern wechseln
`Ctrl-w =`, `Ctrl-w _`	Fenster angleichen / maximieren
`Ctrl-w c`, `:only`	Fenster schließen / nur dieses behalten
`:tabnew datei`, `gt`, `gT`	Neuer Tab, nächster, vorheriger
`:wa`, `:qa`, `:wqa`	Alle speichern / beenden
`vimdiff a b`, `:diffthis`	Dateien vergleichen
`:r datei`, `:r !befehl`	Datei oder Befehlsausgabe einfügen
`:w !sudo tee %`	Schreibgeschützte Datei als root speichern

Visuelle Modi: `v` (zeichenweise), `V` (zeilenweise), `Ctrl-v` (Block). Der Blockmodus erlaubt das Bearbeiten von Spalten: Mit `Ctrl-v`, Bewegung nach unten, `I` + Text + `Esc` wird in jede Zeile am Blockanfang Text eingefügt (etwa ein Kommentarzeichen).

5.6. Die Konfigurationsdatei .vimrc

Niveau: Fortgeschritten

Eigene Einstellungen stehen in `~/.vimrc` (bei Neovim: `~/.config/nvim/init.vim`).

```
" ~/.vimrc - Beispiel
set nocompatible
syntax on
filetype plugin indent on

set number relativenumber      " Zeilennummern
set tabstop=4 shiftwidth=4 expandtab   " 4 Leerzeichen statt Tabs
set autoindent smartindent
set hlsearch incsearch ignorecase smartcase
set scrolloff=5                " Kontext beim Scrollen
set wildmenu                   " Menue fuer Kommandozeilenvervollstaendigung
set hidden                     " Puffer wechseln ohne Speichern
set mouse=a
set clipboard=unnamedplus      " Systemzwischenablage
set undofile undodir=~/.vim/undo   " Undo ueber Sitzungen hinweg

let mapleader = ","
nnoremap <leader>w :w<CR>
nnoremap <leader>q :q<CR>
nnoremap <leader>h :nohlsearch<CR>
" Esc mit jk
inoremap jk <Esc>

autocmd FileType yaml setlocal tabstop=2 shiftwidth=2
autocmd BufWritePre * %s/\s\+$//e    " Leerraum beim Speichern entfernen
```

`nnoremap` definiert eine nicht-rekursive Abbildung im Normalmodus (`n`), analog `inoremap` (Einfügemodus) und `vnoremap` (visuell). `:help` <Thema> öffnet die hervorragende eingebaute Dokumentation, `vimtutor` führt interaktiv in die Grundlagen ein.

5.7. Plugins und Erweiterungen

Niveau: Experte

Plugins erweitern vim um Dateibaum, Fuzzy-Suche, Git-Integration oder Autovervollständigung. Seit Vim 8 gibt es eine eingebaute Paketverwaltung: Plugins werden nach `~/.vim/pack/<name>/start/` geklont und beim Start automatisch geladen.

```
$ mkdir -p ~/.vim/pack/plugins/start
$ git clone https://github.com/tpope/vim-surround ~/.vim/pack/plugins/start/vim-surround
$ vim -c 'helptags ALL' -c q
```

Plugin	Zweck
vim-surround	Klammern/Anführungszeichen um Text setzen und ändern (`cs"'`)
vim-commentary	Auskommentieren mit `gcc`
fzf.vim	Fuzzy-Suche nach Dateien, Puffern, Text
vim-fugitive, gitgutter	Git-Integration
NERDTree / netrw	Dateibaum (netrw ist eingebaut: `:Explore`)
ALE, coc.nvim	Syntaxprüfung, Language-Server-Unterstützung

Übungen und Kontrollfragen

1. Wie ändern Sie in `print("hallo welt")` den Text zwischen den Anführungszeichen?

   Lösung: Cursor in den Text, dann `ci"` eingeben, neuen Text tippen, `Esc`.

2. Wie hängen Sie an alle Zeilen einer Datei ein Semikolon an?

   Lösung: `:%s/$/;/` oder `:g/./normal A;`.

3. Wie kopieren Sie eine Zeile in die Systemzwischenablage?

   Lösung: `"+yy` (benötigt `vim` mit Clipboard-Unterstützung, z. B. Paket `vim-gtk3`).

4. Sie haben eine Datei ohne Schreibrechte geöffnet und viel geändert. Wie speichern Sie?

   Lösung: `:w !sudo tee %` und anschließend `:e!` zum Neuladen.

5. Wie nummerieren Sie in einer Textdatei zehn Zeilen mit einem Makro?

   Lösung: Makro aufnehmen, das `Ctrl-a` (Zahl erhöhen) auf der kopierten Zeile ausführt, z. B. `qayyp^Ctrl-aq`, dann `9@a`.

6. Das Dateisystem

Inhalt: Verzeichnisbaum nach FHS, wichtige Verzeichnisse, absolute und relative Pfade, Dateitypen, Links, Inodes.

Stufe: [E] Einsteiger

Lernziele: Nach diesem Kapitel können Sie

- die Verzeichnisstruktur und den Zweck der Hauptverzeichnisse zu erklären,
- sich mit absoluten und relativen Pfaden sicher zu bewegen,
- Hardlinks und symbolische Links zu unterscheiden und anzulegen.

Vorkenntnisse: Kapitel 2.

6.1. Grundbegriffe

Niveau: Einsteiger

Unter Linux gilt: *Alles ist eine Datei*. Dokumente, Verzeichnisse, Geräte, Prozessinformationen und Netzwerkverbindungen werden über Dateien angesprochen. Alle Dateien liegen in **einem** einzigen Verzeichnisbaum, der bei der Wurzel `/` (*root directory*) beginnt. Laufwerksbuchstaben wie unter Windows gibt es nicht; weitere Datenträger werden an einer Stelle im Baum *eingehängt* (*mount*).

- Der Pfadtrenner ist der Schrägstrich `/`.
- Dateinamen dürfen jedes Zeichen außer `/` und dem Nullzeichen enthalten, sollten aber auf Buchstaben, Ziffern, `.`, `-` und `_` beschränkt bleiben. Leerzeichen und Sonderzeichen erschweren die Arbeit in der Shell.
- Dateien, deren Name mit einem Punkt beginnt, sind **versteckt** (z. B. `.bashrc`).
- Es gibt keine Dateiendungen im technischen Sinn; `.txt` oder `.sh` sind nur Konvention. Mit `file` lässt sich der tatsächliche Typ prüfen.
- Ein Name darf bis zu 255 Bytes lang sein, ein Pfad bis zu 4096.

6.2. Dateitypen

Niveau: Einsteiger

Zeichen	Typ	Beschreibung
`-`	Reguläre Datei	Text, Programm, Bild, Archiv …
`d`	Verzeichnis (*directory*)	Liste von Dateinamen
`l`	Symbolischer Link	Verweis auf einen anderen Pfad
`b`	Blockgerät	z. B. `/dev/sda`, Platten
`c`	Zeichengerät	z. B. `/dev/tty1`, `/dev/null`
`p`	Named Pipe (FIFO)	Verbindung zwischen Prozessen
`s`	Socket	Lokaler Kommunikationskanal

```
$ ls -l /dev/sda /dev/null /etc/os-release
brw-rw---- 1 root disk 8, 0 Okt 14 08:00 /dev/sda
crw-rw-rw- 1 root root 1, 3 Okt 14 08:00 /dev/null
lrwxrwxrwx 1 root root 21 Okt  1 10:12 /etc/os-release -> ../usr/lib/os-release
$ file /bin/ls /etc/passwd
/bin/ls:     symbolic link to usr/bin/ls
/etc/passwd: ASCII text
```

6.3. Der Verzeichnisbaum nach FHS

Niveau: Einsteiger

Der **Filesystem Hierarchy Standard** (FHS) legt fest, was wo liegt. Die Struktur ist auf allen Distributionen sehr ähnlich.

Verzeichnis	Inhalt
`/`	Wurzel des gesamten Baums
`/bin`, `/sbin`	Grundlegende Programme (Benutzer / Administration); auf modernen Systemen Links nach `/usr/bin` und `/usr/sbin`
`/boot`	Kernel, Initramfs, Bootloader-Dateien
`/dev`	Gerätedateien (`sda`, `tty`, `null`, `random`)
`/etc`	**Systemweite Konfigurationsdateien** (nur Text)
`/home`	Heimatverzeichnisse der Benutzer (`/home/tux`)
`/lib`, `/lib64`	Bibliotheken und Kernelmodule (Link nach `/usr/lib`)
`/media`	Einhängepunkte für Wechselmedien (USB, DVD)
`/mnt`	Manuell eingehängte Dateisysteme
`/opt`	Zusatzsoftware einzelner Hersteller
`/proc`	Pseudo-Dateisystem: Prozess- und Kernelinformationen
`/root`	Heimatverzeichnis des Benutzers `root`
`/run`	Laufzeitdaten seit dem Systemstart (PID-Dateien, Sockets)
`/srv`	Daten von Diensten (Web, FTP)
`/sys`	Pseudo-Dateisystem: Geräte und Treiber
`/tmp`	Temporäre Dateien (bei Neustart gelöscht, für alle beschreibbar)
`/usr`	Programme, Bibliotheken, Dokumentation (`/usr/bin`, `/usr/share`, `/usr/local`)
`/var`	Veränderliche Daten: Logs (`/var/log`), Mail, Caches, Datenbanken (`/var/lib`)

Merke

Konfiguration liegt in `/etc`, **Programme** in `/usr`, **Veränderliches** in `/var`, **Benutzerdaten** in `/home`. Wer diese vier Orte kennt, findet auf jedem Linux-System das Wichtigste.

6.4. Pseudo-Dateisysteme

Niveau: Fortgeschritten

`/proc` und `/sys` existieren nur im Arbeitsspeicher. Sie machen Kerneldaten als Dateien lesbar, teilweise auch schreibbar:

```
$ cat /proc/cpuinfo | head -5      # Prozessorinformationen
$ cat /proc/meminfo | head -3      # Speicher
$ cat /proc/uptime                 # Laufzeit in Sekunden
$ cat /proc/$$/status | head -3    # Daten der eigenen Shell ($$ = deren PID)
$ ls /sys/class/net                # Netzwerkschnittstellen
# echo 1 > /proc/sys/net/ipv4/ip_forward   # Kernelparameter (bis zum Neustart)
```

6.5. Dateisysteme und Einhängen im Überblick

Niveau: Fortgeschritten

Ein **Dateisystem** ist die Struktur, in der Daten auf einem Datenträger abgelegt werden (z. B. `ext4`, `xfs`, `btrfs`). Der Befehl `mount` fügt es an einem **Einhängepunkt** in den Baum ein; `umount` löst es wieder. Die Ausführlichkeit steht in Kapitel 29; für den Alltag genügt:

```
$ df -h                          # Belegung der eingehaengten Dateisysteme
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda2        50G   18G   30G  38% /
/dev/sda1       512M  6,1M  506M   2% /boot/efi
$ lsblk                          # Platten und Partitionen als Baum
$ findmnt                        # eingehaengte Dateisysteme als Baum
$ du -sh /var/log                # Platzbedarf eines Verzeichnisses
# mount /dev/sdb1 /mnt           # Datentraeger einhaengen
# umount /mnt                    # und wieder loesen
```

Auf Desktops hängt der Desktop (*udisks*) USB-Medien automatisch unter `/media/<benutzer>/` ein.

6.6. Wo liegt was auf Ubuntu, RHEL und openSUSE?

Niveau: Experte

Thema	Debian/Ubuntu	RHEL/Rocky	openSUSE
Paketdatenbank	`/var/lib/dpkg`	`/var/lib/rpm`	`/var/lib/rpm`
Systemlog	`/var/log/syslog`	`/var/log/messages`	Journal
Netzwerkkonfig.	`/etc/netplan`	NetworkManager	wicked / NM
Admin-Gruppe	`sudo`	`wheel`	`wheel`
Firewall-Standard	`ufw`	`firewalld`	`firewalld`

7. Dateien und Verzeichnisse verwalten

Inhalt: Anlegen, Kopieren, Verschieben, Löschen, Platzhalter (Globbing), Suchen mit `find` und `locate`, Dateiinformationen.

Stufe: [E] Einsteiger bis [F] Fortgeschritten

Lernziele: Nach diesem Kapitel können Sie

- Dateien und Verzeichnisse zu erzeugen, zu kopieren, zu verschieben und zu löschen,
- Dateien mit Platzhaltern auszuwählen,
- Dateien nach Name, Größe, Alter und Besitzer zu finden und Aktionen auszuführen.

Vorkenntnisse: Kapitel 2 und 6.

7.1. Pfade und Navigation

Niveau: Einsteiger

Ein **absoluter Pfad** beginnt bei `/` (`/etc/hosts`), ein **relativer Pfad** beim aktuellen Verzeichnis (`docs/brief.txt`). Besondere Namen:

`.`	Das aktuelle Verzeichnis
`..`	Das übergeordnete Verzeichnis
`~`	Das eigene Heimatverzeichnis (von der Shell ersetzt); `~anna` steht für das Heimatverzeichnis von `anna`
`-`	Bei `cd`: das vorherige Verzeichnis

```
$ pwd                       # aktuelles Verzeichnis
/home/tux
$ cd /var/log               # absolut wechseln
$ cd ..                     # eine Ebene hoeher
$ cd                        # ins Heimatverzeichnis (wie cd ~)
$ cd -                      # zurueck zum vorherigen Verzeichnis
$ ls                        # Inhalt anzeigen
$ ls -l                     # lange Liste: Rechte, Eigentuemer, Groesse, Datum
$ ls -lah                   # inkl. versteckter Dateien, Groessen lesbar
$ ls -ld /etc               # das Verzeichnis selbst statt seines Inhalts
$ ls -lt                    # nach Zeit sortiert (neueste zuerst); -r dreht um
$ ls -R                     # rekursiv
```

Die Ausgabe von `ls -l` im Detail:

```
-rw-r--r-- 1 tux users 4096 Okt 14 09:30 notizen.txt
|\_/\_/\_/  | |   |     |        |        +- Name
| |  |  |   | |   |     |        +- Datum der letzten Aenderung
| |  |  |   | |   |     +- Groesse in Bytes
| |  |  |   | |   +- Gruppe
| |  |  |   | +- Eigentuemer
| |  |  |   +- Anzahl der Hardlinks
| |  |  +- Rechte fuer 'andere'
| |  +- Rechte fuer die Gruppe
| +- Rechte fuer den Eigentuemer
+- Dateityp
```

7.2. Verzeichnisse anlegen und löschen

Niveau: Einsteiger

```
$ mkdir projekt                  # Verzeichnis anlegen
$ mkdir -p projekt/src/test      # inklusive aller Elternverzeichnisse
$ rmdir projekt/src/test         # leeres Verzeichnis loeschen
$ rm -r projekt                  # Verzeichnis mit Inhalt loeschen
$ tree projekt                   # Baumdarstellung (Paket 'tree')
```

7.3. Dateien anlegen, kopieren, verschieben, löschen

Niveau: Einsteiger

```
$ touch neu.txt                  # leere Datei anlegen / Zeitstempel aktualisieren
$ cp quelle.txt ziel.txt         # kopieren
$ cp -r verz1 verz2              # Verzeichnis rekursiv kopieren
$ cp -a verz1 sicherung          # Archivkopie: Rechte, Zeiten, Links erhalten
$ cp -i a.txt b.txt              # vor dem Ueberschreiben nachfragen
$ mv alt.txt neu.txt             # umbenennen
$ mv datei.txt ~/Dokumente/      # verschieben (kein Kopieren auf gleichem Dateisystem)
$ rm datei.txt                   # loeschen
$ rm -i *.log                    # mit Rueckfrage
$ rm -rf verz                    # ohne Rueckfrage, rekursiv: VORSICHT
```

> **Achtung**
>
> Es gibt **keinen Papierkorb** auf der Kommandozeile. Was `rm` löscht, ist weg. Besonders gefährlich: `rm -rf /` und Leerzeichen an falscher Stelle (`rm -rf /home/tux /` statt `rm -rf /home/tux/`). Vor dem Löschen mit Mustern zuerst `ls` mit demselben Muster ausführen. Aktuelle `rm`-Versionen verweigern `rm -rf /` ohne `--no-preserve-root`.

7.4. Dateinamen-Muster (Globbing)

Niveau: Einsteiger

Die Shell ersetzt Muster (*wildcards*) durch passende Dateinamen, *bevor* das Programm startet.

Muster	Bedeutung
`*`	Beliebig viele Zeichen (auch keines); versteckte Dateien werden nicht erfasst
`?`	Genau ein beliebiges Zeichen
`[abc]`	Eines der Zeichen a, b, c
`[a-f]`, `[0-9]`	Zeichen aus einem Bereich
`[!abc]`	Ein Zeichen, das *nicht* aufgeführt ist
`{a,b,c}`	*Brace-Expansion*: erzeugt die Folge a, b, c (keine Dateisuche)

```
$ ls *.txt                       # alle .txt-Dateien
$ ls bild?.png                   # bild1.png, bildA.png, ...
$ ls [A-Z]*                      # beginnt mit Grossbuchstaben
$ ls .??*                        # versteckte Dateien
$ mkdir -p proj/{src,doc,test}   # drei Verzeichnisse auf einmal
$ cp bericht.{txt,bak}           # wie: cp bericht.txt bericht.bak
$ echo *.txt                     # zeigt, was die Shell einsetzt
```

7.5. Links

Niveau: Fortgeschritten

Ein **Hardlink** ist ein zweiter Name für dieselbe Datei (denselben *Inode*); sie wird erst gelöscht, wenn der letzte Name entfernt wurde. Er funktioniert nur innerhalb eines Dateisystems und nicht für Verzeichnisse. Ein **symbolischer Link** (*Symlink*) ist eine eigene kleine Datei mit einem Pfad als Inhalt und kann auf beliebige Ziele zeigen. Zeigt das Ziel ins Leere, ist der Link *kaputt*.

```
$ ln original.txt hart.txt          # Hardlink
$ ln -s /var/log/syslog logs        # symbolischer Link
$ ls -li original.txt hart.txt      # gleiche Inode-Nummer in Spalte 1
$ readlink -f logs                  # Ziel aufloesen
```

7.6. Dateien suchen

Niveau: Fortgeschritten

```
$ find /etc -name "*.conf"                  # nach Namen
$ find ~ -type f -size +100M                # Dateien groesser als 100 MiB
$ find /var/log -mtime -1                   # in den letzten 24 h geaendert
$ find . -type f -name "*.tmp" -delete      # Treffer loeschen
$ find . -name "*.txt" -exec wc -l {} \;    # Befehl auf jeden Treffer anwenden
$ find /srv -user tux -perm -o+w            # Eigentuemer und Rechte
$ locate fstab                              # schnell, ueber Datenbank (updatedb)
$ whereis ls                                # Programm, Quelle, Handbuchseite
```

Test bei `find`	Bedeutung
`-name`, `-iname`	Dateiname mit Muster (`-iname`: Groß-/Kleinschreibung egal)
`-type f`, `d`, `l`	Datei, Verzeichnis, Link
`-size +10M`	Größer als 10 MiB (`-` kleiner; Einheiten `c`, `k`, `M`, `G`)
`-mtime -2`, `-mmin -30`	Geändert in den letzten 2 Tagen / 30 Minuten
`-user`, `-group`, `-perm`	Eigentümer, Gruppe, Rechte
`-maxdepth 2`	Höchstens zwei Ebenen tief
`-exec cmd {} \;`	Befehl für jeden Treffer
`-a`, `-o`, `!`	Und, Oder, Nicht

7.7. Dateiinhalt anzeigen

Niveau: Einsteiger

```
$ cat datei.txt             # gesamte Datei ausgeben
$ less langelog.txt         # seitenweise: Leertaste vor, b zurueck, /suche, q Ende
$ head -n 5 datei.txt       # erste 5 Zeilen
$ tail -n 20 datei.txt      # letzte 20 Zeilen
$ tail -f /var/log/syslog   # Datei live verfolgen (Strg+C beendet)
$ wc -l datei.txt           # Zeilen zaehlen (-w Woerter, -c Bytes)
$ file bild.png             # Dateityp erkennen
$ stat datei.txt            # alle Metadaten
```

> **Tipp**
>
> `less` ist der Standard-Pager: Mit `F` springt es in den *Follow*-Modus (wie `tail -f`), mit `Strg+C` zurück. Mit `less +F datei` startet man direkt dort.

8. Ein- und Ausgabe, Pipes und Filter

Inhalt: Standardkanäle, Umleitungen, Pipes, Filterprogramme wie `sort`, `uniq`, `cut`, `tr`, `wc`, `tee`, `xargs`.
Stufe: [E] Einsteiger bis [F] Fortgeschritten
Lernziele: Nach diesem Kapitel können Sie

- `stdin`, `stdout` und `stderr` umzuleiten,
- Befehle zu Pipelines zu verketten,
- typische Auswertungen mit Filterprogrammen zu bauen.

Vorkenntnisse: Kapitel 2 und 7.

8.1. Standardkanäle

Niveau: Einsteiger

Jedes Linux-Programm hat drei Standardkanäle. Die Shell kann sie umleiten, ohne dass das Programm davon wissen muss.

Nr.	Name	Kürzel	Standardziel
0	Standardeingabe	`stdin`	Tastatur
1	Standardausgabe	`stdout`	Bildschirm
2	Standardfehlerausgabe	`stderr`	Bildschirm

```
$ ls > liste.txt              # stdout in Datei (ueberschreibt)
$ ls >> liste.txt             # anhaengen
$ ls /gibtsnicht 2> fehler.txt   # nur stderr in Datei
$ ls /etc /gibtsnicht > aus.txt 2> fehler.txt   # getrennt
$ ls /etc /gibtsnicht > alles.txt 2>&1          # beides in dieselbe Datei
$ ls /etc /gibtsnicht &> alles.txt              # Kurzform in der bash
$ find / -name passwd 2> /dev/null              # Fehlermeldungen verwerfen
$ sort < ungeordnet.txt       # stdin aus Datei
$ cat > notiz.txt             # Eingabe von der Tastatur bis Strg+D
```

`/dev/null` ist die “schwarze Kiste”: Alles, was man hineinschreibt, verschwindet. `/dev/zero` liefert endlos Nullbytes, `/dev/urandom` Zufallsbytes.

> **Achtung**
> Mit `>` wird die Zieldatei **sofort geleert**, noch bevor das Kommando läuft. Daher niemals `sort datei.txt > datei.txt`: Die Datei ist danach leer. Abhilfe: in eine Zwischendatei schreiben oder `sort -o datei.txt datei.txt` nutzen. Mit `set -o noclobber` verhindert die Shell das Überschreiben bestehender Dateien.

8.2. Pipes

Niveau: Einsteiger

Eine **Pipe** (|) verbindet die Standardausgabe des linken mit der Standardeingabe des rechten Kommandos. So lassen sich einfache Werkzeuge zu mächtigen Abläufen kombinieren: die Unix-Philosophie, "jedes Programm macht eine Sache gut".

```
$ ls /etc | wc -l                                  # wie viele Eintraege hat /etc?
$ ps aux | grep sshd                               # laeuft der SSH-Daemon?
$ cat /etc/passwd | cut -d: -f1 | sort             # alle Benutzernamen, sortiert
$ du -sh /var/* 2>/dev/null | sort -h | tail -5    # die fuenf groessten Verzeichnisse
$ history | awk '{print $2}' | sort | uniq -c | sort -rn | head
```

Mit `tee` lässt sich der Datenstrom abzweigen: Er läuft weiter, wird aber zusätzlich in eine Datei geschrieben.

```
$ make 2>&1 | tee build.log               # sehen UND mitschreiben
$ echo "x" | sudo tee /etc/test.conf      # mit root-Rechten in Datei schreiben
$ echo "y" | sudo tee -a /etc/test.conf   # anhaengen
```

Das letzte Beispiel ist wichtig: In `sudo echo x > datei` läuft nur `echo` als root, die Umleitung führt aber die *eigene* Shell aus, und diese hat keine Rechte.

8.3. Dateien lesen und zusammenfügen

Niveau: Einsteiger

```
$ cat a.txt b.txt > ab.txt         # Dateien aneinanderhaengen
$ cat -n datei.txt                 # mit Zeilennummern
$ tac datei.txt                    # Zeilen in umgekehrter Reihenfolge
$ nl datei.txt                     # Zeilen nummerieren (nur nichtleere)
$ head -n 3 datei.txt | tail -n 1  # genau die dritte Zeile
$ od -c datei.txt | head           # Inhalt als Zeichen/Bytes zeigen
$ xxd datei.bin | head             # Hexdump
```

8.4. Text sortieren und zusammenfassen

Niveau: Einsteiger

```
$ sort namen.txt               # alphabetisch
$ sort -r namen.txt            # umgekehrt
$ sort -n zahlen.txt           # numerisch (sonst: 10 vor 9)
$ sort -h groessen.txt         # menschenlesbare Groessen (1K, 2M)
$ sort -u namen.txt            # sortieren und doppelte Zeilen entfernen
$ sort -t: -k3 -n /etc/passwd  # Trennzeichen ':', nach 3. Feld numerisch
$ sort | uniq                  # benachbarte Duplikate entfernen
$ sort | uniq -c               # ... und zaehlen
$ sort | uniq -d               # nur die doppelten Zeilen
```

> **Hinweis**
>
> `uniq` erkennt nur *unmittelbar benachbarte* gleiche Zeilen. Deshalb steht davor fast immer `sort`. Die Sortierreihenfolge hängt von der Locale ab (siehe Kapitel 18); mit `LC_ALL=C sort` wird nach Bytewerten sortiert.

8.5. Spalten schneiden, ersetzen, verbinden

Niveau: Fortgeschritten

```
$ cut -d: -f1,3 /etc/passwd            # Felder 1 und 3, Trenner ':'
$ cut -c1-8 datei.txt                  # Zeichen 1 bis 8 jeder Zeile
$ tr 'a-z' 'A-Z' < datei.txt           # Klein- in Grossbuchstaben
$ tr -d '\r' < win.txt > unix.txt      # Windows-Zeilenenden entfernen
$ tr -s ' ' < datei.txt                # mehrfache Leerzeichen zusammenfassen
$ paste namen.txt zahlen.txt           # Dateien zeilenweise nebeneinander
$ join -t: datei1 datei2               # Zeilen mit gleichem erstem Feld verbinden
$ column -t -s: /etc/passwd | head -3  # als Tabelle ausgeben
$ fmt -w 60 text.txt                   # Absaetze auf 60 Zeichen umbrechen
```

Kommando	Zweck
`wc`	Zeilen (`-l`), Wörter (`-w`), Bytes (`-c`) zählen
`sort`	Zeilen sortieren
`uniq`	Doppelte benachbarte Zeilen entfernen / zählen
`cut`	Spalten oder Zeichenbereiche ausschneiden
`tr`	Zeichen ersetzen, löschen, zusammenfassen
`head`, `tail`	Anfang bzw. Ende ausgeben
`paste`, `join`	Dateien nebeneinander bzw. nach Schlüssel verbinden
`diff`, `cmp`	Dateien vergleichen
`tee`	Datenstrom in Datei kopieren
`xargs`	Eingabezeilen als Argumente an ein Kommando übergeben

8.6. xargs und Kommandosubstitution

Niveau: Fortgeschritten

`xargs` macht aus der Standardeingabe die *Argumentliste* eines anderen Kommandos. Mit `$(...)` (*Kommandosubstitution*) wird die Ausgabe eines Kommandos direkt im Befehl eingesetzt.

```
$ find . -name "*.tmp" | xargs rm                # bricht bei Leerzeichen im Namen!
$ find . -name "*.tmp" -print0 | xargs -0 rm     # robust
$ ls *.txt | xargs -n1 wc -l                     # je Datei ein Aufruf
$ echo "Heute ist $(date +%A)"                   # Substitution
$ tar czf backup-$(date +%F).tar.gz ~/Dokumente  # Datum im Dateinamen
$ for d in $(cat hosts.txt); do ping -c1 $d; done
```

8.7. Dateien vergleichen

Niveau: Fortgeschritten

```
$ diff alt.conf neu.conf               # Unterschiede zeilenweise
$ diff -u alt.conf neu.conf            # Unified-Format (wie bei Patches)
$ diff -r verz1 verz2                  # Verzeichnisse rekursiv
$ cmp a.bin b.bin                      # Binaerdateien vergleichen
$ sha256sum image.iso                  # Pruefsumme berechnen
$ sha256sum -c SHA256SUMS              # Pruefsummen pruefen
```

9. Reguläre Ausdrücke, grep, sed und awk

Inhalt: Aufbau regulärer Ausdrücke (Basic und Extended), `grep`, Textersetzung mit `sed`, spaltenorientierte Auswertung mit `awk`.

Stufe: [F] Fortgeschritten bis [X] Experte

Lernziele: Nach diesem Kapitel können Sie

- reguläre Ausdrücke zu lesen und zu schreiben,
- Texte mit `grep` zu filtern und mit `sed` umzuformen,
- mit `awk` Felder auszuwerten und einfache Berichte zu erzeugen.

Vorkenntnisse: Kapitel 8 (Pipes und Filter).

9.1. Reguläre Ausdrücke

Niveau: Einsteiger

Ein **regulärer Ausdruck** (*regular expression*, kurz *regex*) beschreibt ein Textmuster. Sie werden von `grep`, `sed`, `awk`, `vi` und vielen Programmiersprachen verwendet. Reguläre Ausdrücke sind **nicht** dasselbe wie die Dateimuster der Shell (`*.txt`): Dort steht `*` für "beliebig viele Zeichen", hier für "beliebig viele Wiederholungen des vorherigen Zeichens".

Ausdruck	Bedeutung
`.`	Genau ein beliebiges Zeichen
`^` / `$`	Zeilenanfang / Zeilenende
`[abc]`, `[a-z]`, `[^0-9]`	Eines der Zeichen / Bereich / alles außer Ziffern
`[[:digit:]]`, `[[:alpha:]]`, `[[:space:]]`	POSIX-Zeichenklassen
`*`	Das vorherige Element 0-mal oder öfter
`+`, `?`	1-mal oder öfter / 0- oder 1-mal (erweiterte Syntax, `-E`)
`{n,m}`	Mindestens n, höchstens m Wiederholungen (erweiterte Syntax)
`a|b`	Alternative: `a` oder `b` (erweitert)
`(...)`	Gruppierung (erweitert), in `sed` als `\1` wiederverwendbar
`\.`	Sonderzeichen mit Backslash maskieren (hier: ein echter Punkt)

9.2. grep: Text durchsuchen

Niveau: Einsteiger

`grep` gibt alle Zeilen aus, die auf ein Muster passen.

```
$ grep root /etc/passwd                    # Zeilen mit 'root'
$ grep -i fehler logdatei                  # Gross-/Kleinschreibung ignorieren
$ grep -v '^#' /etc/ssh/sshd_config        # Zeilen, die NICHT passen (hier: Kommentare)
$ grep -n error /var/log/syslog            # mit Zeilennummern
$ grep -c error /var/log/syslog            # nur Anzahl der Treffer
$ grep -r "PermitRootLogin" /etc           # rekursiv in Verzeichnissen
$ grep -l "tux" /etc/*                     # nur Dateinamen ausgeben
$ grep -w tux /etc/passwd                  # ganzes Wort
$ grep -A2 -B1 "Failed" auth.log           # 2 Zeilen danach, 1 davor
$ grep -E 'ssh|http' /etc/services         # erweiterte Syntax: Alternative
$ grep -E '^[0-9]{3}-' liste.txt           # Zeilen, die mit drei Ziffern und '-' beginnen
$ grep -o '[0-9.]\{7,15\}' log             # nur den passenden Teil ausgeben
$ grep -v '^#' /etc/fstab | grep -v '^$'   # Kommentare und Leerzeilen ausblenden
```

Tipp

Muster mit Sonderzeichen immer in **einfache Anführungszeichen** setzen, damit die Shell sie nicht auswertet. `egrep` entspricht `grep -E`; `fgrep` entspricht `grep -F` (Festtext ohne Regex). `ripgrep` (`rg`) ist eine schnelle Alternative für große Verzeichnisbäume.

9.3. sed: der Stromeditor

Niveau: Fortgeschritten

`sed` (*stream editor*) bearbeitet Text zeilenweise, ohne ihn zu öffnen. Am häufigsten wird das Ersetzen `s/alt/neu/` eingesetzt.

```
$ sed 's/Katze/Hund/' tiere.txt          # erstes Vorkommen je Zeile ersetzen
$ sed 's/Katze/Hund/g' tiere.txt         # alle Vorkommen
$ sed -i 's/Katze/Hund/g' tiere.txt      # direkt in der Datei aendern (-i)
$ sed -i.bak 's/a/b/' datei              # ... mit Sicherheitskopie datei.bak
$ sed -n '5,10p' datei                   # nur Zeilen 5 bis 10 ausgeben
$ sed '/^#/d' datei                      # Kommentarzeilen loeschen
$ sed '/^$/d' datei                      # Leerzeilen loeschen
$ sed '3d' datei                         # Zeile 3 loeschen
$ sed 's/^/    /' datei                  # jede Zeile einruecken
$ sed -E 's/([0-9]+)-([0-9]+)/\2-\1/' d  # Gruppen vertauschen
$ sed 's#/usr/local#/opt#g' pfade.txt    # anderes Trennzeichen bei Pfaden
```

Befehl	Wirkung
`s/muster/ersatz/[g]`	Ersetzen; `&` im Ersatz steht für den Treffer
`d`	Zeile löschen
`p`	Zeile ausgeben (zusammen mit `-n`)
`i\` / `a\`	Text vor / nach der Zeile einfügen
`y/abc/xyz/`	Zeichenweise tauschen (wie `tr`)
Adressen	Zeilennummer (`5`), Bereich (`5,10`), Muster (`/foo/`), letzte Zeile (`$`)

9.4. awk: Spalten verarbeiten

Niveau: Fortgeschritten

`awk` liest Textzeilen, teilt sie in Felder (`$1`, `$2`, …; `$0` ist die ganze Zeile) und führt für passende Zeilen Aktionen aus. Das Schema lautet `Bedingung { Aktion }`.

```
$ awk '{print $1}' datei                    # erstes Feld jeder Zeile
$ awk -F: '{print $1, $3}' /etc/passwd      # Trenner ':' - Benutzer und UID
$ awk -F: '$3 >= 1000 {print $1}' /etc/passwd  # nur normale Benutzer
$ awk '/error/ {n++} END {print n}' log     # Zeilen mit 'error' zaehlen
$ awk '{s += $5} END {print s}' tabelle     # Summe der 5. Spalte
$ awk 'NR==1 || /Fehler/' datei             # erste Zeile und Fehlerzeilen
$ awk '{print NR": "$0}' datei              # Zeilennummern voranstellen
$ awk 'length($0) > 80' datei               # Zeilen laenger als 80 Zeichen
$ df -h | awk 'NR>1 {print $6, $5}'         # Mountpoint und Belegung
```

Element	Bedeutung
`$0`, `$1`, …, `NF`	Ganze Zeile, einzelne Felder, Anzahl der Felder; `$NF` ist das letzte Feld
`NR`	Laufende Zeilennummer
`FS`, `-F`	Feldtrenner (Standard: Leerraum)
`BEGIN {...}`	Wird einmal vor der ersten Zeile ausgeführt
`END {...}`	Wird einmal nach der letzten Zeile ausgeführt
`printf`	Formatierte Ausgabe wie in C, z. B. `printf "%d\n", $1`

9.5. Praxisbeispiele

Niveau: Experte

```
# Die 10 haeufigsten Aufrufer-IPs in einem Webserver-Log
$ awk '{print $1}' /var/log/nginx/access.log | sort | uniq -c | sort -rn | head

# Fehlgeschlagene SSH-Anmeldungen nach Benutzer zaehlen
$ grep "Failed password" /var/log/auth.log | awk '{print $9}' | sort | uniq -c | sort -rn

# Alle aktiven Konfigurationszeilen eines Dienstes
$ grep -Ev '^\s*(#|$)' /etc/ssh/sshd_config

# Suffix aller .jpeg-Dateien in .jpg aendern
$ for f in *.jpeg; do mv -- "$f" "${f%.jpeg}.jpg"; done

# Leerzeichen am Zeilenende entfernen
$ sed -i 's/[[:space:]]*$//' datei.txt
```

10. Textverarbeitung in der Praxis

Inhalt: Praxisrezepte mit `grep`, `sed`, `awk`, `cut`, `sort`, `uniq`, `tr`, `join` und `jq`: Logauswertung, CSV-Verarbeitung, Konfigurationsdateien ändern.

Stufe: [F] Fortgeschritten bis [X] Experte

Lernziele: Nach diesem Kapitel können Sie

- typische Auswertungsaufgaben mit Filterketten zu lösen,
- `awk` für Spalten, Summen und Berichte einzusetzen,
- Konfigurationsdateien automatisiert und sicher mit `sed` zu ändern.

Vorkenntnisse: Kapitel 8 (Streams und Filter) und Kapitel 9 (Reguläre Ausdrücke, sed, awk).

10.1. Filterketten entwickeln

Niveau: Fortgeschritten

Komplexe Auswertungen entstehen schrittweise: Man beginnt mit einem Befehl, prüft die Ausgabe und hängt den nächsten Filter an. Als Beispieldaten dient ein Zugriffslog im üblichen Format:

```
192.0.2.10 - - [14/Oct/2025:08:15:01 +0200] "GET /index.html HTTP/1.1" 200 5123
198.51.100.7 - - [14/Oct/2025:08:15:02 +0200] "GET /img/logo.png HTTP/1.1" 200 20480
192.0.2.10 - - [14/Oct/2025:08:15:05 +0200] "POST /login HTTP/1.1" 302 0
203.0.113.9 - - [14/Oct/2025:08:15:09 +0200] "GET /admin HTTP/1.1" 403 199
```

```
# Die zehn haeufigsten Client-Adressen
$ awk '{print $1}' access.log | sort | uniq -c | sort -rn | head

# Verteilung der HTTP-Statuscodes
$ awk '{print $9}' access.log | sort | uniq -c | sort -rn

# Alle Fehler (4xx und 5xx)
$ awk '$9 ~ /^[45]/' access.log

# Uebertragene Bytes insgesamt (MB)
$ awk '{s += $10} END {printf "%.1f MB\n", s/1024/1024}' access.log

# Anfragen pro Stunde
$ awk -F'[: ]' '{print $5}' access.log | sort | uniq -c

# Meistabgerufene URLs
$ awk '{print $7}' access.log | sort | uniq -c | sort -rn | head -5
```

Das Muster `sort | uniq -c | sort -rn` (Häufigkeitsliste) ist so oft nützlich, dass es sich lohnt, es auswendig zu kennen. `uniq` entfernt nur *aufeinanderfolgende* Duplikate, deshalb steht davor immer `sort`.

10.2. awk als Programmiersprache

Niveau: Fortgeschritten

awk liest Zeilen, zerlegt sie in Felder (`$1`, `$2`, …; `$0` = ganze Zeile; `NF` = Feldanzahl; `NR` = Zeilennummer) und führt Aktionen aus, wenn ein Muster passt. Dazu kommen die Blöcke `BEGIN` (vorher) und `END` (nachher).

```
# Allgemeine Form
awk 'BEGIN { ... }  Muster { Aktion }  END { ... }' datei

# Feldtrenner setzen
awk -F: '{print $1, $7}' /etc/passwd                  # Benutzer und Shell
awk -F: '$3 >= 1000 && $3 < 65534 {print $1}' /etc/passwd   # normale Benutzer
awk -F: 'BEGIN{OFS=";"} {print $1,$3,$6}' /etc/passwd       # Ausgabetrenner

# Summen und Mittelwerte
awk '{s+=$2; n++} END {print "Summe:", s, "Mittel:", s/n}' zahlen.txt

# Assoziative Arrays: Summe pro Schluessel
awk '{summe[$1] += $2} END {for (k in summe) print k, summe[k]}' umsatz.txt

# Zeilen 10 bis 20; letzte Spalte; Zeilen mit mehr als 5 Feldern
awk 'NR>=10 && NR<=20' datei
awk '{print $NF}' datei
awk 'NF > 5' datei

# Formatierte Ausgabe
awk '{printf "%-15s %8.2f\n", $1, $2}' preise.txt

# Duplikate entfernen, Reihenfolge erhalten
awk '!gesehen[$0]++' datei
```

Funktionen (Auszug): `length(s)`, `substr(s,start,len)`, `index(s,t)`, `split(s,arr,sep)`, `sub(re,ers)` und `gsub(re,ers)`, `tolower(s)`, `toupper(s)`, `sprintf(fmt,...)`, `system(befehl)`.

```
# CSV mit Semikolon: Spalte 2 grossschreiben, Kopfzeile unveraendert
awk -F';' 'NR==1 {print; next} {$2 = toupper($2); print}' OFS=';' kunden.csv

# Bericht mit Kopf und Fuss
awk 'BEGIN {print "Name      Alter"}
     {printf "%-10s %5d\n", $1, $2; sum += $2}
     END {print "Durchschnitt:", sum/NR}' personen.txt
```

> **Hinweis**
>
> Echte CSV-Dateien mit Anführungszeichen und Kommas innerhalb von Feldern lassen sich mit `awk -F,` nicht zuverlässig verarbeiten. Dafür gibt es spezialisierte Werkzeuge: `csvkit` (`csvcut`, `csvgrep`), `miller` (`mlr`) oder Python.

10.3. sed für Änderungen in Dateien

Niveau: Fortgeschritten

```
# Zeilen ersetzen, bearbeiten, loeschen
sed 's/foo/bar/g' datei                    # alle Vorkommen je Zeile
sed -i 's/foo/bar/g' datei                 # Datei direkt aendern
sed -i.bak 's/foo/bar/g' datei             # mit Sicherungskopie
sed -n '5,10p' datei                       # nur Zeilen 5-10 ausgeben
sed '/^#/d;/^$/d' datei                    # Kommentare und Leerzeilen weg
sed '3i\Neue Zeile vor Zeile 3' datei      # einfuegen (i), anhaengen (a)
sed '/Abschnitt/a\zusaetzlich' datei       # nach passender Zeile anhaengen
sed -n '/START/,/ENDE/p' datei             # Bereich zwischen Marken
sed 's/\(.*\)@\(.*\)/\2: \1/' adressen     # Gruppen umstellen
sed -E 's/([0-9]{3})-([0-9]+)/\2 (\1)/' t # erweiterte Syntax
sed 's|/usr/local|/opt|g' datei            # anderer Trenner bei Pfaden
```

Beispiel: Konfigurationswert setzen. Häufige Aufgabe in Skripten:

```
# PermitRootLogin auf "no" setzen, egal ob auskommentiert oder nicht
sed -i -E 's/^#?\s*PermitRootLogin\s+.*/PermitRootLogin no/' /etc/ssh/sshd_config

# Zeile nur ergaenzen, wenn sie noch fehlt
grep -q '^PermitRootLogin' /etc/ssh/sshd_config || \
    echo 'PermitRootLogin no' >> /etc/ssh/sshd_config

# Immer erst testen: ohne -i ansehen, was passieren wuerde
sed -E 's/^#?\s*PermitRootLogin\s+.*/PermitRootLogin no/' /etc/ssh/sshd_config | diff /etc/
    ssh/sshd_config -
```

> **Sicherheit**
>
> Vor `sed -i` an Systemdateien eine Kopie anlegen (`-i.bak`) und die Änderung mit `diff` prüfen. Konfigurationen nach der Änderung mit dem Syntaxtest des Dienstes validieren (z. B. `sshd -t`), bevor der Dienst neu geladen wird.

10.4. Weitere Werkzeuge

Niveau: Fortgeschritten

Befehl	Beispiel und Wirkung
`cut`	`cut -d: -f1,7 /etc/passwd` – Felder; `cut -c1-10` – Zeichen
`tr`	`tr a-z A-Z`, `tr -d '\r'`, `tr -s ' '` – Zeichen übersetzen, löschen, zusammenfassen
`sort`	`sort -t: -k3 -n`, `sort -u`, `sort -h` (menschenlesbare Größen)
`join`	Zwei nach Schlüssel sortierte Dateien verbinden
`paste`	Dateien spaltenweise zusammenfügen: `paste a.txt b.txt`
`comm`	Sortierte Dateien vergleichen (nur in 1, nur in 2, in beiden)
`column`	`column -t -s,` – Tabelle ausrichten
`fold`, `fmt`	Zeilen umbrechen
`xargs`	Standardeingabe in Argumente verwandeln: `find ... -print0 | xargs -0 rm`
`tee`	In Datei und auf Standardausgabe schreiben
`jq`	JSON verarbeiten (siehe unten)

```
# Sortiere nach der dritten Spalte numerisch, absteigend
sort -t: -k3 -nr /etc/passwd | head -3

# Wer ist in Liste A, aber nicht in Liste B?
comm -23 <(sort a.txt) <(sort b.txt)

# Tabellarisch ausgeben
mount | column -t | head -3
```

10.5. JSON mit jq

Niveau: Experte

Viele moderne Werkzeuge liefern JSON (`ip -j`, `docker inspect`, `lsblk -J`, `journalctl -o json`). `jq` filtert und formt solche Daten:

```
$ ip -j addr | jq -r '.[] | .ifname + ": " + (.addr_info[0].local // "-")'
$ lsblk -J | jq -r '.blockdevices[] | select(.type=="disk") | .name'
$ echo '{"a":1,"b":[1,2,3]}' | jq '.b | add'
6
$ curl -s https://example.org/api | jq '.items[] | {name, id}'
$ jq -r '.[] | [.name, .size] | @csv' daten.json      # nach CSV
```

10.6. Beispiel: Loganalyse

Niveau: Fortgeschritten

```
#!/usr/bin/env bash
# fehlerbericht.sh LOGDATEI – fasst fehlgeschlagene SSH-Anmeldungen zusammen
set -euo pipefail
log=${1:?Aufruf: $0 LOGDATEI}

echo "== Fehlgeschlagene Anmeldungen je Benutzer =="
grep 'Failed password' "$log" \
  | sed -E 's/.*Failed password for (invalid user )?([^ ]+) from ([0-9.]+).*/\2 \3/' \
  | awk '{u[$1]++} END {for (k in u) printf "%6d %s\n", u[k], k}' \
  | sort -rn | head -10

echo
echo "== Top-Quelladressen =="
grep 'Failed password' "$log" \
  | grep -oE '[0-9]{1,3}(\.[0-9]{1,3}){3}' \
  | sort | uniq -c | sort -rn | head -10
```

Übungen und Kontrollfragen

1. Wie viele verschiedene Benutzer haben Shells in `/etc/passwd`, die nicht `nologin` oder `false` sind?

   Lösung: `awk -F: '$7 !~ /(nologin|false)$/' /etc/passwd | wc -l` bzw. mit `grep -vcE '(nologin|false)$'`.

2. Wie erzeugen Sie aus einer Datei mit Zeilen `name,alter` eine Liste der Namen mit Alter über 30?

   Lösung: `awk -F, '$2 > 30 {print $1}' datei`.

3. Wie ersetzen Sie in allen `.conf`-Dateien eines Verzeichnisbaums `alt.example.org` durch `neu.example.org`?

   Lösung: `find . -name '*.conf' -exec sed -i.bak 's/alt\.example\.org/neu.example.org/g' {} +` – Sicherungen mit `.bak` bleiben liegen.

4. Was gibt `awk '!s[$0]++'` aus?

   Lösung: Jede Zeile nur beim ersten Auftreten; Duplikate werden unterdrückt, die ursprüngliche Reihenfolge bleibt erhalten.

5. Warum muss vor `uniq -c` sortiert werden?

   Lösung: `uniq` fasst nur unmittelbar aufeinanderfolgende gleiche Zeilen zusammen.

11. Die Shell im Detail

Inhalt: Aufgaben der Shell, Variablen und Umgebung, Startdateien, Aliase, Quoting, Expansionen, Verlauf und Prompt.
Stufe: [E] Einsteiger bis [F] Fortgeschritten
Lernziele: Nach diesem Kapitel können Sie
- Shell- und Umgebungsvariablen zu setzen und zu exportieren,
- Aliase, Startdateien und den Prompt anzupassen,
- Quoting und Expansionen (Klammern, Befehlsersetzung, Arithmetik) korrekt zu verwenden.

Vorkenntnisse: Kapitel 2 und 8.

11.1. Shell-Varianten

Niveau: Einsteiger

Die Shell ist ein Programm, das Kommandos einliest und ausführt. Unter Linux ist die **bash** (*Bourne-Again Shell*) der Standard. Weitere: `dash` (schlank, `/bin/sh` unter Debian/Ubuntu), `zsh`, `fish`, `ksh`.

```
$ echo $SHELL                 # Login-Shell des Benutzers
$ cat /etc/shells             # erlaubte Shells
$ chsh -s /bin/zsh            # Login-Shell aendern
$ ps -p $$                    # welche Shell laeuft gerade?
```

11.2. Variablen und Umgebung

Niveau: Einsteiger

Shell-Variablen speichern Werte. Mit `export` wird eine Variable zur **Umgebungsvariable** und an Kindprozesse weitergegeben.

```
$ name=Tux                    # Zuweisung: KEINE Leerzeichen um das '='
$ echo $name                  # Wert lesen mit $
Tux
$ echo "${name}s Pinguin"     # geschweifte Klammern grenzen den Namen ab
$ export EDITOR=nano          # in die Umgebung exportieren
$ unset name                  # Variable loeschen
$ env                         # alle Umgebungsvariablen
$ set                         # alle Variablen und Funktionen
$ printenv HOME               # eine Variable anzeigen
```

Variable	Bedeutung
`HOME`	Heimatverzeichnis
`USER`, `LOGNAME`	Benutzername
`PATH`	Suchpfad für Programme
`PWD`, `OLDPWD`	Aktuelles und vorheriges Verzeichnis
`SHELL`	Login-Shell
`PS1`	Aussehen des Prompts
`LANG`, `LC_*`	Sprache und Zeichensatz
`EDITOR`, `VISUAL`	Bevorzugter Editor
`?` (als `$?`)	Rückgabewert des letzten Kommandos (0 = Erfolg)

Den `PATH` erweitert man so:

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

11.3. Quoting: Sonderzeichen schützen

Niveau: Einsteiger

Die Shell interpretiert Leerzeichen, `*`, `$`, `;`, `|`, `&`, `>`, `<`, `#` und andere Zeichen. Will man sie wörtlich verwenden, muss man sie *maskieren*.

```
$ echo "Hallo $USER"          # doppelte Anfuehrungszeichen: Variablen werden ersetzt
Hallo tux
$ echo 'Hallo $USER'          # einfache: alles bleibt woertlich
Hallo $USER
$ echo Hallo\ Welt            # Backslash maskiert genau ein Zeichen
$ echo "Preis: \$5"           # $ in doppelten Anfuehrungszeichen maskieren
$ touch "Meine Datei.txt"     # Leerzeichen im Namen -> Anfuehrungszeichen
$ rm Meine\ Datei.txt
```

> **Faustregel**
>
> Variablen **immer** in doppelte Anführungszeichen setzen (`"$datei"`). Sonst zerfällt ein Wert mit Leerzeichen in mehrere Argumente. Einfache Anführungszeichen für alles, was die Shell nicht anfassen soll (Regex-Muster, `awk`-Programme).

11.4. Kommandos verknüpfen

Niveau: Einsteiger

```
$ cd /tmp; ls; pwd              # nacheinander, unabhaengig vom Ergebnis
$ mkdir neu && cd neu           # zweites nur, wenn das erste erfolgreich war
$ ping -c1 host || echo "down"  # zweites nur, wenn das erste fehlschlug
$ grep -q root /etc/passwd && echo "ja" || echo "nein"
$ ( cd /tmp && ls )             # in Subshell: aktuelles Verzeichnis bleibt unveraendert
$ { echo a; echo b; } > out.txt # Gruppe: Ausgabe gemeinsam umgeleitet
$ ls -l \
    /etc                        # Zeilenfortsetzung mit Backslash
```

11.5. History, Alias und Vervollständigung

Niveau: Einsteiger

```
$ history | tail                # letzte Kommandos
$ !!                            # letztes Kommando wiederholen
$ sudo !!                       # ... mit sudo davor
$ !123                          # Kommando Nr. 123 der History
$ !grep                         # letztes mit 'grep' beginnendes Kommando
$ alias ll='ls -lah'            # Abkuerzung definieren
$ alias                         # alle Aliase
$ unalias ll
$ \ls                           # Alias umgehen
```

Die History wird beim Abmelden in `~/.bash_history` gespeichert. Ihre Größe steuern `HISTSIZE` und `HISTFILESIZE`; mit `HISTCONTROL=ignoreboth` werden Duplikate und Zeilen mit führendem Leerzeichen ausgelassen (nützlich bei Passwörtern in der Kommandozeile). Dennoch gilt: **Passwörter gehören nie in die Kommandozeile.**

11.6. Startdateien der bash

Niveau: Fortgeschritten

Datei	Wann?
`/etc/profile`, `/etc/profile.d/*.sh`	Systemweit, bei Login-Shells
`~/.profile` oder `~/.bash_profile`	Pro Benutzer, bei Login-Shells (Anmeldung, `su -`, ssh)
`/etc/bash.bashrc` (Debian) bzw. `/etc/bashrc` (RHEL)	Systemweit, bei interaktiven Shells
`~/.bashrc`	Pro Benutzer, bei jeder interaktiven Shell (Terminalfenster); hier gehören Aliase und Prompt hin
`~/.bash_logout`	Beim Beenden einer Login-Shell

```
$ source ~/.bashrc        # Datei in der aktuellen Shell neu einlesen (kurz: . ~/.bashrc)
$ export PS1='\u@\h:\w\$ '   # Prompt: Benutzer@Rechner:Verzeichnis$
```

11.7. Jobs: Vordergrund und Hintergrund

Niveau: Fortgeschritten

Die Shell kann mehrere Programme gleichzeitig verwalten.

```
$ sleep 300 &              # im Hintergrund starten
[1] 4711
$ jobs                     # Jobs dieser Shell
[1]+  Running    sleep 300 &
$ fg %1                    # Job 1 in den Vordergrund holen
(Strg+Z)                   # laufenden Job anhalten
$ bg %1                    # angehaltenen Job im Hintergrund weiterlaufen lassen
$ kill %1                  # Job beenden
$ nohup lange-aufgabe.sh & # laeuft weiter, auch wenn das Terminal geschlossen wird
$ disown %1                # Job von der Shell loesen
```

Für längere Sitzungen auf Servern eignen sich `tmux` oder `screen`: Sie laufen weiter, auch wenn die SSH-Verbindung abbricht, und man kann sich später wieder verbinden (`tmux attach`).

11.8. Shell-Optionen und Erweiterungen

Niveau: Experte

```
$ set -o noclobber         # '>' ueberschreibt keine bestehenden Dateien
$ set -o vi                # vi-Tasten in der Kommandozeile
$ shopt -s globstar        # ** fuer rekursive Muster: ls **/*.txt
$ shopt -s histappend      # History mehrerer Terminals zusammenfuehren
$ echo {1..5}              # 1 2 3 4 5
$ echo file{01..03}.txt    # file01.txt file02.txt file03.txt
$ echo $((3 * 7))          # Arithmetik: 21
$ diff <(ls a) <(ls b)     # Prozesssubstitution: Ausgaben wie Dateien vergleichen
$ cat <<EOF                # Here-Document
Zeile 1
Zeile 2
EOF
```

12. Arbeiten im Terminal: tmux, Locales und Zeit

Inhalt: Terminal-Multiplexer `tmux` und `screen`, Sitzungen auf entfernten Rechnern, Lokalisierung (Locales), Zeichensätze und Kodierung, Zeitzonen und Uhrzeit.

Stufe: [E] Einsteiger bis [F] Fortgeschritten

Lernziele: Nach diesem Kapitel können Sie

- mit `tmux` Sitzungen zu starten, zu trennen und wieder aufzunehmen,
- Sprache, Zeichensatz und Zeitzone korrekt einzustellen,
- Kodierungsprobleme (Umlaute!) zu erkennen und zu beheben.

Vorkenntnisse: Kapitel 2 und 11.

12.1. Terminal-Multiplexer tmux

Niveau: Einsteiger

Bei der Arbeit über SSH bricht eine laufende Sitzung ab, wenn die Verbindung getrennt wird – und damit auch laufende Programme. Ein **Terminal-Multiplexer** löst das Problem: Die Sitzung läuft auf dem Server weiter, man kann sich *trennen* (*detach*) und später wieder *anhängen* (*attach*). Zusätzlich lassen sich mehrere Fenster und Teilfenster in einem Terminal verwalten.

```
$ sudo apt install tmux           # bzw. dnf install tmux
$ tmux                            # neue Sitzung
$ tmux new -s wartung             # benannte Sitzung
$ tmux ls                         # Sitzungen auflisten
$ tmux attach -t wartung          # wieder anhaengen (kurz: tmux a -t wartung)
$ tmux kill-session -t wartung
```

Alle Tastenkombinationen beginnen mit dem **Präfix** `Ctrl-b`; danach folgt eine Taste:

Tasten	Wirkung
`Ctrl-b d`	Sitzung trennen (detach); Programme laufen weiter
`Ctrl-b c`	Neues Fenster
`Ctrl-b n` / `p` / `0..9`	Nächstes / vorheriges Fenster, Fenster nach Nummer
`Ctrl-b ,`	Fenster umbenennen
`Ctrl-b %`	Fenster senkrecht teilen
`Ctrl-b "`	Fenster waagerecht teilen
`Ctrl-b` Pfeiltaste	Zwischen Teilfenstern wechseln
`Ctrl-b z`	Teilfenster maximieren / zurück (Zoom)
`Ctrl-b x`	Teilfenster schließen
`Ctrl-b [`	Kopiermodus (Scrollen mit Pfeiltasten/Bild, `q` beendet)
`Ctrl-b ?`	Alle Tastenkürzel anzeigen
`Ctrl-b :`	Befehlszeile von tmux

```
# ~/.tmux.conf
set -g mouse on                       # Maus fuer Scrollen und Auswahl
set -g history-limit 50000
set -g base-index 1                   # Fenster ab 1 zaehlen
set -g prefix C-a                     # Praefix auf Ctrl-a aendern
unbind C-b
bind C-a send-prefix
bind | split-window -h                # intuitivere Teilung
bind - split-window -v
set -g status-style bg=blue,fg=white
```

Die Alternative `screen` (Präfix `Ctrl-a`; `screen -S name`, `screen -r name`, `Ctrl-a d`) ist auf vielen Systemen vorinstalliert. Für Skripte und lange Aufgaben ohne Multiplexer genügt auch `nohup befehl &` oder `systemd-run --user befehl`.

> **Praxis**
> Vor jeder längeren Wartung auf einem entfernten Server zuerst `tmux new -s wartung` starten. Bricht die Verbindung ab, gehen weder Programme noch Ausgabe verloren: erneut verbinden und `tmux a` eingeben.

12.2. Locales und Spracheinstellungen

Niveau: Einsteiger

Die **Locale** bestimmt Sprache der Meldungen, Zeichensatz, Sortierreihenfolge, Zahlen-, Datums- und Währungsformat. Sie wird über Umgebungsvariablen gesteuert:

Variable	Wirkung
`LANG`	Standardwert für alle Kategorien
`LC_ALL`	Überschreibt alles (nur zur Fehlersuche und in Skripten)
`LC_MESSAGES`	Sprache der Programmmeldungen
`LC_COLLATE`	Sortierreihenfolge
`LC_CTYPE`	Zeichenklassen (Groß-/Kleinbuchstaben), Kodierung
`LC_NUMERIC`, `LC_TIME`, `LC_MONETARY`	Zahlen-, Zeit-, Währungsformat
`LANGUAGE`	Rangfolge von Sprachen für Übersetzungen (GNU gettext)

```
$ locale                                    # aktuelle Einstellungen
$ locale -a                                 # installierte Locales
$ echo $LANG
de_DE.UTF-8
$ localectl status                          # systemweit (systemd)
$ sudo localectl set-locale LANG=de_DE.UTF-8
$ sudo localectl set-keymap de              # Tastaturbelegung (Konsole)
$ sudo localectl set-x11-keymap de          # Tastaturbelegung (X11)

# Debian/Ubuntu: Locale erzeugen
$ sudo dpkg-reconfigure locales             # oder:
$ sudo sed -i 's/^# *\(de_DE.UTF-8\)/\1/' /etc/locale.gen && sudo locale-gen
# RHEL/Fedora
$ sudo dnf install glibc-langpack-de
```

Skripte sollten robust gegen die Locale sein. Die Sortierung von `sort` und die Wirkung von Bereichen wie `[a-z]` hängen von der Locale ab. Maschinenlesbare Ausgaben und Vergleiche mit festen Texten laufen deshalb besser mit `LC_ALL=C`:

```
$ printf 'b\nA\na\nB\n' | sort              # de_DE: a A b B
$ printf 'b\nA\na\nB\n' | LC_ALL=C sort     # C:     A B a b
$ LC_ALL=C grep -c '[[:upper:]]' datei
$ LANG=C ls /nicht-da                       # englische Fehlermeldung (gut fuer Suchmaschinen
    )
```

12.3. Zeichensätze und Kodierung

Niveau: Fortgeschritten

Ein **Zeichensatz** ordnet Zeichen Nummern zu, die **Kodierung** legt fest, wie diese Nummern als Bytes gespeichert werden. Heute ist **UTF-8** der Standard: ASCII-Zeichen belegen ein Byte, Umlaute zwei, viele andere Zeichen drei oder vier. Ältere Texte liegen häufig in `ISO-8859-1` (Latin-1) oder `ISO-8859-15` bzw. `CP1252` (Windows) vor.

```
$ file -i brief.txt                          # Kodierung raten
brief.txt: text/plain; charset=iso-8859-1
$ iconv -f ISO-8859-1 -t UTF-8 brief.txt > brief-utf8.txt
$ iconv -l | head                            # unterstuetzte Kodierungen
$ echo "Gruesse" | iconv -f UTF-8 -t ASCII//TRANSLIT   # Umlaute umschreiben
$ recode ..UTF-8 datei                       # alternativ
$ dos2unix datei.txt                         # Windows-Zeilenenden (CRLF) -> LF
$ unix2dos datei.txt
$ hexdump -C datei | head                    # Bytes ansehen
$ wc -m datei; wc -c datei                   # Zeichen und Bytes
```

Typische Symptome falscher Kodierung: Aus einem Umlaut werden zwei unsinnige Zeichen, meist beginnend mit `Ä` mit Tilde (UTF-8 wurde als Latin-1 gelesen), oder der Umlaut wird zu einem Fragezeichen bzw. Rautenzeichen (Latin-1 wurde als UTF-8 gelesen). Hilfsmittel: richtige Kodierung beim Öffnen wählen, mit `iconv` umwandeln oder `LANG` anpassen.

12.4. Zeitzone und Uhrzeit

Niveau: Einsteiger

```
$ date                                       # aktuelle Zeit
$ date -u                                    # UTC
$ date +"%Y-%m-%d %H:%M:%S"                  # Format
$ date -d "next friday" +%F                  # relative Angaben
$ date -d @1700000000                        # Unix-Zeit -> Datum
$ date +%s                                   # Sekunden seit 1.1.1970
$ timedatectl                                # Zeit, Zeitzone, NTP-Status
$ timedatectl list-timezones | grep Berlin
$ sudo timedatectl set-timezone Europe/Berlin
$ sudo timedatectl set-ntp true              # Zeitsynchronisation aktivieren
$ hwclock --show                             # Hardware-Uhr
```

Server laufen häufig in **UTC**, damit Logs aus verschiedenen Zeitzonen vergleichbar sind und keine Probleme mit Sommerzeit entstehen. Zeitzonen-Daten liegen unter `/usr/share/zoneinfo`; `/etc/localtime` verweist auf die gewählte Zone. Für einzelne Programme setzt man die Zone mit `TZ=America/New_York date`. Die Zeit über das Netz synchronisiert `chrony` bzw. `systemd-timesyncd` (Kapitel 45).

Übungen und Kontrollfragen

1. Sie haben eine Datei mit Latin-1-Kodierung. Wie wandeln Sie sie nach UTF-8?

   Lösung: `iconv -f ISO-8859-1 -t UTF-8 alt.txt > neu.txt`; Kodierung vorab mit `file -i` prüfen.

2. Was bewirkt `Ctrl-b d` in tmux, und wie kehren Sie zurück?

   Lösung: Trennt die Sitzung; die Programme laufen weiter. Zurück mit `tmux attach` (`tmux a -t NAME`).

3. Warum verwendet man in Skripten teilweise `LC_ALL=C`?

   Lösung: Damit Sortierung, Zeichenklassen und Meldungen unabhängig von der Benutzereinstellung gleich und reproduzierbar sind.

4. Wie stellen Sie die Zeitzone eines Servers auf UTC?

   Lösung: `sudo timedatectl set-timezone UTC`.

5. In einem Dokument erscheinen statt der Umlaute jeweils zwei seltsame Zeichen. Was ist passiert?

   Lösung: UTF-8-Text wurde als Latin-1 interpretiert; mit passender Kodierung öffnen bzw. nicht doppelt umwandeln.

13. Shellskripte

Inhalt: Aufbau eines Skripts, Parameter, Bedingungen, Schleifen, Funktionen, Rückgabewerte, Fehlerbehandlung und Debugging.
Stufe: [F] Fortgeschritten
Lernziele: Nach diesem Kapitel können Sie
- ausführbare Skripte mit Shebang zu schreiben,
- Verzweigungen, Schleifen und Funktionen einzusetzen,
- Fehler abzufangen und Skripte zu testen.

Vorkenntnisse: Kapitel 8 bis 11.

Ein Shellskript ist eine Textdatei mit Kommandos, die nacheinander ausgeführt werden. Damit lassen sich wiederkehrende Aufgaben automatisieren: Sicherungen, Benutzeranlage, Systemprüfungen.

13.1. Das erste Skript

Niveau: Einsteiger

```
#!/bin/bash
# hallo.sh - erstes Skript
echo "Hallo $USER, heute ist $(date +%A)."
```

```
$ chmod +x hallo.sh        # ausfuehrbar machen
$ ./hallo.sh               # starten (aktuelles Verzeichnis muss angegeben werden)
$ bash hallo.sh            # alternativ: ohne Ausfuehrungsrecht ueber die Shell
```

Die erste Zeile `#!/bin/bash` (*Shebang*) legt den Interpreter fest. Zeilen mit `#` sind Kommentare. Skripte für Benutzer legt man in `~/bin` (Teil von `PATH` auf vielen Systemen) oder systemweit in `/usr/local/bin` ab.

13.2. Parameter und Variablen

Niveau: Einsteiger

Variable	Bedeutung
`$0`	Name des Skripts
`$1`, `$2`, …	Erstes, zweites, … Argument
`$#`	Anzahl der Argumente
`$@`	Alle Argumente, einzeln (*in Anführungszeichen verwenden*)
`$?`	Rückgabewert des letzten Kommandos
`$$`	Prozess-ID des Skripts

```
#!/bin/bash
# kopiere.sh - Datei mit Zeitstempel sichern
if [ $# -ne 1 ]; then
    echo "Aufruf: $0 <datei>" >&2
    exit 1
fi
cp -- "$1" "$1.$(date +%Y%m%d-%H%M)"
echo "Gesichert: $1"
```

Eingaben vom Benutzer liest `read`:

```
read -p "Name: " name
read -s -p "Passwort: " pw     # -s: Eingabe nicht anzeigen
echo "Hallo $name"
```

13.3. Verzweigungen

Niveau: Einsteiger

```
if [ -f /etc/os-release ]; then
    echo "Datei vorhanden"
elif [ -d /etc/os-release ]; then
    echo "ein Verzeichnis?"
else
    echo "fehlt"
fi

if [[ "$USER" == "root" && $# -gt 0 ]]; then  # [[ ]] ist die bash-Variante
    echo "root mit Argumenten"
fi

case "$1" in
    start)   echo "starte" ;;
    stop)    echo "stoppe" ;;
    restart|reload) echo "neu starten" ;;
    *)       echo "Aufruf: $0 {start|stop|restart}" ;;
esac
```

Test	Wahr, wenn ...
`-e datei`, `-f`, `-d`, `-L`	Datei existiert, reguläre Datei, Verzeichnis, Symlink
`-r`, `-w`, `-x`	lesbar, schreibbar, ausführbar
`-s datei`	Datei existiert und ist nicht leer
`-z "$v"`, `-n "$v"`	Zeichenkette leer / nicht leer
`"$a" = "$b"`, `!=`	Zeichenketten gleich / ungleich
`$a -eq $b`, `-ne`, `-lt`, `-le`, `-gt`, `-ge`	Zahlenvergleich
`!`, `-a`, `-o`	Nicht, und, oder (bei `[[ ]]`: `!`, `&&`, `||`)

13.4. Schleifen

Niveau: Einsteiger

```
for datei in *.txt; do
    echo "Verarbeite $datei"
    wc -l "$datei"
done

for i in {1..5}; do echo "Runde $i"; done

for ((i = 0; i < 3; i++)); do echo $i; done     # C-Stil (bash)

while read -r zeile; do                          # Datei zeilenweise lesen
    echo "-> $zeile"
done < liste.txt

zaehler=0
while [ $zaehler -lt 3 ]; do
    echo $zaehler
    zaehler=$((zaehler + 1))
done

until ping -c1 -W1 192.0.2.10 >/dev/null; do sleep 5; done    # warten bis erreichbar
```

`break` verlässt eine Schleife, `continue` springt zum nächsten Durchlauf.

13.5. Funktionen

Niveau: Fortgeschritten

```
log() {
    echo "[$(date +%T)] $*" >> /tmp/skript.log
}

summe() {
    local a=$1 b=$2           # 'local' begrenzt die Variable auf die Funktion
    echo $((a + b))           # Rueckgabe ueber Ausgabe
}

log "Start"
ergebnis=$(summe 3 4)
echo "Ergebnis: $ergebnis"
```

13.6. Rückgabewerte und Fehlerbehandlung

Niveau: Fortgeschritten

Jedes Kommando liefert einen Rückgabewert: `0` bedeutet Erfolg, alles andere Fehler. Skripte sollten bei Fehlern sauber abbrechen.

```
#!/bin/bash
set -euo pipefail       # -e: bei Fehler beenden, -u: undefinierte Variablen sind Fehler,
                        # -o pipefail: Fehler in Pipes nicht verschlucken
trap 'echo "Fehler in Zeile $LINENO" >&2' ERR
trap 'rm -f "$TMPDATEI"' EXIT      # Aufraeumen bei jedem Ende

TMPDATEI=$(mktemp)
if ! grep -q "^tux:" /etc/passwd; then
    echo "Benutzer fehlt" >&2
    exit 2
fi
```

13.7. Beispiel: Sicherungsskript

Niveau: Fortgeschritten

```
#!/bin/bash
# sicherung.sh - Verzeichnis als tar.gz sichern, alte Sicherungen loeschen
set -euo pipefail

QUELLE="${1:?Aufruf: $0 <verzeichnis>}"
ZIEL="/var/backups/mein"
BEHALTEN=7

mkdir -p "$ZIEL"
DATEI="$ZIEL/$(basename "$QUELLE")-$(date +%F).tar.gz"

tar -czf "$DATEI" -C "$(dirname "$QUELLE")" "$(basename "$QUELLE")"
echo "Sicherung erstellt: $DATEI ($(du -h "$DATEI" | cut -f1))"

# aeltere als $BEHALTEN Tage entfernen
find "$ZIEL" -name "*.tar.gz" -mtime +"$BEHALTEN" -delete
```

13.8. Debugging und gute Praxis

Niveau: Experte

```
$ bash -n skript.sh        # nur Syntax pruefen
$ bash -x skript.sh        # jede Zeile vor der Ausfuehrung anzeigen
$ shellcheck skript.sh     # statische Analyse (Paket 'shellcheck')
```

- Variablen immer in Anführungszeichen setzen (`"$var"`), sonst brechen Leerzeichen im Wert das Skript.
- `set -euo pipefail` an den Anfang, `mktemp` für temporäre Dateien, `local` in Funktionen.
- Skripte, die als `root` laufen, nie mit unvalidierten Eingaben füttern (Befehlseinschleusung).
- Kein Passwort im Skript; stattdessen Dateien mit Rechten `600` oder Geheimnisverwaltung.
- Für Portabilität `#!/bin/sh` nur nutzen, wenn keine bash-Erweiterungen nötig sind; `[[ ]]`, Arrays und `$(( ))`-Erweiterungen sind bash-spezifisch.
- Ab etwa 100 Zeilen oder komplexer Datenverarbeitung lohnt sich ein Wechsel zu Python.

14. Shellskripte vertieft

Inhalt: Funktionen, Arrays, Parameter-Expansion, Argumentverarbeitung mit `getopts`, Here-Documents, `trap`, Fehlerbehandlung, Debugging und Stilregeln.
Stufe: [F] Fortgeschritten bis [X] Experte
Lernziele: Nach diesem Kapitel können Sie

- robuste, wiederverwendbare Skripte mit Funktionen und Optionen zu schreiben,
- Arrays und die Parameter-Expansion der Bash einzusetzen,
- Fehler gezielt abzufangen und Skripte zu debuggen.

Vorkenntnisse: Kapitel 11 (Shell) und Kapitel 13 (Skripte).

14.1. Das Skriptgerüst

Niveau: Fortgeschritten

Ein guter Ausgangspunkt für jedes ernsthafte Skript:

```
#!/usr/bin/env bash
# backup.sh - sichert ein Verzeichnis
set -euo pipefail          # strikter Modus (siehe unten)
IFS=$'\n\t'                # sichere Worttrennung

readonly PROG=${0##*/}
readonly VERSION=1.0

usage() {
    cat <<EOF
Aufruf: $PROG [-v] [-o ZIEL] QUELLE...
  -v        ausfuehrlich
  -o ZIEL   Zielverzeichnis (Standard: /var/backups)
  -h        diese Hilfe
EOF
}

main() {
    ...
}

main "$@"
```

Strikter Modus:

Option	Wirkung
`set -e`	Skript endet beim ersten Befehl mit Fehlerstatus (mit Ausnahmen: Bedingungen, `&&`/`||`-Ketten)
`set -u`	Zugriff auf nicht gesetzte Variable ist ein Fehler
`set -o pipefail`	Pipeline liefert den Fehlerstatus des ersten fehlgeschlagenen Teils
`set -x`	Jeden Befehl vor der Ausführung ausgeben (Debugging)

Achtung

`set -e` ist kein Allheilmittel: Es greift nicht in `if`-Bedingungen, bei ODER-Ketten oder in Teilen einer Pipeline ohne `pipefail`. Kritische Schritte sollten ihren Rückgabewert dennoch explizit prüfen.

14.2. Funktionen

Niveau: Fortgeschritten

```
log() {                             # Argumente: $1 $2 ... wie bei Skripten
    local level=$1; shift           # lokale Variable
    printf '%s [%s] %s\n' "$(date +%T)" "$level" "$*" >&2
}

fehler() { log ERROR "$@"; exit 1; }

ist_zahl() {
    [[ $1 =~ ^[0-9]+$ ]]            # Rueckgabewert = Status des letzten Befehls
}

summe() {
    local s=0 x
    for x in "$@"; do s=$(( s + x )); done
    echo "$s"                       # "Rueckgabe" von Text per stdout
}

if ist_zahl "$1"; then log INFO "ok"; fi
ergebnis=$(summe 1 2 3)             # Befehlsersetzung faengt stdout ein
```

Funktionen *geben* Text über die Standardausgabe zurück und einen Status (0–255) über `return`. Variablen sind standardmäßig global; mit `local` bleiben sie in der Funktion. Funktionen können eine Bibliothek bilden: `source lib.sh` (oder `. lib.sh`) lädt sie in ein Skript.

14.3. Arrays und assoziative Arrays

Niveau: Fortgeschritten

```
dateien=(a.txt "mein bild.png" c.log)     # indiziertes Array
dateien+=(d.txt)                           # anhaengen
echo "${dateien[0]}"                       # erstes Element
echo "${dateien[@]}"                       # alle Elemente (korrekt gequotet)
echo "${#dateien[@]}"                      # Anzahl
echo "${!dateien[@]}"                      # Indizes
echo "${dateien[@]:1:2}"                   # Ausschnitt: ab 1, zwei Elemente
for f in "${dateien[@]}"; do echo "-> $f"; done
unset 'dateien[1]'

declare -A port                            # assoziatives Array (Bash >= 4)
port[ssh]=22; port[http]=80; port[https]=443
echo "${port[ssh]}"
for name in "${!port[@]}"; do echo "$name = ${port[$name]}"; done

mapfile -t zeilen < datei.txt              # Datei zeilenweise in ein Array
```

Wichtig: Immer `"${array[@]}"` mit Anführungszeichen verwenden, sonst zerfallen Elemente mit Leerzeichen.

14.4. Parameter-Expansion

Niveau: Fortgeschritten

Die Parameter-Expansion ersetzt viele Aufrufe von `sed`, `cut` und `basename`:

Ausdruck	Bedeutung	Beispiel
`${v:-x}`	Wert, sonst Vorgabe `x`	`${PORT:-80}`
`${v:=x}`	wie oben, setzt `v` zusätzlich	
`${v:?msg}`	Fehler mit Meldung, falls leer	`${1:?Datei fehlt}`
`${#v}`	Länge	`${#name}`
`${v#muster}`	kürzester Präfix entfernen	
`${v##muster}`	längster Präfix entfernen	`${pfad##*/}` = Dateiname
`${v%muster}`	kürzesten Suffix entfernen	`${f%.txt}`
`${v%%muster}`	längsten Suffix entfernen	`${f%%.*}`
`${v/alt/neu}`	erste Ersetzung	
`${v//alt/neu}`	alle Ersetzungen	
`${v:2:3}`	Teilstring ab Position 2, Länge 3	
`${v^^}` / `${v,,}`	Groß- / Kleinschreibung	

```
f=/home/anna/bericht.final.txt
echo "${f##*/}"        # bericht.final.txt   (basename)
echo "${f%/*}"         # /home/anna          (dirname)
echo "${f%.*}"         # /home/anna/bericht.final
echo "${f##*.}"        # txt                 (Endung)
for bild in *.jpeg; do mv -- "$bild" "${bild%.jpeg}.jpg"; done
```

14.5. Optionen mit getopts

Niveau: Fortgeschritten

```
verbose=0; ziel=/var/backups
while getopts ":vo:h" opt; do
    case $opt in
        v) verbose=1 ;;
        o) ziel=$OPTARG ;;
        h) usage; exit 0 ;;
        :) echo "Option -$OPTARG braucht ein Argument" >&2; exit 2 ;;
        \?) echo "Unbekannte Option -$OPTARG" >&2; usage >&2; exit 2 ;;
    esac
done
shift $((OPTIND - 1))             # uebrig bleiben die Positionsargumente
[[ $# -ge 1 ]] || { usage >&2; exit 2; }
```

Ein Doppelpunkt hinter einem Buchstaben bedeutet “Option mit Argument”; ein führender Doppelpunkt schaltet die eigene Fehlerbehandlung ein. Lange Optionen (`--verbose`) beherrscht das externe Programm `getopt`; für einfache Skripte reicht oft eine `case`-Schleife über `$1` mit `shift`.

14.6. Here-Documents und Here-Strings

Niveau: Fortgeschritten

```
cat <<EOF > /etc/motd                    # Variablen werden ersetzt
Willkommen auf $(hostname)
EOF

cat <<'EOF'                              # in Anfuehrungszeichen: nichts wird ersetzt
Preis: $5, Befehl: $(nicht ausgefuehrt)
EOF

cat <<-EOF                               # fuehrende TABs werden entfernt
        eingerueckt im Skript
        EOF

grep -c fehler <<< "$text"               # Here-String
```

14.7. Fehlerbehandlung und trap

Niveau: Fortgeschritten

```
tmp=$(mktemp -d)                         # sicheres temporaeres Verzeichnis
cleanup() { rm -rf -- "$tmp"; }
trap cleanup EXIT                        # immer beim Beenden
trap 'echo "Abbruch" >&2; exit 130' INT TERM
trap 'echo "Fehler in Zeile $LINENO" >&2' ERR

cp "$quelle" "$tmp/" || fehler "Kopieren fehlgeschlagen"

command -v rsync >/dev/null || fehler "rsync fehlt"
[[ -r $konfig ]] || fehler "Konfig nicht lesbar: $konfig"
```

Nützliche Regeln:

- Variablen **immer** in Anführungszeichen: `"$datei"`.
- Mit `--` das Ende der Optionen markieren: `rm -- "$datei"`.
- Für Bedingungen `[[ ...]]` statt `[ ...]` verwenden (Bash).
- Temporäre Dateien nur mit `mktemp`, nie mit festen Namen in `/tmp`.
- Eingaben (Argumente, Dateinamen, Umgebung) nie ungeprüft an `eval` oder `sh -c` übergeben.
- Exit-Codes sinnvoll wählen: 0 Erfolg, 1 allgemeiner Fehler, 2 Aufruffehler.

14.8. Debugging

Niveau: Fortgeschritten

```
$ bash -n skript.sh              # nur Syntax pruefen
$ bash -x skript.sh              # jede Zeile vor Ausfuehrung zeigen
$ bash -x skript.sh 2> trace.log
$ PS4='+ ${BASH_SOURCE}:${LINENO}: ' bash -x skript.sh    # Zeilennummern im Trace
$ shellcheck skript.sh           # statische Analyse (sehr empfehlenswert)
```

Im Skript lässt sich der Trace gezielt ein- und ausschalten: `set -x` ... `set +x`. `shellcheck` findet typische Fehler (fehlende Anführungszeichen, Tippfehler in Variablen, unsichere Konstrukte) und gehört zum Handwerkszeug.

14.9. Eine praktische Beispielfunktion: Prozesse parallel ausführen

Niveau: Experte

```
# Alle Hosts aus hosts.txt parallel pingen, maximal 8 gleichzeitig
check() { ping -c1 -W1 "$1" >/dev/null && echo "$1 up" || echo "$1 DOWN"; }
export -f check
xargs -a hosts.txt -P 8 -I{} bash -c 'check {}'

# oder mit GNU parallel
parallel -j 8 check ::: $(cat hosts.txt)
```

Übungen und Kontrollfragen

1. Wie ermitteln Sie in einem Skript den Dateinamen ohne Pfad und Endung aus `/tmp/a/b.tar.gz` ohne externe Programme?

   Lösung: `f=/tmp/a/b.tar.gz; n=${f##*/}; n=${n%%.*}` ergibt `b`. (Mit `%.*` bliebe `b.tar`.)

2. Was bewirkt `trap cleanup EXIT`?

   Lösung: Die Funktion `cleanup` läuft beim Beenden des Skripts – bei normalem Ende, `exit` und bei den meisten Fehlern – und eignet sich zum Aufräumen temporärer Dateien.

3. Schreiben Sie ein Skript, das mit `-n ZAHL` die Anzahl der auszugebenden Zeilen festlegt (Standard 10).

   Lösung: `n=10; while getopts ":n:" o; do [[ $o == n ]] && n=$OPTARG; done; shift $((OPTIND-1)); head -n "$n" "$@"`.

4. Warum ist `for f in $(ls)` problematisch?

   Lösung: Die Ausgabe wird an Leerzeichen und Zeilenumbrüchen zerlegt; Dateinamen mit solchen Zeichen brechen. Besser `for f in *` oder `find ... -print0 | xargs -0`.

5. Wie finden Sie heraus, in welcher Zeile ein Skript scheitert?

   Lösung: Mit `bash -x` (und `PS4` mit `$LINENO`) oder `trap '... $LINENO' ERR`; zusätzlich `shellcheck`.

15. Archivieren und Komprimieren

Inhalt: Archive mit `tar`, Kompression (`gzip`, `bzip2`, `xz`, `zstd`), `zip`, Prüfsummen und Integrität.
Stufe: [E] Einsteiger bis [F] Fortgeschritten
Lernziele: Nach diesem Kapitel können Sie

- Archive zu erstellen, anzuzeigen und gezielt zu entpacken,
- geeignete Kompressionsverfahren auszuwählen,
- Prüfsummen zur Integritätskontrolle zu verwenden.

Vorkenntnisse: Kapitel 7.

Archivieren fasst viele Dateien samt Verzeichnisstruktur und Rechten zu einer einzigen Datei zusammen. *Komprimieren* verkleinert Daten. Unter Linux sind beide Aufgaben getrennt: `tar` archiviert, `gzip`, `bzip2` oder `xz` komprimieren. Die Dateiendungen zeigen die Kombination an, z. B. `.tar.gz` oder `.tgz`.

15.1. tar

Niveau: Einsteiger

```
$ tar -cvf projekt.tar projekt/         # c: create, v: verbose, f: Dateiname
$ tar -czvf projekt.tar.gz projekt/     # zusaetzlich mit gzip komprimieren (z)
$ tar -cJf projekt.tar.xz projekt/      # mit xz (J), bessere Kompression
$ tar -tvf projekt.tar.gz               # Inhalt anzeigen (t: list)
$ tar -xvf projekt.tar.gz               # entpacken (x: extract); Format wird erkannt
$ tar -xf projekt.tar.gz -C /tmp        # in anderes Verzeichnis entpacken
$ tar -xf projekt.tar.gz projekt/README   # nur eine Datei entpacken
$ tar -czf etc-$(date +%F).tar.gz -C / etc   # relative Pfade (ohne fuehrendes /)
$ tar --exclude='*.log' -czf bak.tar.gz proj/    # Dateien ausschliessen
```

Option	Bedeutung
`c` / `x` / `t`	Archiv erzeugen / entpacken / Inhalt auflisten
`f datei`	Archivdatei (– steht für Standardein-/ausgabe)
`v`	Ausführliche Ausgabe
`z`, `j`, `J`	Kompression mit gzip, bzip2, xz
`C verz`	Vor dem Ausführen in das Verzeichnis wechseln
`p`	Dateirechte beim Entpacken erhalten (als `root` Standard)

Tipp

Archive sollten immer **ein einziges Verzeichnis** an der Spitze enthalten (`projekt/...`), sonst verstreuen sich die Dateien beim Entpacken im aktuellen Verzeichnis (*tar bomb*). Vorher mit `tar -tf` prüfen, bevor man ein fremdes Archiv entpackt.

15.2. gzip, bzip2 und xz

Niveau: Einsteiger

```
$ gzip datei.txt             # -> datei.txt.gz (Original wird ersetzt)
$ gzip -k datei.txt          # Original behalten
$ gunzip datei.txt.gz        # entpacken
$ gzip -9 datei.txt          # staerkste Kompression (1 = schnell, 9 = klein)
$ zcat datei.txt.gz | head   # Inhalt ansehen, ohne zu entpacken
$ zgrep fehler log.gz        # in komprimierter Datei suchen
$ bzip2 datei                # -> datei.bz2 (bunzip2 zum Entpacken)
$ xz datei                   # -> datei.xz  (unxz zum Entpacken)
$ xz -T0 -9 grosse.iso       # alle CPU-Kerne nutzen
```

Werkzeug	Endung	Geschwindigkeit	Merkmal
`gzip`	`.gz`	schnell	Standard, überall verfügbar
`bzip2`	`.bz2`	langsamer	Etwas besser als gzip
`xz`	`.xz`	am langsamsten	Beste Kompression, hoher Speicherbedarf
`zstd`	`.zst`	sehr schnell	Moderner Kompromiss (Paket `zstd`)

15.3. zip und unzip

Niveau: Einsteiger

Für den Datenaustausch mit Windows eignet sich `zip`, das Archivieren und Komprimieren in einem Schritt erledigt.

```
$ zip -r projekt.zip projekt/       # Verzeichnis rekursiv packen
$ zip -e geheim.zip dokument.pdf    # mit Passwort (schwache Verschluesselung!)
$ unzip projekt.zip                 # entpacken
$ unzip -l projekt.zip              # Inhalt anzeigen
$ unzip projekt.zip -d /tmp/ziel    # in Zielverzeichnis
```

15.4. Weitere Werkzeuge

Niveau: Fortgeschritten

```
$ rsync -av --delete quelle/ ziel/              # Verzeichnisse abgleichen (effizient)
$ rsync -avz -e ssh quelle/ tux@server:/backup/ # ueber SSH
$ dd if=/dev/sdb of=sicherung.img bs=4M status=progress   # Datentraeger-Abbild
$ cpio -o < liste > archiv.cpio                 # Archivformat (initramfs, rpm)
```

> **Hinweis**
>
> `rsync` unterscheidet `quelle` und `quelle/`: Mit Schrägstrich am Ende wird der *Inhalt* des Verzeichnisses kopiert, ohne das Verzeichnis selbst. `--delete` löscht am Ziel alles, was in der Quelle fehlt; vor dem ersten Einsatz mit `-n` (*dry run*) testen.

16. Versionsverwaltung mit Git

Inhalt: Repositories, Commits, Branches, Merge und Rebase, Remote-Repositories, Konfliktlösung, Workflows, nützliche Hilfsmittel.

Stufe: [E] Einsteiger bis [F] Fortgeschritten

Lernziele: Nach diesem Kapitel können Sie

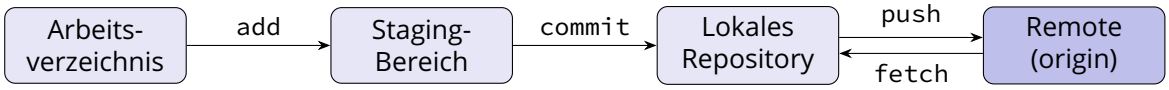
- ein Repository anzulegen und Änderungen nachvollziehbar zu committen,
- mit Branches, Merge und Remotes zu arbeiten,
- Konflikte zu lösen und Fehler rückgängig zu machen.

Vorkenntnisse: Kapitel 6 und 7 (Dateisystem, Dateien) sowie Kapitel 4 (Editor).

16.1. Warum Versionsverwaltung?

Niveau: Einsteiger

Git ist ein verteiltes Versionsverwaltungssystem, das von Linus Torvalds für die Entwicklung des Linux-Kernels geschaffen wurde. Es speichert den Verlauf von Dateien, erlaubt paralleles Arbeiten in Zweigen und macht Änderungen nachvollziehbar. Für Administratoren ist Git längst unverzichtbar: Konfigurationsdateien, Skripte, Ansible-Playbooks und Dokumentation gehören in ein Repository.



16.2. Erste Schritte

Niveau: Einsteiger

```
$ git config --global user.name  "Anna Beispiel"
$ git config --global user.email "anna@example.org"
$ git config --global init.defaultBranch main
$ git config --global core.editor vim
$ git config --global pull.rebase false         # Merge-Strategie beim Pull

$ mkdir projekt && cd projekt
$ git init                                      # neues Repository
$ echo "# Projekt" > README.md
$ git status                                    # Zustand ansehen
$ git add README.md                             # in den Staging-Bereich
$ git commit -m "Erste Version"                 # Commit anlegen
$ git log --oneline                             # Verlauf kurz
$ git diff                                      # Aenderungen gegenueber dem Index
$ git diff --staged                             # Staging-Bereich gegenueber HEAD
```

Ein **Commit** ist ein Schnappschuss des gesamten Projekts mit Autor, Zeitpunkt und Nachricht. Gute Commit-Nachrichten haben eine kurze Betreffzeile (bis ca. 50 Zeichen) im Imperativ, ggf. gefolgt von einer Leerzeile und einer Erläuterung *warum* etwas geändert wurde.

```
$ git add -p                  # Aenderungen stueckweise auswaehlen
$ git commit -a -m "Text"     # alle bekannten, geaenderten Dateien committen
$ git commit --amend          # letzten Commit nachbessern (nur lokal!)
$ git show HEAD               # letzten Commit ansehen
$ git log --graph --oneline --all --decorate    # Verlauf als Graph
$ git log -p datei            # Aenderungen an einer Datei
$ git blame datei             # wer hat welche Zeile geaendert?
```

Dateien, die nicht versioniert werden sollen (Build-Ergebnisse, Passwörter, temporäre Dateien), gehören in die Datei `.gitignore`:

```
*.log
*.swp
build/
.env
secrets/
```

Sicherheit

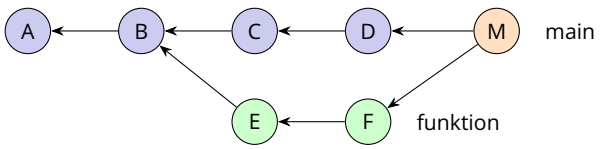
Passwörter, private Schlüssel und Tokens gehören **nicht** in ein Repository – auch nicht in private. Einmal committete Geheimnisse bleiben im Verlauf und müssen als kompromittiert gelten; sie sind zu ersetzen. Hilfsmittel: `.gitignore`, `git-crypt`, `sops`, Secret-Manager.

16.3. Branches und Merge

Niveau: Einsteiger

Ein **Branch** ist ein beweglicher Zeiger auf einen Commit. Zweige sind billig, deshalb legt man für jede Aufgabe einen eigenen an.

```
$ git branch                      # Zweige auflisten
$ git switch -c neue-funktion     # neuen Zweig anlegen und wechseln
$ ... editieren, git add, git commit ...
$ git switch main
$ git merge neue-funktion         # Zweig in main einarbeiten
$ git branch -d neue-funktion     # Zweig loeschen
```



Der Merge-Commit `M` hat zwei Eltern. War `main` seit der Abzweigung nicht verändert, genügt ein *Fast-Forward*: Der Zeiger wird einfach vorgeschoben. Alternativ kann man den eigenen Zweig mit `git rebase main` auf die aktuelle Spitze von `main` setzen; das ergibt eine lineare Historie, schreibt aber die eigenen Commits neu.

Faustregel

Bereits veröffentlichte (gepushte) Commits niemals mit `rebase` oder `commit --amend` umschreiben – das erzeugt Probleme für alle, die den alten Verlauf haben.

16.4. Remote-Repositories

Niveau: Einsteiger

```
$ git clone https://git.example.org/team/projekt.git
$ git clone git@git.example.org:team/projekt.git     # per SSH
$ git remote -v                          # Remotes anzeigen
$ git remote add origin URL              # Remote hinzufuegen
$ git fetch                              # Aenderungen holen, nichts veraendern
$ git pull                               # fetch + merge
$ git push -u origin main                # hochladen, Tracking einrichten
$ git push                               # danach reicht das
$ git switch -c fix origin/fix           # fremden Zweig lokal verfolgen
```

Zugang per SSH funktioniert mit Schlüsseln (siehe Kapitel 42) und ist bequemer und sicherer als Passwörter. Eigene Git-Server: GitLab, Gitea/Forgejo, oder einfach ein Verzeichnis mit `git init --bare` auf einem Server, das über SSH erreichbar ist.

16.5. Konflikte lösen

Niveau: Fortgeschritten

Wenn zwei Zweige dieselbe Stelle unterschiedlich ändern, kann Git nicht automatisch zusammenführen:

```
$ git merge funktion
Auto-merging config.txt
CONFLICT (content): Merge conflict in config.txt
$ git status                         # zeigt "both modified"
$ vim config.txt
<<<<<<< HEAD
port = 8080
=======
port = 9090
>>>>>>> funktion
```

Zwischen `<<<<<<<` und `=======` steht der Stand des aktuellen Zweigs, danach der des anderen. Man editiert die Datei zum gewünschten Ergebnis, entfernt die Markierungen und schließt ab:

```
$ git add config.txt
$ git commit                         # Merge abschliessen
$ git merge --abort                  # oder: Merge abbrechen
$ git mergetool                      # grafisches Werkzeug (z. B. meld, vimdiff)
```

16.6. Änderungen rückgängig machen

Niveau: Fortgeschritten

Situation	Befehl
Datei im Arbeitsverzeichnis verwerfen	`git restore datei`
Datei aus dem Staging nehmen	`git restore --staged datei`
Letzten Commit zurücknehmen, Änderungen behalten	`git reset --soft HEAD~1`
Letzten Commit samt Änderungen verwerfen	`git reset --hard HEAD~1` (gefährlich!)
Veröffentlichten Commit rückgängig machen	`git revert <Commit>` (neuer Gegen-Commit)
Alte Version einer Datei holen	`git restore --source=HEAD~3 datei`
Verlorenen Commit wiederfinden	`git reflog`, dann `git switch -c rettung <Hash>`
Arbeit kurz zur Seite legen	`git stash`, später `git stash pop`

`git reflog` protokolliert alle Bewegungen von `HEAD` und ist die Lebensversicherung bei Fehlern: Auch nach einem `reset --hard` sind die Commits für einige Wochen noch auffindbar.

16.7. Tags, Suche und nützliche Hilfsmittel

Niveau: Fortgeschritten

```
$ git tag -a v1.0 -m "Release 1.0"     # annotierte Markierung
$ git push --tags
$ git describe --tags                  # naechstgelegener Tag
$ git log --since="2 weeks ago" --author=anna --oneline
$ git log -S"passwort" --oneline       # Commits, die eine Zeichenfolge aendern
$ git grep "TODO"                      # in versionierten Dateien suchen
$ git bisect start; git bisect bad; git bisect good v1.0   # Fehler per Bisektion finden
$ git config --global alias.lg "log --graph --oneline --all --decorate"
```

16.8. Typische Workflows

Niveau: Fortgeschritten

- **Trunk-based**: Kurze Zweige, schnell zurück nach `main`; ideal für kleine Teams und Infrastruktur-Repositories.
- **Feature-Branch + Pull/Merge Request**: Jede Änderung in eigenem Zweig, Prüfung (*Review*) auf der Plattform, danach Merge. Standard bei GitHub/GitLab.
- **Gitflow**: Zusätzliche Zweige `develop`, `release`, `hotfix`; für Software mit festen Releases.

Git für die Systemadministration: Mit `git init` in `/etc` (oder dem Werkzeug `etckeeper`) wird jede Konfigurationsänderung nachvollziehbar. Skripte, Dokumentation und Automatisierung (Kapitel 56) lassen sich per Pull Request prüfen und bei Problemen schnell zurückrollen.

Übungen und Kontrollfragen

1. Was ist der Unterschied zwischen `git fetch` und `git pull`?

   Lösung: `fetch` lädt neue Commits, ohne den Arbeitsstand zu ändern; `pull` ist `fetch` plus Merge (bzw. Rebase) in den aktuellen Zweig.

2. Sie haben versehentlich einen Commit mit einem Passwort angelegt, aber noch nicht gepusht. Was tun?

   Lösung: Datei korrigieren und `git commit --amend`; Passwort zusätzlich ändern. Wurde schon gepusht, gilt das Passwort als kompromittiert und ist zu ersetzen.

3. Wie starten Sie ein neues Repository, legen einen Zweig `test` an und wechseln dorthin?

   Lösung: `git init && git switch -c test` (nach dem ersten Commit; vorher legt `switch -c` den Zweig ebenfalls an).

4. Wie finden Sie heraus, wer eine bestimmte Zeile zuletzt geändert hat?

   Lösung: `git blame datei`, ggf. mit `-L Start,Ende`.

5. Wie holen Sie eine gelöschte Datei aus dem Verlauf zurück?

   Lösung: `git log -- datei`, dann `git restore --source=<Hash>^ -- datei` (der Commit vor dem Löschen).

17. Software aus Quellen und Programmierwerkzeuge

Inhalt: Compiler `gcc`, Bibliotheken, `make` und Makefiles, Autotools, CMake, Installation unter `/usr/local`, Pakete selbst bauen, Debugging-Werkzeuge.

Stufe: [F] Fortgeschritten bis [X] Experte

Lernziele: Nach diesem Kapitel können Sie

- ein Programm aus dem Quelltext zu übersetzen und zu installieren,
- einfache Makefiles zu lesen und zu schreiben,
- fehlende Abhängigkeiten zu erkennen und zu beheben.

Vorkenntnisse: Kapitel 7, Kapitel 11 sowie Kapitel 19 (Paketverwaltung).

17.1. Warum aus Quellen bauen?

Niveau: Fortgeschritten

In der Regel installiert man Software über die Paketverwaltung (Kapitel 19). Aus dem Quelltext baut man, wenn

- eine neuere Version benötigt wird als die Distribution bietet,
- Optionen aktiviert oder abgeschaltet werden sollen,
- ein Patch eingespielt werden soll,
- Software für eine andere Architektur gebaut wird,
- man verstehen möchte, wie Open-Source-Software entsteht.

Selbstgebaute Software sollte nie nach `/usr` installiert werden: Die Paketverwaltung weiß davon nichts, und Dateien könnten kollidieren. Der vorgesehene Platz ist `/usr/local` bzw. `/opt`.

17.2. Die Werkzeugkette

Niveau: Fortgeschritten

```
# Debian/Ubuntu
$ sudo apt install build-essential        # gcc, g++, make, libc-dev
# RHEL/Fedora
$ sudo dnf groupinstall "Development Tools"
# openSUSE
$ sudo zypper install -t pattern devel_basis

$ gcc --version
$ make --version
```

Ein C-Programm übersetzen:

```
$ cat hallo.c
#include <stdio.h>
int main(void) {
    printf("Hallo Welt\n");
    return 0;
}
$ gcc -Wall -O2 -o hallo hallo.c          # Quelltext -> Programm
$ ./hallo
Hallo Welt
$ gcc -c modul.c                          # nur zu Objektdatei (modul.o)
$ gcc -o prog main.o modul.o -lm          # linken, Mathe-Bibliothek
$ ldd ./hallo                             # benoetigte Bibliotheken
$ file hallo
hallo: ELF 64-bit LSB pie executable, x86-64 ...
```

Die Schritte heißen *Präprozessor*, *Compiler*, *Assembler* und *Linker*. Bibliotheken liegen als statische Archive (`.a`) oder dynamische Objekte (`.so`) unter `/usr/lib` bzw. `/usr/lib64`; Header-Dateien (`.h`) unter `/usr/include`. Die Header zu Bibliotheken stecken in Entwicklungspaketen mit Namen wie `libssl-dev` (Debian) oder `openssl-devel` (RHEL).

17.3. Der klassische Ablauf: configure, make, make install

Niveau: Fortgeschritten

```
$ wget https://example.org/downloads/tool-2.1.tar.gz
$ sha256sum tool-2.1.tar.gz               # Pruefsumme mit Angabe des Projekts vergleichen
$ gpg --verify tool-2.1.tar.gz.asc        # oder Signatur pruefen
$ tar xzf tool-2.1.tar.gz && cd tool-2.1
$ less README INSTALL                     # lesen!
$ ./configure --prefix=/usr/local --enable-ssl
$ make -j$(nproc)                         # parallel bauen
$ make check                              # Tests, falls vorhanden
$ sudo make install                       # installieren
$ sudo ldconfig                           # Bibliotheks-Cache aktualisieren
```

`./configure` prüft Abhängigkeiten und erzeugt Makefiles. Fehlt etwas, bricht es mit einer Meldung ab (“`configure: error: library foo not found`”). Dann das passende `-dev`/`-devel`-Paket installieren und erneut starten.

Meldung	Wahrscheinliche Ursache
`fatal error: xyz.h: No such file`	Header fehlt: Entwicklungspaket installieren
`cannot find -lxyz`	Bibliothek oder deren Entwicklungspaket fehlt
`command not found: autoreconf`	Autotools fehlen: `autoconf automake libtool`
`error while loading shared libraries`	`ldconfig` nicht ausgeführt oder Pfad fehlt in `/etc/ld.so.conf.d/`
`Permission denied` bei `make install`	Mit `sudo` installieren oder `--prefix=$HOME/.local` wählen

> **Tipp**
>
> Mit `make install DESTDIR=/tmp/stage` lässt sich in ein Probeverzeichnis installieren. So sieht man, welche Dateien entstehen. Praktisch ist auch `checkinstall` (erzeugt ein Paket) oder `stow`: Software unter `/usr/local/stow/tool-2.1` installieren und per `stow` verlinken; Deinstallation per `stow -D`.

17.4. Makefiles

Niveau: Fortgeschritten

Ein **Makefile** beschreibt Abhängigkeiten zwischen Dateien und die Befehle, um sie zu erzeugen. `make` baut nur, was veraltet ist.

```
# Makefile  (Einrueckung: TAB, keine Leerzeichen!)
CC      = gcc
CFLAGS  = -Wall -O2
OBJ     = main.o modul.o

prog: $(OBJ)
        $(CC) -o $@ $(OBJ) -lm

%.o: %.c
        $(CC) $(CFLAGS) -c $< -o $@

.PHONY: clean install
clean:
        rm -f prog $(OBJ)

install: prog
        install -D -m 755 prog $(DESTDIR)/usr/local/bin/prog
```

Aufbau einer Regel: `Ziel: Voraussetzungen`, darunter mit Tabulator die Befehle (*Rezept*). Spezielle Variablen: `$@` (Ziel), `$<` (erste Voraussetzung), `$^` (alle). `.PHONY` kennzeichnet Ziele, die keine Dateien sind. Aufruf: `make`, `make clean`, `make -j4`, `make VAR=wert`.
Auch außerhalb von C sind Makefiles als Aufgabenrunner beliebt:

```
.PHONY: backup test deploy
backup:
        rsync -a /srv/daten/ backup:/srv/backup/
test:
        shellcheck *.sh
deploy: test
        scp skript.sh server:/usr/local/bin/
```

17.5. Autotools, CMake und andere Bausysteme

Niveau: Experte

System	Typischer Ablauf	Erkennung
Autotools	`./configure && make && make install`	`configure`, `Makefile.in`
CMake	`cmake -B build && cmake --build build && cmake --install build`	`CMakeLists.txt`
Meson	`meson setup build && ninja -C build`	`meson.build`
Python	`python3 -m venv venv; pip install .`	`pyproject.toml`, `setup.py`
Go	`go build ./...`	`go.mod`
Rust	`cargo build --release`	`Cargo.toml`
Node.js	`npm install && npm run build`	`package.json`

Bei Git-Quellen fehlt oft das `configure`-Skript; es wird mit `./autogen.sh` oder `autoreconf -i` erzeugt.

17.6. Pakete selbst bauen

Niveau: Experte

Wer Software verteilen will, baut besser ein Paket statt `make install` zu verwenden. Details zur Paketverwaltung stehen in Kapitel 19. Kurz:

```
# Debian: Quellpaket holen und neu bauen
$ sudo apt build-dep nginx
$ apt source nginx && cd nginx-*/
$ dpkg-buildpackage -us -uc -b
# RHEL: Source-RPM neu bauen
$ rpmbuild --rebuild paket-1.0-1.src.rpm
# einfach: fpm erzeugt Pakete aus einem Verzeichnis
$ fpm -s dir -t deb -n meintool -v 1.0 --prefix /usr/local/bin ./meintool
```

17.7. Werkzeuge zur Fehleranalyse

Niveau: Experte

```
$ strace -f -o trace.txt ./prog          # Systemaufrufe verfolgen
$ strace -e trace=openat ./prog          # nur Dateizugriffe
$ ltrace ./prog                          # Bibliotheksaufrufe
$ gdb ./prog core                        # Debugger, Absturzanalyse
$ valgrind --leak-check=full ./prog      # Speicherfehler finden
$ nm -D /usr/lib/libfoo.so | grep funk   # Symbole einer Bibliothek
$ readelf -d ./prog | grep NEEDED        # benoetigte Bibliotheken
```

`strace` ist für Administratoren besonders wertvoll: Wenn ein Programm ohne Fehlermeldung scheitert, zeigt `strace` oft, welche Datei es vergeblich zu öffnen versuchte (`ENOENT`) oder welche Rechte fehlen (`EACCES`).

Übungen und Kontrollfragen

1. Warum installiert man selbst gebaute Programme nicht nach `/usr`?

   Lösung: Dort verwaltet die Paketverwaltung Dateien; selbst installierte Dateien könnten mit Paketen kollidieren oder bei Updates überschrieben werden. `/usr/local` und `/opt` sind dafür reserviert.

2. `./configure` meldet, dass `libpng` nicht gefunden wird. Was tun?

   Lösung: Entwicklungspaket installieren (`libpng-dev` bzw. `libpng-devel`) und `configure` erneut ausführen.

3. Wozu dient `.PHONY` in einem Makefile?

   Lösung: Es kennzeichnet Ziele ohne zugehörige Datei (z. B. `clean`), damit sie immer ausgeführt werden und nicht mit gleichnamigen Dateien verwechselt werden.

4. Ein Programm startet nicht und meldet nur “Datei nicht gefunden”. Wie finden Sie die Datei?

   Lösung: `strace -f -e trace=openat,stat ./prog 2>&1 | grep ENOENT`.

5. Wie installieren Sie ein Programm ohne Root-Rechte?

   Lösung: Mit `./configure --prefix=$HOME/.local`, `make`, `make install`; `$HOME/.local/bin` in den `PATH` aufnehmen.

Teil II.

Systemadministration

18. Systemadministration

Inhalt: Rolle des Administrators, `sudo`, Konfiguration, Systeminformationen, Sprache und Zeit, Kernelparameter (Paketverwaltung siehe Kapitel 19).

Stufe: [E] Einsteiger bis [F] Fortgeschritten

Lernziele: Nach diesem Kapitel können Sie

- administrative Aufgaben sicher mit `sudo` auszuführen,
- Konfigurationsdateien sicher zu ändern und Systeminformationen abzufragen,
- Systemsprache und Zeit einzustellen,
- Systemversion, Kernel und Hardware zu ermitteln.

Vorkenntnisse: Kapitel 2 bis 7.

18.1. Das root-Konto und Administratorrechte

Niveau: Einsteiger

Der Benutzer `root` (UID 0) darf alles: jede Datei lesen und ändern, Prozesse beenden, Software installieren, das System herunterfahren. Ein Tippfehler als `root` kann das System zerstören. Deshalb gilt das **Prinzip der minimalen Rechte**: Man arbeitet als normaler Benutzer und hebt die Rechte nur für einzelne Befehle an.

Methode	Beschreibung
`sudo befehl`	Einzelner Befehl als `root`; Authentifizierung mit dem *eigenen* Passwort; Aktionen werden protokolliert
`sudo -i`	Interaktive Login-Shell als `root`
`sudo -u anna befehl`	Befehl als anderer Benutzer
`su -`	Wechsel zu `root` mit dessen Passwort; `-` lädt die Umgebung von `root`
`pkexec`, `polkit`	Grafische Rechteerhöhung auf dem Desktop

18.1.1. sudo einrichten [F]

Wer `sudo` benutzen darf, bestimmt die Datei `/etc/sudoers`. Sie wird **ausschließlich** mit `visudo` bearbeitet, das die Syntax vor dem Speichern prüft. Zusätzliche Regeln gehören als eigene Dateien nach `/etc/sudoers.d/`.

```
# usermod -aG sudo anna            # Debian/Ubuntu: Gruppe 'sudo'
# usermod -aG wheel anna           # RHEL/openSUSE: Gruppe 'wheel'

# visudo -f /etc/sudoers.d/admins
%admins   ALL=(ALL:ALL) ALL        # Gruppe 'admins' darf alles
anna      ALL=(root) NOPASSWD: /usr/bin/systemctl restart nginx
```

Syntax einer Regel: `wer wo=(als-wer) was`. Mit `sudo -l` listet jeder Benutzer seine erlaubten Befehle. Protokolliert wird nach `/var/log/auth.log` (Debian/Ubuntu) bzw. `/var/log/secure` (RHEL).

18.2. Konfiguration und Systeminformationen

Niveau: Einsteiger

Fast alle Einstellungen stehen als Textdateien unter `/etc` und können mit einem Editor bearbeitet werden. Vor Änderungen sichert man die Originaldatei; viele Dienste bieten ein Prüfkommando (`sshd -t`, `nginx -t`, `visudo -c`).

```
# cp -a /etc/ssh/sshd_config /etc/ssh/sshd_config.orig    # Sicherung
# nano /etc/ssh/sshd_config
# sshd -t && systemctl reload ssh                          # pruefen, dann neu laden
```

```
$ cat /etc/os-release           # Distribution
$ uname -r                      # Kernelversion
$ lscpu                         # Prozessor
$ free -h                       # Arbeitsspeicher und Swap
$ df -h                         # Plattenbelegung
$ lsblk                         # Platten und Partitionen
$ lspci                         # PCI-Geraete (Grafik, Netzwerk)
$ lsusb                         # USB-Geraete
$ ip -br a                      # Netzwerkschnittstellen kurz
$ uptime                        # Laufzeit und Last
$ sudo dmidecode -t system      # Hardwarehersteller / Modell (BIOS-Daten)
$ systemd-detect-virt           # laeuft das System in einer VM?
```

18.3. Sprache, Zeichensatz und Zeit

Niveau: Fortgeschritten

Die **Locale** legt Sprache, Zeichensatz, Datums-, Zahlen- und Sortierformat fest. Heute ist `UTF-8` der Standard.

```
$ locale                                # aktuelle Einstellungen (LANG, LC_*)
$ locale -a                             # installierte Locales
# localectl set-locale LANG=de_DE.UTF-8 # Systemsprache festlegen
# localectl set-keymap de               # Tastaturbelegung (Konsole)
$ LANG=C ls --help                      # Sprache fuer einen Aufruf aendern
```

`LC_ALL` überschreibt alle anderen `LC_*`-Variablen, `LC_COLLATE` steuert die Sortierung, `LC_TIME` das Zeitformat. In Skripten verwendet man oft `LC_ALL=C` für vorhersehbare Ausgaben.
Die Systemzeit wird intern in UTC geführt und nur für die Anzeige in die Zeitzone umgerechnet. Die Uhr sollte per NTP synchronisiert werden.

```
$ timedatectl                           # Zeit, Zeitzone, NTP-Status
# timedatectl set-timezone Europe/Berlin
# timedatectl set-ntp true              # Synchronisation aktivieren
$ date -u                               # UTC
$ chronyc tracking                      # Genauigkeit (bei chrony)
# hostnamectl set-hostname kanban       # Rechnernamen setzen
```

18.4. Kernelparameter und sysctl

Niveau: Experte

Der laufende Kernel lässt sich über `/proc/sys` und `sysctl` abstimmen. Dauerhafte Einstellungen gehören nach `/etc/sysctl.d/*.conf`.

```
$ sysctl net.ipv4.ip_forward                    # Wert lesen
# sysctl -w net.ipv4.ip_forward=1               # bis zum Neustart setzen
# echo "net.ipv4.ip_forward = 1" > /etc/sysctl.d/90-forward.conf
# sysctl --system                               # alle Dateien neu laden
# sysctl vm.swappiness=10                       # weniger Swapping
```

19. Paketverwaltung

Inhalt: Pakete, Repositories und Abhängigkeiten; `apt` und `dpkg` (Debian), `dnf` und `rpm` (Red Hat), `zypper` (SUSE), `pacman` (Arch); Snap, Flatpak, AppImage; eigene Repositories; Aktualisierungsstrategien.

Stufe: [E] Einsteiger bis [F] Fortgeschritten

Lernziele: Nach diesem Kapitel können Sie

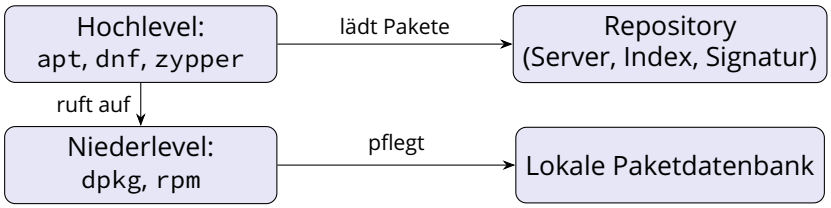
- Software über die Paketverwaltung zu installieren, zu aktualisieren und zu entfernen,
- Repositories und Paketquellen sicher einzubinden,
- Paketdateien zu analysieren und Probleme mit Abhängigkeiten zu lösen.

Vorkenntnisse: Kapitel 18 (Administration) und Kapitel 20 (Benutzer).

19.1. Grundlagen: Pakete und Repositories

Niveau: Einsteiger

Eine Linux-Distribution besteht aus Tausenden von **Paketen**. Ein Paket enthält Programmdateien, Konfigurationsdateien, Dokumentation sowie Metadaten (Name, Version, Abhängigkeiten, Beschreibung, Skripte für Installation und Entfernung). Die **Paketverwaltung** löst Abhängigkeiten automatisch auf, prüft Signaturen und führt Buch darüber, welche Datei zu welchem Paket gehört.
Pakete werden aus **Repositories** (Paketquellen) geladen – Servern, die Pakete und einen signierten Index bereithalten. Es gibt zwei Ebenen von Werkzeugen:



Die Hochlevel-Werkzeuge kümmern sich um Abhängigkeiten und Downloads; die Niederlevel-Werkzeuge installieren einzelne Paketdateien (`.deb`, `.rpm`), ohne Abhängigkeiten aus dem Netz zu holen.

Familie	Distributionen	Paketformat	Werkzeuge	Index
Debian	Debian, Ubuntu, Mint	`.deb`	`apt`, `dpkg`	`Packages`
Red Hat	RHEL, Fedora, Rocky, Alma	`.rpm`	`dnf`, `rpm`	`repodata`
SUSE	openSUSE, SLES	`.rpm`	`zypper`, `rpm`	`repodata`
Arch	Arch, Manjaro	`.pkg.tar.zst`	`pacman`	`*.db`
Alpine	Alpine	`.apk`	`apk`	`APKINDEX`

19.2. Alltägliche Befehle im Vergleich

Niveau: Einsteiger

Aufgabe	Debian/Ubuntu	Fedora/RHEL	openSUSE
Index aktualisieren	`apt update`	(automatisch) `dnf makecache`	`zypper refresh`
Paket installieren	`apt install P`	`dnf install P`	`zypper install P`
Paket entfernen	`apt remove P`	`dnf remove P`	`zypper remove P`
Mit Konfiguration entfernen	`apt purge P`	–	–
Alle aktualisieren	`apt upgrade`	`dnf upgrade`	`zypper update`
Distribution aufrüsten	`apt full-upgrade`	`dnf system-upgrade`	`zypper dup`
Suchen	`apt search X`	`dnf search X`	`zypper search X`
Information	`apt show P`	`dnf info P`	`zypper info P`
Nicht mehr benötigte entfernen	`apt autoremove`	`dnf autoremove`	`zypper packages --unneeded`
Installierte auflisten	`apt list --installed`	`dnf list installed`	`zypper se -i`

```
$ sudo apt update && sudo apt upgrade -y        # Debian/Ubuntu
$ sudo apt install nginx curl htop
$ apt policy nginx                              # installierte und verfuegbare Version
$ sudo apt remove --purge nginx && sudo apt autoremove
$ sudo apt install nginx=1.24.0-1               # bestimmte Version
$ sudo apt-mark hold linux-image-generic        # Paket vor Updates schuetzen
$ sudo apt-mark showhold

$ sudo dnf install httpd                        # RHEL/Fedora
$ sudo dnf check-update
$ sudo dnf history                              # Transaktionen
$ sudo dnf history undo 12                      # Transaktion rueckgaengig
$ sudo dnf versionlock add kernel               # Version festhalten (Plugin)
$ sudo dnf module list                          # Modul-Streams (RHEL 8/9)
```

> **Hinweis**
>
> `apt` ist für die interaktive Nutzung gedacht (Fortschrittsbalken, Farben). In Skripten verwendet man `apt-get` mit stabiler Ausgabe: `DEBIAN_FRONTEND=noninteractive apt-get install -y paket`.

19.3. Die Niederlevel-Werkzeuge dpkg und rpm

Niveau: Fortgeschritten

```
$ sudo dpkg -i paket.deb                    # installieren (ohne Abhaengigkeiten)
$ sudo apt install -f                       # fehlende Abhaengigkeiten nachholen
$ sudo apt install ./paket.deb              # besser: apt kuemmert sich darum
$ sudo dpkg -r paket                        # entfernen
$ sudo dpkg -P paket                        # inkl. Konfiguration
$ sudo dpkg --configure -a                  # abgebrochene Installation fortsetzen
$ dpkg -l | grep ^ii                        # installierte Pakete
$ dpkg --get-selections                     # Auswahl exportieren

$ sudo rpm -ivh paket.rpm                   # installieren
$ sudo rpm -Uvh paket.rpm                   # aktualisieren (oder installieren)
$ sudo rpm -e paket                         # entfernen
$ rpm -qa | sort                            # alle installierten Pakete
$ rpm -q --whatrequires openssl-libs        # welche Pakete brauchen dieses?
$ rpm --import RPM-GPG-KEY-beispiel         # Schluessel importieren
$ rpm -K paket.rpm                          # Signatur pruefen
```

Konfigurationsdateien sind im Paket besonders gekennzeichnet. Wurde eine Datei lokal geändert und das Paket bringt eine neue Version, fragt `apt` ("Behalten oder ersetzen?"), und `rpm` legt die neue Datei als `.rpmnew` oder die alte als `.rpmsave` ab. Diese Dateien nach Updates mit `diff` vergleichen und Änderungen übernehmen.

19.4. Informationen über installierte Pakete

Niveau: Einsteiger

```
# Welches Paket hat diese Datei installiert?
$ dpkg -S /usr/bin/curl                   # Debian
curl: /usr/bin/curl
$ rpm -qf /usr/bin/curl                   # RHEL/SUSE
curl-8.0.1-1.fc38.x86_64

# Welche Dateien gehoeren zu einem Paket?
$ dpkg -L curl
$ rpm -ql curl

# Paket aus einer Datei untersuchen (vor dem Installieren)
$ dpkg -I fremd.deb                       # Metadaten
$ dpkg -c fremd.deb                       # Inhalt
$ rpm -qip fremd.rpm
$ rpm -qlp fremd.rpm

# Pruefen, ob installierte Dateien veraendert wurden
$ sudo debsums -c                         # Debian (Paket debsums)
$ sudo rpm -Va                            # RHEL: alle Pakete verifizieren

# Welche Dateien gehoeren keinem Paket?
$ sudo dnf provides '*/bin/htop'          # welches Paket liefert den Befehl?
$ apt-file search bin/htop                # Debian (Paket apt-file)
```

Die Ausgabe von `rpm -V` nutzt Kürzel: `S` (Größe), `5` (Prüfsumme), `T` (Zeit), `M` (Rechte), `U/G` (Besitzer/Gruppe) und `c` (Konfigurationsdatei). Geänderte Konfigurationsdateien sind normal; geänderte Programmdateien deuten auf Manipulation oder Fehler hin.

19.5. Repositories konfigurieren

Niveau: Einsteiger

Debian/Ubuntu: Paketquellen stehen in `/etc/apt/sources.list` und `/etc/apt/sources.list.d/`.

```
# /etc/apt/sources.list.d/debian.sources  (neues deb822-Format)
Types: deb
URIs: http://deb.debian.org/debian
Suites: bookworm bookworm-updates
Components: main contrib non-free-firmware
Signed-By: /usr/share/keyrings/debian-archive-keyring.gpg

# altes Format
deb http://deb.debian.org/debian bookworm main contrib
```

Red Hat/Fedora: `.repo`-Dateien in `/etc/yum.repos.d/`:

```
# /etc/yum.repos.d/beispiel.repo
[beispiel]
name=Beispiel Repository
baseurl=https://repo.example.org/el9/$basearch/
enabled=1
gpgcheck=1
gpgkey=https://repo.example.org/RPM-GPG-KEY-beispiel
```

```
$ dnf repolist                                  # aktive Repositories
$ sudo dnf config-manager --add-repo https://repo.example.org/beispiel.repo
$ sudo dnf config-manager --set-disabled beispiel
$ sudo dnf --enablerepo=epel install htop       # einmalig ein Repo nutzen
$ sudo zypper addrepo -f https://repo.example.org/suse/ beispiel   # openSUSE
```

Signaturen: Jedes Repository wird mit einem GPG-Schlüssel signiert. Der Schlüssel des Anbieters sollte aus einer vertrauenswürdigen Quelle stammen und *nur für dieses Repository* gelten (`Signed-By`), nicht global im Schlüsselring vertraut werden. Das früher übliche `apt-key add` ist veraltet.

```
$ curl -fsSL https://repo.example.org/key.gpg | sudo gpg --dearmor -o /usr/share/keyrings/
    beispiel.gpg
$ echo "deb [signed-by=/usr/share/keyrings/beispiel.gpg] https://repo.example.org/apt stable
    main" \
    | sudo tee /etc/apt/sources.list.d/beispiel.list
$ sudo apt update
```

> **Sicherheit**
>
> Jedes zusätzliche Repository darf als `root` beliebigen Code installieren. Fremde Quellen (PPAs, Drittanbieter) daher sparsam einsetzen, ihre Herkunft prüfen und Schlüssel mit Fingerabdruck verifizieren. Pakete *nie* mit abgeschalteter Signaturprüfung (`--no-gpg-check`, `--allow-unauthenticated`) installieren. Skripte der Art `curl ... | sudo bash` führen fremden Code mit vollen Rechten aus: den Inhalt zuerst herunterladen und lesen.

19.6. Pakete herunterladen und inspizieren

Niveau: Einsteiger

Ein Paket lässt sich laden, ansehen und auspacken, *ohne* es zu installieren, ähnlich wie ein ZIP-Archiv mit `unzip`. So prüft man vorab, was ein Paket enthält und welche Skripte als `root` laufen würden.

```
# 1. Herunterladen (ohne Installation, ohne root, in das aktuelle Verzeichnis)
$ apt download htop                      # Debian: htop_3.2.0-2_amd64.deb
$ dnf download htop                      # RHEL/Fedora (Paket dnf-plugins-core)
$ yumdownloader htop                     # aelteres RHEL/CentOS (yum-utils)
$ dnf download --resolve --alldeps htop  # auch alle Abhaengigkeiten laden
```

Debian: `.deb`. Eine `.deb`-Datei ist ein `ar`-Archiv mit drei Teilen: `debian-binary` (Formatversion), `control.tar.*` (Metadaten und Skripte) und `data.tar.*` (die eigentlichen Dateien).

```
$ ar t htop_3.2.0-2_amd64.deb            # Aufbau: debian-binary, control.tar.xz, data.tar.xz
$ dpkg-deb -I htop_*.deb                 # Metadaten, Abhaengigkeiten (wie dpkg -I)
$ dpkg-deb -c htop_*.deb                 # Dateiliste mit Rechten    (wie unzip -l)

$ mkdir htop-inhalt
$ dpkg-deb -x htop_*.deb htop-inhalt     # Dateien auspacken         (wie unzip -d)
$ dpkg-deb -e htop_*.deb htop-ctrl       # Kontrolldateien auspacken
$ ls htop-ctrl                           # control, md5sums, evtl. postinst, prerm
$ cat htop-ctrl/postinst                 # Skript, das bei der Installation als root laeuft
$ dpkg-deb -R htop_*.deb htop-alles      # alles in ein Verzeichnis (data und DEBIAN/)
```

RHEL/SUSE: `.rpm`. Eine `.rpm`-Datei besteht aus Kopfdaten (Metadaten, Signatur, Skripte) und einem `cpio`-Archiv mit den Dateien. `rpm` liest die Kopfdaten, `rpm2cpio` liefert das Archiv.

```
$ rpm -qip htop-3.2.2-1.el9.x86_64.rpm   # Metadaten (Version, Lizenz, Beschreibung)
$ rpm -qlp htop-*.rpm                    # Dateiliste                (wie unzip -l)
$ rpm -qlvp htop-*.rpm                   # mit Rechten und Groessen
$ rpm -qp --requires htop-*.rpm          # Abhaengigkeiten
$ rpm -qp --scripts htop-*.rpm           # Installationsskripte (pre/post)
$ rpm -K htop-*.rpm                      # Signatur und Pruefsumme pruefen

$ mkdir htop-inhalt && cd htop-inhalt
$ rpm2cpio ../htop-*.rpm | cpio -idmv    # Dateien auspacken          (wie unzip -d)
```

Auf Debian-Systemen liest `rpm2cpio` fremde RPMs ebenfalls, wenn das Paket `rpm2cpio` oder `rpm` installiert ist. Umgekehrt entpackt `ar x paket.deb` die drei Teile einer `.deb`, auch ohne `dpkg`.

> **Hinweis**
>
> Entpacken Sie Pakete immer in ein leeres, eigenes Verzeichnis und niemals mit Zielpfad `/`. Die Pfade im Archiv sind absolut gedacht (`./usr/bin/htop`); `dpkg-deb -x` und `cpio -idmv` legen sie unterhalb des aktuellen Verzeichnisses an. Vor der Installation einer Fremdquelle lohnt ein Blick in die Skripte (`postinst`, `rpm -qp --scripts`): Sie laufen mit Administratorrechten.

19.7. Abhängigkeitsprobleme lösen

Niveau: Fortgeschritten

Problem	Vorgehen
"Unmet dependencies" / "broken packages"	`apt update`, `apt install -f`, `dpkg --configure -a`; Quellen auf Mischung verschiedener Releases prüfen
Paket nicht gefunden	Index aktualisieren; Repository aktiviert? (`universe`, `EPEL`, `non-free`)
Dateikonflikt zwischen Paketen	Paket entfernen, das die Datei zusätzlich enthält; Meldung genau lesen
Sperre: `Could not get lock`	Anderer Paketprozess läuft (`unattended-upgrades`, GUI-Updater); abwarten statt Sperrdatei löschen
Signatur-/Schlüsselfehler	Schlüssel erneuern, Systemzeit prüfen (abgelaufene Zeit führt zu Fehlern)
Zu wenig Platz	`apt clean`, `dnf clean all`, `journalctl --vacuum-size`, alte Kernel entfernen

19.8. Weitere Paketsysteme: Snap, Flatpak, AppImage

Niveau: Fortgeschritten

Neben Distributionspaketen gibt es **distributionsübergreifende Formate**. Sie bringen ihre Abhängigkeiten mit und laufen isoliert (Sandbox):

Format	Herkunft	Kommandos	Typische Nutzung
Snap	Canonical (Snap Store)	`snap install P`, `snap list`, `snap refresh`	Server und Desktop
Flatpak	Gemeinschaft (Flathub)	`flatpak install flathub P`, `flatpak run P`	Desktop-Anwendungen
AppImage	Einzelne Datei	`chmod +x X.AppImage && ./X.AppImage`	Portables Ausprobieren

Vorteile: aktuelle Software unabhängig von der Distribution, Isolation. Nachteile: höherer Speicherbedarf, doppelte Bibliotheken, Updates liegen in der Hand der Anbieter.

19.9. Aktualisierungsstrategien

Niveau: Fortgeschritten

- **Sicherheitsupdates** zeitnah einspielen. Debian/Ubuntu: Paket `unattended-upgrades`; RHEL: `dnf-automatic`.
- **Testen vor Produktion**: Updates zunächst auf einer Testumgebung einspielen oder gestaffelt (Canary).
- **Kernel-Updates** erfordern einen Neustart; Alternativen: Live-Patching (`kpatch`, Canonical Livepatch).
- **Nach dem Update prüfen**, welche Dienste neu gestartet werden müssen: `needrestart` (Debian), `dnf needs-restarting`.
- **Release-Upgrades** (z. B. Debian 12 auf 13) nur mit Backup und Planung; Anleitung der Distribution befolgen.
- **Eigene Spiegelserver** (z. B. `apt-cacher-ng`, Foreman/Katello, Pulp) sparen Bandbreite und erlauben kontrollierte Paketstände.

```
# Automatische Sicherheitsupdates unter Debian/Ubuntu
$ sudo apt install unattended-upgrades
$ sudo dpkg-reconfigure -plow unattended-upgrades
$ cat /etc/apt/apt.conf.d/20auto-upgrades
APT::Periodic::Update-Package-Lists "1";
APT::Periodic::Unattended-Upgrade "1";

# RHEL: dnf-automatic
$ sudo dnf install dnf-automatic
$ sudo sed -i 's/^apply_updates.*/apply_updates = yes/' /etc/dnf/automatic.conf
$ sudo systemctl enable --now dnf-automatic.timer
```

19.10. Eigene Pakete und Repositories

Niveau: Experte

Wer viele Rechner betreut, verteilt eigene Software als Paket, nicht als Skript. Ein minimales Debian-Paket:

```
meintool/
  DEBIAN/control
  usr/local/bin/meintool

$ cat meintool/DEBIAN/control
Package: meintool
Version: 1.0
Architecture: all
Maintainer: Anna Beispiel <anna@example.org>
Description: Mein kleines Werkzeug
$ dpkg-deb --build meintool
$ dpkg -I meintool.deb
```

Ein einfaches RPM entsteht aus einer `.spec`-Datei mit `rpmbuild -ba meintool.spec` (am besten als normaler Benutzer in `~/rpmbuild`). Für ein eigenes Repository genügen Werkzeuge, die den Index erzeugen:

```
$ createrepo_c /srv/repo/el9/              # RPM-Repository (Index in repodata)
$ cd /srv/repo/apt && dpkg-scanpackages . | gzip > Packages.gz   # einfaches deb-Repo
$ gpg --armor --detach-sign -o Release.gpg Release                # signieren
```

Das Verzeichnis wird über einen Webserver (Kapitel 46) bereitgestellt.

Übungen und Kontrollfragen

1. Wie finden Sie heraus, zu welchem Paket `/usr/bin/dig` gehört?

   Lösung: Debian: `dpkg -S /usr/bin/dig` (Paket `dnsutils/bind9-dnsutils`); RHEL: `rpm -qf /usr/bin/dig`. Ist das Paket nicht installiert: `apt-file search` bzw. `dnf provides`.

2. Worin unterscheiden sich `apt remove` und `apt purge`?

   Lösung: `remove` entfernt Programmdateien, behält aber Konfigurationsdateien; `purge` entfernt auch diese.

3. Wie verhindern Sie, dass ein Paket (z. B. der Kernel) aktualisiert wird?

   Lösung: Debian: `apt-mark hold`; RHEL: `dnf versionlock add` bzw. `exclude=` in der Konfiguration.

4. Warum soll man Paketquellen von Drittanbietern kritisch betrachten?

   Lösung: Pakete laufen bei der Installation mit Root-Rechten und können beliebigen Code ausführen; zudem können Versionskonflikte entstehen. Herkunft und Schlüssel prüfen, wenige Quellen nutzen.

5. Wie bringen Sie einen Server nach einem abgebrochenen Update wieder in einen konsistenten Zustand?

   Lösung: `dpkg --configure -a`, dann `apt install -f`; bei RHEL `dnf history` prüfen, ggf. `dnf history undo` oder `rpm --rebuilddb`.

20. Benutzer und Gruppen

Inhalt: Benutzerkonzept, UID und GID, `/etc/passwd`, `shadow` und `group`, Konten anlegen, ändern, sperren und löschen, Passwörter.

Stufe: [E] Einsteiger bis [F] Fortgeschritten

Lernziele: Nach diesem Kapitel können Sie

- Benutzer und Gruppen anzulegen und zu verwalten,
- die Kontodateien zu lesen und zu verstehen,
- Passwortrichtlinien und Kontosperren einzusetzen.

Vorkenntnisse: Kapitel 6 und 18.

20.1. Grundlagen

Niveau: Einsteiger

Jeder Prozess und jede Datei gehört einem **Benutzer** und einer **Gruppe**. Intern zählen nur Nummern: die UID (*User ID*) und die GID (*Group ID*). Namen sind nur Bezeichnungen, die in Datenbanken zugeordnet werden.

UID-Bereich	Verwendung
0	`root`, der Superuser
1–999	**Systemkonten** für Dienste (`www-data`, `sshd`, `postgres`); meist ohne Login
ab 1000	**Normale Benutzer** (Debian, Ubuntu, RHEL, openSUSE)

```
$ id                          # eigene UID, GID, Gruppen
uid=1000(tux) gid=1000(tux) groups=1000(tux),27(sudo),100(users)
$ id anna                     # Daten eines anderen Benutzers
$ groups                      # eigene Gruppen
$ who                         # wer ist angemeldet?
$ last -n 5                   # letzte Anmeldungen
$ lastlog | head              # letzte Anmeldung je Benutzer
$ getent passwd tux           # Eintrag aus allen Quellen (auch LDAP u. a.)
```

Neben den **primären Gruppen** (die GID im Benutzereintrag, Standardgruppe für neue Dateien) kann ein Benutzer in beliebig vielen **Zusatzgruppen** sein. Rechte ergeben sich aus der Summe seiner Gruppen. Moderne Distributionen legen für jeden Benutzer eine eigene gleichnamige Gruppe an.

20.2. Die Benutzerdatenbanken

Niveau: Einsteiger

Datei	Inhalt
`/etc/passwd`	Benutzerkonten (für alle lesbar)
`/etc/shadow`	Passwort-Hashes und Ablaufdaten (nur `root`)
`/etc/group`	Gruppen und deren Mitglieder
`/etc/gshadow`	Gruppenpasswörter und Verwalter (nur `root`)
`/etc/login.defs`	Standardwerte: UID-Bereiche, Passwortalter, `umask`
`/etc/skel/`	Vorlage für neue Heimatverzeichnisse

Ein Eintrag in `/etc/passwd` hat sieben durch Doppelpunkte getrennte Felder:

```
tux:x:1000:1000:Tux Pinguin,,,:/home/tux:/bin/bash
 |  |  |    |        |            |          +- Login-Shell
 |  |  |    |        |            +- Heimatverzeichnis
 |  |  |    |        +- Kommentar (GECOS): vollstaendiger Name
 |  |  |    +- primaere GID
 |  |  +- UID
 |  +- Passwort ('x': Hash steht in /etc/shadow)
 +- Benutzername
```

In `/etc/shadow` stehen unter anderem der Hash (`$y$` yescrypt, `$6$` SHA-512), das Datum der letzten Änderung und Gültigkeitsfristen. Ein `!` oder `*` im Hash-Feld bedeutet: *kein Passwort-Login möglich* (gesperrt bzw. Systemkonto). Die Datei `/etc/group` hat vier Felder: `name:x:gid:mitglied1,mitglied2`.

> **Sicherheit**
>
> Passwörter werden nie im Klartext gespeichert, sondern als **gesalzener Hash**. `/etc/shadow` darf nicht allgemein lesbar sein. Konten und Gruppen niemals von Hand in diesen Dateien bearbeiten, sondern mit den Werkzeugen; im Notfall mit `vipw` und `vigr`, die die Dateien sperren.

20.3. Benutzerkonten verwalten

Niveau: Einsteiger

```
# useradd -m -s /bin/bash -c "Anna Beispiel" anna    # Konto samt Heimatverzeichnis
# passwd anna                                        # Passwort setzen
# usermod -aG sudo anna                              # zu Gruppe 'sudo' hinzufuegen (-a nicht
    vergessen!)
# usermod -s /usr/sbin/nologin anna                  # Login-Shell sperren
# usermod -L anna                                    # Passwort sperren (-U entsperrt)
# usermod -l anna2 anna                              # Namen aendern
# userdel anna                                       # Konto loeschen
# userdel -r anna                                    # ... samt Heimatverzeichnis und Mail
# chage -l anna                                      # Passwortalter anzeigen
# chage -M 90 -W 7 anna                              # Passwort laeuft nach 90 Tagen ab,
   Warnung 7 Tage vorher
# passwd -e anna                                     # Passwort sofort ablaufen lassen (
   Aenderung beim Login)
```

> **Achtung**
>
> `usermod -G gruppe benutzer` *ersetzt* alle Zusatzgruppen. Zum *Hinzufügen* immer `usermod -aG` verwenden. Änderungen an Gruppenmitgliedschaften werden erst bei der nächsten Anmeldung wirksam.

20.3.1. Komfort unter Debian und Ubuntu [E]

Unter Debian und Ubuntu fragt der Wrapper `adduser` interaktiv nach den Daten, legt das Heimatverzeichnis aus `/etc/skel` an und setzt vernünftige Standardwerte:

```
# adduser anna                          # interaktiv
# adduser anna sudo                     # Benutzer einer Gruppe hinzufuegen
# deluser --remove-home anna            # Konto und Heimatverzeichnis loeschen
```

20.4. Gruppen verwalten

Niveau: Einsteiger

```
# groupadd entwickler                   # neue Gruppe
# groupadd -g 2000 projekt              # mit fester GID
# groupmod -n dev entwickler            # umbenennen
# groupdel dev                          # loeschen
# gpasswd -a anna entwickler            # Mitglied hinzufuegen
# gpasswd -d anna entwickler            # Mitglied entfernen
$ getent group entwickler               # Mitglieder anzeigen
$ newgrp entwickler                     # primaere Gruppe der Sitzung wechseln
```

20.5. Gemeinsame Verzeichnisse

Niveau: Fortgeschritten

Soll ein Team gemeinsam in einem Verzeichnis arbeiten, kombiniert man Gruppe und das *SGID*-Bit (siehe Kapitel 22):

```
# groupadd team
# mkdir /srv/projekt
# chown root:team /srv/projekt
# chmod 2770 /srv/projekt       # rwxrws---: neue Dateien erben die Gruppe 'team'
# usermod -aG team anna
```

20.6. Passwortrichtlinien und Anmeldung

Niveau: Fortgeschritten

- **Passwortqualität**: Das Modul `pam_pwquality` (`/etc/security/pwquality.conf`) erzwingt Mindestlänge und Komplexität.
- **PAM** (*Pluggable Authentication Modules*, `/etc/pam.d/`) regelt, wie sich Benutzer an Diensten anmelden. Fehler in diesen Dateien können alle Logins sperren: immer eine Root-Sitzung offen halten.
- **Sperren nach Fehlversuchen**: `pam_faillock` (RHEL) bzw. `fail2ban` gegen SSH-Angriffe.
- **Konten ohne Login**: Login-Shell `/usr/sbin/nologin`, Passwort mit `passwd -l` gesperrt.
- **SSH-Schlüssel statt Passwörter** für Fernzugriff (Kapitel 42).

20.7. Zentrale Benutzerverwaltung

Niveau: Experte

In Netzen mit vielen Rechnern verwaltet man Konten zentral: LDAP (OpenLDAP, 389 Directory), Active Directory über `sssd` und `realmd`, FreeIPA. Der **Name Service Switch** (`/etc/nsswitch.conf`) bestimmt, in welcher Reihenfolge Quellen befragt werden; `getent passwd` fragt alle konfigurierten Quellen ab.

```
$ grep -E '^(passwd|group|hosts):' /etc/nsswitch.conf
passwd:         files systemd
group:          files systemd
hosts:          files dns
```

21. PAM, zentrale Anmeldung und LDAP

Inhalt: Pluggable Authentication Modules, `nsswitch.conf`, `sudo` im Detail, Zwei-Faktor-Anmeldung, LDAP-Verzeichnisdienste, SSSD, Kerberos und Anbindung an Active Directory.

Stufe: [F] Fortgeschritten bis [X] Experte

Lernziele: Nach diesem Kapitel können Sie

- den Ablauf der Anmeldung über NSS und PAM zu erklären,
- PAM-Konfigurationen zu lesen und vorsichtig anzupassen,
- Linux-Systeme an ein zentrales Verzeichnis (LDAP/AD) anzubinden.

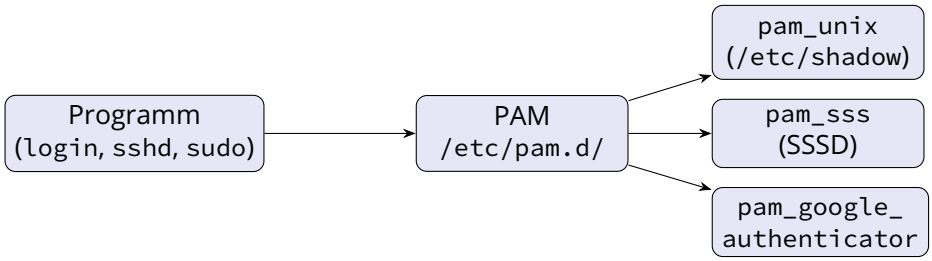
Vorkenntnisse: Kapitel 20 (Benutzer) und Kapitel 22 (Rechte).

21.1. Wie funktioniert die Anmeldung?

Niveau: Fortgeschritten

Wenn sich ein Benutzer anmeldet, sind zwei Fragen zu klären: *Wer ist das?* (Identität: Benutzername, UID, Gruppen, Heimatverzeichnis) und *Darf er herein?* (Authentifizierung und Autorisierung). Linux trennt beides:

- **NSS** (Name Service Switch) liefert Informationen über Benutzer, Gruppen, Hosts usw. Konfiguration: `/etc/nsswitch.conf`.
- **PAM** (Pluggable Authentication Modules) prüft Passwörter und weitere Bedingungen. Konfiguration: `/etc/pam.d/`.



Beim Zugriff auf Benutzerinformationen fragen Programme die Bibliothek (`getpwnam()` u. a.); diese befolgt `nsswitch.conf`:

```
# /etc/nsswitch.conf (Ausschnitt)
passwd:     files systemd sss
group:      files systemd sss
shadow:     files sss
hosts:      files dns
```

"`files sss`" bedeutet: zuerst lokale Dateien (`/etc/passwd`), dann SSSD (LDAP/AD). Das Werkzeug `getent` befragt dieselben Quellen wie die Programme:

```
$ getent passwd anna              # wie ein Programm es sehen wuerde
$ getent group admins
$ getent hosts server1.example.org
$ id anna
```

21.2. PAM im Detail

Niveau: Fortgeschritten

Jede Anwendung hat in `/etc/pam.d/` eine Datei mit Regeln. Jede Zeile hat vier Felder: **Typ**, **Kontrolle**, **Modul**, **Argumente**.

Typ	Aufgabe
`auth`	Identität prüfen (Passwort, Token, Schlüssel)
`account`	Kontoprüfung: abgelaufen, gesperrt, Zugriffszeiten
`password`	Passwort ändern, Qualitätsregeln
`session`	Vor/nach der Sitzung: Limits setzen, Heimatverzeichnis anlegen, Protokoll

Kontrolle	Bedeutung
`required`	Muss gelingen; bei Fehler wird die Prüfung fortgesetzt, aber am Ende abgelehnt
`requisite`	Muss gelingen; bei Fehler sofortiger Abbruch
`sufficient`	Bei Erfolg (und keinem früheren `required`-Fehler) sofort akzeptiert
`optional`	Ergebnis spielt nur eine Rolle, wenn es das einzige Modul ist
`include` / `substack`	Regeln einer anderen Datei einbinden

```
# /etc/pam.d/sshd (vereinfacht)
auth       required     pam_faildelay.so delay=2000000
@include common-auth
account    required     pam_nologin.so
@include common-account
session    required     pam_limits.so
session    optional     pam_motd.so
@include common-session

# /etc/pam.d/common-password (Qualitaetsregeln, Debian)
password   requisite    pam_pwquality.so retry=3 minlen=12 dcredit=-1 ucredit=-1
password   [success=1 default=ignore] pam_unix.so obscure use_authtok sha512 yescrypt
```

Nützliche Module:

Modul	Zweck
`pam_unix`	Klassische Anmeldung gegen `/etc/shadow`
`pam_sss`	Anmeldung über SSSD (LDAP, AD, Kerberos)
`pam_pwquality`	Passwortqualität (Länge, Zeichenklassen)
`pam_faillock`, `pam_tally2`	Konto nach zu vielen Fehlversuchen sperren
`pam_limits`	Ressourcenlimits aus `/etc/security/limits.conf`
`pam_access`	Zugriff nach Quelle/Benutzer (`/etc/security/access.conf`)
`pam_time`	Zugriffszeiten (`/etc/security/time.conf`)
`pam_mkhomedir`, `pam_oddjob_mkhomedir`	Heimatverzeichnis bei erster Anmeldung anlegen
`pam_google_authenticator`	Einmalpasswörter (TOTP) für Zwei-Faktor-Anmeldung
`pam_securetty`, `pam_nologin`	Root-Anmeldung einschränken, Anmeldesperre

Vorsicht

Ein Fehler in PAM kann **alle** Anmeldungen – auch die von `root` – verhindern. Vor Änderungen immer eine zweite, *bereits angemeldete* Root-Sitzung offen halten, erst testen (neue Anmeldung in einem zweiten Terminal), dann die Sitzung schließen. Unter Debian verwaltet `pam-auth-update`, unter RHEL `authselect` die gemeinsamen Dateien – nicht von Hand in Systemdateien wie `system-auth` schreiben.

21.3. Passwortrichtlinien und Sperren

Niveau: Fortgeschritten

```
# Nach 5 Fehlversuchen 10 Minuten sperren (pam_faillock)
auth required pam_faillock.so preauth silent deny=5 unlock_time=600
auth sufficient pam_unix.so
auth [default=die] pam_faillock.so authfail deny=5 unlock_time=600
account required pam_faillock.so

$ faillock --user anna                 # Fehlversuche anzeigen
$ sudo faillock --user anna --reset    # Sperre aufheben

# Ressourcenlimits (/etc/security/limits.conf)
anna     hard   nproc     200
@devs    soft   nofile    8192
*        hard   core      0

# Passwortalterung
$ sudo chage -l anna
$ sudo chage -M 90 -W 14 anna          # maximal 90 Tage, 14 Tage Warnung
```

21.4. sudo im Detail

Niveau: Fortgeschritten

`sudo` erlaubt gezielt administrative Befehle und protokolliert sie. Bearbeitet wird die Konfiguration *immer* mit `visudo`, das die Syntax prüft.

```
# /etc/sudoers.d/webadmin   (mit: sudo visudo -f /etc/sudoers.d/webadmin)
# Benutzer  Rechner = (als wer)  Befehle
anna    ALL=(ALL:ALL) ALL                          # alles, mit Passwort
%admins ALL=(ALL) NOPASSWD: /usr/bin/systemctl restart nginx, /usr/bin/systemctl status nginx
bob     srv1=(www-data) /usr/bin/crontab -l        # nur als www-data auf srv1
Defaults  timestamp_timeout=5
Defaults  logfile=/var/log/sudo.log
Defaults  use_pty
Defaults  log_input, log_output                    # Sitzungen aufzeichnen
Cmnd_Alias NETZ = /usr/sbin/ip, /usr/sbin/ss
%netzteam ALL = NETZ
```

```
$ sudo -l                      # eigene Rechte anzeigen
$ sudo -u www-data id          # als anderer Benutzer
$ sudo -i                      # Root-Login-Shell
$ sudo -k                      # Passwort-Cache loeschen
$ sudo sudoreplay -l           # aufgezeichnete Sitzungen auflisten
```

Grundsätze: Befehle möglichst genau angeben (Vorsicht bei Programmen mit Shell-Ausbruch wie `vi`, `less`, `find`, `awk`: diese erlauben praktisch Root), Dateien in `/etc/sudoers.d/` statt `/etc/sudoers` ändern und `NOPASSWD` sparsam einsetzen.

21.5. Zwei-Faktor-Anmeldung per TOTP

Niveau: Experte

```
$ sudo apt install libpam-google-authenticator
$ google-authenticator                  # als Benutzer ausfuehren; QR-Code scannen

# /etc/pam.d/sshd - zusaetzlich zur Anmeldung einen Code verlangen
auth required pam_google_authenticator.so

# /etc/ssh/sshd_config
KbdInteractiveAuthentication yes
AuthenticationMethods publickey,keyboard-interactive
```

So wird für die Anmeldung per SSH zuerst ein Schlüssel und anschließend der Einmalcode benötigt. Auch hier gilt: Sitzung offen lassen, in einem zweiten Fenster testen.

21.6. LDAP-Verzeichnisdienste

Niveau: Experte

LDAP (Lightweight Directory Access Protocol) ist ein Protokoll für hierarchische Verzeichnisse, die Benutzer, Gruppen, Rechner und Adressen speichern. Typische Server: OpenLDAP, 389 Directory Server (Red Hat), FreeIPA, Microsoft Active Directory. Ein Eintrag wird über seinen **Distinguished Name** (DN) identifiziert:

```
dn: uid=anna,ou=people,dc=example,dc=org
objectClass: inetOrgPerson
objectClass: posixAccount
uid: anna
cn: Anna Beispiel
sn: Beispiel
uidNumber: 10001
gidNumber: 10000
homeDirectory: /home/anna
loginShell: /bin/bash
mail: anna@example.org
```

```
# Suche (ldapsearch aus Paket ldap-utils / openldap-clients)
$ ldapsearch -x -H ldap://ldap.example.org -b "dc=example,dc=org" "(uid=anna)"
$ ldapsearch -x -H ldaps://ldap.example.org -D "cn=admin,dc=example,dc=org" -W \
      -b "ou=people,dc=example,dc=org" "(objectClass=posixAccount)" uid uidNumber
$ ldapadd -x -D "cn=admin,dc=example,dc=org" -W -f neuer-benutzer.ldif
$ ldapmodify  ...    $ ldappasswd ...
```

Verbindungen immer mit **TLS** absichern (`ldaps://` auf Port 636 oder StartTLS auf 389); Anmeldedaten gehen sonst im Klartext über das Netz.

21.7. SSSD: Anbindung an LDAP und Active Directory

Niveau: Experte

Der **System Security Services Daemon** (SSSD) verbindet Linux mit zentralen Verzeichnissen. Er puffert Daten (Offline-Anmeldung möglich) und stellt NSS- und PAM-Module bereit.

```
# /etc/sssd/sssd.conf  (Rechte 0600, Besitzer root!)
[sssd]
services = nss, pam, ssh
domains = example.org

[domain/example.org]
id_provider = ldap
auth_provider = ldap
ldap_uri = ldaps://ldap.example.org
ldap_search_base = dc=example,dc=org
ldap_tls_reqcert = demand
cache_credentials = true
enumerate = false

$ sudo chmod 600 /etc/sssd/sssd.conf
$ sudo systemctl enable --now sssd
$ id anna                                  # sollte nun UID aus LDAP zeigen
$ sudo sss_cache -E                        # Cache leeren
```

Für **Active Directory** vereinfacht `realmd` die Einrichtung:

```
$ sudo apt install realmd sssd adcli
$ realm discover example.org
$ sudo realm join --user=administrator example.org
$ realm list
$ id administrator@example.org
$ sudo realm permit --groups linuxadmins@example.org
$ sudo pam-auth-update --enable mkhomedir
```

Kerberos ermöglicht Single Sign-on: Nach einmaliger Anmeldung stellt das *Key Distribution Center* Tickets aus (`kinit`, `klist`); weitere Dienste verlangen kein Passwort mehr. Alternativ bietet **FreeIPA** (Red Hat Identity Management) eine integrierte Lösung aus LDAP, Kerberos, DNS und Zertifikaten.

Übungen und Kontrollfragen

1. Welche Datei bestimmt, woher `getent passwd` seine Daten bezieht?

   Lösung: `/etc/nsswitch.conf`, Zeile `passwd:`.

2. Was ist der Unterschied zwischen `required` und `requisite`?

   Lösung: Bei einem Fehler wird `required` noch zu Ende durchlaufen (Ablehnung am Schluss), `requisite` bricht sofort ab.

3. Warum bearbeitet man `/etc/sudoers` nur mit `visudo`?

   Lösung: `visudo` sperrt die Datei gegen gleichzeitiges Bearbeiten und prüft die Syntax; ein Fehler könnte sonst `sudo` unbrauchbar machen.

4. Wie verhindern Sie, dass sich nach fünf Fehlversuchen weitere Anmeldungen durchprobieren lassen?

   Lösung: Mit `pam_faillock` (`deny=5 unlock_time=...`) und zusätzlich Schutz auf Netzwerkebene (z. B. `fail2ban`).

5. Warum soll LDAP nicht ohne TLS betrieben werden?

   Lösung: Passwörter und Daten würden im Klartext übertragen und könnten mitgelesen oder manipuliert werden; `ldaps://` oder StartTLS verwenden.

22. Zugriffsrechte

Inhalt: Lese-, Schreib- und Ausführrechte, Oktaldarstellung, `umask`, Spezialrechte (SUID, SGID, Sticky), ACLs, Dateiattribute.
Stufe: [E] Einsteiger bis [X] Experte
Lernziele: Nach diesem Kapitel können Sie

- Rechte symbolisch und oktal zu lesen und zu setzen,
- Spezialrechte und `umask` gezielt einzusetzen,
- mit ACLs feingranulare Rechte zu vergeben.

Vorkenntnisse: Kapitel 6 und 20.

22.1. Das Rechtekonzept

Niveau: Einsteiger

Jede Datei hat einen **Eigentümer** (*user*), eine **Gruppe** (*group*) und Rechte für drei Klassen: den Eigentümer `u`, die Gruppe `g` und alle anderen `o` (*others*). Jede Klasse hat drei Rechte:

Recht	Wert	Bei Dateien	Bei Verzeichnissen
`r` (read)	4	Inhalt lesen	Inhalt auflisten (`ls`)
`w` (write)	2	Inhalt ändern	Dateien anlegen, umbenennen, löschen
`x` (execute)	1	Als Programm ausführen	Verzeichnis betreten (`cd`), auf Inhalt zugreifen

```
$ ls -l skript.sh
-rwxr-xr-- 1 tux team 220 Okt 14 10:00 skript.sh
```

`rwxr-xr--` bedeutet: Eigentümer `tux` darf lesen, schreiben, ausführen (`rwx` = 7), die Gruppe `team` darf lesen und ausführen (`r-x` = 5), alle anderen dürfen nur lesen (`r--` = 4). Das ergibt **754**.

> **Wichtig**
> Ob man eine Datei **löschen** darf, bestimmt das Recht `w` am *Verzeichnis*, nicht an der Datei. Ohne `x` am Verzeichnis kann man auch dessen lesbare Dateien nicht öffnen. `root` unterliegt diesen Prüfungen nicht.

22.2. Rechte ändern: chmod

Niveau: Einsteiger

```
$ chmod 754 skript.sh          # oktal: rwxr-xr--
$ chmod 644 brief.txt          # rw-r--r-- (typisch fuer Dateien)
$ chmod 755 programm           # rwxr-xr-x (typisch fuer Programme und Verzeichnisse)
$ chmod 600 ~/.ssh/id_ed25519  # nur Eigentuemer (private Schluessel)
$ chmod u+x skript.sh          # symbolisch: Eigentuemer darf ausfuehren
$ chmod g-w,o-rwx datei        # Gruppe verliert w, andere verlieren alles
$ chmod a+r datei              # alle duerfen lesen (a = ugo)
$ chmod -R u=rwX,g=rX,o= projekt/    # rekursiv; X = x nur bei Verzeichnissen/ausf. Dateien
```

Eigentümer ändern

```
# chown anna datei                  # Eigentuemer aendern (nur root)
# chown anna:team datei             # Eigentuemer und Gruppe
# chown -R www-data: /var/www       # rekursiv, Gruppe = Standardgruppe des Benutzers
$ chgrp team datei                  # Gruppe aendern (Eigentuemer: nur eigene Gruppen)
```

22.3. Standardrechte: umask

Niveau: Einsteiger

Neue Dateien erhalten die Rechte `666`, neue Verzeichnisse `777`, jeweils *abzüglich* der **umask**. Üblich sind `022` (Ergebnis: Dateien 644, Verzeichnisse 755) und `077` (nur Eigentümer: 600/700). Die Ubuntu-Standardeinstellung ist `022`.

```
$ umask               # aktuellen Wert anzeigen
0022
$ umask 077           # fuer diese Sitzung strenger
$ touch a; mkdir b; ls -ld a b
-rw------- 1 tux tux 0 Okt 14 10:05 a
drwx------ 2 tux tux 4096 Okt 14 10:05 b
```

Dauerhaft setzt man die `umask` in `~/.bashrc`, `/etc/profile` oder `/etc/login.defs`.

22.4. Besondere Rechte: SUID, SGID und Sticky-Bit

Niveau: Fortgeschritten

Bit	Wert	Bei Programmen	Bei Verzeichnissen
SUID (`s` bei `u`)	4000	Läuft mit den Rechten des *Eigentümers*	ohne Wirkung
SGID (`s` bei `g`)	2000	Läuft mit den Rechten der *Gruppe*	Neue Dateien erben die Gruppe des Verzeichnisses
Sticky (`t` bei `o`)	1000	–	Nur Eigentümer (und `root`) dürfen eigene Dateien löschen

```
$ ls -l /usr/bin/passwd /tmp
-rwsr-xr-x 1 root root 59976 Mai 30  2025 /usr/bin/passwd      # SUID: Benutzer aendert so /
    etc/shadow
drwxrwxrwt 18 root root 4096 Okt 14 10:10 /tmp                 # Sticky: jeder darf anlegen,
    aber nur eigene loeschen
$ chmod u+s programm       # SUID setzen       (oder chmod 4755)
$ chmod g+s verzeichnis    # SGID setzen       (oder chmod 2775)
$ chmod +t verzeichnis     # Sticky setzen     (oder chmod 1777)
$ find / -perm -4000 -type f 2>/dev/null     # alle SUID-Programme auflisten
```

Sicherheit

SUID-Programme sind ein bevorzugtes Angriffsziel. Nur setzen, wenn zwingend nötig, und die Liste (`find / -perm -4000`) regelmäßig auf Unerwartetes prüfen. SUID auf Shell-Skripten wird vom Kernel ignoriert.

22.5. Access Control Lists (ACL)

Niveau: Fortgeschritten

Die ACL erlauben Rechte für *einzelne* Benutzer oder Gruppen, über das Dreiklassenmodell hinaus. Das Dateisystem muss ACLs unterstützen (ext4, XFS und Btrfs tun das standardmäßig). Eine Datei mit ACL zeigt ein `+` am Ende der Rechte.

```
$ setfacl -m u:anna:rw bericht.txt          # Benutzer anna darf lesen und schreiben
$ setfacl -m g:audit:r bericht.txt          # Gruppe audit darf lesen
$ getfacl bericht.txt                       # ACL anzeigen
# file: bericht.txt
# owner: tux
# group: tux
user::rw-
user:anna:rw-
group::r--
group:audit:r--
mask::rw-
other::r--
$ setfacl -x u:anna bericht.txt             # Eintrag entfernen
$ setfacl -b bericht.txt                    # alle ACLs entfernen
$ setfacl -d -m g:team:rwx /srv/projekt     # Standard-ACL: wird an neue Dateien vererbt
```

Die **Maske** (`mask`) begrenzt die Rechte aller Benutzer- und Gruppeneinträge außer dem Eigentümer. Sie wird durch `chmod` auf die Gruppenrechte gesetzt, was ACL-Rechte unerwartet einschränken kann.

22.6. Dateiattribute, Capabilities und Mandatory Access Control

Niveau: Experte

Zusätzlich zu den Rechten kennen ext4 und XFS **Dateiattribute**, die auch `root` einschränken:

```
# chattr +i /etc/resolv.conf     # immutable: nicht aenderbar, nicht loeschbar
# chattr -i /etc/resolv.conf
# chattr +a /var/log/audit.log   # append-only: nur anhaengen
$ lsattr /etc/resolv.conf        # Attribute anzeigen
----i---------e------- /etc/resolv.conf
```

Capabilities zerlegen die Allmacht von `root` in einzelne Privilegien. So darf ein Webserver niedrige Ports öffnen, ohne `root` zu sein:

```
# setcap cap_net_bind_service=+ep /usr/local/bin/meinserver
$ getcap /usr/local/bin/meinserver
```

Ergänzend erzwingen **SELinux** (RHEL, Fedora) und **AppArmor** (Ubuntu, openSUSE) *zwingende Zugriffskontrollen* (MAC) für Prozesse, unabhängig von den Dateirechten.

```
$ getenforce                     # SELinux: Enforcing / Permissive / Disabled
$ ls -Z /var/www/html            # SELinux-Kontext von Dateien
# restorecon -Rv /var/www/html   # Standardkontext wiederherstellen
# aa-status                      # AppArmor: geladene Profile
```

> **Hinweis**
>
> Wenn trotz korrekter Rechte ein Dienst unter RHEL nicht auf Dateien zugreifen kann, ist fast immer SELinux die Ursache. Fehlermeldungen: `ausearch -m avc -ts recent` bzw. das Journal. SELinux nicht pauschal abschalten, sondern den Kontext korrigieren.

23. SELinux, AppArmor und Auditing

Inhalt: Mandatory Access Control: SELinux (Kontexte, Modi, Booleans, Fehlerbehebung), AppArmor (Profile), Linux Audit Framework, Capabilities, Härtung von Diensten mit systemd und Kernel-Schutzmechanismen.

Stufe: [X] Experte

Lernziele: Nach diesem Kapitel können Sie

- den Unterschied zwischen DAC und MAC zu erklären,
- SELinux-Meldungen zu analysieren und Richtlinien gezielt anzupassen,
- AppArmor-Profile zu verwalten und Zugriffe mit `auditd` zu protokollieren.

Vorkenntnisse: Kapitel 22 (Rechte), Kapitel 37 (Logging) und Kapitel 52 (Sicherheit).

23.1. Discretionary und Mandatory Access Control

Niveau: Experte

Die klassischen Unix-Rechte (Besitzer, Gruppe, `rwx`) heißen **DAC** (*Discretionary Access Control*): Der Besitzer einer Datei entscheidet über den Zugriff, und `root` darf alles. Das hat eine Schwäche: Ein kompromittierter Webserver-Prozess hat die Rechte des Benutzers `www-data` und kann alles tun, was dieser darf.
MAC (*Mandatory Access Control*) ergänzt systemweit durchgesetzte Regeln, die der Besitzer nicht umgehen kann – auch `root` nicht. Ein Webserver darf dann z. B. nur Dateien unter `/var/www` lesen und nur Port 80/443 öffnen, selbst wenn er als Root liefe. Zugriff erfordert beides: DAC *und* MAC müssen erlauben.

Merkmal	SELinux	AppArmor
Verbreitung	RHEL, Fedora, Rocky, Alma, Android	Ubuntu, Debian, openSUSE
Prinzip	Labels (Kontexte) an Dateien, Prozessen, Ports	Profile je Programm mit Pfadregeln
Granularität	Sehr fein, komplex	Einfacher, pfadbasiert
Lernkurve	Steil	Flach

23.2. SELinux

Niveau: Experte

```
$ getenforce                        # Enforcing / Permissive / Disabled
Enforcing
$ sestatus
$ sudo setenforce 0                 # Permissive (nur bis zum Neustart)
$ sudo setenforce 1
# dauerhaft: /etc/selinux/config -> SELINUX=enforcing
```

- **Enforcing**: Regeln werden durchgesetzt und Verstöße geblockt.
- **Permissive**: Verstöße werden nur protokolliert – ideal zur Fehlersuche.
- **Disabled**: Aus. Nicht empfohlen; spätere Reaktivierung erfordert vollständiges Neu-Labeln.

Kontexte haben die Form `Benutzer:Rolle:Typ:Stufe`. Entscheidend ist der **Typ** (`_t`). Die Richtlinie legt fest, welcher Prozesstyp auf welche Dateitypen zugreifen darf.

```
$ ls -Z /var/www/html/index.html
unconfined_u:object_r:httpd_sys_content_t:s0 /var/www/html/index.html
$ ps -eZ | grep httpd
system_u:system_r:httpd_t:s0  1234 ?  00:00:01 httpd
$ id -Z
$ ss -Ztlnp                                  # Kontexte der lauschenden Dienste
$ sudo semanage port -l | grep http
http_port_t   tcp   80, 81, 443, 488, 8008, 8009, 8443, 9000
```

Häufige Aufgaben:

```
# 1. Dateien haben falschen Kontext (z. B. mit mv aus /home verschoben)
$ sudo restorecon -Rv /var/www/html           # Standardkontext wiederherstellen

# 2. Eigenes Datenverzeichnis fuer den Webserver freigeben
$ sudo semanage fcontext -a -t httpd_sys_content_t "/srv/web(/.*)?"
$ sudo restorecon -Rv /srv/web

# 3. Dienst soll auf einem ungewoehnlichen Port lauschen
$ sudo semanage port -a -t http_port_t -p tcp 8888

# 4. Booleans: vorgesehene Schalter der Richtlinie
$ getsebool -a | grep httpd
$ sudo setsebool -P httpd_can_network_connect on      # -P = dauerhaft
$ sudo setsebool -P httpd_can_sendmail on
$ sudo setsebool -P httpd_enable_homedirs on

# 5. Nach Dateisystem-Neuaufbau: komplettes Neu-Labeln beim naechsten Start
$ sudo touch /.autorelabel && sudo reboot
```

Fehler untersuchen: Blockierte Zugriffe (“AVC denials”) stehen im Audit-Log. Das Werkzeug `setroubleshoot` liefert Erklärungen und Lösungsvorschläge.

```
$ sudo ausearch -m AVC -ts recent                   # AVC-Meldungen der letzten Zeit
$ sudo journalctl -t setroubleshoot
$ sudo sealert -a /var/log/audit/audit.log          # Erklaerung mit Vorschlaegen
$ sudo ausearch -m AVC -ts recent | audit2why
$ sudo ausearch -m AVC -ts recent | audit2allow -M meinmodul   # Modul erzeugen
$ sudo semodule -i meinmodul.pp                      # Modul laden
```

Faustregel

SELinux nicht abschalten, sondern die Ursache beheben: Zuerst prüfen, ob der Dateikontext stimmt (`restorecon`), dann ob ein Boolean passt (`getsebool`), dann ob ein Port freizugeben ist (`semanage port`). Nur wenn all das nicht hilft, ein eigenes Modul mit `audit2allow` erzeugen – und es vor dem Laden *lesen*, denn es erlaubt genau das, was geblockt wurde.

23.3. AppArmor

Niveau: Experte

AppArmor beschreibt für jedes Programm ein **Profil** mit erlaubten Dateizugriffen, Netzwerkzugriffen und Capabilities. Es ist unter Ubuntu und SUSE standardmäßig aktiv.

```
$ sudo aa-status                              # geladene Profile und Modi
$ ls /etc/apparmor.d/                         # Profile
$ sudo aa-enforce /etc/apparmor.d/usr.sbin.nginx
$ sudo aa-complain /etc/apparmor.d/usr.sbin.nginx    # Lernmodus: nur protokollieren
$ sudo aa-disable  /etc/apparmor.d/usr.sbin.nginx
$ sudo apparmor_parser -r /etc/apparmor.d/usr.sbin.nginx   # Profil neu laden
$ sudo aa-genprof /usr/local/bin/meintool      # Profil durch Beobachtung erstellen
$ sudo aa-logprof                              # Profil anhand des Logs verfeinern
$ sudo journalctl -k | grep -i apparmor        # Verstoesse
```

```
# /etc/apparmor.d/usr.local.bin.meintool
#include <tunables/global>
/usr/local/bin/meintool {
  #include <abstractions/base>
  capability net_bind_service,
  network inet stream,
  /etc/meintool.conf        r,
  /var/lib/meintool/**      rw,
  /var/log/meintool.log     w,
  deny /etc/shadow          r,
}
```

23.4. Das Audit-Framework

Niveau: Experte

Das **Linux Audit Framework** protokolliert sicherheitsrelevante Ereignisse im Kernel: Dateizugriffe, Systemaufrufe, Anmeldungen, Rechteänderungen. Der Dienst `auditd` schreibt das Protokoll nach `/var/log/audit/audit.log`. Regeln setzt man mit `auditctl`; dauerhaft liegen sie in `/etc/audit/rules.d/`.

```
$ sudo apt install auditd                                  # Debian: Paket auditd
$ sudo auditctl -l                                         # aktive Regeln
$ sudo auditctl -w /etc/passwd -p wa -k identitaet         # Schreib-/Attributzugriffe
    beobachten
$ sudo auditctl -w /etc/ssh/sshd_config -p wa -k sshd
$ sudo auditctl -a always,exit -F arch=b64 -S execve -F auid>=1000 -k befehle
$ sudo ausearch -k identitaet                              # Ereignisse suchen
$ sudo ausearch -m USER_LOGIN -ts today
$ sudo aureport --auth --summary                           # Berichte
$ sudo aureport -x --summary                               # ausgefuehrte Programme
```

```
# /etc/audit/rules.d/99-meine.rules
-w /etc/passwd -p wa -k identitaet
-w /etc/shadow -p wa -k identitaet
-w /etc/sudoers -p wa -k sudo
-w /etc/sudoers.d/ -p wa -k sudo
-a always,exit -F arch=b64 -S mount -S umount2 -k mounts
-e 2     # Regeln bis zum Neustart sperren
```

Die Regeln `-w` (watch) bedeuten: Pfad beobachten; `-p` wählt Zugriffsarten (`read`, `write`, `execute`, `attribute`); `-k` vergibt ein Stichwort zum späteren Suchen. Das Audit-Log ist schnell groß; Rotation

über `max_log_file` in `auditd.conf` und zentrale Sammlung (Kapitel 57) sind üblich. Sicherheitsstandards wie die CIS-Benchmarks und der BSI-Grundschutz fordern oft eine solche Protokollierung.

23.5. Härtung von Diensten

Niveau: Experte

Neben MAC gibt es weitere Schutzschichten, die sich gut kombinieren lassen:

- **systemd-Sandboxing** (Kapitel 32): u. a. `ProtectSystem=strict` und `PrivateTmp=yes`. Bewertung eines Dienstes: `systemd-analyze security`.
- **Kernel-Härtung** per `sysctl`, z. B. `kernel.kptr_restrict=2` und `kernel.dmesg_restrict=1`.
- **Mount-Optionen**: `noexec,nosuid,nodev` für `/tmp`, `/var/tmp`, `/dev/shm`.
- **Secure Boot** und **Lockdown** des Kernels, signierte Module.
- **Updates**, minimale Pakete, abgeschaltete unnötige Dienste (`ss -tulpn`).
- **Prüfwerkzeuge**: `lynis audit system`, OpenSCAP (`oscap`) gegen CIS/STIG-Profile.

```
$ systemd-analyze security sshd.service | tail -3
-> Overall exposure level for sshd.service: 9.6 UNSAFE
$ sudo lynis audit system                # Systemhaertung bewerten
$ sudo oscap xccdf eval --profile cis --report report.html /usr/share/xml/scap/ssg/content/
    ssg-rhel9-ds.xml
```

Übungen und Kontrollfragen

1. Ein Webserver kann Dateien in `/srv/web` nicht lesen, obwohl die Unix-Rechte stimmen (RHEL). Was prüfen Sie?

   Lösung: Ob SELinux blockiert (`ausearch -m AVC`); `ls -Z /srv/web`. Kontext mit `semanage fcontext -a -t httpd_sys_content_t "/srv/web(/.*)?"` setzen und `restorecon -Rv /srv/web` ausführen.

2. Wie testen Sie, ob SELinux die Ursache eines Problems ist, ohne es abzuschalten?

   Lösung: `setenforce 0` (Permissive): Verstöße werden nur protokolliert. Läuft es dann, liegt es an SELinux; danach mit `setenforce 1` und gezielter Korrektur beheben.

3. Worin unterscheiden sich SELinux und AppArmor grundsätzlich?

   Lösung: SELinux arbeitet mit Labels an Dateien und Prozessen, AppArmor mit pfadbasierten Profilen je Programm; SELinux ist feiner, aber komplexer.

4. Wie protokollieren Sie jede Änderung an `/etc/passwd`?

   Lösung: `auditctl -w /etc/passwd -p wa -k identitaet`, dauerhaft in `/etc/audit/rules.d/`; Suche mit `ausearch -k identitaet`.

5. Wozu dient `systemd-analyze security`?

   Lösung: Es bewertet die Sandboxing-Einstellungen einer Unit und zeigt, welche Schutzfunktionen noch fehlen.

24. Prozessverwaltung

Inhalt: Prozessmodell, Prozesszustände, `ps` und `top`, Signale, Jobsteuerung, Prioritäten, `/proc`.
Stufe: [E] Einsteiger bis [F] Fortgeschritten
Lernziele: Nach diesem Kapitel können Sie

- laufende Prozesse zu finden und zu beurteilen,
- Prozesse mit Signalen zu steuern und zu beenden,
- Hintergrundjobs und Prioritäten zu verwalten.

Vorkenntnisse: Kapitel 2 und 20.

24.1. Was ist ein Prozess?

Niveau: Einsteiger

Ein **Prozess** ist ein laufendes Programm samt seinem Speicher und Zustand. Jeder Prozess hat eine eindeutige **Prozess-ID** (PID), die PID seines Elternprozesses (PPID) sowie einen Eigentümer. Prozesse entstehen durch *fork*: Ein Elternprozess erzeugt eine Kopie, die dann ein neues Programm startet (*exec*). Der Urahn aller Prozesse hat die PID 1 (`systemd`, früher `init`).

Code	Zustand	Bedeutung
R	Running	Läuft oder ist rechenbereit
S	Sleeping	Wartet auf ein Ereignis (Normalzustand)
D	Uninterruptible	Wartet auf E/A (z. B. Platte, NFS); lässt sich nicht beenden
T	Stopped	Angehalten (`Strg+Z`, `SIGSTOP`)
Z	Zombie	Beendet, aber Elternprozess hat den Rückgabewert noch nicht abgeholt

24.2. Prozesse anzeigen

Niveau: Einsteiger

```
$ ps                        # Prozesse der aktuellen Shell
$ ps aux                    # alle Prozesse, benutzerorientiert (BSD-Syntax)
$ ps -ef                    # alle Prozesse, Standard-Syntax, mit PPID
$ ps -u tux                 # Prozesse eines Benutzers
$ ps -eo pid,ppid,user,%cpu,%mem,stat,cmd --sort=-%mem | head    # Spalten und Sortierung
$ pstree -p                 # Prozessbaum mit PIDs
$ pgrep -a sshd             # PIDs nach Namen suchen
$ top                       # laufende Uebersicht (q beendet)
$ htop                      # komfortabler (Paket 'htop')
```

```
$ ps aux | head -3
USER   PID %CPU %MEM    VSZ   RSS TTY   STAT START   TIME COMMAND
root     1  0.0  0.1 167880 11404 ?     Ss   08:00   0:02 /sbin/init
root     2  0.0  0.0      0     0 ?     S    08:00   0:00 [kthreadd]
```

VSZ ist der virtuelle, **RSS** der tatsächlich im RAM belegte Speicher. Eckige Klammern kennzeichnen Kernel-Threads. In `top` sind nützlich: `M` (nach Speicher sortieren), `P` (nach CPU), `k` (Prozess beenden), `1` (einzelne Kerne), `u` (nach Benutzer filtern).

24.2.1. Die Systemlast [F]

Die ersten Zeilen von `top` und `uptime` zeigen die **Load Average**: die mittlere Zahl wartender oder laufender Prozesse der letzten 1, 5 und 15 Minuten. Als Faustregel gilt: Liegt der Wert dauerhaft über der Zahl der CPU-Kerne (`nproc`), ist das System überlastet.

```
$ uptime
 10:15:01 up 3 days,  2:14,  2 users,  load average: 0.45, 0.60, 0.58
$ free -h                  # Speicher: 'available' ist die wichtige Spalte
$ vmstat 1 5               # CPU, Speicher, E/A, Swap im Sekundentakt
$ iostat -x 1              # Plattenlast (Paket sysstat)
```

24.3. Prozesse beenden: Signale

Niveau: Einsteiger

Prozesse werden über **Signale** gesteuert. `kill` sendet ein Signal an eine PID (der Name ist irreführend).

Nr.	Signal	Wirkung
1	`SIGHUP`	"Auflegen"; viele Dienste lesen daraufhin ihre Konfiguration neu
2	`SIGINT`	Unterbrechung (`Strg+C`)
9	`SIGKILL`	Sofort beenden, nicht abfangbar
15	`SIGTERM`	Höfliche Beendigung (Standard von `kill`); Programm kann aufräumen
18 / 19	`SIGCONT` / `SIGSTOP`	Fortsetzen / anhalten

```
$ kill 4711                # SIGTERM an PID 4711
$ kill -9 4711             # SIGKILL: nur wenn SIGTERM nicht hilft
$ kill -HUP 1234           # Konfiguration neu laden
$ killall firefox          # alle Prozesse mit diesem Namen
$ pkill -u tux -f "python backup"   # nach Muster beenden (Kommandozeile)
$ kill -l                  # alle Signale
```

> **Merke**
>
> Erst `SIGTERM` (15) senden und kurz warten. `SIGKILL` (9) gibt dem Programm keine Chance, Dateien zu schließen oder aufzuräumen. Prozesse im Zustand `D` lassen sich auch mit `kill -9` nicht beenden; Ursache ist meist ein hängendes Laufwerk. Zombies verschwinden, wenn man den *Elternprozess* beendet.

24.4. Prioritäten: nice und renice

Niveau: Fortgeschritten

Der **Nice-Wert** (`-20` bis `19`) beeinflusst, wie großzügig sich ein Prozess gegenüber anderen verhält. Hoher Wert = "netter" = niedrigere Priorität. Normale Benutzer dürfen den Wert nur erhöhen, `root` auch senken.

```
$ nice -n 10 backup.sh              # mit niedrigerer Prioritaet starten
$ renice -n 5 -p 4711               # laufenden Prozess aendern
# renice -n -5 -p 4711              # hoehere Prioritaet (nur root)
$ ps -o pid,ni,cmd -p 4711          # Nice-Wert anzeigen
$ ionice -c3 tar czf a.tgz /home    # niedrige E/A-Prioritaet (idle)
```

24.5. Offene Dateien und Ressourcen

Niveau: Fortgeschritten

```
$ lsof -p 4711                      # Dateien eines Prozesses (Paket lsof)
$ lsof -i :22                       # wer benutzt Port 22?
$ lsof +D /var/log                  # wer hat Dateien in diesem Verzeichnis offen?
$ lsof | grep deleted               # geloeschte, aber noch offene Dateien (Platz nicht frei
    !)
$ ls -l /proc/4711/fd               # offene Datei-Deskriptoren
$ cat /proc/4711/cmdline | tr '\0' ' '    # vollstaendige Kommandozeile
$ ulimit -a                         # Ressourcengrenzen der Shell
$ ulimit -n 4096                    # max. offene Dateien (Sitzung)
```

24.6. cgroups, OOM-Killer und Dienste im Hintergrund

Niveau: Experte

Der Kernel gruppiert Prozesse in **Control Groups** (*cgroups*), um Ressourcen zu begrenzen. `systemd` nutzt sie für jeden Dienst; auch Container bauen darauf auf.

```
$ systemd-cgls                      # cgroup-Baum
$ systemd-cgtop                     # Ressourcenverbrauch je cgroup
# systemd-run --scope -p MemoryMax=500M -p CPUQuota=50% ./rechenjob
```

Wird der Speicher knapp, beendet der **OOM-Killer** (*out of memory*) Prozesse. Dies erkennt man an Meldungen wie `Out of memory: Killed process` in `dmesg` oder im Journal (`journalctl -k | grep -i oom`). Abhilfe: mehr RAM, Swap, Speicherlimits oder das Beheben eines Speicherlecks.

25. Kernel, Module und Hardware

Inhalt: Aufgaben des Kernels, Versionen und Kernel-Pakete, Module mit `modprobe`, Kernelparameter und `sysctl`, `initramfs`, `udev`, Hardware-Informationen, Firmware, Kernel selbst bauen.
Stufe: [F] Fortgeschritten bis [X] Experte
Lernziele: Nach diesem Kapitel können Sie

- Kernelversionen, Module und Parameter zu verwalten,
- Hardware zu identifizieren und Treiberprobleme einzugrenzen,
- `udev`-Regeln und `sysctl`-Einstellungen anzuwenden.

Vorkenntnisse: Kapitel 24 (Prozesse), Kapitel 31 (Booten) und Kapitel 19.

25.1. Aufgaben des Kernels

Niveau: Fortgeschritten

Der **Kernel** ist der Kern des Betriebssystems. Er verwaltet Prozesse und Speicher, spricht mit der Hardware (Treiber), stellt Dateisysteme und das Netzwerk bereit und erzwingt Sicherheitsgrenzen. Programme sprechen mit ihm über **Systemaufrufe** (*system calls*); die Ein- und Ausgabe erfolgt über Gerätedateien in `/dev`. Linux ist ein *monolithischer Kernel mit ladbaren Modulen*: Treiber können im Betrieb nachgeladen werden.

```
$ uname -r                  # laufende Kernelversion
6.1.0-18-amd64
$ uname -a
$ cat /proc/version
$ cat /proc/cmdline         # beim Start uebergebene Parameter
$ ls /boot                  # installierte Kernel (vmlinuz-*), initramfs-Abbilder
$ dpkg -l 'linux-image-*' | grep ^ii      # Debian: installierte Kernel
$ rpm -q kernel                            # RHEL: installierte Kernel
$ zcat /proc/config.gz | grep CONFIG_EXT4  # Kernelkonfiguration (falls vorhanden)
```

Versionsnummern: `6.1.0-18-amd64` bedeutet Kernel 6.1, Distributions-Patchlevel 18, Architektur `amd64`. Kernel-Versionen mit Langzeitpflege (*LTS*) erhalten jahrelang Sicherheitskorrekturen.

25.2. Kernelmodule

Niveau: Fortgeschritten

```
$ lsmod                              # geladene Module
$ lsmod | grep -i e1000
$ modinfo e1000e                     # Beschreibung, Parameter, Abhaengigkeiten
$ sudo modprobe e1000e               # Modul samt Abhaengigkeiten laden
$ sudo modprobe -r e1000e            # entladen
$ sudo modprobe usb_storage delay_use=3    # mit Parameter
$ ls /lib/modules/$(uname -r)/kernel/drivers
$ sudo depmod -a                     # Abhaengigkeitsliste neu erzeugen
```

`modprobe` ist dem älteren `insmod`/`rmmod` vorzuziehen, weil es Abhängigkeiten auflöst. Dauerhafte Einstellungen:

```
# /etc/modules-load.d/vm.conf       - Module beim Start laden
kvm_intel
br_netfilter

# /etc/modprobe.d/blacklist-nouveau.conf  - Modul sperren
blacklist nouveau

# /etc/modprobe.d/e1000e.conf       - Optionen festlegen
options e1000e InterruptThrottleRate=3000
```

Nach Änderungen an `modprobe.d` muss das `initramfs` neu erzeugt werden (`update-initramfs -u` bzw. `dracut -f`), wenn das Modul früh beim Start gebraucht wird. Ob ein Modul den Kernel "verunreinigt" (*tainted*), zeigt `cat /proc/sys/kernel/tainted`; proprietäre Treiber und nicht signierte Module setzen das Flag.

25.3. Kernelparameter zur Laufzeit: sysctl

Niveau: Fortgeschritten

Das Verzeichnis `/proc/sys` enthält einstellbare Kernelparameter. Man liest und schreibt sie mit `sysctl` oder direkt per `echo`.

```
$ sysctl -a | wc -l                              # alle Parameter
$ sysctl net.ipv4.ip_forward
net.ipv4.ip_forward = 0
$ sudo sysctl -w net.ipv4.ip_forward=1           # sofort, nicht dauerhaft
$ cat /proc/sys/net/ipv4/ip_forward
$ echo 1 | sudo tee /proc/sys/net/ipv4/ip_forward

# dauerhaft
$ echo 'net.ipv4.ip_forward = 1' | sudo tee /etc/sysctl.d/90-forward.conf
$ sudo sysctl --system                           # alle Dateien neu laden
```

Parameter	Wirkung	Typischer Wert
`net.ipv4.ip_forward`	Router-Funktion (Weiterleitung)	1 bei Router/Gateway
`net.ipv4.tcp_syncookies`	Schutz vor SYN-Flood	1
`net.ipv4.conf.all.rp_filter`	Prüfung der Rückroute (Spoofing)	1
`vm.swappiness`	Neigung zum Auslagern (0–100)	10 bei Servern
`vm.max_map_count`	Max. Speicherbereiche (Elasticsearch u. a.)	262144
`fs.file-max`	Max. offene Dateien systemweit	je nach Last
`kernel.pid_max`	Max. Prozess-IDs	4194304
`kernel.sysrq`	Magic SysRq-Tasten	1 oder Maske
`fs.inotify.max_user_watches`	Datei-Beobachter (IDEs, Container)	524288

25.4. Das initramfs

Niveau: Fortgeschritten

Beim Start lädt der Bootloader den Kernel und ein kleines Dateisystemabbild im Speicher, das **initramfs** (früher *initrd*). Es enthält Module und Werkzeuge, um das eigentliche Wurzeldateisystem zu finden und einzubinden – etwa Treiber für RAID, LVM, Verschlüsselung oder Netzwerk.

```
$ ls -lh /boot/initrd.img-* /boot/initramfs-*
$ lsinitramfs /boot/initrd.img-$(uname -r) | head     # Debian: Inhalt
$ lsinitrd /boot/initramfs-$(uname -r).img | head     # RHEL
$ sudo update-initramfs -u -k all                     # Debian: neu erzeugen
$ sudo dracut -f --kver $(uname -r)                   # RHEL
```

Ist das initramfs fehlerhaft oder unvollständig, landet das System beim Start in einer Notfall-Shell (“`(initramfs)`” bzw. dracut-Shell). Typische Ursachen: fehlende Module für den Speichercontroller, falsche UUID in `fstab` oder `GRUB`, fehlende LVM-/RAID-Konfiguration.

25.5. udev: Geräteverwaltung

Niveau: Fortgeschritten

udev erzeugt die Gerätedateien in `/dev` und reagiert auf Hardware-Ereignisse (*Hotplug*). Es lädt Treiber, vergibt stabile Namen und führt Aktionen aus.

```
$ udevadm monitor                          # Ereignisse live beobachten (USB einstecken!)
$ udevadm info /dev/sdb                    # Eigenschaften eines Geraets
$ udevadm info -a /dev/sdb | head -30      # Attribute fuer Regeln
$ sudo udevadm control --reload            # Regeln neu laden
$ sudo udevadm trigger                     # Regeln auf vorhandene Geraete anwenden
$ udevadm test /sys/block/sdb
$ ls -l /dev/disk/by-id /dev/disk/by-uuid /dev/disk/by-label   # stabile Namen
```

Eigene Regeln stehen in `/etc/udev/rules.d/` (Dateiname mit Nummer und Endung `.rules`):

```
# /etc/udev/rules.d/99-usbstick.rules
# Stabilen Namen fuer einen bestimmten USB-Stick
SUBSYSTEM=="block", ENV{ID_SERIAL_SHORT}=="1234ABCD", SYMLINK+="backupstick"

# Rechte fuer ein USB-Geraet setzen
SUBSYSTEM=="usb", ATTR{idVendor}=="0403", ATTR{idProduct}=="6001", MODE="0660", GROUP="
    dialout"

# Skript bei Einstecken ausfuehren
ACTION=="add", SUBSYSTEM=="usb", ATTR{idVendor}=="1234", RUN+="/usr/local/bin/stick-erkannt.
    sh"
```

Netzwerkschnittstellen erhalten über `systemd` vorhersagbare Namen wie `enp3s0` (Ethernet, PCI-Bus 3, Slot 0) oder `wlp2s0`; diese Namen sind unabhängig von der Erkennungsreihenfolge.

25.6. Hardware-Informationen

Niveau: Einsteiger

Befehl	Anzeige
`lscpu`	Prozessor: Modell, Kerne, Cache, Virtualisierungsfähigkeit
`lsmem`, `free -h`	Hauptspeicher
`lspci -k`	PCI-Geräte und der verwendete Treiber (`Kernel driver in use`)
`lsusb -t`	USB-Geräte als Baum
`lsblk -f`	Blockgeräte, Dateisysteme, UUIDs
`lshw -short` (root)	Gesamtübersicht der Hardware
`dmidecode -t system` (root)	BIOS/UEFI-Angaben: Hersteller, Modell, Seriennummer, RAM-Bänke
`inxi -Fxz`	Zusammenfassung (Paket `inxi`)
`sensors`	Temperaturen und Lüfter (Paket `lm-sensors`)
`upower -d`, `acpi -V`	Akkustatus bei Laptops
`dmesg -T`	Kernelmeldungen mit Zeitstempel – Treiber, Fehler, Erkennung

```
$ lspci -k | grep -A3 -i ethernet
00:19.0 Ethernet controller: Intel Corporation Ethernet Connection I217-LM
        Subsystem: Lenovo ThinkPad T440s
        Kernel driver in use: e1000e
        Kernel modules: e1000e
$ sudo dmesg -T --level=err,warn | tail
$ journalctl -k -b -p warning              # Kernelmeldungen dieses Starts
```

Firmware: Manche Geräte (WLAN, Grafik) benötigen binäre Firmware aus `/lib/firmware`. Fehlt sie, meldet `dmesg` etwa `failed to load firmware`. Unter Debian liegt sie im Paket `firmware-linux` (`non-free-firmware`), unter RHEL/Fedora in `linux-firmware`. Mit `fwupdmgr` lässt sich Firmware von BIOS, SSD und Peripherie aktualisieren.

25.7. Kernel installieren, wechseln und selbst bauen

Niveau: Experte

```
$ sudo apt install linux-image-amd64 linux-headers-amd64    # Debian
$ sudo apt install linux-generic-hwe-22.04                   # Ubuntu: neuerer Kernel
$ sudo dnf install kernel kernel-devel                       # RHEL
$ sudo grubby --default-kernel                               # RHEL: Standardkernel
$ sudo apt autoremove --purge                                # alte Kernel loeschen
```

Wer einen eigenen Kernel baut (z. B. für Spezialhardware oder Lernzwecke), folgt dem Ablauf:

```
$ wget https://cdn.kernel.org/pub/linux/kernel/v6.x/linux-6.6.tar.xz
$ tar xf linux-6.6.tar.xz && cd linux-6.6
$ cp /boot/config-$(uname -r) .config          # Konfiguration der laufenden Kernels
$ make olddefconfig                            # neue Optionen mit Standardwert
$ make menuconfig                              # optional: Konfiguration anpassen
$ make -j$(nproc)
$ sudo make modules_install
$ sudo make install                            # kopiert nach /boot, erzeugt initramfs
$ sudo update-grub
```

Mit `make deb-pkg` bzw. `make rpm-pkg` entsteht stattdessen ein installierbares Paket. Den alten Kernel behalten, bis der neue sich bewährt hat; im GRUB-Menü kann er jederzeit gewählt werden. Wird **Secure Boot** verwendet, müssen Kernel und Module signiert sein.

> Hinweis

Für die meisten Aufgaben genügt der Distributionskernel. Kompilieren lohnt sich vor allem zum Lernen; im Betrieb ist ein Paket mit Sicherheitsupdates wartungsärmer.

Übungen und Kontrollfragen

1. Wie sehen Sie, welcher Treiber eine Netzwerkkarte steuert?

   Lösung: `lspci -k` (Zeile `Kernel driver in use`) oder `ethtool -i eth0`.

2. Wie aktivieren Sie IP-Weiterleitung dauerhaft?

   Lösung: Datei `/etc/sysctl.d/90-forward.conf` mit `net.ipv4.ip_forward = 1` anlegen und `sysctl --system` ausführen.

3. Wie verhindern Sie, dass der Treiber `nouveau` geladen wird?

   Lösung: In `/etc/modprobe.d/` eine Datei mit `blacklist nouveau` anlegen und das initramfs neu erzeugen.

4. Ein USB-Gerät soll immer als `/dev/messgeraet` erscheinen. Wie?

   Lösung: Per `udevadm info` Hersteller-/Produkt-ID oder Seriennummer bestimmen und in einer `udev`-Regel mit `SYMLINK+=` festlegen; `udevadm control --reload` und `trigger`.

5. Das System startet nach einem Kernel-Update nicht. Was tun?

   Lösung: Im GRUB-Menü unter “Erweiterte Optionen” den vorherigen Kernel wählen; danach den defekten Kernel analysieren/entfernen und das initramfs neu erzeugen.

26. Platten und Partitionen

Inhalt: Blockgeräte, Partitionstabellen (MBR, GPT), Partitionieren mit `fdisk` und `parted`, Loop-Geräte.
Stufe: [E] Einsteiger bis [X] Experte
Lernziele: Nach diesem Kapitel können Sie

- Laufwerke und Partitionen zu identifizieren,
- Platten zu partitionieren,
- Loop-Geräte als Übungsumgebung zu nutzen (RAID, LVM und Verschlüsselung: folgende Kapitel).

Vorkenntnisse: Kapitel 6, 18 und 22; Verständnis von Mount-Punkten.

26.1. Massenspeicher unter Linux

Niveau: Einsteiger

Linux spricht Datenträger über **Gerätedateien** in `/dev` an. Der Name verrät die Anschlussart:

Gerätename	Bedeutung
`/dev/sda`, `sdb`, ...	SATA-, SAS- und USB-Platten sowie SCSI/virtuelle Datenträger in der Reihenfolge der Erkennung
`/dev/sda1`, `sda2`	Partitionen auf `sda`
`/dev/nvme0n1`, `nvme0n1p1`	NVMe-SSD (Controller 0, Namespace 1) und deren Partition 1
`/dev/vda`, `vdb`	Virtio-Platten in virtuellen Maschinen (KVM)
`/dev/mmcblk0p1`	SD-Karten (z. B. Raspberry Pi)
`/dev/sr0`	Optisches Laufwerk
`/dev/mapper/...`, `/dev/dm-*`	Device-Mapper: LVM, RAID, Verschlüsselung
`/dev/loop0`	Loop-Gerät: Datei als Datenträger

Wichtig

Die Reihenfolge `sda`, `sdb` kann sich bei jedem Start ändern (z. B. durch USB-Sticks). In `/etc/fstab` und in Skripten daher **UUIDs** oder Labels statt Gerätenamen verwenden (`blkid`, `lsblk -f`, `/dev/disk/by-uuid/`).

```
$ lsblk                              # Baumansicht
NAME   MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
sda      8:0    0    50G  0 disk
|-sda1   8:1    0   512M  0 part /boot/efi
`-sda2   8:2    0  49,5G  0 part /
sdb      8:16   0    20G  0 disk
$ lsblk -f                           # mit Dateisystem, Label und UUID
$ sudo blkid                         # UUIDs aller Dateisysteme
$ sudo fdisk -l /dev/sdb             # Partitionstabelle anzeigen
$ ls -l /dev/disk/by-uuid/           # stabile Namen
$ cat /proc/partitions               # vom Kernel erkannte Platten
$ sudo smartctl -H /dev/sda          # SMART-Gesundheit (Paket smartmontools)
```

26.2. Partitionierung: MBR und GPT

Niveau: Einsteiger

Eine Platte wird in **Partitionen** unterteilt, die der Kernel als eigene Geräte behandelt. Zwei Partitionstabellen sind gebräuchlich:

	MBR (DOS)	GPT
Zugehöriger Start	BIOS	UEFI (auch mit BIOS möglich)
Maximale Plattengröße	2 TiB	9,4 ZB
Partitionen	4 primäre, oder 3 + 1 erweiterte mit logischen	128 (Standard)
Redundanz	keine	Kopie der Tabelle am Plattenende, Prüfsummen

Bei UEFI-Systemen benötigt man eine **ESP** (*EFI System Partition*, FAT32, 100–1000 MiB, eingehängt unter `/boot/efi`). Typische Aufteilung eines Servers:

Partition	Größe	Dateisystem	Zweck
ESP	512 MiB	FAT32	UEFI-Bootloader (`/boot/efi`)
`/boot`	1 GiB	ext4	Kernel, Initramfs (optional)
/ (Root)	ab 20 GiB	ext4 / XFS	System und Programme
Swap	siehe Kapitel 29	swap	Auslagerung
`/var`, `/home`	nach Bedarf	ext4 / XFS	Optional getrennt, damit volllaufende Logs das System nicht blockieren

26.3. Partitionieren mit fdisk, gdisk und parted

Niveau: Einsteiger

`fdisk` (MBR und GPT), `gdisk` (GPT) und `cfdisk` (Menü) arbeiten interaktiv. Änderungen werden erst mit `w` geschrieben.

```
# fdisk /dev/sdb
Command (m for help): g               # neue GPT-Tabelle (o = MBR)
Command (m for help): n               # neue Partition
Partition number (1-128, default 1):  # Enter
First sector (2048-41943006, default 2048):      # Enter
Last sector, +/-sectors or +/-size{K,M,G,T,P}: +10G
Command (m for help): t               # Partitionstyp (z. B. 'Linux LVM')
Command (m for help): p               # Tabelle anzeigen
Command (m for help): w               # schreiben und beenden (q = verwerfen!)
# partprobe /dev/sdb                  # Kernel ueber neue Tabelle informieren
```

```
# parted /dev/sdb --script mklabel gpt mkpart data ext4 1MiB 100%     # nicht interaktiv
# sgdisk -n1:0:+10G -t1:8300 /dev/sdb         # skriptbar (GPT)
# sfdisk -d /dev/sda > sda.dump               # Tabelle sichern
# sfdisk /dev/sdb < sda.dump                  # Tabelle auf andere Platte uebertragen
```

> **Achtung**
>
> Fehler beim Partitionieren zerstören Daten. **Zuerst** mit `lsblk` das richtige Gerät identifizieren, die Tabelle mit `sfdisk -d` sichern und möglichst in einer virtuellen Maschine üben. Mit `wipefs -a /dev/sdb` löscht man Signaturen alter Dateisysteme; auch das ist nicht umkehrbar.

26.4. Loop-Geräte als Übungsumgebung

Niveau: Experte

```
# truncate -s 1G test.img                    # 1-GiB-Datei als "Platte"
# losetup -fP --show test.img                # als /dev/loopN verfuegbar machen
/dev/loop0
# mkfs.ext4 /dev/loop0 && mount /dev/loop0 /mnt     # Uebungsumgebung ohne echte Platte
# umount /mnt && losetup -d /dev/loop0
```

Weiterführend

Software-RAID mit `mdadm` behandelt Kapitel 27. Anschließend folgen in Kapitel 28 LVM (flexible Volumes, Snapshots, RAID in LVM) und die Verschlüsselung mit LUKS.

27. Software-RAID mit mdadm

Inhalt: Ziele und Level von RAID, Software-, Hardware- und Fake-RAID, RAID mit `mdadm` anlegen, Überwachung, Plattenausfall und Austausch, Hot-Spare, Verbund erweitern, Scrubbing, Zusammenspiel mit LVM und LUKS, Grenzen von RAID.

Stufe: [E] Einsteiger bis [X] Experte

Lernziele: Nach diesem Kapitel können Sie

- die gängigen RAID-Level zu unterscheiden und deren nutzbare Kapazität sowie Ausfalltoleranz zu berechnen,
- mit `mdadm` ein Software-RAID anzulegen, zu überwachen und dauerhaft einzubinden,
- den Ausfall einer Platte zu erkennen, die Platte zu ersetzen und den Verbund wiederherzustellen.

Vorkenntnisse: Kapitel 26 (Platten und Partitionen), Kapitel 29 (Dateisysteme). Im Anschluss folgt LVM mit eigener RAID-Funktion (Kapitel 28).

27.1. Wozu RAID?

Niveau: Einsteiger

RAID (*Redundant Array of Independent Disks*) fasst mehrere Platten zu einem logischen Laufwerk zusammen. Je nach *Level* steht eines der folgenden Ziele im Vordergrund:

- **Ausfallsicherheit**: Fällt eine Platte aus, läuft der Betrieb weiter (Spiegelung oder Parität).
- **Leistung**: Daten werden auf mehrere Platten verteilt (*Striping*), Lese- und Schreibzugriffe laufen parallel.
- **Kapazität**: Mehrere kleine Platten erscheinen als ein großes Laufwerk.

> **RAID ist kein Backup**
>
> RAID schützt vor dem Ausfall einer *Platte*. Es schützt nicht vor versehentlichem Löschen, Dateisystemfehlern, Schadsoftware, Brand oder Diebstahl: Jede Änderung wird sofort auf alle Platten übernommen. Ein Backup (Kapitel 53) bleibt unverzichtbar.

Es gibt drei Umsetzungen:

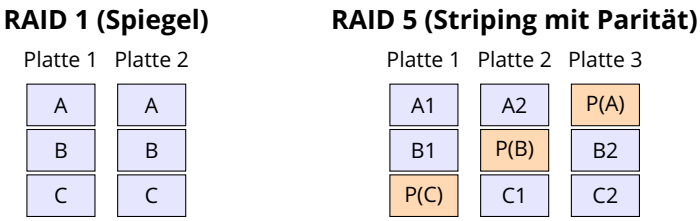
Art	Wer rechnet?	Eigenschaften
Software-RAID (`md`)	Der Linux-Kernel	Keine Spezialhardware, portabel zwischen Rechnern, gut dokumentiert; belegt etwas CPU
Hardware-RAID	Eigener RAID-Controller	Eigener Cache mit Batterie; Verbund ist an das Controller-Modell gebunden
Fake-RAID (BIOS/UEFI-RAID)	Treiber im Betriebssystem	Eingestellt im Firmware-Menü, aber rechnet in Software; unter Linux besser direkt `mdadm` nutzen

27.2. RAID-Level im Überblick

Niveau: Einsteiger

Die Datenverteilung bestimmt der RAID-Level. Mit n Platten gleicher Größe G gilt:

Level	Platten	Nutzbar	Ausfall verkraftet	Eigenschaft
RAID 0	≥ 2	$n \cdot G$	keine Platte	Striping: schnell, *keine* Redundanz; ein Ausfall zerstört alles
RAID 1	≥ 2	G	$n-1$ Platten	Spiegelung: einfach und robust, halbe Kapazität bei zwei Platten
RAID 5	≥ 3	$(n-1) \cdot G$	1 Platte	Striping mit verteilter Parität; guter Kompromiss
RAID 6	≥ 4	$(n-2) \cdot G$	2 Platten	Doppelte Parität; für große Platten und große Verbünde
RAID 10	≥ 4	$n/2 \cdot G$	1 je Spiegelpaar	Spiegel plus Striping: schnell und sicher, teuer



Die **Parität** (P) ist eine Prüfsumme (XOR) über die Datenblöcke einer Zeile. Fällt eine Platte aus, lässt sich ihr Inhalt aus den übrigen Blöcken und der Parität rekonstruieren. Dafür muss jeder Schreibzugriff auch die Parität aktualisieren, was RAID 5 und 6 bei vielen kleinen Schreibzugriffen langsamer macht als RAID 1 oder 10.

- **Gleich große Platten**: Der Verbund richtet sich nach der *kleinsten* Platte; Mehrkapazität größerer Platten bleibt ungenutzt.
- **Linear** und **JBOD**: Platten werden nur hintereinandergehängt, ohne Redundanz; kaum empfehlenswert.
- **Praxisempfehlung**: RAID 1 für Systemplatten und kleine Server, RAID 10 für Datenbanken, RAID 6 für große Datenplatten (viele Terabyte, lange Wiederherstellungszeit).

27.3. RAID mit mdadm anlegen

Niveau: Fortgeschritten

Der Linux-Kernel stellt Software-RAID über den **md**-Treiber (*multiple devices*) bereit, verwaltet wird er mit `mdadm`. Als Bausteine dienen ganze Platten oder Partitionen; üblich ist eine Partition je Platte mit dem Typ *Linux RAID* (bei GPT `fd00`).

```
$ sudo apt install mdadm                          # RHEL: dnf install mdadm
$ lsblk                                           # richtige Platten identifizieren!

# Partitionen als RAID-Mitglieder vorbereiten (Beispiel: GPT)
$ sudo sgdisk -n1:0:0 -t1:fd00 /dev/sdb
$ sudo sgdisk -n1:0:0 -t1:fd00 /dev/sdc

# RAID 1 (Spiegel) aus zwei Partitionen
$ sudo mdadm --create /dev/md0 --level=1 --raid-devices=2 /dev/sdb1 /dev/sdc1
# RAID 5 aus drei Platten
$ sudo mdadm --create /dev/md1 --level=5 --raid-devices=3 /dev/sdb1 /dev/sdc1 /dev/sdd1
# RAID 10 aus vier Platten
$ sudo mdadm --create /dev/md2 --level=10 --raid-devices=4 /dev/sd[b-e]1
```

Direkt nach dem Anlegen beginnt der **Resync**: Die Platten werden abgeglichen. Das RAID ist währenddessen bereits benutzbar, wenn auch langsamer. Danach verhält sich `/dev/md0` wie eine normale Partition:

```
$ sudo mkfs.ext4 /dev/md0                          # Dateisystem anlegen
$ sudo mkdir -p /srv/daten
$ sudo mount /dev/md0 /srv/daten
$ sudo blkid /dev/md0                              # UUID fuer /etc/fstab
$ echo 'UUID=1b2c...  /srv/daten  ext4  defaults,nofail  0  2' | sudo tee -a /etc/fstab
```

Damit der Verbund nach dem Neustart unter einem festen Namen zusammengesetzt wird, trägt man ihn in die `mdadm.conf` ein und aktualisiert das initramfs (Kapitel 25):

```
$ sudo mdadm --detail --scan | sudo tee -a /etc/mdadm/mdadm.conf     # RHEL: /etc/mdadm.conf
ARRAY /dev/md0 metadata=1.2 name=server01:0 UUID=8f3c...
$ sudo update-initramfs -u                         # Debian/Ubuntu
$ sudo dracut --force                              # RHEL/Fedora
```

> **Zum Üben**
>
> Echte Platten sind nicht nötig: Mit Loop-Geräten (Kapitel 26) lässt sich jeder Befehl dieses Kapitels gefahrlos ausprobieren, z. B. `truncate -s 500M d1.img d2.img d3.img`, jeweils `losetup -f --show d1.img` und anschließend `mdadm --create /dev/md9 --level=5 --raid-devices=3 /dev/loop0 /dev/loop1 /dev/loop2`.

27.4. Status und Überwachung

Niveau: Fortgeschritten

```
$ cat /proc/mdstat                                 # Kurzstatus aller Verbuende
Personalities : [raid1] [raid6] [raid5] [raid4] [raid10]
md0 : active raid1 sdc1[1] sdb1[0]
      20954112 blocks super 1.2 [2/2] [UU]
      [==>..................]  resync = 12.4% (2600000/20954112) finish=8.2min speed=37000K/
    sec

$ sudo mdadm --detail /dev/md0                     # ausfuehrliche Informationen
$ sudo mdadm --examine /dev/sdb1                   # Superblock einer Mitgliedsplatte
```

In `/proc/mdstat` zeigt die Angabe `[UU]` an, dass beide Platten aktiv sind (*Up*). Ein Unterstrich wie in `[U_]` bedeutet: Ein Mitglied fehlt oder ist ausgefallen, der Verbund läuft *degradiert*. Wichtige Zustände in `mdadm --detail`:

State	Bedeutung
`clean, active`	Verbund in Ordnung
`degraded`	Mindestens eine Platte fehlt oder ist ausgefallen; Redundanz teilweise verloren
`resyncing, recovering`	Abgleich bzw. Wiederherstellung läuft
`failed`	Mehr Platten ausgefallen als der Level verkraftet: Daten nicht mehr verfügbar

Ein degradierter Verbund fällt im Alltag nicht auf, solange er läuft. Deshalb ist die **automatische Benachrichtigung** entscheidend:

```
# /etc/mdadm/mdadm.conf
MAILADDR admin@example.org

$ sudo systemctl status mdmonitor              # RHEL; Debian: mdadm.service bzw. mdmonitor
$ sudo mdadm --monitor --scan --oneshot --test # Testmeldung ausloesen
$ journalctl -k | grep -i md                   # Kernelmeldungen zum RAID
```

Zusätzlich sollte die Überwachung (Kapitel 57) den Zustand abfragen, z. B. per Skript, das `/proc/mdstat` auf _ prüft, und die SMART-Werte der Platten beobachten (Kapitel 28).

27.5. Plattenausfall und Austausch

Niveau: Fortgeschritten

Fällt eine Platte aus, markiert der Kernel sie als *faulty* und läuft degradiert weiter. Das Vorgehen beim Austausch:

```
# 1. Fehler erkennen
$ cat /proc/mdstat                             # [U_]
$ sudo mdadm --detail /dev/md0                 # welche Platte ist "faulty"?

# 2. Defekte Platte aus dem Verbund nehmen (bei Test erst als fehlerhaft markieren)
$ sudo mdadm /dev/md0 --fail /dev/sdb1         # nur beim Ueben noetig
$ sudo mdadm /dev/md0 --remove /dev/sdb1

# 3. Platte tauschen, Partitionstabelle von der gesunden Platte uebernehmen
$ sudo sfdisk -d /dev/sdc | sudo sfdisk /dev/sdb      # MBR/GPT; bei GPT danach: sgdisk -G /
    dev/sdb
$ lsblk                                        # neue Platte erkannt?

# 4. Ersatz hinzufuegen: Rekonstruktion startet automatisch
$ sudo mdadm /dev/md0 --add /dev/sdb1
$ watch cat /proc/mdstat                       # Fortschritt der Wiederherstellung
```

> **Vorsicht**
>
> Vor dem Ziehen der Platte die *richtige* Platte eindeutig identifizieren (Seriennummer mit `smartctl -i /dev/sdb` oder `lsblk -o NAME,SERIAL`). Wird versehentlich die gesunde Platte entfernt, ist der Verbund ausgefallen. Während der Wiederherstellung ist die Redundanz noch nicht hergestellt; ein weiterer Ausfall in dieser Phase bedeutet bei RAID 1 und 5 Datenverlust.

Hot-Spare: Eine zusätzliche, im Normalbetrieb ungenutzte Platte springt bei einem Ausfall *automatisch* ein. Die Wiederherstellung beginnt sofort, ohne dass ein Administrator eingreifen muss.

```
$ sudo mdadm /dev/md1 --add /dev/sde1          # zu einem intakten Verbund: wird Spare
$ sudo mdadm --detail /dev/md1 | grep -iE 'spare|state'
```

27.6. Verbund stoppen und wieder zusammensetzen

Niveau: Fortgeschritten

Die Metadaten (*Superblock*) liegen auf den Mitgliedsplatten selbst. Dadurch kann ein Verbund auch auf einem anderen Rechner oder in einem Rettungssystem wieder zusammengesetzt werden.

```
$ sudo umount /srv/daten
$ sudo mdadm --stop /dev/md0                    # Verbund anhalten

$ sudo mdadm --assemble --scan                  # alle Verbuende anhand der Superblocks suchen
$ sudo mdadm --assemble /dev/md0 /dev/sdb1 /dev/sdc1     # gezielt zusammensetzen
$ sudo mdadm --assemble --force /dev/md0 /dev/sdb1 /dev/sdc1   # Notfall: veraltetes Mitglied
    erzwingen
```

Erscheint ein Verbund nach dem Start als `/dev/md127` statt `/dev/md0`, fehlt meist der Eintrag in der `mdadm.conf` im initramfs. Soll eine Platte endgültig aus einem Verbund gelöst und anderweitig genutzt werden, entfernt man ihren Superblock:

```
$ sudo mdadm --zero-superblock /dev/sdb1       # loescht die RAID-Metadaten dieser Partition
```

27.7. Erweitern und umbauen

Niveau: Experte

Verbünde lassen sich im Betrieb vergrößern oder in ihrem Level ändern (*reshape*). Das dauert je nach Größe Stunden bis Tage und belastet die Platten stark; vorher ein Backup anlegen.

```
# RAID 5 um eine vierte Platte erweitern
$ sudo mdadm /dev/md1 --add /dev/sde1
$ sudo mdadm --grow /dev/md1 --raid-devices=4
$ watch cat /proc/mdstat                        # Reshape-Fortschritt
$ sudo resize2fs /dev/md1                       # danach Dateisystem anpassen (XFS: xfs_growfs
    )

# Alle Platten durch groessere ersetzen, dann Verbund auf neue Groesse bringen
$ sudo mdadm /dev/md0 --fail /dev/sdb1 --remove /dev/sdb1
$ sudo mdadm /dev/md0 --add /dev/sdb1           # (neue, groessere Platte) - fuer jede Platte
    wiederholen
$ sudo mdadm --grow /dev/md0 --size=max

# RAID 1 mit zwei Platten in ein RAID 5 mit drei Platten umwandeln
$ sudo mdadm --grow /dev/md0 --level=5 --raid-devices=3 --add /dev/sdd1
```

Write-Intent-Bitmap: Sie merkt sich, welche Bereiche gerade beschrieben werden. Nach einem Absturz oder wenn eine Platte kurz weg war, müssen nur diese Bereiche abgeglichen werden statt des ganzen Verbunds.

```
$ sudo mdadm --grow /dev/md0 --bitmap=internal
```

Scrubbing: Beim Lesen bleiben schleichende Fehler (defekte Sektoren, die lange niemand gelesen hat) unbemerkt. Ein regelmäßiger Prüflauf liest den ganzen Verbund und deckt sie auf, solange noch Redundanz vorhanden ist.

```
$ echo check | sudo tee /sys/block/md0/md/sync_action     # Pruefung starten
$ cat /sys/block/md0/md/mismatch_cnt                       # 0 = keine Abweichungen
$ sudo /usr/share/mdadm/checkarray --all                   # Debian: monatlicher Cron/Timer
```

Leistung: Beim Anlegen lässt sich die *Chunk-Größe* (Stripe-Segment) mit `--chunk=` festlegen (Standard 512 KiB). Die Geschwindigkeit einer Wiederherstellung begrenzen die Kernelparameter `dev.raid.speed_limit_min` und `speed_limit_max`:

```
$ sudo sysctl dev.raid.speed_limit_min dev.raid.speed_limit_max
$ sudo sysctl -w dev.raid.speed_limit_min=100000          # Rebuild beschleunigen (KB/s je
    Platte)
$ echo 8192 | sudo tee /sys/block/md1/md/stripe_cache_size # RAID 5/6: Schreibleistung
    erhoehen
```

27.8. RAID mit LVM, LUKS und beim Systemstart

Niveau: Experte

Die Schichten lassen sich frei stapeln. Eine verbreitete Reihenfolge von unten nach oben:

Schicht	Beispiel
Dateisystem	ext4, XFS
Logical Volume	`/dev/vg_daten/lv_web`
Verschlüsselung (optional)	`/dev/mapper/geheim` (LUKS)
Volume Group mit PV	PV `/dev/md0`
RAID	`/dev/md0` aus `/dev/sdb1` und `/dev/sdc1`
Platten	`/dev/sdb`, `/dev/sdc`

```
$ sudo pvcreate /dev/md0                          # RAID als Physical Volume
$ sudo vgcreate vg_daten /dev/md0
$ sudo lvcreate -L 20G -n lv_web vg_daten
```

Die Spiegelung sorgt für Ausfallsicherheit, LVM für flexible Größen und Snapshots (Kapitel 28). LVM kann Spiegelung und Parität auch selbst übernehmen; dieser Weg wird dort im Abschnitt *RAID in LVM: Spiegelung und Parität* beschrieben. Wird die Verschlüsselung *über* dem RAID angelegt, muss sie nur einmal entsperrt werden und schützt alle Platten gemeinsam.

Systemplatte spiegeln: Auch `/` lässt sich auf RAID 1 legen. Das Installationsprogramm (Debian, Ubuntu, Fedora) bietet dies bei der Partitionierung an. Bei UEFI-Systemen muss die EFI-Systempartition (ESP) *auf jeder Platte* vorhanden sein; sie wird entweder mit RAID 1 und den Metadaten `--metadata=1.0` (Superblock am Ende) angelegt oder manuell auf beiden Platten gepflegt. Der Bootloader (Kapitel 31) muss auf allen Platten installiert sein, damit das System auch nach Ausfall der ersten Platte startet.

```
$ sudo grub-install /dev/sdb                      # Bootloader auch auf die zweite Platte
$ sudo mdadm --create /dev/md0 --level=1 --raid-devices=2 --metadata=1.0 /dev/sda1 /dev/sdb1
      # ESP als RAID 1
```

27.9. Grenzen und Planung

Niveau: Fortgeschritten

- **Kein Backup**: Siehe oben; zusätzlich schützt RAID nicht vor Fehlern im Dateisystem oder in der Anwendung.
- **Große Platten, lange Wiederherstellung**: Bei Platten mit vielen Terabyte dauert ein Rebuild Stunden bis Tage. In dieser Zeit ist ein RAID 5 anfällig: Stößt der Abgleich auf einen unlesbaren Sektor einer weiteren Platte, ist der Verbund verloren. Für große Datenplatten daher RAID 6 oder RAID 10 bevorzugen.
- **Gleichzeitiger Ausfall**: Platten gleichen Typs und gleicher Charge altern ähnlich. Platten mischen (Hersteller, Alter) und Ersatz bereithalten.
- **Write-Hole**: Bricht bei RAID 5/6 der Strom mitten im Schreiben ab, können Daten und Parität inkonsistent bleiben. Abhilfe bieten Write-Intent-Bitmap, ein Journal-Gerät (`--write-journal`), eine USV und das regelmäßige Scrubbing.

- **Alternativen**: Btrfs und ZFS bringen Spiegelung, Parität und Prüfsummen im Dateisystem mit und erkennen stille Datenfehler (Kapitel 30). Für eine Spiegelung ganzer Server über das Netz dient DRBD (Kapitel 60).
- **Überwachung testen**: Benachrichtigung und Austausch einmal mit Loop-Geräten oder in einer VM durchspielen, bevor es ernst wird.

Übungen und Kontrollfragen

1. Wie viel nutzbare Kapazität hat ein RAID 5 aus vier 4-TB-Platten, wie viel ein RAID 6 aus sechs 8-TB-Platten?

   Lösung: RAID 5: $(4-1)\cdot 4 = 12$ TB. RAID 6: $(6-2)\cdot 8 = 32$ TB.

2. Welche Ausfälle verkraftet ein RAID 10 aus vier Platten?

   Lösung: Eine Platte je Spiegelpaar, also bis zu zwei Platten, solange nicht beide Platten desselben Paares ausfallen.

3. Was ist der Unterschied zwischen RAID 1 und einem Backup?

   Lösung: RAID 1 hält zwei Kopien *gleichzeitig*; Löschfehler, Verschlüsselungstrojaner oder Dateisystemfehler werden sofort gespiegelt. Ein Backup bewahrt frühere Zustände getrennt auf.

4. Woran erkennen Sie in `/proc/mdstat`, dass ein Verbund degradiert ist?

   Lösung: Am Unterstrich in der Statusanzeige, z. B. `[U_]` statt `[UU]`; `mdadm --detail` zeigt zusätzlich `degraded`.

5. Wie ersetzen Sie die ausgefallene Platte `/dev/sdb` in `/dev/md0`?

   Lösung: Mit `mdadm /dev/md0 --remove /dev/sdb1` entfernen, neue Platte einbauen und partitionieren, mit `mdadm /dev/md0 --add /dev/sdb1` hinzufügen und die Wiederherstellung in `/proc/mdstat` beobachten.

6. Wozu dient ein Hot-Spare?

   Lösung: Eine zusätzliche Platte, die bei einem Ausfall automatisch einspringt; die Wiederherstellung beginnt sofort, die Zeit im degradierten Zustand verkürzt sich.

7. Wie stellen Sie sicher, dass das RAID nach einem Neustart wieder unter `/dev/md0` erscheint?

   Lösung: Verbund mit `mdadm --detail --scan` in die `mdadm.conf` eintragen, initramfs aktualisieren und das Dateisystem per UUID in `/etc/fstab` einbinden.

8. Warum ist RAID 5 bei sehr großen Platten problematisch?

   Lösung: Der Rebuild dauert sehr lange; ein unlesbarer Sektor auf einer weiteren Platte in dieser Zeit führt zum Totalverlust. RAID 6 oder RAID 10 sind sicherer.

28. LVM und Verschlüsselung

Inhalt: Logical Volume Manager (Physical Volumes, Volume Groups, Logical Volumes, Snapshots, Thin Provisioning), RAID direkt in LVM (Spiegelung, Parität, Umbau, Reparatur), Festplattenverschlüsselung mit LUKS, SMART-Überwachung.

Stufe: [F] Fortgeschritten bis [X] Experte

Lernziele: Nach diesem Kapitel können Sie

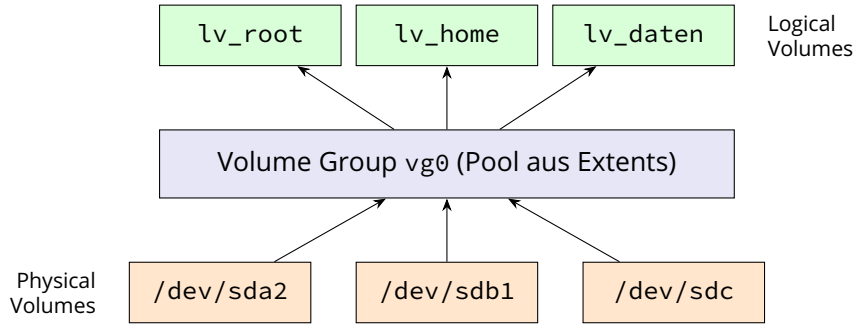
- Speicher flexibel mit LVM zu organisieren und online zu erweitern,
- mit LVM ohne `mdadm` gespiegelte und paritätsgesicherte Volumes anzulegen und nach einem Plattenausfall zu reparieren,
- Datenträger mit LUKS zu verschlüsseln und beim Start einzubinden,
- den Zustand von Platten mit SMART zu überwachen.

Vorkenntnisse: Kapitel 26 (Platten), Kapitel 29 (Dateisysteme) und Kapitel 27 (Software-RAID mit `mdadm`; die RAID-Level werden dort erklärt).

28.1. Warum LVM?

Niveau: Fortgeschritten

Klassische Partitionen sind starr: Größe und Lage werden bei der Installation festgelegt und lassen sich nur mühsam ändern. Der **Logical Volume Manager** (LVM) schiebt eine Abstraktionsschicht dazwischen. Mehrere physische Datenträger werden zu einem Speicherpool zusammengefasst, aus dem man flexibel Volumes entnimmt.



- **PV** (Physical Volume): Partition oder ganzer Datenträger, der LVM zur Verfügung gestellt wird.
- **VG** (Volume Group): Pool aus einem oder mehreren PVs, aufgeteilt in *Extents* (Standard 4 MiB).
- **LV** (Logical Volume): Aus der VG herausgeschnittenes Volume, das wie eine Partition mit einem Dateisystem versehen wird.

Vorteile: Größenänderung im laufenden Betrieb, Zusammenfassung mehrerer Platten, Snapshots, Verschieben von Daten zwischen Platten ohne Unterbrechung.

28.2. LVM einrichten

Niveau: Fortgeschritten

```
# 1. Physical Volumes anlegen
$ sudo pvcreate /dev/sdb /dev/sdc
$ sudo pvs                                    # Uebersicht (pvdisplay: Details)

# 2. Volume Group anlegen
$ sudo vgcreate vg_daten /dev/sdb /dev/sdc
$ sudo vgs

# 3. Logical Volumes anlegen
$ sudo lvcreate -L 20G -n lv_web vg_daten      # feste Groesse
$ sudo lvcreate -l 100%FREE -n lv_db vg_daten  # den Rest nutzen
$ sudo lvs

# 4. Dateisystem anlegen und einhaengen
$ sudo mkfs.ext4 /dev/vg_daten/lv_web
$ sudo mkdir -p /srv/web
$ sudo mount /dev/vg_daten/lv_web /srv/web
$ echo '/dev/vg_daten/lv_web /srv/web ext4 defaults 0 2' | sudo tee -a /etc/fstab
```

Der Gerätename ist stets `/dev/<VG>/<LV>` (alternativ `/dev/mapper/<VG>-<LV>`). Mit `lsblk` und `vgs`/`lvs` behält man die Übersicht.

28.3. Volumes vergrößern und verkleinern

Niveau: Fortgeschritten

```
# Vergroessern - im laufenden Betrieb, Dateisystem gleich mit
$ sudo lvextend -L +10G -r /dev/vg_daten/lv_web        # -r = Dateisystem anpassen
$ sudo lvextend -l +100%FREE -r /dev/vg_daten/lv_web

# Neue Platte in die Gruppe aufnehmen
$ sudo pvcreate /dev/sdd
$ sudo vgextend vg_daten /dev/sdd

# Verkleinern - nur ext4, NICHT XFS; erst aushaengen, Dateisystem verkleinern
$ sudo umount /srv/web
$ sudo e2fsck -f /dev/vg_daten/lv_web
$ sudo lvreduce -L 15G -r /dev/vg_daten/lv_web

# Platte aus der Gruppe entfernen (Daten werden verschoben)
$ sudo pvmove /dev/sdb                  # Extents auf andere PVs verschieben
$ sudo vgreduce vg_daten /dev/sdb
$ sudo pvremove /dev/sdb
```

> **Achtung**
>
> Verkleinern ist riskanter als Vergrößern: Zuerst das Dateisystem, dann das Volume verkleinern – nie umgekehrt. Das Dateisystem `XFS` lässt sich überhaupt nicht verkleinern. Vorher immer ein Backup anlegen.

28.4. Snapshots und Thin Provisioning

Niveau: Fortgeschritten

Ein **Snapshot** friert den Zustand eines Volumes ein (Copy-on-Write: nur Änderungen belegen Platz). Er eignet sich für konsistente Backups und zum gefahrlosen Testen.

```
$ sudo lvcreate -s -L 5G -n snap_web /dev/vg_daten/lv_web   # Snapshot anlegen
$ sudo mount -o ro /dev/vg_daten/snap_web /mnt/snap          # lesend einhaengen
$ tar czf /backup/web.tar.gz -C /mnt/snap .                  # sichern
$ sudo umount /mnt/snap
$ sudo lvremove /dev/vg_daten/snap_web                       # wieder loeschen
$ sudo lvconvert --merge /dev/vg_daten/snap_web              # Zustand zuruecksetzen
```

Wichtig: Ein klassischer Snapshot wird ungültig, wenn sein reservierter Platz voll läuft (`lvs` zeigt `Data%`). Snapshots deshalb nur kurz halten.
Thin Provisioning legt Volumes an, die mehr Speicher versprechen, als physisch vorhanden ist; Platz wird erst bei Bedarf belegt (ähnlich wie bei virtuellen Maschinen):

```
$ sudo lvcreate -L 100G --thinpool pool0 vg_daten
$ sudo lvcreate -V 50G --thin -n lv_vm1 vg_daten/pool0
$ sudo lvcreate -V 50G --thin -n lv_vm2 vg_daten/pool0     # 100 G versprochen, Pool 100 G
$ sudo lvs -a                                              # Fuellstand des Pools beobachten
```

28.5. RAID in LVM: Spiegelung und Parität

Niveau: Fortgeschritten

LVM kann Redundanz selbst bereitstellen, ohne dass vorher ein `/dev/md0` mit `mdadm` angelegt werden muss (Kapitel 27). Intern nutzt LVM dieselbe Kerneltechnik (`dm-raid`, der RAID-Code von `md`) und dieselben Level: `raid1`, `raid5`, `raid6`, `raid10` sowie `raid0`. Der Unterschied: Die Redundanz gilt pro *Logical Volume*, nicht pro Platte. Ein Volume kann gespiegelt sein, ein anderes derselben Volume Group ungeschützt.
Voraussetzung ist eine VG mit genügend Physical Volumes auf *verschiedenen* Platten: mindestens zwei für `raid1`, drei für `raid5`, vier für `raid6`.

```
$ sudo pvcreate /dev/sdb /dev/sdc /dev/sdd
$ sudo vgcreate vg_raid /dev/sdb /dev/sdc /dev/sdd

# RAID 1: Spiegelung (-m 1 = eine zusaetzliche Kopie)
$ sudo lvcreate --type raid1 -m 1 -L 20G -n lv_spiegel vg_raid

# RAID 5: 2 Datenstreifen + Paritaet (-i = Anzahl Datenstreifen)
$ sudo lvcreate --type raid5 -i 2 -L 20G -n lv_parity vg_raid

# RAID 10 und RAID 6 entsprechend
$ sudo lvcreate --type raid10 -i 2 -m 1 -L 20G -n lv_r10 vg_raid
$ sudo lvcreate --type raid6 -i 3 -L 20G -n lv_r6 vg_raid   # braucht 5 PVs

$ sudo mkfs.ext4 /dev/vg_raid/lv_spiegel
$ sudo mount /dev/vg_raid/lv_spiegel /srv/daten
```

Mit `lvs` sieht man Typ, Synchronisationsstand und die beteiligten Platten. Die internen Teilvolumes (`[lv_spiegel_rimage_0]` für Daten, `[lv_spiegel_rmeta_0]` für Metadaten) zeigt `lvs -a`.

```
$ sudo lvs -a -o name,segtype,sync_percent,devices vg_raid
  LV                   Type   Cpy%Sync Devices
  lv_spiegel           raid1  100.00   lv_spiegel_rimage_0(0),...
  [lv_spiegel_rimage_0] linear          /dev/sdb(1)
  [lv_spiegel_rimage_1] linear          /dev/sdc(1)
```

Die Anfangssynchronisation läuft im Hintergrund; das Volume ist währenddessen schon nutzbar. `Cpy%Sync` zeigt den Fortschritt. Die Kapazitäten entsprechen den RAID-Leveln aus Kapitel 27.

	`mdadm` (Kapitel 27)	**RAID in LVM**
Granularität	ganze Platten oder Partitionen	einzelne Logical Volumes
Flexibilität	Größe des Verbunds fest, Umbau aufwendig	Volumes online vergrößern, verschieben und umwandeln
Schichten	RAID, dann LVM, dann Dateisystem	LVM allein, ein Werkzeug (`lvm`)
Platzbedarf	nutzbar ist der ganze Verbund	Redundanz nur dort, wo sie nötig ist
Boot und Root	gut unterstützt (initramfs, Installer)	`/boot` meist besser außerhalb von LVM-RAID
Verbreitung	universell, auch ohne LVM	nur mit LVM, Erfahrung im Betrieb geringer

Welche Variante?

Soll das gesamte System einheitlich gespiegelt werden und gilt Einfachheit mehr als Flexibilität, ist `mdadm` mit LVM darüber (Kapitel 27) der verbreitete Weg. Sollen nur ausgewählte Volumes redundant sein oder sich der Schutz später ändern lassen, ist RAID in LVM bequemer. Beides ersetzt kein Backup.

28.6. RAID-Volumes umwandeln, reparieren und prüfen

Niveau: Experte

Umwandeln: Ein vorhandenes lineares Volume lässt sich im laufenden Betrieb spiegeln und wieder zurückstellen. Dafür muss in der VG ein weiteres PV mit genug freien Extents vorhanden sein.

```
$ sudo lvconvert --type raid1 -m 1 vg_daten/lv_web     # linear -> gespiegelt
$ sudo lvconvert -m +1 vg_daten/lv_web                 # weitere Kopie
$ sudo lvconvert -m 0 vg_daten/lv_web                  # zurueck zu linear
$ sudo lvconvert --splitmirrors 1 --name lv_kopie vg_raid/lv_spiegel
```

Plattenausfall: Fällt ein PV aus, läuft das RAID-Volume degradiert weiter. `lvs` zeigt in der Spalte `Health` den Wert `partial`, die VG meldet fehlende PVs. Reparatur in drei Schritten:

```
$ sudo lvs -a -o name,segtype,lv_health_status vg_raid   # Zustand: partial
$ sudo pvcreate /dev/sde                                 # Ersatzplatte
$ sudo vgextend vg_raid /dev/sde
$ sudo lvconvert --repair vg_raid/lv_spiegel             # Teile auf neuem PV aufbauen
$ sudo vgreduce --removemissing vg_raid                  # defektes PV entfernen
```

Eine funktionierende Platte lässt sich auch gezielt austauschen: `lvconvert --replace /dev/sdb vg_raid/lv_spiegel /dev/sde` baut die Daten von `sdb` auf `sde` neu auf.

Prüfen (Scrubbing): Wie bei `md` sollte ein RAID-Volume regelmäßig auf Inkonsistenzen geprüft werden.

```
$ sudo lvchange --syncaction check vg_raid/lv_spiegel           # lesen und vergleichen
$ sudo lvs -o name,raid_sync_action,raid_mismatch_count vg_raid
$ sudo lvchange --syncaction repair vg_raid/lv_spiegel          # Unterschiede korrigieren
```

Stellt `raid_mismatch_count` Abweichungen fest, sollte man die Ursache klären (defekte Platte, Kabel, Speicher), bevor man `repair` ausführt. Dienste wie `lvm2-monitor` überwachen Ausfälle und können

mit `raid_fault_policy` in der `/etc/lvm/lvm.conf` festlegen, ob LVM ausgefallene Teile selbstständig ersetzt (`allocate`) oder nur warnt (`warn`). Alle Möglichkeiten beschreibt `man lvmraid`.

28.7. Verschlüsselung mit LUKS

Niveau: Fortgeschritten

LUKS (Linux Unified Key Setup) ist der Standard zur Verschlüsselung von Blockgeräten. Der Kernel-Teil heißt `dm-crypt`, das Werkzeug `cryptsetup`. LUKS verschlüsselt mit einem zufälligen Hauptschlüssel, der seinerseits durch bis zu acht Passwörter oder Schlüsseldateien geschützt wird.

```
$ sudo apt install cryptsetup
$ sudo cryptsetup luksFormat /dev/sdb1          # ACHTUNG: zerstoert vorhandene Daten
$ sudo cryptsetup open /dev/sdb1 geheim         # entsperren -> /dev/mapper/geheim
$ sudo mkfs.ext4 /dev/mapper/geheim
$ sudo mount /dev/mapper/geheim /mnt/geheim
...
$ sudo umount /mnt/geheim
$ sudo cryptsetup close geheim                  # sperren

$ sudo cryptsetup luksDump /dev/sdb1            # Header und Schluesselplaetze
$ sudo cryptsetup luksAddKey /dev/sdb1          # weiteres Passwort
$ sudo cryptsetup luksHeaderBackup /dev/sdb1 --header-backup-file hdr.img
```

Automatisches Entsperren beim Start regelt `/etc/crypttab` zusammen mit `/etc/fstab`:

```
# /etc/crypttab
geheim  UUID=3b5f...  none  luks
# /etc/fstab
/dev/mapper/geheim  /mnt/geheim  ext4  defaults  0  2
```

Bei `none` fragt das System beim Start nach der Passphrase. Für unbeaufsichtigte Server nutzt man Schlüsseldateien, netzwerkgebundene Entsperrung (Clevis und Tang) oder TPM2 mit `systemd-cryptenroll`. Bei Verlust des LUKS-Headers oder aller Passwörter sind die Daten **unwiederbringlich verloren** – Header sichern!

28.8. SMART und Gesundheit der Platten

Niveau: Fortgeschritten

```
$ sudo apt install smartmontools
$ sudo smartctl -H /dev/sda                     # Gesamtbewertung: PASSED / FAILED
$ sudo smartctl -a /dev/sda                     # alle Werte
$ sudo smartctl -t short /dev/sda               # Kurztest starten
$ sudo smartctl -l selftest /dev/sda            # Ergebnis ansehen
$ sudo nvme smart-log /dev/nvme0                # NVMe
```

Kritisch sind steigende Werte bei `Reallocated_Sector_Ct`, `Current_Pending_Sector`, `Offline_Uncorrectable` und `UDMA_CRC_Error_Count`. Der Dienst `smartd` überwacht Platten dauerhaft und kann Warnungen per E-Mail verschicken.

> **Schichten kombinieren**
>
> LUKS lässt sich auf einem RAID-Volume ebenso anlegen wie auf einem normalen Volume oder einem `mdadm`-Gerät (Kapitel 27). Wird erst gespiegelt und dann verschlüsselt, ist nur eine Passphrase nötig und die Spiegelung ist unabhängig vom Schlüssel.

Übungen und Kontrollfragen

1. Sie haben eine VG mit 100 GB, davon sind 80 GB belegt. Wie erweitern Sie `lv_web` um 10 GB samt ext4?

   Lösung: `lvextend -L +10G -r /dev/vg/lv_web` (genug freie Extents vorhanden: 20 GB frei).

2. Wozu dient `pvmove`?

   Lösung: Es verschiebt belegte Extents von einem PV auf andere PVs derselben VG, z. B. um eine Platte im laufenden Betrieb zu ersetzen.

3. Wie legen Sie in der VG `vg_raid` ein gespiegeltes Volume `lv_spiegel` mit 20 GB an?

   Lösung: `lvcreate --type raid1 -m 1 -L 20G -n lv_spiegel vg_raid`; die VG braucht dafür mindestens zwei PVs auf verschiedenen Platten.

4. Ein PV der VG ist ausgefallen, `lvs` zeigt `partial`. Wie stellen Sie die Spiegelung wieder her?

   Lösung: Ersatzplatte mit `pvcreate` vorbereiten, mit `vgextend` der VG hinzufügen, `lvconvert --repair vg_raid/lv_spiegel` ausführen und das defekte PV mit `vgreduce --removemissing` entfernen.

5. Wann ist RAID in LVM, wann `mdadm` mit LVM darüber sinnvoller?

   Lösung: RAID in LVM, wenn nur einzelne Volumes redundant sein oder nachträglich umgewandelt werden sollen; `mdadm` darunter, wenn das gesamte System einheitlich gespiegelt werden soll und Einfachheit sowie gute Boot-Unterstützung wichtiger sind.

6. Welche Daten müssen Sie bei LUKS unbedingt sichern?

   Lösung: Den LUKS-Header (`cryptsetup luksHeaderBackup`) und die Passphrase bzw. Schlüsseldatei an einem sicheren, getrennten Ort.

29. Dateisysteme und Swap

Inhalt: Dateisystemtypen, Anlegen und Prüfen, Einhängen, `/etc/fstab`, Swap, Inodes, Quotas, `dd`.
Stufe: [E] Einsteiger bis [X] Experte
Lernziele: Nach diesem Kapitel können Sie

- passende Dateisysteme zu wählen und anzulegen,
- Dateisysteme dauerhaft über `fstab` einzubinden,
- Swap einzurichten und Platzprobleme zu analysieren.

Vorkenntnisse: Kapitel 6 und 26.

29.1. Linux-Dateisysteme im Überblick

Niveau: Einsteiger

Ein Dateisystem organisiert Dateien, Verzeichnisse und Metadaten auf einem Datenträger. Linux unterstützt sehr viele, in der Praxis begegnen einem diese:

Typ	Eigenschaften
ext4	Standard unter Debian und Ubuntu; ausgereift, Journaling, bis 1 EiB; wachsen online, schrumpfen offline
XFS	Standard unter RHEL; sehr gut bei großen Dateien und paralleler E/A; nur wachsen, nicht schrumpfen
Btrfs	Copy-on-Write, Snapshots, Prüfsummen, Subvolumes, integriertes RAID; Standard bei openSUSE und Fedora Workstation
ZFS	Sehr funktionsreich (Pools, Snapshots, Replikation); wegen der Lizenz nicht im Kernel, aber über Pakete nutzbar
FAT32 / exFAT	Austauschformat für USB-Sticks, EFI-Systempartition
NTFS	Windows-Partitionen; Zugriff über `ntfs-3g` bzw. den Kernel-Treiber
tmpfs	Dateisystem im Arbeitsspeicher (`/run`, `/dev/shm`, oft `/tmp`)
NFS, SMB/CIFS	Netzwerkdateisysteme (siehe Kapitel 49)

Journaling (ext4, XFS) protokolliert Änderungen vor dem Schreiben. Nach einem Absturz genügt es, das Journal abzuarbeiten, statt die ganze Platte zu prüfen.

29.2. Dateisysteme anlegen und prüfen

Niveau: Einsteiger

```
# mkfs.ext4 /dev/sdb1                    # ext4 anlegen (Daten gehen verloren!)
# mkfs.ext4 -L daten /dev/sdb1           # mit Label
# mkfs.xfs /dev/sdb1                     # XFS
# mkfs.btrfs /dev/sdb1                   # Btrfs
# mkfs.vfat -F32 /dev/sdb1               # FAT32
# fsck /dev/sdb1                         # Pruefung; nur bei AUSGEHAENGTEM Dateisystem!
# fsck -f -y /dev/sdb1                   # erzwingen, alle Rueckfragen mit 'ja' beantworten
# xfs_repair /dev/sdb1                   # Reparatur bei XFS
# tune2fs -l /dev/sdb1                   # ext4-Parameter anzeigen
# tune2fs -L neuesLabel /dev/sdb1        # Label aendern
# tune2fs -m 1 /dev/sdb1                 # reservierte Blocks von 5 % auf 1 % reduzieren
```

29.3. Einhängen und Aushängen

Niveau: Einsteiger

```
# mkdir -p /mnt/daten
# mount /dev/sdb1 /mnt/daten                          # einhaengen
# mount -o ro /dev/sdb1 /mnt/daten                    # nur lesend
# mount -t ext4 -o noexec,nosuid /dev/sdb1 /mnt/daten # mit Optionen
# mount -o loop abbild.iso /mnt/iso                   # ISO-Datei
# mount -o remount,rw /                               # laufend aendern
$ findmnt /mnt/daten                                  # Einhaengepunkt pruefen
# umount /mnt/daten                                   # aushaengen
# umount -l /mnt/daten                                # "lazy": sobald nicht mehr benutzt
$ lsof +f -- /mnt/daten                               # wer blockiert das Aushaengen?
$ fuser -vm /mnt/daten                                # dto.
```

Meldet `umount` "target is busy", hält ein Prozess das Dateisystem offen (auch eine Shell, die in diesem Verzeichnis steht).

Mount-Option	Wirkung
`ro`, `rw`	Nur lesen / lesen und schreiben
`noexec`	Keine Programme ausführen (sinnvoll für `/tmp`, Datenpartitionen)
`nosuid`	SUID-/SGID-Bits ignorieren
`nodev`	Keine Gerätedateien interpretieren
`noatime`	Zugriffszeit nicht schreiben (weniger Schreibzugriffe, schneller)
`defaults`	`rw,suid,dev,exec,auto,nouser,async`
`nofail`	Start nicht abbrechen, wenn das Gerät fehlt
`user`, `users`	Normale Benutzer dürfen einhängen

29.4. Dauerhaft einhängen: /etc/fstab

Niveau: Einsteiger

Die Datei `/etc/fstab` legt fest, was beim Start eingehängt wird. Jede Zeile hat sechs Felder: Gerät, Einhängepunkt, Typ, Optionen, `dump` (fast immer 0) und `fsck`-Reihenfolge (`1` für `/`, `2` für andere, `0` = nicht prüfen).

```
# <Geraet>                                  <Mount>      <Typ>  <Optionen>         <dump> <fsck
    >
UUID=2f4e1c3a-7a5b-4c10-8d4e-0123456789ab  /            ext4   defaults           0      1
UUID=A1B2-C3D4                             /boot/efi    vfat   umask=0077         0      1
UUID=9d8c7b6a-1111-2222-3333-444455556666  /srv/daten   xfs    defaults,noatime   0      2
/dev/vg_daten/lv_web                       /srv/web     ext4   defaults           0      2
192.0.2.20:/export/home                    /mnt/nfs     nfs    defaults,_netdev   0      0
tmpfs                                      /tmp         tmpfs  defaults,size=2G   0      0
```

```
# blkid /dev/sdb1                 # UUID ermitteln
# mount -a                        # fstab testen: alle Eintraege einhaengen
# findmnt --verify                # Syntax der fstab pruefen
# systemctl daemon-reload         # systemd ueber fstab-Aenderungen informieren
```

> **Achtung**
>
> Ein fehlerhafter Eintrag in `/etc/fstab` kann den Start in den *Emergency Mode* zwingen. **Immer** mit `mount -a` testen, bevor man neu startet, und für nicht-kritische Datenträger `nofail` verwenden.

29.5. Swap

Niveau: Einsteiger

Swap ist ausgelagerter Arbeitsspeicher auf der Platte. Er fängt Speicherspitzen ab und erlaubt es dem Kernel, selten benutzte Seiten zu verdrängen. Swap ist kein Ersatz für RAM: Aktives Swapping macht ein System sehr langsam.

```
$ swapon --show                       # aktive Swap-Bereiche
$ free -h

# --- Swap-Datei anlegen ---
# fallocate -l 2G /swapfile           # (bei Btrfs: siehe 'btrfs filesystem mkswapfile')
# chmod 600 /swapfile                 # nur root!
# mkswap /swapfile
# swapon /swapfile
# echo '/swapfile none swap sw 0 0' >> /etc/fstab     # dauerhaft

# --- Swap-Partition ---
# mkswap /dev/sdb2 && swapon /dev/sdb2
# swapoff -a                          # allen Swap abschalten
```

Arbeitsspeicher	Richtwert für Swap
bis 2 GB	2 × RAM
2–8 GB	ca. gleich RAM
8–64 GB	4–8 GB (mit Ruhezustand: mindestens RAM-Größe)
über 64 GB	Nach Anwendung (Datenbanken: Herstellerangaben)

Die Neigung zum Auslagern steuert `vm.swappiness` (0–100; Standard 60); für Datenbankserver setzt man oft 10. `zram` legt komprimierten Swap im RAM an und ist auf Desktops mit wenig Speicher sinnvoll.

29.6. Belegung, Inodes und Quotas

Niveau: Fortgeschritten

```
$ df -hT                        # Belegung inkl. Dateisystemtyp
$ df -i                         # Inodes: Datei-Anzahl kann ausgehen, obwohl Platz frei ist
$ du -sh /var/*                 # Platzfresser finden
$ du -h --max-depth=1 / 2>/dev/null | sort -h | tail
$ ncdu /var                     # interaktiv (Paket ncdu)
$ stat -f /                     # Dateisystemdaten
```

Jede Datei benötigt einen **Inode** (Metadatenstruktur). Beim Anlegen eines ext4-Dateisystems wird deren Anzahl festgelegt. Melden Programme “No space left on device”, obwohl `df -h` Platz zeigt, prüft man `df -i`.
Quotas begrenzen den Speicherplatz je Benutzer oder Gruppe: Mount-Optionen `usrquota,grpquota`, dann `quotacheck`, `quotaon`, `edquota -u anna`.

29.7. dd, Abbilder und Datenrettung

Niveau: Experte

`dd` kopiert Daten blockweise und ist das Werkzeug für Abbilder:

```
# dd if=/dev/sda of=/mnt/backup/sda.img bs=4M status=progress     # ganze Platte sichern
# dd if=ubuntu.iso of=/dev/sdb bs=4M status=progress conv=fsync   # Installationsstick
    schreiben
# dd if=/dev/zero of=/dev/sdb bs=1M count=10                      # Anfang loeschen (
    Signaturen)
# dd if=/dev/urandom of=/dev/sdb bs=4M                            # Platte ueberschreiben
$ dd if=/dev/zero of=testdatei bs=1M count=1024 oflag=direct      # grober Schreibtest
```

> **Achtung**
>
> Bei `dd` sind `if=` (Eingabe) und `of=` (Ausgabe) leicht zu verwechseln; ein vertauschtes Paar überschreibt die Quelle. Den Spitznamen "*disk destroyer*" trägt das Programm zu Recht. Vor dem Schreiben das Zielgerät mit `lsblk` prüfen.

Bei defekten Platten ist `ddrescue` (Paket `gddrescue`) besser, weil es Lesefehler überspringt und später erneut versucht. Gelöschte Dateien lassen sich mit `testdisk` und `photorec` mitunter wiederherstellen, sofern das Gerät danach nicht weiter beschrieben wurde.

30. Moderne Dateisysteme: Btrfs und ZFS

Inhalt: Copy-on-Write-Dateisysteme, Btrfs mit Subvolumes, Snapshots und RAID, Kompression und Prüfsummen; Grundlagen von ZFS: Pools, Datasets, Snapshots, Send/Receive; XFS-Besonderheiten.

Stufe: [F] Fortgeschritten bis [X] Experte

Lernziele: Nach diesem Kapitel können Sie

- die Vorteile von Copy-on-Write-Dateisystemen zu benennen,
- Subvolumes, Snapshots und Kompression bei Btrfs einzusetzen,
- ZFS-Pools und -Datasets anzulegen und Snapshots zu übertragen.

Vorkenntnisse: Kapitel 26 und 29 (Platten, Dateisysteme) sowie Kapitel 28.

30.1. Warum neue Dateisysteme?

Niveau: Fortgeschritten

Klassische Dateisysteme wie ext4 überschreiben Daten an Ort und Stelle. Bei einem Stromausfall mitten im Schreibvorgang kann das Ergebnis inkonsistent sein (Journaling mildert dies für Metadaten). **Copy-on-Write** (CoW) schreibt Änderungen stets in neue Blöcke und stellt erst danach den Verweis um – der alte Zustand bleibt gültig, bis der neue vollständig ist. Daraus ergeben sich:

- **Prüfsummen** über Daten und Metadaten: stille Datenfehler (*bit rot*) werden erkannt, bei Redundanz sogar repariert.
- **Snapshots** in Sekundenbruchteilen, ohne Platzbedarf, bis sich Daten ändern.
- **Integrierte Volumenverwaltung**: Mehrere Datenträger, RAID, Größenänderung ohne LVM.
- **Kompression**, Deduplizierung (eingeschränkt), Send/Receive für effiziente Backups.

Merkmal	**ext4**	**XFS**	**Btrfs / ZFS**
Prinzip	Journal	Journal	Copy-on-Write
Prüfsummen für Daten	nein	nein	ja
Snapshots	nein (LVM)	nein (LVM)	ja, integriert
Verkleinern	ja	**nein**	ja / eingeschränkt
Stärken	Robust, einfach, überall	Große Dateien, Parallelität	Datenintegrität, Snapshots
Standard bei	Debian, Ubuntu	RHEL	openSUSE, Fedora (Btrfs)

30.2. Btrfs

Niveau: Fortgeschritten

```
$ sudo mkfs.btrfs -L daten /dev/sdb              # Dateisystem anlegen
$ sudo mkfs.btrfs -L raid1 -d raid1 -m raid1 /dev/sdb /dev/sdc   # gespiegelt
$ sudo mount /dev/sdb /mnt/daten
$ sudo btrfs filesystem show
$ sudo btrfs filesystem df /mnt/daten            # Belegung nach Typ
$ sudo btrfs filesystem usage /mnt/daten
$ sudo btrfs device add /dev/sdd /mnt/daten      # Platte hinzufuegen
$ sudo btrfs balance start -dusage=50 /mnt/daten # Daten neu verteilen
$ sudo btrfs scrub start /mnt/daten              # Pruefsummen pruefen und reparieren
$ sudo btrfs scrub status /mnt/daten
```

Subvolumes sind eigenständige Verzeichnisbäume innerhalb eines Btrfs, die sich wie eigene Dateisysteme einhängen und snapshotten lassen. Verbreitet ist die Aufteilung in `@` (Root), `@home`, `@snapshots` usw.

```
$ sudo btrfs subvolume create /mnt/daten/projekte
$ sudo btrfs subvolume list /mnt/daten
$ sudo btrfs subvolume snapshot /mnt/daten/projekte /mnt/daten/snap-2025-01-15       #
    schreibbar
$ sudo btrfs subvolume snapshot -r /mnt/daten/projekte /mnt/daten/snap-ro            # nur
    lesbar
$ sudo btrfs subvolume delete /mnt/daten/snap-2025-01-15

# Subvolumes einhaengen (fstab)
UUID=...  /home  btrfs  subvol=@home,compress=zstd:1,noatime  0 0

# Dateien kopieren ohne Platzbedarf (Reflink)
$ cp --reflink=always grosse.img kopie.img
```

Mount-Option	Wirkung
`compress=zstd:3`	Transparente Kompression (spart Platz, oft auch schneller)
`noatime`	Keine Zugriffszeit-Aktualisierung (weniger Schreiblast)
`subvol=@home`	Bestimmtes Subvolume einhängen
`discard=async`	TRIM für SSDs

Werkzeuge wie **Snapper** und **Timeshift** legen zeitgesteuert Snapshots an und erlauben das Zurückrollen des Systems nach fehlerhaften Updates (`snapper rollback`); openSUSE integriert dies mit GRUB-Einträgen für frühere Systemzustände. Backups auf ein zweites Btrfs gelingen effizient mit `btrfs send/receive`:

```
$ sudo btrfs subvolume snapshot -r /mnt/daten/projekte /mnt/daten/s1
$ sudo btrfs send /mnt/daten/s1 | sudo btrfs receive /mnt/backup/
# Inkrementell: nur Unterschiede zum Vorgaenger
$ sudo btrfs send -p /mnt/daten/s1 /mnt/daten/s2 | ssh backup sudo btrfs receive /mnt/backup/
```

> **Hinweis**
>
> Btrfs-RAID 5/6 gilt weiterhin als eingeschränkt zuverlässig; für Parität besser `mdadm` oder ZFS verwenden. RAID 1/10 sind ausgereift. Und: Ein Snapshot ist kein Backup, da er auf demselben Datenträger liegt.

30.3. ZFS

Niveau: Experte

ZFS (ursprünglich von Sun Microsystems) verbindet Dateisystem und Volumenverwaltung. Unter Linux wird es als Modul **OpenZFS** nachgeladen (Ubuntu: `zfsutils-linux`; wegen der Lizenz nicht im Kernel). Bausteine:

- **Pool** (`zpool`): Zusammenfassung von Platten zu *vdevs* (einzeln, Mirror, `raidz1`/`raidz2`/`raidz3`).
- **Dataset**: Dateisystem innerhalb des Pools mit eigenen Eigenschaften (Kompression, Quota, Mountpunkt); beliebig viele, hierarchisch.
- **Zvol**: Blockgerät aus dem Pool (z. B. für VMs).
- **ARC**: Lesecache im Arbeitsspeicher – ZFS nutzt freien RAM intensiv.

```
$ sudo apt install zfsutils-linux
$ sudo zpool create tank mirror /dev/sdb /dev/sdc          # gespiegelter Pool
$ sudo zpool create tank2 raidz1 /dev/sdd /dev/sde /dev/sdf    # RAID5-aehnlich
$ zpool status                                              # Zustand
$ zpool list
$ sudo zfs create tank/home
$ sudo zfs create tank/home/anna
$ sudo zfs set compression=lz4 tank
$ sudo zfs set quota=50G tank/home/anna
$ sudo zfs set mountpoint=/srv/daten tank/daten
$ zfs list
$ zfs get compressratio tank
```

```
# Snapshots, Rollback, Klone
$ sudo zfs snapshot tank/home@vor-update
$ zfs list -t snapshot
$ sudo zfs rollback tank/home@vor-update
$ ls /srv/daten/.zfs/snapshot/                      # Dateien aus Snapshots lesen
$ sudo zfs clone tank/home@vor-update tank/test

# Replikation auf einen anderen Rechner
$ sudo zfs send tank/home@s1 | ssh backup sudo zfs receive backuppool/home
$ sudo zfs send -i s1 tank/home@s2 | ssh backup sudo zfs receive backuppool/home   #
    inkrementell

# Pflege
$ sudo zpool scrub tank                             # Daten pruefen/reparieren (regelmaessig!)
$ sudo zpool replace tank /dev/sdc /dev/sdg         # defekte Platte ersetzen
$ sudo zpool import; sudo zpool export tank         # Pool auf anderem Rechner nutzen
```

> **Hinweis**
>
> ZFS benötigt ausreichend Arbeitsspeicher (Faustregel: mehrere GiB, mehr bei Deduplizierung – diese nur in Ausnahmefällen aktivieren). ZFS-Pools sollten *ganze Platten* bekommen, kein Hardware-RAID dazwischen, damit ZFS Fehler selbst erkennen kann.

30.4. XFS und ext4: Pflege im Alltag

Niveau: Fortgeschritten

```
# ext4
$ sudo tune2fs -l /dev/sda1 | head          # Eigenschaften
$ sudo tune2fs -c 0 -i 0 /dev/sda1          # periodische Pruefung abschalten
$ sudo resize2fs /dev/vg0/lv_home           # nach lvextend
$ sudo e2fsck -f /dev/sda1                  # Pruefen (nur ausgehaengt)

# XFS
$ sudo xfs_info /srv/daten
$ sudo xfs_growfs /srv/daten                # vergroessern (eingehaengt!)
$ sudo xfs_repair /dev/sdb1                 # reparieren (ausgehaengt)
$ sudo xfs_admin -L daten /dev/sdb1         # Label setzen
$ sudo xfs_quota -x -c 'report' /srv/daten
```

Übungen und Kontrollfragen

1. Was bedeutet Copy-on-Write, und welcher Vorteil ergibt sich daraus?

   Lösung: Änderungen werden in neue Blöcke geschrieben und erst danach referenziert. Das Dateisystem bleibt jederzeit konsistent, und Snapshots sind praktisch kostenlos.

2. Wie legen Sie einen schreibgeschützten Btrfs-Snapshot von `/mnt/daten/projekte` an?

   Lösung: `btrfs subvolume snapshot -r /mnt/daten/projekte /mnt/daten/snap`.

3. Wozu dient `zpool scrub`?

   Lösung: Es liest alle Daten, prüft sie anhand der Prüfsummen und repariert Fehler aus Redundanz (Mirror/RAIDZ). Regelmäßig (z. B. monatlich) ausführen.

4. Warum ist ein Snapshot kein Backup?

   Lösung: Er liegt auf demselben Datenträger; Hardwaredefekt, Pool-Verlust oder Löschen des Datasets vernichten Daten und Snapshots gleichzeitig. Snapshots per `send`/`receive` auf ein anderes System übertragen.

5. Welches Dateisystem lässt sich nicht verkleinern?

   Lösung: XFS.

31. Der Bootvorgang

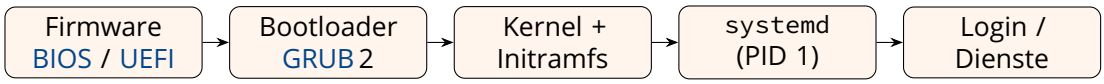
Inhalt: Ablauf vom Einschalten bis zur Anmeldung, UEFI und BIOS, GRUB 2, Kernelparameter, Rettungsmodus, Herunterfahren.
Stufe: [E] Einsteiger bis [X] Experte
Lernziele: Nach diesem Kapitel können Sie
- die Stufen des Systemstarts zu benennen,
- GRUB zu konfigurieren und Kernelparameter zu nutzen,
- ein nicht startendes System zu retten.

Vorkenntnisse: Kapitel 26 und 29.

31.1. Vom Einschalten zur Anmeldung

Niveau: Einsteiger

Der Systemstart durchläuft mehrere Stufen, bei denen jeweils die nächste geladen wird:



1. **Firmware**: Beim Einschalten führt das UEFI (früher BIOS) einen Selbsttest durch (POST), initialisiert die Hardware und sucht ein bootfähiges Medium. Bei UEFI lädt es den Bootloader aus der ESP; bei BIOS liest es den Bootsektor (MBR).
2. **Bootloader**: GRUB 2 zeigt ein Menü, lädt Kernel und *Initramfs* in den Speicher und übergibt Kernelparameter.
3. **Kernel**: Initialisiert Hardware, hängt das Initramfs als vorläufiges Root-Dateisystem ein.
4. **Initramfs**: Enthält Treiber und Skripte, um das eigentliche Root-Dateisystem zu finden (RAID, LVM, Verschlüsselung, Netzwerk) und einzuhängen.
5. **init (`systemd`)**: Der erste Prozess (PID 1) startet alle Dienste bis zum Zielzustand (Kapitel 32).

31.2. Informationen zum Start

Niveau: Einsteiger

```
$ cat /proc/cmdline             # Kernelparameter dieses Starts
BOOT_IMAGE=/vmlinuz-6.8.0-45-generic root=UUID=2f4e... ro quiet splash
$ ls /boot                      # Kernel, Initramfs, GRUB
$ ls /sys/firmware/efi          # existiert nur bei UEFI-Start
$ systemd-analyze               # Dauer des Starts
Startup finished in 3.8s (kernel) + 9.2s (userspace) = 13.0s
$ systemd-analyze blame | head  # langsamste Dienste
$ dmesg | less                  # Kernelmeldungen (Treiber, Hardware)
$ journalctl -b                 # komplette Meldungen dieses Starts
$ journalctl -b -1              # Meldungen des vorherigen Starts
```

31.3. GRUB 2

Niveau: Fortgeschritten

Der Bootloader GRUB 2 wird nicht direkt bearbeitet: Die Datei `/boot/grub/grub.cfg` (bzw. `/boot/grub2/grub.cfg`) wird **generiert**. Einstellungen stehen in `/etc/default/grub` und `/etc/grub.d/`.

```
# nano /etc/default/grub
GRUB_DEFAULT=0                          # Standardeintrag (saved = zuletzt gewaehlter)
GRUB_TIMEOUT=5                          # Menue 5 s anzeigen
GRUB_CMDLINE_LINUX_DEFAULT="quiet"      # zusaetzliche Kernelparameter

# update-grub                           # Debian/Ubuntu: grub.cfg neu erzeugen
# grub2-mkconfig -o /boot/grub2/grub.cfg    # RHEL/openSUSE (BIOS)
# grub2-mkconfig -o /boot/grub2/grub.cfg    # bei UEFI: Pfad unter /boot/efi/EFI/<distro>/
# grub-install /dev/sda                 # GRUB in den MBR (BIOS) installieren
```

Im GRUB-Menü bearbeitet `e` den Eintrag *einmalig* (z. B. zusätzliche Kernelparameter), `Strg+X` startet damit, `c` öffnet die GRUB-Kommandozeile.

Kernelparameter	Wirkung
`quiet`, `splash`	Weniger Meldungen / grafischer Startbildschirm
`nomodeset`	Grafiktreiber nicht laden (hilft bei Grafikproblemen)
`systemd.unit=rescue.target`	In den Rettungsmodus (Einzelbenutzer) starten
`systemd.unit=multi-user.target`	Ohne grafische Oberfläche starten
`single`, `1`	Alte Schreibweise für Einzelbenutzermodus
`init=/bin/bash`	Statt `systemd` eine Shell starten (Notfall)
`root=UUID=...`, `ro`	Root-Dateisystem und schreibgeschützt einhängen
`rd.break`	Anhalten im Initramfs (RHEL)
`fsck.mode=force`	Dateisystemprüfung erzwingen

31.4. Probleme beim Start beheben

Niveau: Fortgeschritten

Symptom	Vorgehen
Kein Bootmenü, `grub rescue>`	Bootloader beschädigt: Live-System booten, `chroot`, `grub-install` und `update-grub`
"Kernel panic" nach Update	Im GRUB-Menü unter "Advanced options" einen älteren Kernel wählen
Start bleibt in *Emergency Mode*	Meist fehlerhafte `/etc/fstab`; `journalctl -xb` lesen, Eintrag korrigieren, `systemctl default`
Grafik bleibt schwarz	Parameter `nomodeset`, Konsole mit `Strg+Alt+F3`, Treiber prüfen
Vergessenes `root`-Passwort	Siehe unten
Dateisystemfehler	Im Rettungsmodus `fsck` auf dem *ausgehängten* Dateisystem ausführen

Rettung mit einem Live-System

```
# (Live-USB starten, Terminal oeffnen)
# mount /dev/sda2 /mnt                      # Root-Dateisystem einhaengen
# mount /dev/sda1 /mnt/boot/efi             # ESP einhaengen (UEFI)
# for d in dev proc sys run; do mount --rbind /$d /mnt/$d; done
# chroot /mnt                               # in das installierte System wechseln
# update-grub && grub-install /dev/sda      # Bootloader reparieren
# exit; reboot
```

31.4.1. Passwort von root zurücksetzen [X]

Wer physischen Zugriff auf die Konsole hat, kann das Passwort zurücksetzen. Das ist ein Grund, Server physisch zu schützen und den Bootloader sowie die Platte zu verschlüsseln.

- Im GRUB-Menü `e` drücken und an die Zeile `linux` den Parameter `init=/bin/bash` (Debian/Ubuntu) bzw. `rd.break` (RHEL) anhängen, mit `Strg+X` starten.
- Root-Dateisystem schreibbar einhängen: `mount -o remount,rw /`. Bei RHEL stattdessen `mount -o remount,rw /sysroot` und `chroot /sysroot`.
- `passwd` ausführen. Bei RHEL zusätzlich `touch /.autorelabel`, damit SELinux die Kontexte erneuert.
- Neu starten: `exec /sbin/init` bzw. `reboot -f`.

31.5. Herunterfahren und Neustart

Niveau: Einsteiger

```
# systemctl poweroff              # herunterfahren (auch: poweroff, shutdown -h now)
# systemctl reboot                # neu starten (reboot)
# shutdown -h +10 "Wartung"       # in 10 Minuten herunterfahren, Meldung an Benutzer
# shutdown -r 22:00               # um 22:00 Uhr neu starten
# shutdown -c                     # geplantes Herunterfahren abbrechen
# systemctl suspend               # Bereitschaft; hibernate = Ruhezustand
$ last -x | head                  # Neustarts und Herunterfahren protokolliert
```

Dabei beendet `systemd` Dienste geordnet (`SIGTERM`, nach Wartezeit `SIGKILL`), hängt Dateisysteme aus und schaltet ab. Ein hartes Ausschalten (Stromkabel) riskiert beschädigte Dateisysteme und Datenverlust.

> **Tipp**
>
> `Strg+Alt+Entf` löst auf Servern meist einen Neustart aus. Bei einem komplett eingefrorenen System hilft die *Magic SysRq*-Folge `Alt+SysRq+R E I S U B` (Tastatur übernehmen, beenden, töten, synchronisieren, ausgehängt, Neustart), sofern `kernel.sysrq` aktiviert ist.

32. systemd

Inhalt: Units und Targets, `systemctl`, eigene Service-Units, Abhängigkeiten, Absicherung und Ressourcenbegrenzung.
Stufe: [E] Einsteiger bis [X] Experte
Lernziele: Nach diesem Kapitel können Sie
- Dienste zu starten, zu stoppen und dauerhaft zu aktivieren,
- eigene Service-Units zu schreiben,
- Dienste mit systemd abzusichern und zu begrenzen.

Vorkenntnisse: Kapitel 18 und 24.

32.1. Was ist systemd?

Niveau: Einsteiger

`systemd` ist der Init-Prozess (PID 1) und Dienstverwalter fast aller großen Distributionen. Er startet Dienste parallel, überwacht sie, protokolliert ihre Ausgaben (`journald`) und bietet einheitliche Verwaltung über das Kommando `systemctl`.
Historisch gab es davor *SysV-Init* (Skripte in `/etc/init.d`, Runlevel 0–6, sequenzieller Start) und *Upstart*. Die Runlevel wurden durch *Targets* ersetzt:

Runlevel	Target	Bedeutung
0	`poweroff.target`	Herunterfahren
1	`rescue.target`	Einzelbenutzermodus (Wartung)
3	`multi-user.target`	Mehrbenutzer ohne grafische Oberfläche
5	`graphical.target`	Mehrbenutzer mit grafischer Oberfläche
6	`reboot.target`	Neustart
–	`emergency.target`	Minimale Notfall-Shell

32.2. Dienste mit systemctl steuern

Niveau: Einsteiger

```
$ systemctl status nginx              # Zustand, PID, letzte Logzeilen
# systemctl start nginx               # jetzt starten
# systemctl stop nginx                # jetzt stoppen
# systemctl restart nginx             # stoppen und starten
# systemctl reload nginx              # Konfiguration neu laden, ohne Neustart
# systemctl enable nginx              # beim Systemstart starten
# systemctl disable nginx             # nicht mehr beim Start
# systemctl enable --now nginx        # aktivieren und sofort starten
# systemctl mask nginx                # komplett sperren (auch manuell nicht startbar)
# systemctl unmask nginx
$ systemctl is-active nginx           # active / inactive / failed
$ systemctl is-enabled nginx
```

```
$ systemctl list-units --type=service              # laufende Dienste
$ systemctl list-unit-files --type=service         # alle installierten Dienste + Autostart
$ systemctl --failed                               # fehlgeschlagene Units
$ systemctl list-dependencies multi-user.target    # Abhaengigkeitsbaum
$ systemctl get-default                            # Standard-Target
# systemctl set-default multi-user.target          # Standard-Target aendern
# systemctl isolate rescue.target                  # jetzt in anderes Target wechseln
```

Hinweis

Der Unterschied zwischen `start` und `enable`: `start` wirkt nur *jetzt*, `enable` nur beim *nächsten Start*. Für beides gleichzeitig dient `enable --now`. Unter Debian/Ubuntu starten neu installierte Dienste meist automatisch, unter RHEL nicht.

32.3. Unit-Typen und Dateien

Niveau: Fortgeschritten

Alles, was `systemd` verwaltet, heißt **Unit**. Der Typ ergibt sich aus der Endung:

Endung	Aufgabe
`.service`	Dienst/Daemon oder einmaliges Programm
`.socket`	Lauscht an Socket/Port und startet den Dienst bei Bedarf (*Socket Activation*)
`.target`	Gruppierung von Units (Startziel)
`.timer`	Zeitgesteuerter Start einer Unit (Ersatz für `cron`)
`.mount`, `.automount`	Einhängepunkte (aus `/etc/fstab` generiert)
`.path`	Startet bei Änderung an Dateien/Verzeichnissen
`.slice`, `.scope`	Ressourcengruppen (cgroups)

Verzeichnis	Inhalt
`/usr/lib/systemd/system/`	Units der Pakete (nicht ändern!)
`/etc/systemd/system/`	Eigene Units und Änderungen (haben Vorrang)
`/run/systemd/system/`	Zur Laufzeit erzeugte Units

```
$ systemctl cat sshd                    # die tatsaechlich wirksame Unit-Datei
# systemctl edit sshd                   # Drop-in anlegen (/etc/systemd/system/sshd.service.d/
    override.conf)
# systemctl edit --full sshd            # komplette Kopie bearbeiten
# systemctl daemon-reload               # nach manuellen Aenderungen an Unit-Dateien
```

32.4. Eigene Service-Unit

Niveau: Fortgeschritten

Beispiel: Ein selbst geschriebenes Programm soll als Dienst laufen.

```
# nano /etc/systemd/system/meinapp.service
[Unit]
Description=Meine Anwendung
After=network-online.target
Wants=network-online.target

[Service]
Type=simple
User=meinapp
WorkingDirectory=/opt/meinapp
ExecStart=/opt/meinapp/bin/server --port 8080
Restart=on-failure
RestartSec=5
Environment=LOGLEVEL=info

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

```
# useradd -r -s /usr/sbin/nologin meinapp      # Systembenutzer ohne Login
# systemctl daemon-reload
# systemctl enable --now meinapp
$ systemctl status meinapp
$ journalctl -u meinapp -f                     # Logausgabe live verfolgen
```

Direktive	Bedeutung
`After=`, `Before=`	Nur *Reihenfolge*, keine Abhängigkeit
`Wants=`	Schwache Abhängigkeit: startet mit, Fehlschlag egal
`Requires=`	Harte Abhängigkeit: Fehlschlag führt zum Abbruch
`Type=simple`	Prozess in `ExecStart` *ist* der Dienst (Standard)
`Type=forking`	Programm forkt in den Hintergrund (alte Daemons, `PIDFile=`)
`Type=oneshot`	Läuft einmal und endet (Skripte); mit `RemainAfterExit=yes`
`Type=notify`	Dienst meldet Bereitschaft selbst an `systemd`
`Restart=`	`no`, `on-failure`, `always`; mit `RestartSec=`
`User=`, `Group=`	Dienst unter anderem Benutzer ausführen
`WantedBy=`	Bei `enable` wird in dieses Target gehängt

32.5. Absicherung und Ressourcenbegrenzung

Niveau: Experte

`systemd` bietet viele Sandbox-Optionen, die Dienste stark einschränken, ohne dass der Dienst selbst angepasst werden muss:

```
[Service]
NoNewPrivileges=yes          # kein Rechtegewinn (z.B. durch SUID)
ProtectSystem=strict         # / ist schreibgeschuetzt
ProtectHome=yes              # /home unsichtbar
PrivateTmp=yes               # eigenes /tmp
PrivateDevices=yes           # keine echten Geraete
ReadWritePaths=/var/lib/meinapp
MemoryMax=512M               # Speichergrenze (cgroup)
CPUQuota=50%                 # max. halber CPU-Kern
TasksMax=200
```

```
$ systemd-analyze security meinapp     # Bewertung der Absicherung (Exposure-Score)
$ systemd-cgls                         # cgroup-Baum
$ systemd-cgtop                        # Ressourcenverbrauch je cgroup (wie top)
$ systemctl show meinapp -p MainPID,MemoryCurrent
```

32.6. Weitere Werkzeuge

Niveau: Fortgeschritten

```
$ hostnamectl                          # Hostname, OS, Kernel, virtuelle Umgebung
# hostnamectl set-hostname server01
$ timedatectl                          # Zeit, Zeitzone, NTP-Status
# timedatectl set-timezone Europe/Berlin
# timedatectl set-ntp true
$ localectl                            # Sprache und Tastaturlayout
# localectl set-keymap de
$ loginctl list-sessions               # angemeldete Sitzungen
$ systemd-analyze critical-chain       # was verzoegert den Start?
```

33. Container-Grundlagen: Namespaces und cgroups

Inhalt: Isolation von Prozessen mit Namespaces, Ressourcenkontrolle mit cgroups, `chroot`, `systemd-nspawn`, LXC/LXD, Capabilities und `seccomp`; Unterschied zwischen Container und virtueller Maschine.

Stufe: [X] Experte

Lernziele: Nach diesem Kapitel können Sie

- zu erklären, was ein Container technisch ist und wie er sich von einer VM unterscheidet,
- Namespaces und cgroups von Hand zu untersuchen und einzusetzen,
- die Sicherheitsmechanismen für Container einzuordnen.

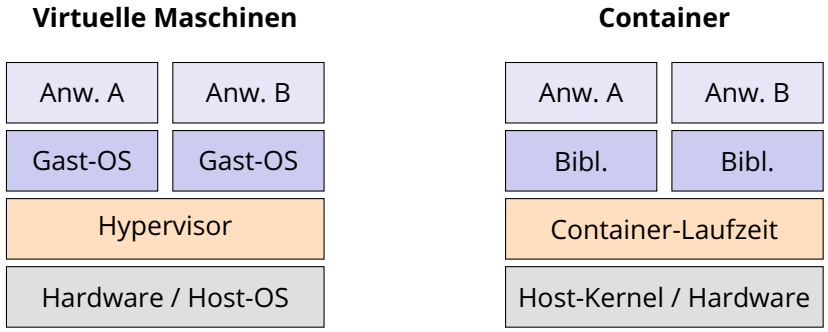
Vorkenntnisse: Kapitel 24 (Prozesse), Kapitel 32 (systemd) und Kapitel 25.

33.1. Container sind Prozesse

Niveau: Experte

Ein **Container** ist kein eigenes Betriebssystem, sondern eine *isolierte Gruppe von Prozessen* auf dem gemeinsamen Kernel des Hosts. Die Isolation entsteht durch Funktionen des Linux-Kernels:

- **Namespaces** begrenzen, was ein Prozess *sieht* (Prozesse, Netzwerk, Dateisysteme, Benutzer …).
- **Control Groups (cgroups)** begrenzen, was ein Prozess *verbrauchen* darf (CPU, Speicher, E/A).
- **Capabilities, seccomp, SELinux/AppArmor** begrenzen, was ein Prozess *tun* darf.
- Ein **Image** liefert das Dateisystem (Root-Dateisystem aus Schichten).



Merkmal	Virtuelle Maschine	Container
Kernel	Eigener Kernel je VM	Kernel des Hosts wird geteilt
Startzeit	Sekunden bis Minuten	Millisekunden bis Sekunden
Overhead	Höher (komplettes Betriebssystem)	Gering
Isolation	Stark (Hardware-Virtualisierung)	Schwächer (gemeinsamer Kernel)
Betriebssystem	Beliebig (Windows, BSD …)	Linux-Userland auf Linux-Kernel
Typischer Einsatz	Server-Konsolidierung, unterschiedliche OS	Anwendungen, Microservices, CI/CD

33.2. Namespaces

Niveau: Experte

Jeder Prozess gehört zu je einem Namespace jeder Art. Sie lassen sich in `/proc/<PID>/ns/` ansehen.

Namespace	Isoliert	Wirkung im Container
`pid`	Prozess-IDs	Erster Prozess hat PID 1, Host-Prozesse unsichtbar
`net`	Netzwerk: Schnittstellen, Routen, Ports	Eigene IP-Adresse, eigene Firewall
`mnt`	Einhängepunkte	Eigenes Wurzeldateisystem
`uts`	Hostname, Domainname	Eigener Hostname
`ipc`	Shared Memory, Semaphoren	Keine Kommunikation mit Host-Prozessen
`user`	UIDs/GIDs	`root` im Container ist Normalbenutzer auf dem Host
`cgroup`	Sicht auf die cgroup-Hierarchie	Eigener cgroup-Baum
`time`	Systemuhr-Offsets	Eigene Zeit (selten)

```
$ ls -l /proc/$$/ns                      # Namespaces der eigenen Shell
$ lsns                                   # alle Namespaces des Systems

# Neuen PID- und Mount-Namespace starten
$ sudo unshare --pid --fork --mount-proc bash
# ps aux                                 # nur wenige Prozesse, bash hat PID 1
# exit

# Eigener Netzwerk-Namespace
$ sudo unshare --net bash
# ip link                                # nur "lo", noch nicht aktiv
# exit

# Benannter Netzwerk-Namespace mit virtuellem Kabel (veth)
$ sudo ip netns add test
$ sudo ip link add veth0 type veth peer name veth1
$ sudo ip link set veth1 netns test
$ sudo ip addr add 10.0.0.1/24 dev veth0 && sudo ip link set veth0 up
$ sudo ip netns exec test ip addr add 10.0.0.2/24 dev veth1
$ sudo ip netns exec test ip link set veth1 up
$ sudo ip netns exec test ping -c1 10.0.0.1

# In den Namespace eines laufenden Prozesses wechseln
$ sudo nsenter --target 1234 --net --pid ip addr
```

33.3. cgroups: Ressourcen begrenzen

Niveau: Experte

Control Groups ordnen Prozesse in eine Hierarchie und verwalten Ressourcen. Moderne Systeme verwenden **cgroups v2** (eine einheitliche Hierarchie unter `/sys/fs/cgroup`); systemd legt für jeden Dienst eine eigene cgroup an.

Controller	Aufgabe
`cpu`	Rechenzeit: Gewichtung (`cpu.weight`) und Obergrenze (`cpu.max`)
`memory`	Arbeitsspeicher: `memory.max`, `memory.high`; bei Überschreitung greift der OOM-Killer
`io`	Ein-/Ausgabe: Bandbreite und IOPS pro Gerät
`pids`	Maximale Anzahl Prozesse (Schutz vor Fork-Bomben)
`cpuset`	Bestimmte CPU-Kerne und NUMA-Knoten

```
$ stat -fc %T /sys/fs/cgroup                # "cgroup2fs" = v2
$ cat /proc/$$/cgroup                       # cgroup der eigenen Shell
$ systemd-cgls                              # Baum aller cgroups
$ systemd-cgtop                             # Ressourcenverbrauch je cgroup (wie top)

# Direkt mit systemd-run begrenzen
$ systemd-run --scope -p MemoryMax=200M -p CPUQuota=50% stress --vm 1 --vm-bytes 500M
$ systemd-run --user -p MemoryMax=1G -p TasksMax=100 ./build.sh

# Dauerhaft fuer einen Dienst
$ sudo systemctl set-property nginx.service MemoryMax=512M CPUWeight=200

# Von Hand (v2)
$ sudo mkdir /sys/fs/cgroup/test
$ echo "+memory +cpu" | sudo tee /sys/fs/cgroup/cgroup.subtree_control
$ echo 100M | sudo tee /sys/fs/cgroup/test/memory.max
$ echo $$   | sudo tee /sys/fs/cgroup/test/cgroup.procs   # Shell hineinschieben
```

33.4. chroot, systemd-nspawn und LXC

Niveau: Experte

`chroot` ist der Urahn der Container: Es ändert für einen Prozess das scheinbare Wurzelverzeichnis. Es isoliert aber weder Prozesse noch Netzwerk und ist keine Sicherheitsgrenze, eignet sich jedoch hervorragend zur Reparatur eines installierten Systems von einem Live-Medium aus:

```
$ sudo mount /dev/vg0/root /mnt
$ for d in dev proc sys; do sudo mount --rbind /$d /mnt/$d; done
$ sudo chroot /mnt /bin/bash          # nun "im" installierten System
# update-grub; passwd; ...
```

systemd-nspawn startet eine komplette Betriebssystem-Umgebung in einem Verzeichnis mit Namespaces und cgroups; **LXC/LXD (Incus)** bietet "Systemcontainer", die sich wie leichte VMs verhalten:

```
$ sudo debootstrap stable /var/lib/machines/deb https://deb.debian.org/debian
$ sudo systemd-nspawn -D /var/lib/machines/deb          # Shell im Container
$ sudo systemd-nspawn -b -D /var/lib/machines/deb       # Container booten (systemd als PID
    1)
$ machinectl list

$ incus launch images:debian/12 web1                    # LXC/Incus
$ incus exec web1 -- bash
$ incus list
```

33.5. Sicherheitsmechanismen

Niveau: Experte

- **Capabilities** zerlegen die Root-Macht in Einzelrechte (z. B. `CAP_NET_BIND_SERVICE`: Ports unter 1024 binden; `CAP_SYS_ADMIN`: sehr umfassend). Container erhalten nur eine kleine Teilmenge. Anzeige: `capsh --print`, `getpcaps <PID>`.
- **seccomp** filtert zulässige Systemaufrufe. Docker liefert ein Standardprofil, das rund 40 gefährliche Aufrufe blockiert.
- **SELinux/AppArmor** erzwingen Zugriffsregeln unabhängig von Dateirechten (Kapitel 23).
- **User-Namespaces** und *rootless*-Container: Auch "root" im Container hat auf dem Host keine Root-Rechte.
- **Schreibgeschützte Dateisysteme**, `no-new-privileges` und entfernte Capabilities verringern die Angriffsfläche weiter.

> **Wichtig**
>
> Container teilen den Kernel. Eine Sicherheitslücke im Kernel oder ein zu freizügig konfigurierter Container (`--privileged`, Docker-Socket eingebunden, Host-Verzeichnisse mit Schreibrecht) kann den Host gefährden. Für Mandantentrennung mit hohen Anforderungen sind VMs oder spezialisierte Sandboxes (gVisor, Kata Containers) die stärkere Wahl.

Übungen und Kontrollfragen

1. Was ist der Unterschied zwischen Namespaces und cgroups?

   Lösung: Namespaces begrenzen die *Sicht* (Prozesse, Netzwerk, Dateisysteme ...), cgroups begrenzen *Ressourcen* (CPU, RAM, E/A).

2. Wie begrenzen Sie einen Befehl auf 500 MB RAM?

   Lösung: `systemd-run --scope -p MemoryMax=500M befehl`.

3. Warum ist `chroot` kein Sicherheitsmechanismus?

   Lösung: Es ändert nur das Wurzelverzeichnis; ein Prozess mit Root-Rechten kann ausbrechen. Prozesse, Netzwerk und Benutzer bleiben ungetrennt.

4. Wie starten Sie eine Shell mit eigenem Prozessbaum?

   Lösung: `sudo unshare --pid --fork --mount-proc bash`.

5. Warum sind `--privileged` und der eingebundene Docker-Socket riskant?

   Lösung: Sie geben dem Container praktisch Root-Zugriff auf den Host (alle Geräte/Capabilities bzw. Kontrolle über den Docker-Daemon).

34. Docker und Podman

Inhalt: Images und Container, Registries, Dockerfile, Volumes und Netzwerke, Docker Compose, Podman und rootless-Betrieb, Aufräumen, Sicherheit und Betrieb von Containern.

Stufe: [F] Fortgeschritten bis [X] Experte

Lernziele: Nach diesem Kapitel können Sie

- Container zu starten, zu untersuchen und zu beenden,
- eigene Images mit einem Dockerfile zu bauen,
- mehrteilige Anwendungen mit Compose zu beschreiben und sicher zu betreiben.

Vorkenntnisse: Kapitel 33 (Container-Grundlagen), Kapitel 24 (Prozesse) und Kapitel 39 (Netzwerk).

34.1. Konzepte

Niveau: Fortgeschritten

- **Image**: Unveränderliche Vorlage aus übereinander liegenden Schichten (*layers*): Basissystem, Bibliotheken, Anwendung.
- **Container**: Laufende (oder gestoppte) Instanz eines Images mit einer dünnen beschreibbaren Schicht obenauf. Wird der Container gelöscht, sind die Änderungen weg – dauerhafte Daten gehören in **Volumes**.
- **Registry**: Server für Images (Docker Hub, GitHub Container Registry, Quay, eigene Registry). Ein Image heißt `registry/name:tag`, z. B. `docker.io/library/nginx:1.27`.
- **Docker** (Daemon `dockerd`, benötigt Root-Rechte) und **Podman** (daemonlos, standardmäßig rootless) sind kompatible Werkzeuge; die meisten Befehle sind identisch (`alias docker=podman`).

```
# Installation
$ sudo apt install docker.io            # Debian/Ubuntu (oder docker-ce aus dem Docker-
    Repository)
$ sudo dnf install podman               # RHEL/Fedora
$ sudo systemctl enable --now docker
$ sudo usermod -aG docker anna          # Achtung: gleichbedeutend mit Root-Rechten!
$ docker version; docker info
$ docker run --rm hello-world
```

Sicherheit

Wer in der Gruppe `docker` ist, kann den Host kontrollieren (z. B. mit `docker run -v /:/host ...`) – die Mitgliedschaft ist praktisch ein Root-Äquivalent. Podman im rootless-Modus vermeidet dieses Problem.

34.2. Container starten und verwalten

Niveau: Fortgeschritten

```
$ docker run -d --name web -p 8080:80 nginx:1.27          # im Hintergrund, Port 8080 -> 80
$ docker ps                                               # laufende Container
$ docker ps -a                                            # alle, auch gestoppte
$ docker logs -f web                                      # Ausgabe verfolgen
$ docker exec -it web bash                                # Shell im Container
$ docker inspect web | jq '.[0].NetworkSettings.IPAddress'
$ docker stats                                            # Ressourcenverbrauch live
$ docker top web                                          # Prozesse im Container
$ docker stop web; docker start web; docker restart web
$ docker rm -f web                                        # loeschen (auch laufend)
$ docker run --rm -it debian:12 bash                      # Wegwerf-Container
```

Option von `docker run`	Wirkung
`-d`	Im Hintergrund (*detached*)
`-it`	Interaktiv mit Terminal
`--name N`	Namen vergeben
`--rm`	Container nach Beenden automatisch löschen
`-p 8080:80`	Host-Port 8080 an Container-Port 80 weiterleiten
`-v name:/pfad`, `-v /host:/pfad`	Volume bzw. Verzeichnis einbinden (`:ro` = schreibgeschützt)
`-e VAR=wert`, `--env-file f`	Umgebungsvariablen
`--restart unless-stopped`	Nach Neustart/Absturz automatisch starten
`--memory 512m --cpus 1.5`	Ressourcen begrenzen (cgroups)
`--network N`	Mit einem bestimmten Netzwerk verbinden
`--user 1000:1000`	Als normalen Benutzer ausführen
`--read-only`	Wurzeldateisystem schreibgeschützt

34.3. Images

Niveau: Fortgeschritten

```
$ docker pull debian:12                  # Image laden
$ docker images                          # lokale Images
$ docker search nginx
$ docker history nginx:1.27              # Schichten
$ docker tag meinbild:1.0 registry.example.org/team/meinbild:1.0
$ docker login registry.example.org
$ docker push registry.example.org/team/meinbild:1.0
$ docker save -o bild.tar meinbild:1.0   # exportieren
$ docker load -i bild.tar
$ docker rmi nginx:1.27
```

Tags sind veränderlich: `nginx:latest` zeigt morgen auf ein anderes Image. In Produktion feste Versionen oder Digests verwenden (`nginx@sha256:...`) und Images aus vertrauenswürdigen Quellen (*Official Images*) beziehen.

34.4. Eigene Images mit dem Dockerfile

Niveau: Fortgeschritten

```
# Dockerfile
FROM python:3.12-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
RUN useradd -r -u 1001 appuser
USER appuser
EXPOSE 8000
ENV PORT=8000
HEALTHCHECK CMD curl -f http://localhost:8000/health || exit 1
ENTRYPOINT ["python"]
CMD ["app.py"]
```

Anweisung	Bedeutung
`FROM`	Basisimage
`RUN`	Befehl beim *Bauen* ausführen (erzeugt eine Schicht)
`COPY`, `ADD`	Dateien ins Image kopieren (`COPY` bevorzugen)
`WORKDIR`, `USER`, `ENV`	Arbeitsverzeichnis, Benutzer, Umgebungsvariable
`EXPOSE`	Dokumentiert den Port (veröffentlicht ihn nicht)
`CMD`, `ENTRYPOINT`	Programm beim *Starten* des Containers
`HEALTHCHECK`	Prüfung, ob der Dienst funktioniert

```
$ docker build -t meinbild:1.0 .                # bauen (Kontext = aktuelles Verzeichnis)
$ docker build -f Dockerfile.prod -t meinbild:prod .
$ docker run --rm -p 8000:8000 meinbild:1.0
```

Gute Praxis: kleine Basisimages (`-slim`, `alpine`), selten ändernde Schritte zuerst (Cache nutzen), `.dockerignore` verwenden, einen Prozess pro Container, kein Root, keine Geheimnisse im Image (sie bleiben in den Schichten sichtbar). *Multi-Stage-Builds* halten das Endimage klein: In einer ersten Stufe wird kompiliert, in der zweiten nur das Ergebnis kopiert.

```
FROM golang:1.22 AS build
WORKDIR /src
COPY . .
RUN CGO_ENABLED=0 go build -o /out/tool .

FROM gcr.io/distroless/static
COPY --from=build /out/tool /tool
ENTRYPOINT ["/tool"]
```

34.5. Volumes und Netzwerke

Niveau: Fortgeschritten

```
$ docker volume create daten
$ docker run -d --name db -v daten:/var/lib/postgresql/data -e POSTGRES_PASSWORD=geheim
    postgres:16
$ docker volume ls; docker volume inspect daten
$ docker run --rm -v $PWD:/work -w /work alpine ls    # Verzeichnis des Hosts einbinden

# Sichern eines Volumes
$ docker run --rm -v daten:/d -v $PWD:/backup alpine tar czf /backup/daten.tgz -C /d .

$ docker network ls
$ docker network create appnet
$ docker run -d --name api --network appnet meineapi
$ docker run --rm --network appnet alpine ping -c1 api    # Namensaufloesung im Netz
```

Standardmäßig werden Container an ein Bridge-Netz (`docker0`) gehängt und über NAT nach außen vermittelt; veröffentlichte Ports (`-p`) legt Docker selbst als Firewall-Regeln an. In *benutzerdefinierten* Netzen lösen sich Containernamen gegenseitig per DNS auf. Weitere Treiber: `host` (kein Netz-Namespace), `none`, `macvlan`, `overlay` (mehrere Hosts).

> **Hinweis**
>
> Docker setzt eigene Regeln in `iptables`/`nftables`. Mit `-p 8080:80` ist der Dienst *auch von außen* erreichbar und umgeht oft die Regeln von `ufw`. Soll ein Dienst nur lokal erreichbar sein, bindet man ihn an die Loopback-Adresse: `-p 127.0.0.1:8080:80`.

34.6. Docker Compose

Niveau: Fortgeschritten

Mehrere Container, Netze und Volumes beschreibt man deklarativ in einer Datei `compose.yaml`:

```
services:
  web:
    image: nginx:1.27
    ports: ["127.0.0.1:8080:80"]
    volumes:
      - ./html:/usr/share/nginx/html:ro
    depends_on: [app]
    restart: unless-stopped
  app:
    build: ./app
    environment:
      DB_HOST: db
      DB_PASSWORD_FILE: /run/secrets/dbpass
    secrets: [dbpass]
    depends_on:
      db: {condition: service_healthy}
  db:
    image: postgres:16
    volumes: [dbdata:/var/lib/postgresql/data]
    environment:
      POSTGRES_PASSWORD_FILE: /run/secrets/dbpass
    secrets: [dbpass]
    healthcheck:
      test: ["CMD", "pg_isready", "-U", "postgres"]
      interval: 10s
volumes:
  dbdata:
secrets:
  dbpass:
    file: ./dbpass.txt
```

```
$ docker compose up -d              # alles starten
$ docker compose ps
$ docker compose logs -f app
$ docker compose exec db psql -U postgres
$ docker compose pull && docker compose up -d     # aktualisieren
$ docker compose down               # stoppen und Container loeschen (Volumes bleiben)
$ docker compose down -v            # auch Volumes loeschen!
```

34.7. Podman und rootless

Niveau: Fortgeschritten

`podman` verwendet dieselbe Befehlssyntax, benötigt aber keinen Daemon und läuft ohne Root-Rechte im **User-Namespace**. Zusätzlich bringt es **Pods** (Gruppen von Containern mit gemeinsamem Netz wie in Kubernetes) und die Integration mit systemd mit:

```
$ podman run -d --name web -p 8080:80 docker.io/library/nginx
$ podman ps
$ podman generate systemd --new --name web > ~/.config/systemd/user/container-web.service
$ systemctl --user enable --now container-web.service
$ loginctl enable-linger anna            # Benutzerdienste auch ohne Anmeldung starten

# Quadlet: Container als systemd-Unit beschreiben (~/.config/containers/systemd/web.container
    )
[Container]
Image=docker.io/library/nginx:1.27
PublishPort=127.0.0.1:8080:80
[Install]
WantedBy=default.target
```

Rootless-Container können nur Ports ab 1024 öffnen (Anpassung: `net.ipv4.ip_unprivileged_port_start`); Dateien erscheinen auf dem Host mit verschobenen UIDs (`/etc/subuid`).

34.8. Aufräumen und Betrieb

Niveau: Experte

```
$ docker system df                      # Platzverbrauch
$ docker system prune -a --volumes      # ACHTUNG: loescht ungenutzte Images, Container,
    Volumes
$ docker container prune; docker image prune; docker volume prune
$ journalctl -u docker                  # Dienst-Log
$ cat /etc/docker/daemon.json
{ "log-driver": "json-file",
  "log-opts": {"max-size": "10m", "max-file": "3"} }
```

- **Protokolle** begrenzen (`max-size`), sonst füllen sie die Festplatte.
- **Images regelmäßig neu bauen**, um Sicherheitsupdates der Basis zu erhalten; mit Scannern wie `trivy` oder `grype` auf bekannte Schwachstellen prüfen.
- **Geheimnisse** nicht in Umgebungsvariablen oder Images, sondern über Secrets/Dateien (`*_FILE`) einbringen.
- **Daten** gehören in benannte Volumes; Sicherungen planen (Kapitel 53).
- **Orchestrierung**: Für viele Hosts und Hochverfügbarkeit setzt man Kubernetes (oder Nomad, Docker Swarm) ein; für einzelne Server genügt Compose.

Übungen und Kontrollfragen

1. Was passiert mit Daten im Container, wenn er gelöscht wird?

   Lösung: Sie gehen verloren, sofern sie nicht in einem Volume oder einem eingebundenen Verzeichnis liegen.

2. Wie starten Sie nginx so, dass er nur lokal auf Port 8080 erreichbar ist?

   Lösung: `docker run -d -p 127.0.0.1:8080:80 nginx`.

3. Worin unterscheiden sich `CMD` und `RUN`?

   Lösung: `RUN` läuft beim Bauen und erzeugt eine Schicht; `CMD` legt den Standardbefehl beim Start des Containers fest.

4. Warum soll man Container nicht als Root betreiben?

   Lösung: Bei einer Schwachstelle hätte ein Angreifer im Container Root-Rechte und damit bessere Chancen auf einen Ausbruch. Mit `USER`, `--user` oder rootless-Podman verringert man das Risiko.

5. Wie sichern Sie die Daten eines Volumes?

   Lösung: Mit einem Hilfscontainer, der das Volume und ein Host-Verzeichnis einbindet und per `tar` sichert (siehe Beispiel), oder dienstspezifisch (z. B. `pg_dump`).

35. Virtualisierung mit KVM und libvirt

Inhalt: Hardware-Virtualisierung, KVM und QEMU, `libvirt` und `virsh`, virtuelle Netze und Speicher, Snapshots, Verwaltung mit Cockpit und `virt-manager`.

Stufe: [F] Fortgeschritten bis [X] Experte

Lernziele: Nach diesem Kapitel können Sie

- die Virtualisierungstechniken unter Linux einzuordnen,
- virtuelle Maschinen mit `virt-install` und `virsh` anzulegen und zu steuern,
- Netzwerk und Speicher für VMs zu konfigurieren.

Vorkenntnisse: Kapitel 26 (Platten), Kapitel 39 (Netzkonfiguration) und Kapitel 28.

35.1. Virtualisierung im Überblick

Niveau: Fortgeschritten

Mit **Virtualisierung** laufen mehrere vollständige Betriebssysteme (*Gäste*) auf einem Rechner (*Host*). Jede **virtuelle Maschine** (VM) erhält virtuelle CPU, Speicher, Platte und Netzwerkkarte. Eingesetzt wird das für Serverkonsolidierung, Testumgebungen, Labore und zur Trennung von Anwendungen.

Technik	Typ	Hinweis
KVM + QEMU	Hypervisor im Kernel (Typ 1)	Standard unter Linux, nutzt Intel VT-x / AMD-V
Xen	Hypervisor	Älter, in Rechenzentren und Clouds verbreitet
VirtualBox, VMware Workstation	Typ 2 (Desktop-Programm)	Bequem für Desktop-Nutzung
Proxmox VE, oVirt, OpenStack	Plattformen auf KVM-Basis	Webbasierte Verwaltung, Cluster
Container (Docker, LXC)	Betriebssystem-Virtualisierung	Teilen den Kernel, siehe Kapitel 33

KVM (Kernel-based Virtual Machine) ist ein Kernelmodul, das die Virtualisierungserweiterungen der CPU nutzt. **QEMU** emuliert die restliche Hardware (Chipsatz, Netzwerkkarte, Grafik). Mit `virtio`-Geräten (paravirtualisierte Treiber) erreichen Gäste nahezu native Geschwindigkeit. **libvirt** ist die einheitliche Verwaltungsschicht (Bibliothek, Dienst `libvirtd`, Werkzeuge `virsh`, `virt-manager`).

```
$ lscpu | grep -i virtualization          # VT-x oder AMD-V vorhanden?
$ grep -Ec '(vmx|svm)' /proc/cpuinfo      # > 0 = ja
$ lsmod | grep kvm
$ sudo apt install qemu-kvm libvirt-daemon-system libvirt-clients virtinst virt-manager
    bridge-utils
$ sudo dnf install @virtualization
$ sudo systemctl enable --now libvirtd
$ sudo usermod -aG libvirt anna
$ virsh list --all
$ virt-host-validate                       # Pruefung der Host-Konfiguration
```

35.2. Virtuelle Maschinen anlegen

Niveau: Fortgeschritten

```
# Installation aus ISO-Abbild
$ sudo virt-install \
    --name debian12 --memory 2048 --vcpus 2 \
    --disk size=20,format=qcow2 \
    --cdrom /srv/iso/debian-12-netinst.iso \
    --os-variant debian12 \
    --network network=default \
    --graphics spice

# Textinstallation ohne Grafik (Server)
$ sudo virt-install --name srv1 --memory 1024 --vcpus 1 \
    --disk size=10 --location http://deb.debian.org/debian/dists/bookworm/main/installer-
    amd64/ \
    --extra-args 'console=ttyS0' --graphics none --os-variant debian12

# Aus fertigem Cloud-Image
$ sudo qemu-img create -f qcow2 -b debian-12-genericcloud-amd64.qcow2 -F qcow2 vm1.qcow2 20G
$ sudo virt-install --name vm1 --memory 2048 --vcpus 2 --import \
    --disk vm1.qcow2 --os-variant debian12 --network network=default --noautoconsole
```

Cloud-Images lassen sich beim ersten Start automatisch mit `cloud-init` einrichten (Benutzer, SSH-Schlüssel, Pakete); das Vorgehen einschließlich lokalem KVM-Test beschreibt Kapitel 61.

35.3. VMs steuern mit virsh

Niveau: Fortgeschritten

Befehl	Wirkung
`virsh list --all`	Alle VMs und ihr Zustand
`virsh start VM` / `shutdown VM`	Starten / sauber herunterfahren (ACPI)
`virsh destroy VM`	Hart ausschalten (Stecker ziehen)
`virsh reboot VM`, `suspend`, `resume`	Neustart, anhalten, fortsetzen
`virsh console VM`	Serielle Konsole (Verlassen: `Ctrl-]`)
`virsh dominfo VM`, `domifaddr VM`	Eigenschaften, IP-Adresse
`virsh edit VM`	XML-Definition bearbeiten
`virsh dumpxml VM > vm.xml`	Konfiguration exportieren
`virsh define vm.xml`	VM aus XML anlegen
`virsh undefine VM --remove-all-storage`	VM und Datenträger löschen
`virsh autostart VM`	Beim Hostbeginn automatisch starten
`virsh setmem VM 4G --live`, `setvcpus`	Ressourcen anpassen
`virsh snapshot-create-as VM s1`	Snapshot anlegen
`virsh snapshot-list VM`, `snapshot-revert VM s1`	Snapshots auflisten / zurückkehren
`virsh migrate --live VM qemu+ssh://host2/system`	Laufende VM auf anderen Host verschieben

Die VM-Definition ist eine XML-Datei (`/etc/libvirt/qemu/VM.xml`), die man mit `virsh edit` bearbeitet:

```
<domain type='kvm'>
  <name>debian12</name>
  <memory unit='GiB'>2</memory>
  <vcpu>2</vcpu>
  <os><type arch='x86_64' machine='q35'>hvm</type></os>
  <devices>
    <disk type='file' device='disk'>
      <driver name='qemu' type='qcow2'/>
      <source file='/var/lib/libvirt/images/debian12.qcow2'/>
      <target dev='vda' bus='virtio'/>
    </disk>
    <interface type='network'>
      <source network='default'/>
      <model type='virtio'/>
    </interface>
  </devices>
</domain>
```

35.4. Speicher für VMs

Niveau: Fortgeschritten

- **Abbilddateien**: `qcow2` (Snapshots, Thin Provisioning, Kompression) oder `raw` (schnell, einfach). Ablage standardmäßig in `/var/lib/libvirt/images`.
- **Logical Volumes (LVM)**: Direkt als Blockgerät, geringer Overhead (Kapitel 28).
- **Storage Pools**: libvirt fasst Verzeichnisse, LVM-Gruppen, NFS, iSCSI oder Ceph in Pools zusammen (`virsh pool-list`, `vol-list`).

```
$ qemu-img create -f qcow2 /var/lib/libvirt/images/daten.qcow2 50G
$ qemu-img info vm1.qcow2                       # virtuelle und belegte Groesse
$ qemu-img resize vm1.qcow2 +10G                # vergroessern (danach im Gast Partition/FS
    anpassen)
$ qemu-img convert -O qcow2 -c in.vmdk out.qcow2    # Formate umwandeln
$ virsh attach-disk vm1 /var/lib/libvirt/images/daten.qcow2 vdb --persistent --driver qemu --
    subdriver qcow2
$ virt-df -h vm1                                # Platz im Gast, ohne Anmeldung (libguestfs)
$ virt-customize -a vm1.qcow2 --root-password password:test --install nginx
```

35.5. Netzwerk für VMs

Niveau: Fortgeschritten

Modus	Beschreibung
NAT (`default`)	VMs hängen an einer virtuellen Bridge `virbr0` (192.168.122.0/24), libvirt verteilt Adressen (dnsmasq) und übersetzt per NAT nach außen. Von außen nicht direkt erreichbar.
Bridge	VMs hängen über eine Bridge (z. B. `br0`) direkt im LAN und erhalten Adressen aus dem realen Netz.
Isoliert	Nur VMs untereinander.
macvtap/direct	Direkter Zugriff über die physische Schnittstelle (Host kann nicht mit der VM kommunizieren).

```
$ virsh net-list --all
$ virsh net-dhcp-leases default               # Adressen der VMs
$ virsh net-edit default
# Bridge br0 mit NetworkManager
$ nmcli con add type bridge ifname br0 con-name br0
$ nmcli con add type bridge-slave ifname enp3s0 master br0
$ nmcli con up br0
$ virt-install ... --network bridge=br0
```

35.6. Verwaltungsoberflächen

Niveau: Fortgeschritten

- **virt-manager**: Grafische Oberfläche (lokal oder per SSH mit der Option `-c` und einer `qemu+ssh`-URL).
- **Cockpit** (`cockpit-machines`): Weboberfläche für Server (`https://server:9090`).
- **Proxmox VE**: Komplette Plattform mit Cluster, Backup, Web-UI – auf Debian und KVM/LXC aufgebaut.
- **Vagrant** und **Multipass**: Reproduzierbare Entwicklungs-VMs aus einer Beschreibungsdatei.

Übungen und Kontrollfragen

1. Wie prüfen Sie, ob ein Rechner KVM nutzen kann?

   Lösung: `grep -Ec '(vmx|svm)' /proc/cpuinfo` (Ergebnis größer 0), `lsmod | grep kvm` und `virt-host-validate`; ggf. Virtualisierung im BIOS/UEFI aktivieren.

2. Was ist der Unterschied zwischen `virsh shutdown` und `virsh destroy`?

   Lösung: `shutdown` sendet ein ACPI-Signal (sauberes Herunterfahren), `destroy` beendet die VM sofort wie ein Ausschalten ohne Herunterfahren; kein Löschen der VM.

3. Warum sind virtio-Geräte vorzuziehen?

   Lösung: Sie sind paravirtualisiert, brauchen deutlich weniger CPU-Aufwand als emulierte Hardware und erreichen höheren Durchsatz.

4. Wie erreichen Sie, dass eine VM direkt eine Adresse aus dem LAN erhält?

   Lösung: Bridge auf dem Host einrichten und die VM mit `--network bridge=br0` anbinden.

36. Zeitgesteuerte Aufgaben

Inhalt: Einmalige Aufgaben mit `at`, wiederkehrende mit `cron`, systemd-Timer.
Stufe: [E] Einsteiger bis [F] Fortgeschritten
Lernziele: Nach diesem Kapitel können Sie
- Aufgaben einmalig und regelmäßig zu planen,
- die Crontab-Syntax sicher zu lesen und zu schreiben,
- systemd-Timer als Alternative einzusetzen.

Vorkenntnisse: Kapitel 13 und 32.

36.1. Einmalige Aufgaben mit at

Niveau: Einsteiger

`at` führt einen Befehl *einmal* zu einem späteren Zeitpunkt aus (Paket `at`, Dienst `atd`).

```
$ at 22:30                          # Befehle eingeben, Ende mit Strg+D
at> /usr/local/bin/backup.sh
at> <EOT>
$ echo "reboot" | at now + 2 hours  # relative Angabe
$ at 10:00 tomorrow < skript.sh     # aus Datei
$ atq                               # wartende Auftraege (Nummer, Zeit)
$ at -c 3                           # Inhalt von Auftrag 3
$ atrm 3                            # Auftrag loeschen
```

Berechtigungen regeln `/etc/at.allow` und `/etc/at.deny`. Ausgaben schickt `at` per Mail an den Benutzer.

36.2. Wiederkehrende Aufgaben mit cron

Niveau: Einsteiger

`cron` (Dienst `cron` bzw. `crond`) führt Aufgaben regelmäßig aus. Jeder Benutzer hat eine eigene *Crontab*.

```
$ crontab -e                        # eigene Crontab bearbeiten
$ crontab -l                        # anzeigen
$ crontab -r                        # loeschen (Vorsicht, keine Rueckfrage!)
# crontab -u anna -e                # Crontab eines anderen Benutzers
```

Jede Zeile enthält fünf Zeitfelder und den Befehl:

```
# +-------------- Minute        (0-59)
# | +------------ Stunde        (0-23)
# | | +---------- Tag im Monat  (1-31)
# | | | +-------- Monat         (1-12)
# | | | | +------ Wochentag     (0-7; 0 und 7 = Sonntag)
# | | | | |
# * * * * *  Befehl
  30 2 * * *       /usr/local/bin/backup.sh                 # taeglich 02:30
  */15 * * * *     /usr/bin/curl -s https://example.org/ping # alle 15 Minuten
  0 8 * * 1-5      /home/anna/bin/bericht.sh                # Mo-Fr 08:00
  0 0 1 * *        /usr/local/bin/monatsabschluss.sh        # am 1. jedes Monats
  @reboot          /home/anna/bin/start.sh                  # beim Systemstart
```

Zeichen	Bedeutung
`*`	Jeder Wert
`,`	Aufzählung: `1,15`
`-`	Bereich: `1-5`
`/`	Schrittweite: `*/10` = alle 10
`@daily`, `@hourly`, `@weekly`, `@monthly`	Abkürzungen

Achtung

`cron` startet Aufgaben mit einer minimalen Umgebung (kurzer `PATH`, keine Profile). Daher *absolute Pfade* verwenden. Ein `%` in der Crontab-Zeile muss als `\%` geschrieben werden. Ausgaben landen per Mail beim Benutzer; mit `>> /var/log/job.log 2>&1` umleiten.

Systemweite Aufgaben

- `/etc/crontab` und `/etc/cron.d/*`: Format mit *zusätzlichem Benutzerfeld* vor dem Befehl.
- `/etc/cron.hourly`, `cron.daily`, `cron.weekly`, `cron.monthly`: Skripte hineinlegen (ausführbar, ohne Punkt im Namen).
- `anacron` holt Aufgaben nach, wenn der Rechner zur Startzeit ausgeschaltet war (typisch auf Notebooks).
- Zugriffssteuerung: `/etc/cron.allow`, `/etc/cron.deny`.

```
# cat /etc/cron.d/backup
# m h dom mon dow user  command
30 2 * * * root  /usr/local/bin/backup.sh
```

36.3. systemd-Timer

Niveau: Fortgeschritten

Timer sind die moderne Alternative: Sie sind in `journald` protokolliert, können Abhängigkeiten nutzen, holen verpasste Läufe nach (`Persistent=true`) und lassen sich wie jede Unit verwalten. Ein Timer startet eine gleichnamige Service-Unit.

```
# /etc/systemd/system/backup.service
[Unit]
Description=Tagesbackup

[Service]
Type=oneshot
ExecStart=/usr/local/bin/backup.sh

# /etc/systemd/system/backup.timer
[Unit]
Description=Tagesbackup taeglich 02:30

[Timer]
OnCalendar=*-*-* 02:30:00
RandomizedDelaySec=10min
Persistent=true

[Install]
WantedBy=timers.target
```

```
# systemctl daemon-reload
# systemctl enable --now backup.timer
$ systemctl list-timers                     # naechste/letzte Ausfuehrung
$ journalctl -u backup.service              # Ausgaben des Laufs
$ systemd-analyze calendar "Mon..Fri 08:00" # Zeitausdruck pruefen
```

Timer-Option	**Bedeutung**
`OnCalendar=`	Kalenderzeit, z. B. `daily`, `weekly`, `Mon *-*-* 08:00`
`OnBootSec=`, `OnUnitActiveSec=`	Relativ zum Systemstart bzw. zum letzten Lauf (*monotone* Timer)
`Persistent=true`	Verpassten Lauf nach dem Einschalten nachholen
`RandomizedDelaySec=`	Zufällige Verzögerung (verteilt Last)

Wann man was nutzt: `cron` ist einfach, überall verfügbar und für Benutzer ohne `root`-Rechte geeignet. Timer sind besser für Systemdienste mit Logging, Abhängigkeiten und Ressourcenlimits.

37. Protokollierung

Inhalt: Logdateien in `/var/log`, Journal und `journalctl`, `rsyslog`, Logrotation, Auswertung.
Stufe: [E] Einsteiger bis [X] Experte
Lernziele: Nach diesem Kapitel können Sie

- Ereignisse im Journal und in Logdateien zu finden,
- journald und rsyslog zu konfigurieren,
- Logs zu rotieren und systematisch auszuwerten.

Vorkenntnisse: Kapitel 8, 9 und 32.

37.1. Wo finde ich Logs?

Niveau: Einsteiger

Linux protokolliert Ereignisse an zwei Orten, die sich ergänzen: im binären **Journal** (Dienst `journald`) und in klassischen **Textdateien** unter `/var/log`, die meist von `rsyslog` geschrieben werden.

Datei	Inhalt
`/var/log/syslog` bzw. `/var/log/messages`	Allgemeine Systemmeldungen (Debian/Ubuntu bzw. RHEL)
`/var/log/auth.log` bzw. `/var/log/secure`	Anmeldungen, `sudo`, `ssh`
`/var/log/kern.log`	Kernelmeldungen
`/var/log/dmesg`, Kommando `dmesg`	Kernel-Ringpuffer (Hardware, Treiber)
`/var/log/dpkg.log`, `/var/log/apt/`	Paketinstallationen (Debian)
`/var/log/dnf.log`, `yum.log`	Paketinstallationen (RHEL)
`/var/log/wtmp`, `btmp`	Anmeldungen/Fehlanmeldungen (Binärformat: `last`, `lastb`)
`/var/log/nginx/`, `apache2/`	Logs der Webserver

```
$ tail -f /var/log/syslog                  # live mitlesen
$ sudo grep -i "failed password" /var/log/auth.log | tail
$ sudo less /var/log/messages
$ last -n 10                               # letzte Anmeldungen
$ sudo lastb | head                        # fehlgeschlagene Anmeldungen
$ lastlog                                  # letzte Anmeldung je Benutzer
$ dmesg -T --level=err,warn                # Kernelwarnungen mit Zeitstempel
```

37.2. journalctl: das Journal lesen

Niveau: Einsteiger

```
$ journalctl                          # alles, aelteste zuerst
$ journalctl -r                       # neueste zuerst
$ journalctl -f                       # live verfolgen (wie tail -f)
$ journalctl -n 50                    # letzte 50 Zeilen
$ journalctl -b                       # seit dem aktuellen Start
$ journalctl -b -1                    # vorheriger Start
$ journalctl --list-boots
$ journalctl -u nginx                 # nur eine Unit
$ journalctl -u nginx -u php-fpm      # mehrere Units
$ journalctl -p err                   # Prioritaet err und schlimmer
$ journalctl -p warning..err
$ journalctl -k                       # nur Kernelmeldungen
$ journalctl --since "2025-03-01" --until "2025-03-02 12:00"
$ journalctl --since "1 hour ago"
$ journalctl _UID=1000                # nach Feldern filtern
$ journalctl _COMM=sshd
$ journalctl -o json-pretty -n 1      # Ausgabeformat (cat, short-iso, json, ...)
$ journalctl --disk-usage             # Platzbedarf
# journalctl --vacuum-time=2weeks     # Eintraege aelter als 2 Wochen loeschen
# journalctl --vacuum-size=500M
```

Stufe	Name	Bedeutung
0	`emerg`	System unbenutzbar
1	`alert`	Sofortiges Eingreifen nötig
2	`crit`	Kritischer Zustand
3	`err`	Fehler
4	`warning`	Warnung
5	`notice`	Wichtige Information
6	`info`	Information
7	`debug`	Debug-Meldungen

37.3. journald dauerhaft speichern und konfigurieren

Niveau: Fortgeschritten

Standardmäßig legt `journald` unter manchen Distributionen das Journal nur im Speicher (`/run/log/journal`) ab, sodass es beim Neustart verloren geht.

```
# mkdir -p /var/log/journal                  # dauerhaft speichern (Storage=auto)
# systemctl restart systemd-journald

# nano /etc/systemd/journald.conf
[Journal]
Storage=persistent
SystemMaxUse=1G              # Gesamtgroesse begrenzen
MaxRetentionSec=1month
Compress=yes
ForwardToSyslog=yes          # an rsyslog weiterreichen
```

37.4. rsyslog

Niveau: Fortgeschritten

`rsyslog` nimmt Meldungen entgegen und verteilt sie nach **Facility** (Herkunft, z. B. `auth`, `kern`, `mail`, `cron`, `local0`–`local7`) und **Priorität** (Tabelle oben) in Dateien, an Programme oder über das Netzwerk an einen Logserver.

```
# /etc/rsyslog.conf  bzw.  /etc/rsyslog.d/*.conf
# Facility.Prioritaet     Ziel
auth,authpriv.*           /var/log/auth.log
*.*;auth,authpriv.none    -/var/log/syslog          # '-' = nicht sofort synchronisieren
kern.*                    -/var/log/kern.log
mail.err                  /var/log/mail.err
*.emerg                   :omusrmsg:*               # an alle angemeldeten Benutzer
*.*                       @@loghost.example.org:514 # TCP (@ = UDP) an zentralen Logserver
```

```
# systemctl restart rsyslog
$ logger "Test aus der Shell"                       # eigene Meldung erzeugen
$ logger -p local0.warning -t meinskript "Platte fast voll"
$ journalctl -t meinskript
```

In Shell-Skripten ist `logger` der Standardweg, um sauber ins Systemlog zu schreiben.

37.5. Logrotation mit logrotate

Niveau: Fortgeschritten

Logdateien wachsen unbegrenzt. `logrotate` (täglich über `cron`/Timer aufgerufen) benennt Logs um, komprimiert alte und löscht sehr alte Dateien.

```
# /etc/logrotate.d/meinapp
/var/log/meinapp/*.log {
    weekly                  # woechentlich rotieren
    rotate 8                # 8 Generationen behalten
    compress                # mit gzip komprimieren
    delaycompress           # die neueste rotierte Datei noch nicht
    missingok               # kein Fehler wenn Datei fehlt
    notifempty              # leere Logs nicht rotieren
    create 0640 meinapp adm
    postrotate
        systemctl reload meinapp > /dev/null 2>&1 || true
    endscript
}
```

```
# logrotate -d /etc/logrotate.d/meinapp     # Testlauf (debug), aendert nichts
# logrotate -f /etc/logrotate.conf          # Rotation erzwingen
```

37.6. Logs auswerten

Niveau: Experte

```
# Haeufigste IP-Adressen mit fehlgeschlagenen SSH-Anmeldungen
$ grep "Failed password" /var/log/auth.log | awk '{print $(NF-3)}' | sort | uniq -c | sort -
    rn | head

# HTTP-Statuscodes im nginx-Log zaehlen
$ awk '{print $9}' /var/log/nginx/access.log | sort | uniq -c | sort -rn

# Fehler der letzten Stunde je Unit
$ journalctl -p err --since "1 hour ago" -o json | jq -r '._SYSTEMD_UNIT' | sort | uniq -c
```

Für größere Umgebungen empfiehlt sich ein **zentraler Logserver** (rsyslog, Loki, ELK-Stack): Logs bleiben auch erhalten, wenn ein Rechner kompromittiert wird. `fail2ban` wertet Logs aus und sperrt Angreifer automatisch per Firewall.

> **Tipp**
> Bei einem unbekannten Fehler hilft die Reihenfolge: `systemctl status <Dienst>`, dann `journalctl -u <Dienst> -b`, danach `journalctl -p err -b` und zuletzt die Logdateien der Anwendung selbst.

Teil III.

Netzwerk

38. TCP/IP-Grundlagen

Inhalt: Schichtenmodell, Adressen, IPv4-Subnetting, Routing, TCP und UDP, Ports, IPv6, DNS.
Stufe: [E] Einsteiger bis [X] Experte
Lernziele: Nach diesem Kapitel können Sie
- das TCP/IP-Modell zu erklären,
- Netze zu berechnen und Adressbereiche zuzuordnen,
- Ports, Routing und Namensauflösung zu verstehen.

Vorkenntnisse: Keine besonderen; Kapitel 2 hilft bei den Beispielen.

38.1. Schichtenmodell

Niveau: Einsteiger

Netzwerkkommunikation ist in Schichten aufgeteilt; jede nutzt die Dienste der darunterliegenden. Das **TCP/IP-Modell** hat vier Schichten (das OSI-Modell sieben):

TCP/IP-Schicht	Beispiele	Aufgabe	OSI
Anwendung	HTTP, SSH, DNS, SMTP	Dienste für Programme	5–7
Transport	TCP, UDP	Ende-zu-Ende zwischen Prozessen (Ports)	4
Internet	IP, ICMP	Adressierung und Weiterleitung (Routing)	3
Netzzugang	Ethernet, WLAN, ARP	Übertragung im lokalen Netz (MAC-Adressen)	1–2

Beim Senden verpackt jede Schicht die Daten mit eigenem Kopf (*Encapsulation*): Anwendungsdaten → TCP-Segment → IP-Paket → Ethernet-Frame.

38.2. Adressen im lokalen Netz

Niveau: Einsteiger

- **MAC-Adresse**: 48 Bit, hexadezimal (`52:54:00:ab:cd:ef`), fest in der Netzwerkkarte, nur im lokalen Netz relevant.
- **IP-Adresse**: Logische Adresse; wird einer Schnittstelle zugewiesen und ermöglicht Routing über Netzgrenzen.
- **ARP** (*Address Resolution Protocol*): Übersetzt im selben Netz die IPv4-Adresse in eine MAC-Adresse („Wer hat 192.168.1.1?" – Broadcast). Bei IPv6 übernimmt das *Neighbor Discovery*.

```
$ ip neigh                              # ARP-/Nachbartabelle
192.168.1.1 dev eth0 lladdr 3c:90:66:aa:bb:cc REACHABLE
```

38.3. IPv4-Adressen und Netzmasken

Niveau: Einsteiger

Eine IPv4-Adresse besteht aus 32 Bit, geschrieben als vier Dezimalzahlen (`192.168.1.20`). Die **Netzmaske** trennt den Netzanteil vom Hostanteil. Die Kurzschreibweise **CIDR** gibt die Anzahl der Netzbits an: `192.168.1.20/24` entspricht der Maske `255.255.255.0`.

CIDR	Maske	Nutzbare Hosts
/8	255.0.0.0	16 777 214
/16	255.255.0.0	65 534
/24	255.255.255.0	254
/25	255.255.255.128	126
/26	255.255.255.192	62
/27	255.255.255.224	30
/28	255.255.255.240	14
/30	255.255.255.252	2

Pro Netz sind zwei Adressen reserviert: die erste (**Netzadresse**) und die letzte (**Broadcast**). Die Zahl der Hosts ist $2^{32-n} - 2$ bei einem `/n`-Netz.
Beispiel: `192.168.1.77/26`. Ein /26 hat Blöcke von 64 Adressen (0–63, 64–127, …). Die 77 liegt im Block 64–127. Netzadresse: `192.168.1.64`, Broadcast: `192.168.1.127`, Hosts: `.65` bis `.126`.

```
$ ipcalc 192.168.1.77/26                # Berechnung (Paket ipcalc / sipcalc)
$ python3 -c "import ipaddress as i; n=i.ip_network('192.168.1.77/26',strict=False); print(n,
    n.broadcast_address)"
```

Private und besondere Adressen

Bereich	Verwendung
`10.0.0.0/8`, `172.16.0.0/12`, `192.168.0.0/16`	Private Adressen (RFC 1918), im Internet nicht geroutet; Zugang über NAT
`127.0.0.0/8`	Loopback (`localhost`)
`169.254.0.0/16`	Link-Local (Selbstzuweisung ohne DHCP)
`224.0.0.0/4`	Multicast
`0.0.0.0`	"Alle" / unbestimmt (z. B. Dienst lauscht auf allen Schnittstellen)

38.4. Routing und Gateway

Niveau: Einsteiger

Ziele im eigenen Netz erreicht ein Rechner direkt (per ARP/MAC). Für alle anderen Ziele sendet er das Paket an das **Standard-Gateway** (*default route*), meist den Router. Dieser leitet es anhand seiner **Routingtabelle** weiter, bis es das Ziel erreicht.

```
$ ip route
default via 192.168.1.1 dev eth0 proto dhcp metric 100
192.168.1.0/24 dev eth0 proto kernel scope link src 192.168.1.20
$ ip route get 8.8.8.8                  # welchen Weg nimmt ein Paket zu diesem Ziel?
8.8.8.8 via 192.168.1.1 dev eth0 src 192.168.1.20
```

Es gewinnt der *spezifischste* passende Eintrag (längstes Präfix); die `default`-Route (`0.0.0.0/0`) passt auf alles und kommt zuletzt. Ein Linux-Rechner leitet Pakete nur weiter, wenn `net.ipv4.ip_forward=1` gesetzt ist.

38.5. TCP, UDP und Ports

Niveau: Einsteiger

Die Transportschicht unterscheidet Anwendungen über **Portnummern** (16 Bit, 0–65535). Eine Verbindung wird durch das Tupel *(Quell-IP, Quell-Port, Ziel-IP, Ziel-Port, Protokoll)* identifiziert.

	TCP	**UDP**
Eigenschaft	Verbindungsorientiert, zuverlässig, Reihenfolge, Flusskontrolle	Verbindungslos, ohne Garantie, geringer Overhead
Aufbau	Dreiwege-Handshake (SYN, SYN/ACK, ACK)	Kein Aufbau
Typisch	Web, SSH, Mail, Dateitransfer	DNS-Anfragen, Streaming, VoIP, NTP, DHCP

Port	**Dienst**	**Port**	**Dienst**
20/21	FTP	123/udp	NTP
22	SSH	143 / 993	IMAP / IMAPS
25 / 587	SMTP / Submission	161/udp	SNMP
53	DNS (UDP und TCP)	389 / 636	LDAP / LDAPS
67/68	DHCP	443	HTTPS
80	HTTP	445	SMB
110 / 995	POP3 / POP3S	3306 / 5432	MySQL / PostgreSQL

Die Zuordnung steht in `/etc/services`. Ports unter 1024 (*well-known ports*) darf nur `root` öffnen. Dynamische Quellports liegen meist zwischen 32768 und 60999.

```
$ ss -tulpn                          # lauschende Dienste (TCP/UDP, mit Prozess)
$ getent services ssh                # Port zu Dienstname
ssh                  22/tcp
```

38.6. IPv6

Niveau: Fortgeschritten

IPv6 verwendet 128-Bit-Adressen, geschrieben in acht Blöcken hexadezimal. Führende Nullen entfallen, *eine* Folge von Nullblöcken wird durch `::` ersetzt: `2001:0db8:0000:0000:0000:0000:0000:0001` → `2001:db8::1`.

Adresse	**Bedeutung**
`::1`	Loopback
`fe80::/10`	Link-Local, automatisch auf jeder Schnittstelle
`fc00::/7` (`fd..`)	Unique Local (privat)
`2000::/3`	Global Unicast (öffentlich geroutet)
`ff00::/8`	Multicast (ersetzt Broadcast)

Üblich sind `/64`-Subnetze: 64 Bit Netzpräfix, 64 Bit Schnittstellenkennung. Die Adressvergabe läuft per SLAAC (Router-Advertisements) oder DHCPv6. Dank des riesigen Adressraums ist NAT nicht nötig; eine Firewall bleibt trotzdem unverzichtbar.

```
$ ping -6 ::1
$ ping6 -I eth0 fe80::1              # Link-Local braucht Angabe der Schnittstelle
$ ip -6 addr
$ ip -6 route
```

38.7. Namensauflösung mit DNS

Niveau: Einsteiger

Menschen verwenden Namen, das Netz Adressen. Das DNS ist ein verteiltes, hierarchisches Verzeichnis: von der Wurzel (`.`) über Top-Level-Domains (`de`) zu Domains (`example.de`) und Hosts (`www.example.de`). Ein *Resolver* stellt Anfragen an einen DNS-Server, der rekursiv auflöst und das Ergebnis zwischenspeichert (*Cache*, TTL).

Record	Inhalt
`A` / `AAAA`	IPv4- / IPv6-Adresse zu einem Namen
`CNAME`	Alias auf einen anderen Namen
`MX`	Mailserver der Domain (mit Priorität)
`NS`	Zuständige Nameserver
`PTR`	Rückwärtsauflösung: IP → Name
`TXT`	Freitext (SPF, DKIM, Verifizierungen)
`SOA`	Zonenverwaltung (Seriennummer, Zeiten)

```
$ dig example.org                       # A-Record abfragen
$ dig +short example.org AAAA
$ dig MX example.org +short
$ dig @1.1.1.1 example.org              # bestimmten DNS-Server fragen
$ dig -x 93.184.216.34 +short           # Reverse-Lookup
$ dig +trace example.org                # Aufloesung von der Wurzel an verfolgen
$ host example.org
$ getent hosts example.org              # wie ein Programm aufloesen (nutzt nsswitch.conf)
```

38.8. Nützliche Kennzahlen und Details

Niveau: Experte

- **MTU**: Maximale Paketgröße im Netz, bei Ethernet meist 1500 Byte. Zu große Pakete werden fragmentiert oder verworfen; `ping -M do -s 1472 ziel` testet den Pfad.
- **NAT**: Ein Router tauscht private Quelladressen gegen seine öffentliche und merkt sich die Zuordnung (Port-Umsetzung, *PAT*).
- **Broadcast** erreicht alle im Netz; **Multicast** eine Gruppe; **Unicast** genau einen Empfänger.
- **ICMP** transportiert Fehler- und Diagnosemeldungen (`ping`, `traceroute`); es ist kein Transportprotokoll im Sinne von TCP/UDP.
- **TTL**/*Hop Limit*: Jeder Router verringert den Wert; bei 0 wird das Paket verworfen (Schleifenschutz, Grundlage von `traceroute`).

39. Netzwerk konfigurieren

Inhalt: `ip`, Netplan, NetworkManager, DHCP, Hostname und DNS, Firewall-Überblick, Fehlersuche mit `ping`, `ss`, `dig`, `tcpdump`.
Stufe: [E] Einsteiger bis [X] Experte
Lernziele: Nach diesem Kapitel können Sie

- Netzwerkschnittstellen anzuzeigen und dauerhaft zu konfigurieren,
- Namensauflösung und Firewall-Grundlagen einzurichten,
- Netzwerkprobleme systematisch einzugrenzen.

Vorkenntnisse: Kapitel 18 und 38.

39.1. Netzwerkstatus mit ip

Niveau: Einsteiger

Das Kommando `ip` (Paket `iproute2`) ersetzt die veralteten `ifconfig`, `route` und `arp`.

```
$ ip -br addr                          # kompakte Uebersicht aller Schnittstellen
lo       UNKNOWN  127.0.0.1/8 ::1/128
eth0     UP       192.168.1.20/24 fe80::5054:ff:feab:cdef/64
$ ip addr show eth0                    # Details einer Schnittstelle
$ ip link                              # Layer 2: Zustand, MAC, MTU
$ ip route                             # Routingtabelle
$ ip neigh                             # ARP-Tabelle
$ ip -s link show eth0                 # Statistik (Pakete, Fehler, Drops)
```

Temporär (bis zum Neustart) lässt sich direkt konfigurieren – gut zum Testen:

```
# ip link set eth0 up                          # Schnittstelle aktivieren (down = aus)
# ip addr add 192.168.1.50/24 dev eth0         # Adresse hinzufuegen
# ip addr del 192.168.1.50/24 dev eth0
# ip route add default via 192.168.1.1         # Standardroute
# ip route add 10.10.0.0/16 via 192.168.1.254  # statische Route
# ip route del 10.10.0.0/16
# ip link set eth0 mtu 9000
```

39.2. Werkzeuge zur Netzkonfiguration

Niveau: Einsteiger

Die dauerhafte Konfiguration unterscheidet sich je nach Distribution und Einsatz:

Werkzeug	Einsatz	Konfiguration
Netplan	Ubuntu Server/Cloud; erzeugt Konfig für `networkd` oder NetworkManager	`/etc/netplan/*.yaml`
NetworkManager	Desktops, Notebooks, RHEL/Fedora/SUSE; WLAN, VPN	`nmcli`, `nmtui`, `/etc/NetworkManager/`
systemd-networkd	Schlanke Server, Container	`/etc/systemd/network/`
ifupdown	Klassisch Debian	`/etc/network/interfaces`

39.3. Hostname und Namensauflösung

Niveau: Einsteiger

```
$ hostname                              # Namen anzeigen
$ hostnamectl
# hostnamectl set-hostname web01        # dauerhaft setzen (schreibt /etc/hostname)
$ cat /etc/hosts                        # lokale Namenszuordnung (hat Vorrang vor DNS)
127.0.0.1   localhost
127.0.1.1   web01
192.168.1.30  db01.example.org db01
$ cat /etc/resolv.conf                  # DNS-Server und Suchdomaene
nameserver 192.168.1.1
search example.org
$ cat /etc/nsswitch.conf | grep hosts   # Reihenfolge der Quellen
hosts: files dns
$ resolvectl status                     # DNS-Konfiguration bei systemd-resolved
```

Auf Systemen mit `systemd-resolved` zeigt `/etc/resolv.conf` oft auf `127.0.0.53` (lokaler Stub-Resolver). Die tatsächlichen DNS-Server nennt `resolvectl status`; man ändert sie in Netplan oder NetworkManager, nicht in der Datei.

39.4. Netplan (Ubuntu)

Niveau: Fortgeschritten

```
# nano /etc/netplan/01-netcfg.yaml
network:
  version: 2
  renderer: networkd
  ethernets:
    eth0:
      dhcp4: false
      addresses: [192.168.1.20/24]
      routes:
        - to: default
          via: 192.168.1.1
      nameservers:
        search: [example.org]
        addresses: [192.168.1.1, 9.9.9.9]
    eth1:
      dhcp4: true            # Adresse per DHCP
```

```
# netplan try                   # anwenden mit automatischem Rollback nach 120 s
# netplan apply                 # dauerhaft anwenden
# netplan generate              # nur Syntax pruefen und Konfig erzeugen
```

YAML verwendet Leerzeichen zur Einrückung (keine Tabulatoren). Ein Fehler im Einzug ist die häufigste Fehlerquelle; `netplan try` verhindert, dass man sich aussperrt.

39.5. NetworkManager und nmcli

Niveau: Fortgeschritten

```
$ nmcli device status                    # Geraete und Zustand
$ nmcli connection show                  # gespeicherte Verbindungen
$ nmcli device wifi list                 # WLANs
# nmcli device wifi connect "MeinWLAN" password "geheim"

# Statische Adresse konfigurieren
# nmcli connection modify "Wired connection 1" \
      ipv4.method manual \
      ipv4.addresses 192.168.1.20/24 \
      ipv4.gateway 192.168.1.1 \
      ipv4.dns "192.168.1.1 9.9.9.9" \
      ipv4.dns-search example.org
# nmcli connection up "Wired connection 1"

# Zurueck zu DHCP
# nmcli connection modify "Wired connection 1" ipv4.method auto
$ nmtui                                  # textbasierte Oberflaeche
```

39.6. DHCP

Niveau: Fortgeschritten

DHCP (*Dynamic Host Configuration Protocol*) vergibt automatisch IP-Adresse, Netzmaske, Gateway, DNS-Server und Suchdomäne auf Zeit (*Lease*). Ablauf: *Discover* (Broadcast des Clients) → *Offer* → *Request* → *Acknowledge* (DORA). Der Server lauscht auf UDP-Port 67, der Client auf 68.

```
$ ip -br addr show eth0                  # bekommt die Schnittstelle eine Adresse?
$ journalctl -u systemd-networkd | grep -i dhcp
$ nmcli -f DHCP4 device show eth0        # empfangene DHCP-Optionen (NetworkManager)
# dhclient -r eth0 && dhclient eth0      # Lease freigeben und erneuern (dhclient)
```

39.7. Firewall im Überblick

Niveau: Fortgeschritten

Der Kernel filtert Pakete mit `netfilter`. Verwaltet wird er heute mit `nftables` (`nft`), die Frontends sind einfacher:

Werkzeug	Distribution	Merkmal
`ufw`	Ubuntu, Debian	“Uncomplicated Firewall”, einfache Regeln
`firewalld`	RHEL, Fedora, SUSE	Zonen und Dienste, `firewall-cmd`
`nft`, `iptables`	überall	Direktes, vollständiges Regelwerk

```
# ufw default deny incoming              # Standard: eingehend sperren
# ufw default allow outgoing
# ufw allow 22/tcp                       # SSH erlauben (VOR dem Aktivieren!)
# ufw allow from 192.168.1.0/24 to any port 5432 proto tcp
# ufw enable
$ ufw status verbose

# firewall-cmd --state
# firewall-cmd --permanent --add-service=http
# firewall-cmd --permanent --add-port=8080/tcp
# firewall-cmd --reload
$ firewall-cmd --list-all

# nft list ruleset                       # alle aktiven Regeln
```

> **Achtung**
>
> Wer per SSH arbeitet, muss den SSH-Port *erlauben, bevor* die Firewall aktiviert wird – sonst sperrt man sich selbst aus. Auf Cloud-Servern zusätzlich die Regeln der Anbieter-Firewall (Security Groups) beachten.

39.8. Fehlersuche im Netzwerk

Niveau: Einsteiger

Vorgehen von unten nach oben, Schicht für Schicht:

1. **Link**: `ip link` – Schnittstelle UP? Kabel/WLAN? (`ethtool eth0`: `Link detected`)
2. **Adresse**: `ip -br addr` – Adresse vorhanden und im richtigen Netz?
3. **Gateway**: `ip route`, `ping 192.168.1.1`
4. **Internet per IP**: `ping 1.1.1.1`
5. **DNS**: `ping example.org`, `dig example.org`
6. **Dienst**: `curl`, `ss -tlpn`, Firewall prüfen

```
$ ping -c 4 example.org                # Erreichbarkeit, Laufzeit, Paketverlust
$ traceroute example.org               # Weg ueber die Router (ICMP/UDP)
$ mtr -rw -c 10 example.org            # traceroute + ping fortlaufend, als Bericht
$ tracepath example.org
$ ss -tulpn                            # lauschende Ports mit Prozess (als root alle
    Prozesse)
$ ss -tan state established            # bestehende TCP-Verbindungen
$ ss -s                                # Zusammenfassung
$ curl -I https://example.org          # nur HTTP-Header
$ curl -v http://localhost:8080/       # ausfuehrlich: Verbindung, TLS, Header
$ curl -o /dev/null -s -w "%{http_code} %{time_total}\n" https://example.org
$ nc -zv 192.168.1.30 22               # Port offen? (netcat)
$ nmap -p 22,80,443 192.168.1.30       # Portscan -- nur im eigenen Netz!
$ nmap -sn 192.168.1.0/24              # welche Hosts sind erreichbar?
```

39.8.1. Pakete mitschneiden mit tcpdump [x]

```
# tcpdump -i eth0                                  # alles auf eth0
# tcpdump -i eth0 -nn port 53                      # nur DNS, keine Namensaufloesung
# tcpdump -i any -nn host 192.168.1.30 and tcp port 443
# tcpdump -i eth0 -w mitschnitt.pcap -c 1000       # in Datei, 1000 Pakete
$ tcpdump -nn -r mitschnitt.pcap                   # Datei lesen
$ wireshark mitschnitt.pcap                        # grafisch analysieren
```

Hinweis

Mitschneiden fremden Verkehrs und Portscans sind nur in eigenen Netzen oder mit ausdrücklicher Erlaubnis zulässig.

39.8.2. Routing und Weiterleitung aktivieren [x]

```
# sysctl net.ipv4.ip_forward                       # Zustand abfragen
# sysctl -w net.ipv4.ip_forward=1                  # bis zum Neustart aktivieren
# echo "net.ipv4.ip_forward=1" > /etc/sysctl.d/90-forward.conf   # dauerhaft
# sysctl --system                                  # alle sysctl-Dateien laden
```

40. Netzwerkdienste

Inhalt: SSH-Einstieg mit Schlüsseln, `scp` und `rsync` sowie ein Überblick über die wichtigsten Netzwerkdienste.

Stufe: [E] Einsteiger bis [F] Fortgeschritten

Lernziele: Nach diesem Kapitel können Sie

- sich per SSH anzumelden und ein Schlüsselpaar einzurichten,
- Dateien zu übertragen und zu synchronisieren,
- die wichtigsten Netzwerkdienste zu kennen und das Vorgehen beim Inbetriebnehmen eines Dienstes anzuwenden.

Vorkenntnisse: Kapitel 32, 38 und 39.

40.1. SSH – sicher auf entfernten Rechnern arbeiten

Niveau: Einsteiger

SSH (*Secure Shell*, TCP-Port 22) ersetzt `telnet` und `rsh`: Alles ist verschlüsselt, der Server wird per Hostschlüssel authentifiziert, der Benutzer per Passwort oder Schlüssel.

```
$ ssh anna@server01                    # anmelden
$ ssh -p 2222 anna@server01            # anderer Port
$ ssh anna@server01 'uptime; df -h /'  # einzelnen Befehl ausfuehren
$ ssh -i ~/.ssh/id_ed25519 anna@server01
$ ssh -v anna@server01                 # ausfuehrlich bei Problemen (-vv, -vvv)
```

Beim ersten Verbindungsaufbau fragt der Client, ob der Fingerabdruck des Servers stimmt, und speichert ihn in `~/.ssh/known_hosts`. Ändert sich der Schlüssel später, warnt `ssh` (“REMOTE HOST IDENTIFICATION HAS CHANGED”) – entweder wurde der Server neu installiert oder jemand täuscht ihn vor.

40.2. Anmeldung mit Schlüsseln

Niveau: Einsteiger

Schlüsselpaare sind sicherer und bequemer als Passwörter: Der *private* Schlüssel bleibt beim Benutzer, der *öffentliche* liegt auf dem Server.

```
$ ssh-keygen -t ed25519 -C "anna@laptop"    # Schluesselpaar erzeugen (Passphrase setzen!)
$ ls ~/.ssh/
id_ed25519  id_ed25519.pub  known_hosts
$ ssh-copy-id anna@server01                 # oeffentlichen Schluessel auf den Server legen
$ ssh anna@server01                         # jetzt ohne Serverpasswort (nur Passphrase)

$ eval "$(ssh-agent)"                       # Agent: Passphrase nur einmal eingeben
$ ssh-add ~/.ssh/id_ed25519
$ ssh-add -l                                # geladene Schluessel
```

Der öffentliche Schlüssel steht in `~/.ssh/authorized_keys` auf dem Server. Rechte: `~/.ssh` `700`, `authorized_keys` `600` – sonst ignoriert `sshd` die Datei.

> **Achtung**
> Den privaten Schlüssel (ohne `.pub`) **niemals** weitergeben, in Repositories ablegen oder per Mail senden. Eine Passphrase schützt ihn bei Diebstahl.

40.3. Dateien übertragen: scp und rsync

Niveau: Einsteiger

```
$ scp bericht.pdf anna@server01:/home/anna/          # Datei hochladen
$ scp anna@server01:/var/log/syslog .                # herunterladen
$ scp -r projekt/ anna@server01:/srv/                # Verzeichnis rekursiv
$ scp -P 2222 datei anna@server01:

$ rsync -avh projekt/ anna@server01:/srv/projekt/    # synchronisieren
$ rsync -avh --delete --progress quelle/ ziel/       # Ziel spiegeln (loescht Ueberfluessiges
    !)
$ rsync -avhn --delete quelle/ ziel/                 # -n: Testlauf, nichts aendern
$ rsync -avz -e "ssh -p 2222" daten/ server01:/backup/
$ sftp anna@server01                                 # interaktiv: get, put, ls, cd
```

`rsync` überträgt nur geänderte Teile und ist daher für Backups und große Datenmengen deutlich besser geeignet. Ein Schrägstrich am Ende des Quellpfads (`projekt/`) bedeutet "Inhalt des Verzeichnisses", ohne ihn wird das Verzeichnis selbst mitkopiert.

> **Weiterführend**
> Die Client-Konfiguration (`~/.ssh/config`, `ProxyJump`), die Absicherung des Servers (`sshd_config`, `sshd -t`) sowie Tunnel und Portweiterleitung (`ssh -L`, `-R`, `-D`) behandelt Kapitel 42.

40.4. Weitere Dienste im Überblick

Niveau: Fortgeschritten

Dienst	Paket/Port	Zweck
Webserver	`nginx`, `apache2`/`httpd`; 80, 443	Webseiten, Reverse-Proxy, TLS
Zeit (NTP)	`chrony`, `systemd-timesyncd`; 123/udp	Uhrzeit synchronisieren
DNS-Server	`bind9`, `unbound`, `dnsmasq`; 53	Namensauflösung, Caching
Mail	`postfix`, `dovecot`; 25, 587, 993	Versand (SMTP), Abruf (IMAP)
NFS	`nfs-kernel-server`; 2049	Netzwerkdateisystem unter Unix
Samba	`samba`; 445	Windows-Freigaben (SMB/CIFS)
Datenbanken	`mariadb`, `postgresql`; 3306, 5432	Relationale Datenbanken

40.5. Grundschablone: einen Dienst in Betrieb nehmen

Niveau: Fortgeschritten

Für nahezu jeden Netzwerkdienst gilt dieselbe Reihenfolge:

1. **Installieren** (`apt install` bzw. `dnf install`) und **aktivieren** (`systemctl enable --now`),
2. **Port prüfen** (`ss -tlpn`) und Erreichbarkeit testen (`curl`, `nc`),
3. **Firewall öffnen** (Kapitel 43),
4. **Konfiguration testen** und ohne Unterbrechung neu laden (`systemctl reload`),
5. **Logs lesen** (`journalctl -u`, `/var/log/`).

Ein vollständiges Beispiel mit `nginx` steht in Kapitel 46, Freigaben per NFS und Samba in Kapitel 49.

41. Netzwerkwerkzeuge in der Praxis

Inhalt: Diagnose in Schichten: Verbindung, Routing, Namensauflösung, Ports; `ip`, `ss`, `ping`, `traceroute`, `mtr`, `curl`, `nc`, `nmap`, `tcpdump`, `iperf3` und Wireshark.
Stufe: [F] Fortgeschritten bis [X] Experte
Lernziele: Nach diesem Kapitel können Sie

- Netzwerkprobleme systematisch von unten nach oben einzugrenzen,
- die wichtigsten Diagnosewerkzeuge gezielt einzusetzen und ihre Ausgaben zu lesen,
- Verkehr mit `tcpdump` mitzuschneiden und Bandbreite zu messen.

Vorkenntnisse: Kapitel 38 (TCP/IP), Kapitel 39 (Netzwerk) und Kapitel 44 (DNS).

41.1. Vorgehen: Schicht für Schicht

Niveau: Einsteiger

Bei Netzwerkproblemen hilft ein fester Ablauf von unten nach oben. So findet man die Ebene, auf der es hakt:

Frage	Befehl	Aussage
Link da?	`ip -br link`, `ethtool eth0`	Schnittstelle `UP`, Kabel/WLAN, Geschwindigkeit
Adresse?	`ip -br addr`	IP und Netzmaske vorhanden?
Gateway?	`ip route`, `ping GW`	Standardroute vorhanden und erreichbar?
Internet per IP?	`ping 9.9.9.9`	Routing und NAT funktionieren
DNS?	`dig name`, `getent hosts name`	Namensauflösung
Port offen?	`nc -vz host 443`, `ss -tln`	Dienst lauscht und Firewall lässt durch
Anwendung?	`curl -v URL`	Protokoll, Zertifikat, Antwort

41.2. Schnittstellen, Adressen und Routen: ip

Niveau: Einsteiger

```
$ ip -br link                       # Kurzuebersicht Links
$ ip -br addr                       # Adressen
$ ip -s link show eth0              # Statistik: Fehler, Drops
$ ip route                          # Routing-Tabelle
$ ip route get 8.8.8.8              # welchen Weg nimmt ein Paket?
$ ip neigh                          # ARP/NDP-Cache (Nachbarn)
$ ip -6 addr
$ ethtool eth0                      # Link-Geschwindigkeit, Duplex
$ ethtool -S eth0 | grep -i err     # Fehlerzaehler der Karte
```

Fehlerzähler (`errors`, `dropped`, `overruns`) steigender Werte deuten auf Kabel-, Duplex- oder Überlastprobleme hin.

41.3. Erreichbarkeit und Weg: ping, traceroute, mtr

Niveau: Einsteiger

```
$ ping -c 4 192.168.10.1            # vier Pakete
$ ping -c 4 -s 1400 -M do host      # grosse Pakete ohne Fragmentierung (MTU-Test)
$ traceroute -n 9.9.9.9             # Hops anzeigen
$ traceroute -T -p 443 example.org  # TCP statt ICMP/UDP
$ mtr -rwn -c 20 example.org        # Dauer-Traceroute mit Verlust und Latenz
```

In `mtr` sind `Loss%` (Paketverlust) und `Avg`/`Wrst` (Latenz) je Hop wichtig. Verlust nur auf einem Zwischenhop, aber nicht auf den folgenden, ist harmlos (der Router beantwortet Ping nachrangig); Verlust, der sich bis zum Ziel fortsetzt, ist echt. Viele Netze blockieren ICMP – ein fehlender Ping beweist keinen Ausfall.

41.4. Sockets und Ports: ss, lsof, nc

Niveau: Einsteiger

```
$ ss -tlnp                          # TCP-Listener mit Prozessen (root fuer -p)
$ ss -ulnp                          # UDP
$ ss -tn state established          # aktive Verbindungen
$ ss -tn dst 192.168.10.5
$ ss -s                             # Zusammenfassung
$ sudo lsof -i :443                 # wer nutzt Port 443?
$ sudo lsof -iTCP -sTCP:LISTEN -P -n
$ nc -vz 192.168.10.5 22            # Port erreichbar? (TCP)
$ nc -vzu 192.168.10.5 53           # UDP (unzuverlaessig)
$ timeout 3 bash -c '</dev/tcp/192.168.10.5/443' && echo offen || echo zu
```

Ergebnis von `nc -vz`	Bedeutung
`succeeded`	Dienst lauscht und ist erreichbar
`Connection refused`	Host erreichbar, aber kein Dienst auf dem Port (oder `reject`-Regel)
Zeitüberschreitung	Paket wird verworfen: Firewall (`drop`), Routing- oder Netzproblem

`netcat` ist zudem ein vielseitiges Werkzeug für schnelle Tests:

```
# Empfaenger
$ nc -l 9000 > empfangen.dat
# Sender
$ nc 192.168.10.5 9000 < datei.dat          # einfacher Dateitransfer (unverschluesselt!)
$ printf 'GET / HTTP/1.0\r\n\r\n' | nc example.org 80
```

41.5. HTTP testen mit curl

Niveau: Fortgeschritten

```
$ curl -v https://example.org/                  # Details inkl. TLS
$ curl -sS -o /dev/null -w '%{http_code} %{time_total}s\n' https://example.org/
$ curl -I https://example.org/                  # nur Header
$ curl -L https://example.org/                  # Weiterleitungen folgen
$ curl --resolve example.org:443:192.168.10.5 https://example.org/   # DNS ueberschreiben
$ curl -H 'Host: www.example.org' http://192.168.10.5/
$ curl -X POST -d 'name=anna' -H 'Content-Type: application/x-www-form-urlencoded' http://
    localhost/api
$ curl -u anna:geheim -O https://example.org/datei.zip
$ curl --max-time 5 --retry 3 URL
$ curl -w 'dns=%{time_namelookup} connect=%{time_connect} tls=%{time_appconnect} total=%{
    time_total}\n' -o /dev/null -s URL
```

Die letzte Zeile liefert eine Aufschlüsselung der Zeiten: Wo geht die Zeit verloren – im DNS, beim Verbindungsaufbau, im TLS oder in der Anwendung? Mit `wget -c URL` lassen sich große Downloads fortsetzen.

41.6. Ports und Dienste erkunden: nmap

Niveau: Fortgeschritten

```
$ nmap 192.168.10.5                    # haeufige 1000 Ports
$ nmap -p 1-65535 192.168.10.5         # alle Ports
$ nmap -sV -p 22,80,443 192.168.10.5   # Dienst und Version erkennen
$ nmap -sn 192.168.10.0/24             # Host-Erkennung (Ping-Scan)
$ nmap -sU -p 53,123,161 192.168.10.5  # UDP (langsam)
$ nmap -O 192.168.10.5                 # Betriebssystem raten (root)
$ nmap --script ssl-enum-ciphers -p 443 192.168.10.5
```

> **Recht und Ethik**
>
> Port-Scans nur in *eigenen* Netzen oder mit ausdrücklicher schriftlicher Erlaubnis. Unautorisierte Scans können als Angriffsvorbereitung gelten und rechtliche Folgen haben (§ 202c StGB in Deutschland).

41.7. Mitschneiden mit tcpdump und Wireshark

Niveau: Fortgeschritten

`tcpdump` zeigt Pakete auf der Schnittstelle. Mit *Capture-Filtern* (BPF-Syntax) begrenzt man die Ausgabe.

```
$ sudo tcpdump -ni eth0                              # alles (n = keine Namen)
$ sudo tcpdump -ni eth0 host 192.168.10.5
$ sudo tcpdump -ni eth0 port 53
$ sudo tcpdump -ni eth0 'tcp port 443 and host 192.168.10.5'
$ sudo tcpdump -ni eth0 'tcp[tcpflags] & (tcp-syn) != 0'   # nur SYN-Pakete
$ sudo tcpdump -ni any icmp
$ sudo tcpdump -ni eth0 -c 100 -w mitschnitt.pcap    # in Datei schreiben (100 Pakete)
$ sudo tcpdump -nr mitschnitt.pcap                   # Datei lesen
$ sudo tcpdump -ni eth0 -A port 80                   # Inhalt als ASCII
```

Die `pcap`-Datei lässt sich in **Wireshark** grafisch analysieren (Filter wie `tcp.analysis.retransmission`, "Follow TCP Stream"). Für Textkonsolen: `tshark`. Typische Beobachtungen:

- Viele `SYN` ohne `SYN-ACK`: Ziel antwortet nicht (Firewall/Dienst aus).

- `RST` als Antwort auf `SYN`: Port geschlossen.
- Wiederholte Pakete (*Retransmissions*): Verluste auf dem Weg.
- DNS-Anfragen ohne Antwort: Resolver nicht erreichbar.

Mitschnitte enthalten oft vertrauliche Daten (Passwörter in Klartextprotokollen, Cookies). Nur mit Berechtigung erstellen und sorgfältig aufbewahren.

41.8. Bandbreite und Qualität messen: iperf3

Niveau: Experte

```
# Server
$ iperf3 -s
# Client
$ iperf3 -c 192.168.10.5                # TCP-Durchsatz
$ iperf3 -c 192.168.10.5 -R             # umgekehrte Richtung
$ iperf3 -c 192.168.10.5 -P 4 -t 30     # 4 parallele Stroeme, 30 s
$ iperf3 -c 192.168.10.5 -u -b 100M     # UDP mit 100 MBit/s: Verlust und Jitter
```

- Ergebnisse vergleichen: erwartet 940 MBit/s auf Gigabit-Ethernet. Deutlich weniger deutet auf Duplex-Fehler, schlechte Kabel, Switch-Engpässe oder CPU-Limit hin.
- **Weitere Werkzeuge**: `iftop` und `nethogs` (Verkehr pro Verbindung bzw. Prozess), `vnstat` (Verlauf), `arping` und `arp-scan` (Layer 2), `ethtool`, `bmon`.
- **Traffic Control**: Mit `tc qdisc add dev eth0 root netem delay 100ms loss 1%` lassen sich schlechte Verbindungen testweise simulieren (danach `tc qdisc del dev eth0 root`).

Übungen und Kontrollfragen

1. In welcher Reihenfolge prüfen Sie ein Netzwerkproblem?

   Lösung: Von unten nach oben: Link, IP-Adresse, Gateway, Internet per IP, DNS, Port/Dienst, Anwendung.

2. Was sagt `Connection refused` im Unterschied zu einer Zeitüberschreitung?

   Lösung: *Refused*: Der Host antwortet, aber auf dem Port lauscht nichts; eine Zeitüberschreitung deutet darauf hin, dass Pakete verworfen werden (Firewall/Routing).

3. Wie finden Sie heraus, welcher Prozess Port 8080 belegt?

   Lösung: `sudo ss -tlnp | grep :8080` oder `sudo lsof -i :8080`.

4. Wie messen Sie mit `curl` getrennt die Zeit für DNS, Verbindung und TLS?

   Lösung: Mit `-w` und den Variablen `time_namelookup`, `time_connect`, `time_appconnect`, `time_total`.

5. Wie schneiden Sie nur den DNS-Verkehr in eine Datei mit?

   Lösung: `sudo tcpdump -ni eth0 port 53 -w dns.pcap`.

6. Wofür eignet sich `iperf3`?

   Lösung: Zur Messung des Netzwerkdurchsatzes (TCP/UDP) zwischen zwei Rechnern, auch von Verlust und Jitter.

42. SSH im Detail

Inhalt: Funktionsweise von SSH, Schlüsselpaare und Agent, Konfiguration von Client und Server, Härtung, Portweiterleitung und Tunnel, Jump-Hosts, `scp`, `sftp` und `rsync`, Zertifikate und Mehrfaktor-Anmeldung.

Stufe: [F] Fortgeschritten bis [X] Experte

Lernziele: Nach diesem Kapitel können Sie

- sich mit Schlüsseln anzumelden und den Client über `~/.ssh/config` komfortabel zu konfigurieren,
- einen SSH-Server zu härten und Tunnel (Port-Forwarding, Jump-Host) einzusetzen,
- Dateien sicher mit `scp`, `sftp` und `rsync` zu übertragen.

Vorkenntnisse: Kapitel 20 (Benutzer), Kapitel 22 (Rechte) und Kapitel 40 (Netzdienste, Grundlagen zu SSH).

42.1. Grundlagen

Niveau: Einsteiger

SSH (*Secure Shell*, TCP-Port 22) ermöglicht verschlüsselte Anmeldung, Befehlsausführung, Dateiübertragung und Tunnel. Es ersetzt `telnet`, `rsh` und `ftp`. Unter Linux ist **OpenSSH** Standard: Client `ssh`, Server `sshd`. Die grundlegenden Aufrufe (Anmelden, Befehl ausführen, anderer Port) stehen in Kapitel 40; hier folgt die Vertiefung.

```
$ ssh -v anna@server                          # ausfuehrliche Fehlersuche (-vv, -vvv)
$ ssh -i ~/.ssh/id_ed25519 anna@server        # bestimmten Schluessel verwenden
```

Beim ersten Verbindungsaufbau fragt SSH nach dem **Fingerabdruck des Server-Hostschlüssels**; nach Bestätigung wird er in `~/.ssh/known_hosts` gespeichert. Ändert er sich später, warnt SSH – das kann ein Hinweis auf einen Man-in-the-Middle-Angriff sein, aber auch auf eine Neuinstallation des Servers. Vor dem Entfernen (`ssh-keygen -R server`) Ursache klären.

42.2. Schlüsselbasierte Anmeldung

Niveau: Einsteiger

Statt Passwörtern nutzt man ein **Schlüsselpaar**: Der private Schlüssel bleibt auf dem Client, der öffentliche wird auf dem Server hinterlegt. Erzeugen (`ssh-keygen`) und Übertragen (`ssh-copy-id`) zeigt Kapitel 40; hier die Hintergründe:

```
$ ls -l ~/.ssh/
-rw------- 1 anna anna  411 id_ed25519          # privat (nie weitergeben)
-rw-r--r-- 1 anna anna   96 id_ed25519.pub      # oeffentlich
$ cat ~/.ssh/authorized_keys                     # auf dem Server: erlaubte Schluessel
```

- **Algorithmus**: `ed25519` ist modern, schnell und kurz. Alternativ `rsa` mit mindestens 3072 Bit.
- **Rechte**: `~/.ssh` muss `700`, `authorized_keys` `600` haben, sonst ignoriert `sshd` sie (`StrictModes`).
- **Passphrase**: schützt den privaten Schlüssel bei Diebstahl der Datei. Der **ssh-agent** merkt sie sich für die Sitzung.

```
$ eval "$(ssh-agent -s)"                          # Agent starten (Desktop-Sitzungen haben
    meist einen)
$ ssh-add ~/.ssh/id_ed25519                        # Schluessel laden
$ ssh-add -l                                       # geladene Schluessel
$ ssh -A anna@jump                                 # Agent-Forwarding (nur vertrauenswuerdigen
    Hosts!)
```

42.3. Der Client: `~/.ssh/config`

Niveau: Fortgeschritten

In der Konfigurationsdatei lassen sich Verbindungen mit Kurznamen und Optionen hinterlegen:

```
# ~/.ssh/config
Host *
    ServerAliveInterval 60
    ServerAliveCountMax 3
    AddKeysToAgent yes

Host web1
    HostName web1.example.lan
    User anna
    Port 22
    IdentityFile ~/.ssh/id_ed25519

Host intern-*
    ProxyJump bastion
    User admin

Host bastion
    HostName bastion.example.org
    User anna
    ForwardAgent no

Host nas
    HostName 192.168.10.5
    User root
    ControlMaster auto
    ControlPath ~/.ssh/cm-%r@%h:%p
    ControlPersist 10m
```

```
$ ssh web1                                         # statt ssh -i ... anna@web1.example.lan
$ ssh intern-db1                                   # geht automatisch ueber den Bastion-Host
$ ssh -J bastion anna@10.0.0.20                    # ad-hoc Jump-Host
$ ssh -G web1 | head                               # wirksame Optionen anzeigen
```

`ControlMaster` (Verbindungs-Multiplexing) nutzt eine bestehende Verbindung für weitere Sitzungen – sofortiger Aufbau und weniger Anmeldungen.

42.4. Dateien übertragen: sftp und sshfs

Niveau: Fortgeschritten

Die Grundlagen zu `scp` und `rsync` stehen in Kapitel 40, Sicherungsstrategien in Kapitel 53. Ergänzend:

```
$ sftp anna@server                                  # interaktiv: get, put, ls, cd, lcd,
    mkdir
$ scp -P 2222 datei anna@server:                    # Port (grosses P!)
$ rsync -avz --delete -e 'ssh -p 2222' quelle/ anna@server:/ziel/
$ sshfs anna@server:/srv /mnt/remote                # Verzeichnis einhaengen (FUSE)
$ fusermount -u /mnt/remote
```

Bei `rsync` macht der Schrägstrich am Ende des Quellpfades den Unterschied: `quelle/` kopiert den *Inhalt*, `quelle` das Verzeichnis selbst. Details und Sicherungsstrategien: Kapitel 53.

42.5. Der Server: sshd und Härtung

Niveau: Fortgeschritten

Die Konfiguration liegt in `/etc/ssh/sshd_config` (und `sshd_config.d/`).

```
# /etc/ssh/sshd_config.d/10-haertung.conf
PermitRootLogin no
PasswordAuthentication no
KbdInteractiveAuthentication no
PubkeyAuthentication yes
AllowUsers anna bob              # oder AllowGroups sshusers
MaxAuthTries 3
LoginGraceTime 30
ClientAliveInterval 300
ClientAliveCountMax 2
X11Forwarding no
AllowTcpForwarding no            # bei Bedarf pro Benutzer erlauben (Match)
Banner /etc/issue.net

Match User backup
    ForceCommand /usr/local/bin/backup-wrapper
    AllowTcpForwarding no

$ sudo sshd -t                                      # Syntax pruefen (VOR dem Neuladen!)
$ sudo systemctl reload ssh                         # Dienst heisst ssh (Debian) oder sshd (RHEL
    )
$ sudo sshd -T | grep -iE 'permitroot|passwordauth'   # wirksame Werte
```

> **Nicht aussperren**
>
> Vor Änderungen an der SSH-Konfiguration: eine zweite Sitzung offen lassen, `sshd -t` ausführen, nach dem Neuladen in einem *neuen* Terminal testen, erst dann die alte Sitzung schließen. Passwort-Anmeldung erst abschalten, wenn die Schlüsselanmeldung nachweislich funktioniert.

- **Port ändern** ist bestenfalls kosmetisch (weniger Log-Rauschen), kein Sicherheitsgewinn.
- **fail2ban** oder Rate-Limits in der Firewall (Kapitel 43) bremsen Brute-Force-Angriffe; Zugriff auf Admin-Netze beschränken.
- **Protokoll**: `journalctl -u ssh` (Debian) bzw. `sshd`; Anmeldeversuche: `lastb`, `last`.
- **Moderne Kryptografie**: Veraltete Algorithmen (SSHv1, CBC, `ssh-rsa` mit SHA-1) abschalten; `ssh-audit` prüft die Konfiguration.

42.6. Tunnel und Portweiterleitung

Niveau: Fortgeschritten

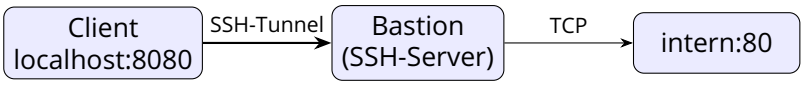
```
# Lokale Weiterleitung: lokaler Port -> Ziel, erreichbar vom Server aus
$ ssh -L 8080:intern.example.lan:80 anna@bastion
#  -> http://localhost:8080 zeigt nun intern.example.lan:80

# Datenbank im Remote-Netz erreichen
$ ssh -N -L 5433:db.intern:5432 anna@bastion          # -N: nur Tunnel, keine Shell
$ psql -h localhost -p 5433 -U shopuser shop

# Entfernte Weiterleitung: Port auf dem Server -> lokaler Dienst
$ ssh -R 9000:localhost:3000 anna@server              # Entwicklungsserver zeigen

# Dynamischer SOCKS-Proxy
$ ssh -D 1080 -N anna@bastion
$ curl --socks5-hostname localhost:1080 http://intern.example.lan/

# Dauerhafter Tunnel im Hintergrund
$ ssh -fN -L 8080:intern:80 anna@bastion
$ ss -tlnp | grep 8080
```



42.7. Zertifikate, Mehrfaktor und Automatisierung

Niveau: Experte

- **SSH-Zertifikate**: Statt Hunderte öffentliche Schlüssel zu verteilen, signiert eine interne CA die Benutzerschlüssel (befristet): `ssh-keygen -s ca_key -I anna -n anna -V +1d id_ed25519.pub`; die Server vertrauen der CA (`TrustedUserCAKeys`).
- **Mehrfaktor**: PAM-Module (Google Authenticator, `pam_u2f`) oder FIDO2-Schlüssel (`ssh-keygen -t ed25519-sk`) für Hardware-gebundene Schlüssel (Kapitel 21).
- **Automatisierung**: Schlüssel ohne Passphrase nur für Skripte, dann mit `command="..."` und `from="..."` in `authorized_keys` auf einen Zweck und Ursprung beschränken; Konfigurationswerkzeuge wie Ansible nutzen SSH (Kapitel 56).
- **Hostschlüssel überprüfen**: `ssh-keyscan server` und `ssh-keygen -lf` liefern Fingerabdrücke zum Vergleich; per DNS (SSHFP) lassen sie sich veröffentlichen.
- **Fehlersuche**: `ssh -vvv`, auf dem Server `journalctl -u ssh -f` bzw. `sshd -ddd` auf einem Testport.

```
# authorized_keys mit Einschraenkungen
from="192.168.10.0/24",command="/usr/local/bin/sync.sh",no-pty,no-port-forwarding ssh-ed25519
     AAAA... backup@host
$ sudo sshd -ddd -p 2222                        # Test-Instanz im Vordergrund mit Debug-
    Ausgabe
$ ssh-keygen -lf ~/.ssh/id_ed25519.pub          # Fingerabdruck anzeigen
$ ssh-keyscan -t ed25519 server                 # Hostschluessel abfragen
```

Übungen und Kontrollfragen

1. Welche Dateien gehören zu einem SSH-Schlüsselpaar und welche darf niemand außer Ihnen besitzen?

   Lösung: `id_ed25519` (privat, Modus 600) und `id_ed25519.pub` (öffentlich). Der private Schlüssel darf nie weitergegeben werden.

2. Wie richten Sie die Anmeldung ohne Passwort auf einem Server ein?

   Lösung: `ssh-keygen -t ed25519`, dann `ssh-copy-id anna@server`; der öffentliche Schlüssel landet in `~/.ssh/authorized_keys`.

3. Wie erreichen Sie einen internen Webserver, der nur über einen Bastion-Host zugänglich ist?

   Lösung: `ssh -L 8080:intern:80 anna@bastion`, danach `http://localhost:8080`; alternativ `ProxyJump` in der `config`.

4. Welche drei Einstellungen in `sshd_config` bringen den größten Sicherheitsgewinn?

   Lösung: `PermitRootLogin no`, `PasswordAuthentication no` (mit Schlüsseln) und Beschränkung der Benutzer (`AllowUsers`/`AllowGroups`).

5. Was müssen Sie vor dem Neuladen von `sshd` tun, um sich nicht auszusperren?

   Lösung: Syntax mit `sshd -t` prüfen, eine bestehende Sitzung offen lassen und die neue Konfiguration in einer zweiten Sitzung testen.

6. Worin unterscheiden sich `ssh -L` und `ssh -R`?

   Lösung: `-L`: lokaler Port wird zu einem Ziel hinter dem Server weitergeleitet; `-R`: Port auf dem Server wird zu einem lokalen Dienst zurückgeleitet.

43. Firewall: nftables, iptables, ufw und firewalld

Inhalt: Paketfilter im Kernel, Tabellen, Ketten und Regeln, Zustandsverfolgung, NAT und Portweiterleitung, nftables, die Frontends ufw und firewalld, Protokollierung, Fehlersuche und Härtung.
Stufe: [F] Fortgeschritten bis [X] Experte
Lernziele: Nach diesem Kapitel können Sie

- den Weg eines Pakets durch den Linux-Paketfilter zu beschreiben,
- mit nftables einen restriktiven Regelsatz für einen Server zu schreiben,
- NAT und Portweiterleitung einzurichten sowie ufw und firewalld sinnvoll einzusetzen.

Vorkenntnisse: Kapitel 38 und Kapitel 39 (TCP/IP, Netzkonfiguration) sowie Kapitel 52 (Sicherheit).

43.1. Wozu eine Firewall?

Niveau: Einsteiger

Eine Firewall entscheidet anhand von Regeln, welche Netzwerkpakete einen Rechner erreichen, verlassen oder durchqueren dürfen. Unter Linux arbeitet der Paketfilter **im Kernel** (Netfilter); die Werkzeuge im Benutzerbereich schreiben nur die Regeln.

- **Host-Firewall**: schützt den einzelnen Rechner; erlaubt nur die Dienste, die er anbietet.
- **Netz-Firewall / Router**: filtert und übersetzt Verkehr zwischen Netzen (Forwarding, NAT).
- Eine Firewall ersetzt *keine* Sicherheitsupdates und keine sichere Konfiguration der Dienste – sie ist eine zusätzliche Schicht (Defense in Depth).

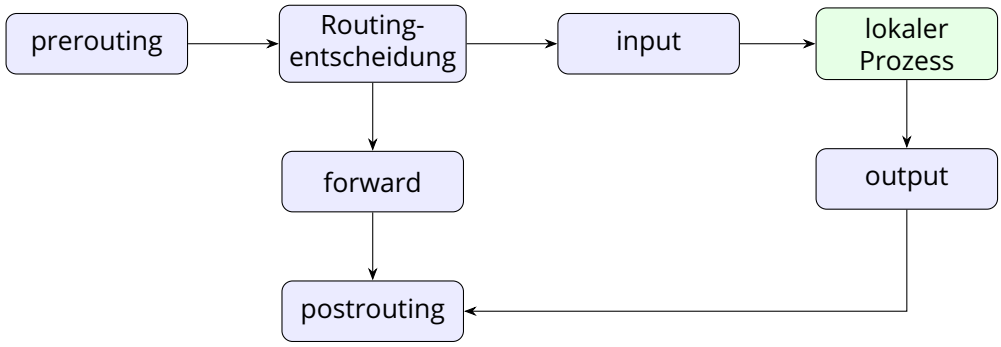
Grundsatz

Alles verbieten, was nicht ausdrücklich erlaubt ist (*default deny*). Wer nur bekannte Schlechtes sperrt, übersieht zwangsläufig etwas. Vor jeder Änderung per SSH: einen zweiten Zugang offen halten oder die Regeln mit einem Zeitlimit zurücknehmen.

43.2. Der Weg eines Pakets durch Netfilter

Niveau: Fortgeschritten

Netfilter bietet an mehreren Stellen des Netzwerk-Stapels *Hooks*. Regeln werden in **Ketten** (*chains*) abgelegt, Ketten in **Tabellen** (*tables*).



- **input**: Pakete *an* den Rechner selbst.
- **output**: Pakete, die der Rechner selbst erzeugt.

- **forward**: Pakete, die der Rechner nur weiterreicht (Router; benötigt `net.ipv4.ip_forward=1`).
- **prerouting** / **postrouting**: vor bzw. nach der Routing-Entscheidung; hier sitzt NAT.

Zustandsverfolgung (*connection tracking*): Der Kernel merkt sich Verbindungen. Pakete sind `new`, `established`, `related` (gehört zu einer bestehenden Verbindung, etwa ICMP-Fehler) oder `invalid`. Dadurch genügt es, neue Verbindungen zu erlauben; die Antworten folgen automatisch.

43.3. nftables

Niveau: Fortgeschritten

nftables ist der Nachfolger von iptables, ip6tables, arptables und ebtables. Ein Werkzeug (`nft`), eine Syntax, eine Tabelle kann IPv4 und IPv6 gemeinsam abdecken (Familie `inet`).

```
$ sudo apt install nftables
$ sudo systemctl enable --now nftables
$ sudo nft list ruleset                 # alle Regeln anzeigen
$ sudo nft flush ruleset                # ALLE Regeln loeschen (Vorsicht!)
$ sudo nft -c -f /etc/nftables.conf     # Syntax pruefen, nichts laden
$ sudo nft -f /etc/nftables.conf        # Regeln laden
```

Ein vollständiger, restriktiver Regelsatz für einen Server in `/etc/nftables.conf`:

```
#!/usr/sbin/nft -f
flush ruleset

table inet filter {
    set erlaubte_admins {
        type ipv4_addr
        flags interval
        elements = { 192.168.10.0/24 }
    }

    chain input {
        type filter hook input priority 0; policy drop;

        ct state established,related accept
        ct state invalid drop
        iif "lo" accept

        # ICMP / ICMPv6 (Ping, Pfad-MTU, IPv6-Nachbarsuche)
        ip protocol icmp icmp type { echo-request, destination-unreachable, time-exceeded }
    accept
        ip6 nexthdr icmpv6 accept

        # SSH nur aus dem Admin-Netz, mit Begrenzung
        tcp dport 22 ip saddr @erlaubte_admins ct state new limit rate 10/minute accept

        # Webserver oeffentlich
        tcp dport { 80, 443 } accept

        # Alles andere protokollieren und verwerfen
        limit rate 5/minute log prefix "nft-drop: " counter drop
    }

    chain forward {
        type filter hook forward priority 0; policy drop;
    }

    chain output {
        type filter hook output priority 0; policy accept;
    }
}
```

Erklärung: `policy drop` verwirft alles, was keine Regel erlaubt. Die erste Regel lässt Antworten auf bestehende Verbindungen zu, die Loopback-Schnittstelle bleibt frei. Ein `set` fasst Adressen zusammen und macht Regeln lesbar. `counter` zählt Pakete und Bytes.

Ausdruck	**Bedeutung**
`tcp dport 22`	Ziel-Port 22 (TCP)
`ip saddr 10.0.0.0/8`	Quelladresse aus dem Netz
`iif "eth0"`, `oif "eth1"`	Eingangs- bzw. Ausgangsschnittstelle
`ct state new`	Neue Verbindung
`limit rate 10/minute`	Rate begrenzen (Schutz vor Brute-Force/Flut)
`accept`, `drop`, `reject`	Erlauben, stillschweigend verwerfen, mit Fehlermeldung abweisen
`log prefix "x: "`	Ins Kernel-Log schreiben (`journalctl -k`)
`counter`	Zähler

drop oder **reject**? `drop` verrät dem Angreifer nichts, lässt aber legitime Clients auf Zeitüberschreitungen warten. `reject` antwortet sofort mit einem Fehler – für interne Netze meist angenehmer.

```
# Regeln im laufenden Betrieb aendern
$ sudo nft add rule inet filter input tcp dport 8080 accept
$ sudo nft -a list chain inet filter input     # mit Handles (Nummern)
$ sudo nft delete rule inet filter input handle 17
$ sudo nft list ruleset | sudo tee /etc/nftables.conf   # dauerhaft sichern
```

43.4. NAT und Portweiterleitung

Niveau: Fortgeschritten

NAT (*Network Address Translation*) ersetzt Adressen oder Ports in Paketen. Zwei Arten sind wichtig:

- **Masquerading / SNAT**: Interne Rechner (private Adressen) gehen über die öffentliche Adresse des Routers ins Internet.
- **DNAT / Portweiterleitung**: Verbindungen aus dem Internet an einem Port werden an einen internen Rechner umgeleitet.

```
# Router: Weiterleitung aktivieren
$ echo 'net.ipv4.ip_forward=1' | sudo tee /etc/sysctl.d/90-forward.conf
$ sudo sysctl --system

table ip nat {
    chain prerouting {
        type nat hook prerouting priority -100;
        iif "eth0" tcp dport 8080 dnat to 192.168.10.20:80
    }
    chain postrouting {
        type nat hook postrouting priority 100;
        oif "eth0" masquerade
    }
}

table inet filter {
    chain forward {
        type filter hook forward priority 0; policy drop;
        ct state established,related accept
        iif "eth1" oif "eth0" accept                    # LAN -> Internet
        iif "eth0" ip daddr 192.168.10.20 tcp dport 80 ct status dnat accept
    }
}
```

Wichtig: Auch für die Portweiterleitung muss die `forward`-Kette den Verkehr erlauben; NAT allein öffnet nichts.

43.5. iptables (Altsystem)

Niveau: Fortgeschritten

Auf älteren Systemen und in vielen Anleitungen begegnet `iptables`. Moderne Distributionen stellen den Befehl als Übersetzer bereit (`iptables-nft`), der Regeln nach nftables umsetzt. Wichtige Entsprechungen:

```
$ sudo iptables -L -n -v --line-numbers          # Regeln anzeigen
$ sudo iptables -P INPUT DROP                    # Standardrichtlinie
$ sudo iptables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
$ sudo iptables -A INPUT -i lo -j ACCEPT
$ sudo iptables -A INPUT -p tcp --dport 22 -j ACCEPT
$ sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
$ sudo iptables-save > /etc/iptables/rules.v4    # sichern
$ sudo iptables-restore < /etc/iptables/rules.v4
```

Bei Neueinrichtungen sollte nftables bevorzugt werden. Unter Debian/Ubuntu sichert das Paket `iptables-persistent`, unter RHEL der Dienst `iptables-services` die Regeln.

43.6. Frontends: ufw und firewalld

Niveau: Fortgeschritten

Wer keine Regeln von Hand schreiben möchte, nutzt ein Frontend.
ufw (*Uncomplicated Firewall*, Ubuntu/Debian):

```
$ sudo ufw default deny incoming
$ sudo ufw default allow outgoing
$ sudo ufw allow 22/tcp                        # SSH
$ sudo ufw allow from 192.168.10.0/24 to any port 3306 proto tcp
$ sudo ufw limit ssh                           # Brute-Force bremsen
$ sudo ufw enable
$ sudo ufw status verbose
$ sudo ufw status numbered
$ sudo ufw delete 3
```

firewalld (RHEL, Fedora, SUSE) arbeitet mit **Zonen**: Jede Schnittstelle gehört zu einer Zone mit einem Vertrauensniveau (`public`, `internal`, `dmz`, `trusted`, `drop`). Dienste werden pro Zone freigegeben.

```
$ sudo systemctl enable --now firewalld
$ sudo firewall-cmd --state
$ sudo firewall-cmd --get-active-zones
$ sudo firewall-cmd --zone=public --list-all
$ sudo firewall-cmd --zone=public --add-service=https             # nur laufend
$ sudo firewall-cmd --permanent --zone=public --add-service=https # dauerhaft
$ sudo firewall-cmd --permanent --add-port=8080/tcp
$ sudo firewall-cmd --permanent --add-rich-rule='rule family="ipv4" source address="10.0.0.5"
    port port="22" protocol="tcp" accept'
$ sudo firewall-cmd --reload
```

Firewalld unterscheidet die **Runtime-** von der **permanenten** Konfiguration: Ohne `--permanent` gilt eine Änderung nur bis zum nächsten Neuladen. Praktisch zum Ausprobieren; mit `firewall-cmd --runtime-to-permanent` übernimmt man sie.

> **Nicht mischen**
>
> Nur *eine* Verwaltungsebene verwenden: entweder eigene nftables-Dateien, ufw oder firewalld. Mehrere Werkzeuge überschreiben sich gegenseitig oder erzeugen schwer durchschaubare Regel-

werke. Auch Docker legt eigene Regeln an (Kapitel 34) – veröffentlichte Container-Ports umgehen oft die Host-Firewall.

43.7. Protokollierung, Tests und Härtung

Niveau: Experte

```
$ sudo journalctl -k | grep nft-drop            # verworfene Pakete
$ sudo nft list ruleset | grep -A2 counter      # Zaehlerstaende
$ sudo conntrack -L | head                       # Verbindungstabelle
$ sudo conntrack -S                              # Statistik, Ueberlauf?
$ nmap -sS -p- 192.168.10.5                      # Port-Scan von aussen (nur eigene Systeme!)
$ ss -tulpen                                     # was lauscht lokal?
```

- **Zuerst prüfen, was lauscht**: Dienste, die nicht laufen, brauchen keine Firewallregel. Mit `ss -tulpen` sieht man offene Sockets.
- **Egress-Filter**: Auch ausgehenden Verkehr einschränken, damit kompromittierte Server nicht frei nach außen verbinden.
- **Rate-Limits** gegen Brute-Force, ergänzt um `fail2ban` (Kapitel 52), das bei wiederholten Fehlversuchen automatisch Regeln setzt.
- **Anti-Spoofing**: `net.ipv4.conf.all.rp_filter=1`; Pakete mit unmöglicher Quelladresse werden verworfen.
- **Zeitlimit beim Testen**: `echo 'nft flush ruleset' | at now + 5 min` als Rettungsanker, falls man sich aussperrt.
- **Änderungen versionieren** (Git, Kapitel 16) und das Regelwerk dokumentieren.

Typischer Fehler

Die Regel für `established,related` fehlt oder steht hinter einem `drop`: Dann funktionieren ausgehende Verbindungen nur scheinbar – die Antworten kommen nie an. Regeln werden von oben nach unten geprüft; die erste passende Regel mit Endaktion (`accept`, `drop`) entscheidet.

Übungen und Kontrollfragen

1. Was bedeutet `policy drop` an einer Kette?

   Lösung: Pakete, auf die keine Regel zutrifft, werden verworfen (Default deny).

2. Warum genügt es meist, neue Verbindungen zu erlauben?

   Lösung: Die Zustandsverfolgung lässt Antworten (`established,related`) automatisch durch; man muss nicht für jede Richtung eine Regel schreiben.

3. Welche Ketten durchläuft ein Paket, das ein Router von LAN nach Internet weiterreicht?

   Lösung: `prerouting`, `forward`, `postrouting`. Die Ketten `input` und `output` betreffen nur den Router selbst.

4. Schreiben Sie eine nftables-Regel, die SSH nur aus 10.0.0.0/24 erlaubt.

   Lösung: `tcp dport 22 ip saddr 10.0.0.0/24 accept` in der `input`-Kette.

5. Worin unterscheidet sich `drop` von `reject`?

   Lösung: `drop` verwirft still (Client wartet bis zum Timeout); `reject` sendet sofort eine Ablehnung.

6. Warum reicht NAT allein nicht, um einen internen Webserver zu veröffentlichen?

   Lösung: Zusätzlich muss die `forward`-Kette den DNAT-Verkehr erlauben; außerdem muss `ip_forward` aktiv sein.

44. DNS-Server: BIND, Unbound und dnsmasq

Inhalt: Aufbau des DNS, Resolver und autoritative Server, Zonen und Eintragstypen, Betrieb von BIND 9, Unbound als Cache, dnsmasq für kleine Netze, Reverse-Zonen, DNSSEC-Grundlagen, Fehlersuche mit `dig`.

Stufe: [F] Fortgeschritten bis [X] Experte

Lernziele: Nach diesem Kapitel können Sie

- die Abfragekette im DNS und die wichtigsten Eintragstypen zu erklären,
- eine einfache Zone mit BIND bereitzustellen und einen Cache-Resolver einzurichten,
- DNS-Probleme systematisch mit `dig` einzugrenzen.

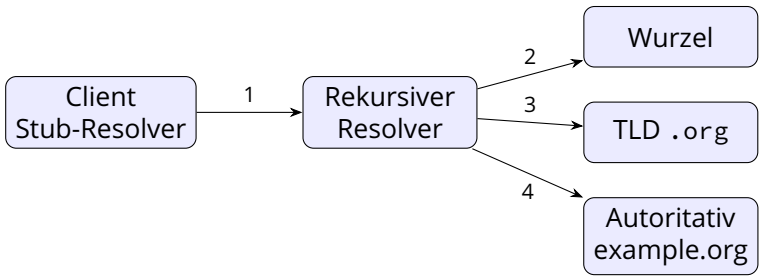
Vorkenntnisse: Kapitel 38 (TCP/IP und DNS-Grundlagen) und Kapitel 39 (Netzwerk konfigurieren).

44.1. Wie DNS funktioniert

Niveau: Einsteiger

Das **Domain Name System** übersetzt Namen in Adressen und umgekehrt. Es ist eine weltweit verteilte, hierarchische Datenbank: Die Wurzel (`.`) kennt die Server der Top-Level-Domains (`de`, `org`), diese die Server der Domains (`example.org`), und so weiter.

- **Stub-Resolver**: die Bibliothek im Client (`getaddrinfo`); fragt einen konfigurierten Resolver.
- **Rekursiver Resolver** (Cache): beantwortet Anfragen für Clients, indem er die Hierarchie von der Wurzel an durchläuft, und speichert Antworten für die *TTL*-Dauer. Beispiele: Unbound, BIND, der Router.
- **Autoritativer Server**: kennt die Zone einer Domain und liefert verbindliche Antworten. Beispiele: BIND, NSD, PowerDNS, Knot.



Wichtige Eintragstypen (*Resource Records*):

Typ	Bedeutung
`A` / `AAAA`	Name → IPv4- bzw. IPv6-Adresse
`CNAME`	Alias auf einen anderen Namen (nicht an der Zonenspitze)
`MX`	Mailserver der Domain mit Priorität (Kapitel 50)
`NS`	Zuständige Nameserver der Zone
`SOA`	Zonenkopf: Haupt-Nameserver, Seriennummer, Timer
`PTR`	Adresse → Name (Reverse-Auflösung)
`TXT`	Freitext, genutzt für SPF, DKIM, Domain-Bestätigungen
`SRV`	Dienst mit Host und Port
`CAA`	Welche Zertifizierungsstellen Zertifikate ausstellen dürfen

44.2. Abfragen mit dig, host und resolvectl

Niveau: Einsteiger

```
$ dig example.org                          # A-Eintrag, mit Details
$ dig +short example.org AAAA
$ dig example.org MX
$ dig @9.9.9.9 example.org                 # bestimmten Server fragen
$ dig +trace example.org                   # Aufloesung ab der Wurzel
$ dig -x 93.184.216.34                     # Reverse-Abfrage
$ dig example.org NS +short
$ dig example.org SOA +multiline
$ host -t txt example.org
$ resolvectl query example.org             # systemd-resolved
$ resolvectl status                        # welche Server, welche Domains
$ getent hosts example.org                 # so loest die libc auf (inkl. /etc/hosts)
```

In der Ausgabe von `dig` sind wichtig: `status` (`NOERROR`, `NXDOMAIN` = Name existiert nicht, `SERVFAIL` = Serverfehler), das Flag `aa` (autoritative Antwort), die TTL sowie die `Query time`.

Reihenfolge der Namensauflösung

Welche Quellen die Bibliothek verwendet, steht in `/etc/nsswitch.conf` (Zeile `hosts:`), typischerweise `files dns`: zuerst `/etc/hosts`, dann DNS. `dig` umgeht `/etc/hosts`, `getent hosts` nicht – ein häufiger Grund für scheinbare Widersprüche.

44.3. Unbound als Cache-Resolver

Niveau: Fortgeschritten

Unbound ist ein schlanker, sicherer rekursiver Resolver mit DNSSEC-Validierung – ideal als lokaler Cache für ein Netz.

```
$ sudo apt install unbound
$ sudo unbound-checkconf
$ sudo systemctl enable --now unbound

# /etc/unbound/unbound.conf.d/lokal.conf
server:
    interface: 0.0.0.0
    access-control: 127.0.0.0/8 allow
    access-control: 192.168.10.0/24 allow
    hide-identity: yes
    hide-version: yes
    prefetch: yes
    cache-min-ttl: 60
    local-zone: "lan." static
    local-data: "nas.lan. IN A 192.168.10.5"
    local-data-ptr: "192.168.10.5 nas.lan."

forward-zone:                      # optional: alles an einen Upstream weiterreichen
    name: "."
    forward-tls-upstream: yes
    forward-addr: 9.9.9.9@853#dns.quad9.net
```

```
$ sudo unbound-control stats_noreset | grep -E 'total.num.(queries|cachehits)'
$ sudo unbound-control flush example.org
$ dig @127.0.0.1 nas.lan
```

Wichtig: Ein rekursiver Resolver, der aus dem Internet erreichbar ist ("offener Resolver"), kann für DNS-Verstärkungsangriffe missbraucht werden. Deshalb nur eigene Netze zulassen (`access-control`) und die Firewall (Kapitel 43) entsprechend einstellen.

44.4. dnsmasq für kleine Netze

Niveau: Fortgeschritten

`dnsmasq` vereint DNS-Weiterleitung, Cache und DHCP-Server in einem kleinen Programm. Es ist die Standardlösung vieler Heimrouter und für Labor- und VM-Netze (libvirt nutzt es) gut geeignet.

```
# /etc/dnsmasq.d/lab.conf
interface=eth1
domain=lab.local
local=/lab.local/
expand-hosts
server=9.9.9.9
address=/test.lab.local/192.168.50.10     # eigener Eintrag
dhcp-range=192.168.50.100,192.168.50.200,12h
dhcp-host=aa:bb:cc:dd:ee:ff,drucker,192.168.50.20
$ sudo dnsmasq --test
$ sudo systemctl restart dnsmasq
```

Zusätzliche Hosts nimmt dnsmasq aus `/etc/hosts` auf – eine einfache Möglichkeit, interne Namen zu pflegen. Mehr zum DHCP-Teil in Kapitel 45.

44.5. Autoritativer Server mit BIND 9

Niveau: Experte

`BIND` (`named`) ist der Klassiker unter den DNS-Servern und kann gleichzeitig Zonen bedienen und rekursiv auflösen (beides auf demselben Server zu mischen wird aber nicht empfohlen).

```
$ sudo apt install bind9 bind9-utils dnsutils    # Debian: Paket bind9
$ sudo dnf install bind bind-utils               # RHEL: Dienst named
$ sudo systemctl enable --now named              # Debian: bind9
$ ls /etc/bind/                                  # Debian; RHEL: /etc/named.conf, /var/named/
```

Hauptkonfiguration (Ausschnitt) und Zonendefinition:

```
# /etc/bind/named.conf.options
options {
    directory "/var/cache/bind";
    listen-on { 127.0.0.1; 192.168.10.2; };
    allow-query { 192.168.10.0/24; localhost; };
    recursion no;                       # nur autoritativ
    dnssec-validation auto;
};

# /etc/bind/named.conf.local
zone "example.lan" {
    type primary;
    file "/etc/bind/zones/db.example.lan";
    allow-transfer { 192.168.10.3; };   # Sekundaerer Server
    also-notify    { 192.168.10.3; };
};
zone "10.168.192.in-addr.arpa" {
    type primary;
    file "/etc/bind/zones/db.192.168.10";
};
```

Die Zonendatei:

```
; /etc/bind/zones/db.example.lan
$TTL 1h
@   IN  SOA ns1.example.lan. admin.example.lan. (
            2025010101  ; Serial (JJJJMMTTNN) - bei jeder Aenderung erhoehen!
            1h          ; Refresh
            15m         ; Retry
            1w          ; Expire
            1h )        ; Negative Caching
@       IN  NS   ns1.example.lan.
@       IN  NS   ns2.example.lan.
@       IN  MX   10 mail.example.lan.
ns1     IN  A    192.168.10.2
ns2     IN  A    192.168.10.3
mail    IN  A    192.168.10.4
www     IN  A    192.168.10.10
web     IN  CNAME www
```

```
; /etc/bind/zones/db.192.168.10     (Reverse-Zone)
$TTL 1h
@   IN SOA ns1.example.lan. admin.example.lan. ( 2025010101 1h 15m 1w 1h )
@   IN NS  ns1.example.lan.
2   IN PTR ns1.example.lan.
10  IN PTR www.example.lan.
```

```
$ sudo named-checkconf
$ sudo named-checkzone example.lan /etc/bind/zones/db.example.lan
$ sudo rndc reload                          # Zonen neu laden
$ sudo rndc status
$ sudo rndc flush                           # Cache leeren
$ dig @192.168.10.2 www.example.lan +norec
$ dig @192.168.10.2 example.lan AXFR        # Zonentransfer testen (sollte fuer Fremde
    scheitern)
```

- **Seriennummer** erhöhen – sonst übernehmen sekundäre Server die Änderung nicht. Der Punkt am Ende von Namen (`ns1.example.lan.`) bedeutet “vollständig”; fehlt er, hängt BIND den Zonennamen an – ein klassischer Fehler.

- **Primär/Sekundär**: Der Sekundäre holt Zonen per `AXFR/IXFR` vom Primären. Transfers nur an bekannte Server erlauben, ggf. mit TSIG-Schlüsseln absichern.
- **DNSSEC** signiert Zonen kryptografisch; Resolver (`dnssec-validation`) prüfen die Kette bis zur Wurzel. Die Verwaltung der Schlüssel übernimmt BIND mit `dnssec-policy` weitgehend automatisch.
- **Nach Änderungen**: `named-checkzone`, Seriennummer, `rndc reload`, mit `dig` prüfen.

44.6. Systematische Fehlersuche

Niveau: Fortgeschritten

1. `ping 9.9.9.9` – Funktioniert das Netz ohne Namen?
2. `getent hosts name` vs. `dig name` – Liegt es an `/etc/hosts` oder `nsswitch.conf`?
3. `resolvectl status` / `cat /etc/resolv.conf` – Welchen Resolver nutzt der Client?
4. `dig @resolver name` und `dig @autoritativer-server name` – Wo bricht die Kette ab?
5. `dig +trace name` – Delegation korrekt? Stimmen die `NS`-Einträge?
6. `journalctl -u named` / unbound – Fehler beim Laden der Zone?
7. TTL beachten: Nach Änderungen sehen Clients den alten Wert bis zum Ablauf der TTL.

Übungen und Kontrollfragen

1. Worin unterscheiden sich ein rekursiver und ein autoritativer DNS-Server?

   Lösung: Der rekursive Resolver fragt für Clients die Hierarchie ab und cached; der autoritative Server kennt die Zone selbst und liefert verbindliche Antworten.

2. Welchen Eintragstyp benötigen Sie für die Mailzustellung einer Domain?

   Lösung: `MX`-Einträge (mit Priorität), die auf Hostnamen mit `A`/`AAAA`-Einträgen zeigen.

3. Sie ändern eine Zonendatei, der sekundäre Server liefert aber noch den alten Stand. Warum?

   Lösung: Die Seriennummer im `SOA`-Eintrag wurde nicht erhöht (oder der Transfer ist nicht erlaubt/benachrichtigt).

4. Wie prüfen Sie, welche IP-Adresse `www.example.org` bei einem bestimmten Server hat?

   Lösung: `dig @server www.example.org +short`.

5. Warum ist ein offener rekursiver Resolver ein Problem?

   Lösung: Er kann für DNS-Reflection-/Verstärkungsangriffe missbraucht werden; Zugriff auf eigene Netze beschränken.

6. Was bedeutet `NXDOMAIN`?

   Lösung: Der abgefragte Name existiert nicht.

45. DHCP und Zeitsynchronisation

Inhalt: Ablauf von DHCP, DHCP-Server mit Kea und dnsmasq, statische Zuweisungen, Zeitsynchronisation mit NTP, chrony und systemd-timesyncd, Überwachung der Uhr, Zeitserver im eigenen Netz.

Stufe: [F] Fortgeschritten

Lernziele: Nach diesem Kapitel können Sie

- den Vier-Schritte-Ablauf von DHCP zu erklären und Leases zu untersuchen,
- einen DHCP-Server mit Adressbereich und festen Zuordnungen zu konfigurieren,
- die Systemzeit mit chrony zu synchronisieren und deren Qualität zu prüfen.

Vorkenntnisse: Kapitel 38 (TCP/IP), Kapitel 39 (Netzwerk) und Kapitel 32 (systemd).

45.1. DHCP: Adressen automatisch vergeben

Niveau: Einsteiger

Das **Dynamic Host Configuration Protocol** versorgt Clients beim Start mit IP-Adresse, Netzmaske, Standard-Gateway, DNS-Servern und weiteren Optionen. Der Ablauf heißt **DORA**:

Schritt	Nachricht	Inhalt
1	`DISCOVER`	Client sucht per Broadcast einen DHCP-Server
2	`OFFER`	Server bietet eine Adresse samt Optionen an
3	`REQUEST`	Client fordert die angebotene Adresse an
4	`ACK`	Server bestätigt; die Adresse wird für die *Lease-Dauer* vergeben

Vor Ablauf der Hälfte der Lease versucht der Client, sie zu verlängern. Server und Client nutzen UDP (Server-Port 67, Client-Port 68). Weil der Client zu Beginn keine Adresse hat, arbeitet DHCP im lokalen Segment mit Broadcasts; über Router hinweg leitet ein *DHCP-Relay* Anfragen weiter.

```
$ sudo dhclient -v eth0                       # klassischer Client
$ sudo dhclient -r eth0                       # Lease freigeben
$ nmcli device show eth0 | grep -i dhcp       # NetworkManager: Optionen
$ networkctl status eth0                      # systemd-networkd: Lease anzeigen
$ ls /var/lib/dhcp/ /var/lib/NetworkManager/  # Lease-Dateien
$ sudo tcpdump -ni eth0 port 67 or port 68    # DHCP-Verkehr mitschneiden
```

45.2. DHCP-Server mit dnsmasq

Niveau: Fortgeschritten

Für kleine Netze reicht `dnsmasq` (siehe auch Kapitel 44):

```
# /etc/dnsmasq.d/dhcp.conf
interface=eth1
bind-interfaces
dhcp-range=192.168.10.100,192.168.10.199,255.255.255.0,12h
dhcp-option=option:router,192.168.10.1
dhcp-option=option:dns-server,192.168.10.2
dhcp-host=aa:bb:cc:11:22:33,drucker,192.168.10.20     # feste Zuordnung per MAC
dhcp-authoritative
log-dhcp
$ sudo dnsmasq --test && sudo systemctl restart dnsmasq
$ cat /var/lib/misc/dnsmasq.leases                    # aktuelle Leases
```

45.3. DHCP-Server mit Kea

Niveau: Fortgeschritten

`Kea` ist der Nachfolger des ISC-DHCP-Servers (`dhcpd`). Die Konfiguration erfolgt in JSON, Leases lassen sich in Datenbanken speichern, und eine REST-Schnittstelle erlaubt Automatisierung.

```
$ sudo apt install kea-dhcp4-server
$ sudo nano /etc/kea/kea-dhcp4.conf
{
  "Dhcp4": {
    "interfaces-config": { "interfaces": [ "eth1" ] },
    "lease-database": { "type": "memfile", "lfc-interval": 3600 },
    "valid-lifetime": 43200,
    "subnet4": [
      {
        "id": 1,
        "subnet": "192.168.10.0/24",
        "pools": [ { "pool": "192.168.10.100 - 192.168.10.199" } ],
        "option-data": [
          { "name": "routers",             "data": "192.168.10.1" },
          { "name": "domain-name-servers", "data": "192.168.10.2" },
          { "name": "domain-name",         "data": "example.lan" }
        ],
        "reservations": [
          { "hw-address": "aa:bb:cc:11:22:33", "ip-address": "192.168.10.20" }
        ]
      }
    ]
  }
}
$ sudo kea-dhcp4 -t /etc/kea/kea-dhcp4.conf        # Konfiguration pruefen
$ sudo systemctl enable --now kea-dhcp4-server
```

- **Adresspool** nicht zu klein wählen und von festen Adressen (Server, Drucker) trennen.
- **Reservierungen** (feste Zuordnung per MAC) halten Geräte auf bekannter Adresse, ohne an jedem Gerät eine statische Konfiguration zu pflegen.
- **Nur ein DHCP-Server pro Segment**: Ein zweiter, unbeabsichtigter Server (z. B. ein Heimrouter im Laborkabel) verursacht schwer erklärliche Fehler (“Rogue DHCP”).

- Bei Firewalls: UDP 67/68 erlauben (Kapitel 43).

45.4. Warum die Uhrzeit wichtig ist

Niveau: Einsteiger

Eine falsche Systemzeit macht Probleme, die auf den ersten Blick nichts mit der Zeit zu tun haben: ungültige TLS-Zertifikate, fehlschlagende Kerberos-Anmeldungen, unsinnige Logs, fehlerhafte Zeitsteuerung, Konflikte in verteilten Datenbanken. Deshalb synchronisiert jeder Server seine Uhr über das **Network Time Protocol** (NTP, UDP-Port 123).

- NTP-Server sind in **Stratum**-Stufen organisiert: Stratum 0 sind Referenzuhren (GPS, Atomuhr), Stratum 1 Server mit direkter Verbindung, Stratum 2 beziehen von Stratum 1 usw.
- Die Systemzeit läuft in **UTC**; die Anzeige in Ortszeit regelt die Zeitzone (Kapitel 12).
- Die **Hardware-Uhr** (RTC) hält die Zeit im ausgeschalteten Zustand; `hwclock` liest und setzt sie.

```
$ timedatectl                          # Zeitzone, NTP-Status, RTC
$ timedatectl set-timezone Europe/Berlin
$ timedatectl set-ntp true             # systemd-timesyncd aktivieren
$ date -u                              # UTC
$ sudo hwclock --show
```

45.5. chrony und systemd-timesyncd

Niveau: Fortgeschritten

systemd-timesyncd ist ein einfacher SNTP-Client, der als Client genügt. **chrony** ist ein vollwertiger NTP-Dienst: genauer, schnell bei der Anpassung, geeignet für Server und instabile Netze, und kann selbst als Zeitserver dienen.

```
$ sudo apt install chrony
$ sudo systemctl enable --now chrony          # RHEL: chronyd
$ chronyc tracking                             # Abweichung, Stratum, Frequenz
$ chronyc sources -v                           # Quellen und Qualitaet
$ chronyc sourcestats
$ sudo chronyc makestep                        # Uhr sofort korrigieren (grosse Abweichung)
```

```
# /etc/chrony/chrony.conf
pool 2.debian.pool.ntp.org iburst
# oder eigene Server:
# server ntp1.example.lan iburst
driftfile /var/lib/chrony/chrony.drift
makestep 1.0 3                 # in den ersten 3 Updates grosse Spruenge erlauben
rtcsync                        # RTC mitfuehren
allow 192.168.10.0/24          # Clients im LAN bedienen (macht chrony zum Server)
local stratum 10               # Notfall-Quelle ohne Internet
```

Wichtig in der Ausgabe von `chronyc tracking`: `System time` (aktueller Versatz, sollte bei Millisekunden oder weniger liegen), `Stratum` und `Leap status` (`Normal`). In `chronyc sources` markiert `*` die aktuell gewählte Quelle, `+` brauchbare Alternativen; die Spalte `Reach` zeigt als Oktalzahl, ob die letzten Abfragen erfolgreich waren (377 = alle acht).

> Nur ein Zeitdienst

`systemd-timesyncd`, `chrony` und `ntpd` dürfen nicht gleichzeitig laufen. Wer chrony installiert, ersetzt timesyncd automatisch; mit `timedatectl` prüft man, welcher Dienst aktiv ist (`NTP service`, `System clock synchronized`).

45.6. Zeitserver im eigenen Netz und Virtualisierung

Niveau: Experte

- Im internen Netz betreibt man **zwei bis vier** Zeitserver; Clients beziehen von diesen statt aus dem Internet (weniger Verkehr, Kontrolle, Funktion ohne Internet).
- Verbreitet ist auch **NTS** (Network Time Security): authentisiertes NTP über TLS. In chrony: `server time.cloudflare.com iburst nts`.
- Für höchste Genauigkeit (Finanzwesen, Telekommunikation) kommt **PTP** (IEEE 1588, `linuxptp`) mit Hardware-Zeitstempeln in Frage.
- **Virtuelle Maschinen** können ihre Uhr vom Host erhalten (`ptp_kvm`); trotzdem ist eine eigene NTP-Synchronisation üblich. Nach dem Pausieren oder Migrieren einer VM springt die Uhr ggf. stark – `makestep` hilft.
- **Zeitsprünge** können Anwendungen stören. chrony passt die Zeit normalerweise schrittweise an (*slew*) und springt nur bei großen Abweichungen.

```
# Zeitabweichung ueberwachen (z. B. fuer Monitoring, Kapitel Monitoring)
$ chronyc tracking | awk '/System time/ {print $4, $5}'
$ journalctl -u chrony --since today | tail
$ ss -ulpn | grep ':123'               # lauscht NTP?
```

Übungen und Kontrollfragen

1. Nennen Sie die vier Nachrichten des DHCP-Ablaufs in der richtigen Reihenfolge.

   Lösung: `DISCOVER`, `OFFER`, `REQUEST`, `ACK`.

2. Wie erreichen Sie, dass ein Drucker immer die Adresse 192.168.10.20 erhält?

   Lösung: Per DHCP-Reservierung anhand der MAC-Adresse (z. B. `dhcp-host` in dnsmasq, `reservations` in Kea) – außerhalb des dynamischen Pools.

3. Warum sind zwei DHCP-Server im selben Segment problematisch?

   Lösung: Clients erhalten unterschiedliche, womöglich falsche Konfigurationen und Adresskonflikte können entstehen.

4. Warum ist eine korrekte Uhrzeit auf Servern wichtig?

   Lösung: Zertifikate, Kerberos, Logauswertung, Zeitsteuerung und verteilte Systeme hängen von konsistenten Zeitstempeln ab.

5. Wie prüfen Sie, ob chrony synchronisiert ist und wie groß die Abweichung ist?

   Lösung: `chronyc tracking` (Versatz, Stratum) und `chronyc sources` (mit `*` markierte Quelle); ergänzend `timedatectl`.

6. Was bedeutet Stratum 2?

   Lösung: Der Server bezieht seine Zeit von einem Stratum-1-Server, der direkt an einer Referenzuhr hängt.

46. Webserver: nginx und Apache

Inhalt: Funktionsweise von HTTP, Installation und Konfiguration von nginx und Apache, virtuelle Hosts, TLS mit Let's Encrypt, Reverse-Proxy, Zugriffsschutz, Protokolle und Leistungsoptimierung.

Stufe: [F] Fortgeschritten bis [X] Experte

Lernziele: Nach diesem Kapitel können Sie

- einen Webserver zu installieren und statische Seiten über virtuelle Hosts auszuliefern,
- HTTPS mit einem Zertifikat einzurichten und HTTP korrekt umzuleiten,
- nginx als Reverse-Proxy vor eine Anwendung zu stellen und die Protokolle auszuwerten.

Vorkenntnisse: Kapitel 40 (Netzdienste), Kapitel 32 (systemd) und Kapitel 22 (Rechte).

46.1. HTTP in Kürze

Niveau: Einsteiger

HTTP ist ein textbasiertes Anfrage-Antwort-Protokoll auf TCP (Port 80; verschlüsselt als **HTTPS** auf Port 443). Der Client sendet eine Anfrage mit Methode (`GET`, `POST`, …), Pfad und Kopfzeilen; der Server antwortet mit einem **Statuscode** und dem Inhalt.

Statuscode	Bedeutung
200 OK	Erfolg
301 / 302	Dauerhafte / vorübergehende Weiterleitung
400 / 401 / 403	Fehlerhafte Anfrage / Anmeldung nötig / verboten
404 Not Found	Ressource nicht vorhanden
500 / 502 / 503	Serverfehler / fehlerhafte Antwort des Backends / nicht verfügbar

Eine Anfrage lässt sich von Hand nachvollziehen:

```
$ curl -v http://localhost/             # Anfrage und Antwort im Detail
$ curl -I https://example.org/          # nur die Kopfzeilen (HEAD)
$ printf 'GET / HTTP/1.1\r\nHost: localhost\r\nConnection: close\r\n\r\n' | nc localhost 80
```

46.2. nginx installieren und konfigurieren

Niveau: Einsteiger

`nginx` (sprich: "Engine-X") ist ein ereignisgesteuerter Webserver, der sich gut als Auslieferer statischer Dateien und als Reverse-Proxy eignet.

```
$ sudo apt install nginx                    # Debian/Ubuntu
$ sudo dnf install nginx                    # RHEL/Fedora
$ sudo systemctl enable --now nginx
$ curl -I http://localhost/
$ sudo nginx -t                             # Konfiguration pruefen
$ sudo systemctl reload nginx               # ohne Unterbrechung neu laden
$ ls /etc/nginx/ /etc/nginx/conf.d/ /var/log/nginx/
```

Die Konfiguration besteht aus **Direktiven** und **Blöcken**. Die Hauptdatei `/etc/nginx/nginx.conf` bindet Dateien aus `conf.d/` (RHEL) bzw. `sites-enabled/` (Debian) ein.

```
# /etc/nginx/conf.d/beispiel.conf
server {
    listen 80;
    listen [::]:80;
    server_name www.example.org example.org;

    root /var/www/beispiel;
    index index.html;

    access_log /var/log/nginx/beispiel.access.log;
    error_log  /var/log/nginx/beispiel.error.log;

    location / {
        try_files $uri $uri/ =404;
    }

    location /bilder/ {
        expires 7d;                      # Browser-Cache
    }

    location ~ /\.(git|ht) {
        deny all;                        # versteckte Dateien sperren
    }
}
```

- Ein **`server`-Block** ist ein virtueller Host; `server_name` wählt ihn anhand des `Host`-Headers.
- **`location`-Blöcke** wählen anhand des Pfads aus (Präfix, `=` exakt, `~` Regex).
- `root` gibt das Dokumentenverzeichnis an; die Dateien müssen für den Benutzer `www-data` (Debian) bzw. `nginx` lesbar sein.

```
$ sudo mkdir -p /var/www/beispiel
$ echo '<h1>Hallo Linux</h1>' | sudo tee /var/www/beispiel/index.html
$ sudo chown -R www-data: /var/www/beispiel
$ curl -H 'Host: www.example.org' http://localhost/    # Host-Header testen
```

46.3. Apache HTTP Server

Niveau: Fortgeschritten

Der `Apache` (Paket `apache2` bzw. `httpd`) arbeitet mit **Modulen** und unterstützt Konfiguration pro Verzeichnis über `.htaccess`.

```
$ sudo apt install apache2
$ sudo a2enmod rewrite ssl headers proxy proxy_http   # Module aktivieren
$ sudo a2ensite beispiel                              # Site aktivieren
$ sudo a2dissite 000-default
$ sudo apache2ctl configtest
$ sudo systemctl reload apache2
$ apache2ctl -S                                       # virtuelle Hosts anzeigen
```

```
# /etc/apache2/sites-available/beispiel.conf
<VirtualHost *:80>
    ServerName www.example.org
    DocumentRoot /var/www/beispiel

    <Directory /var/www/beispiel>
        Options -Indexes +FollowSymLinks
        AllowOverride None
        Require all granted
    </Directory>

    ErrorLog  ${APACHE_LOG_DIR}/beispiel.error.log
    CustomLog ${APACHE_LOG_DIR}/beispiel.access.log combined
</VirtualHost>
```

Aspekt	nginx	Apache
Architektur	Ereignisgesteuert, wenige Prozesse	Prozess-/Thread-Modelle (MPM)
Stärken	Statische Inhalte, Proxy, geringe Last	Module, `.htaccess`, Flexibilität
Konfig-Prüfung	`nginx -t`	`apache2ctl configtest`
Neu laden	`systemctl reload nginx`	`systemctl reload apache2`
Benutzer (Debian)	`www-data`	`www-data`

46.4. HTTPS und Let's Encrypt

Niveau: Fortgeschritten

Moderne Websites laufen nur noch über **TLS** (Kapitel 58). Zertifikate stellt **Let's Encrypt** kostenlos aus; der Client `certbot` automatisiert Anforderung und Erneuerung.

```
$ sudo apt install certbot python3-certbot-nginx
$ sudo certbot --nginx -d example.org -d www.example.org
$ sudo certbot renew --dry-run               # Erneuerung testen
$ systemctl list-timers | grep certbot       # automatische Erneuerung
```

Voraussetzung: Der DNS-Name zeigt auf den Server, und Port 80 ist von außen erreichbar (HTTP-Challenge). Der Server-Block mit TLS sieht dann etwa so aus:

```
server {
    listen 80;
    server_name example.org www.example.org;
    return 301 https://example.org$request_uri;       # HTTP -> HTTPS
}

server {
    listen 443 ssl http2;
    server_name example.org;

    ssl_certificate     /etc/letsencrypt/live/example.org/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/example.org/privkey.pem;
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_prefer_server_ciphers off;

    add_header Strict-Transport-Security "max-age=31536000" always;
    add_header X-Content-Type-Options nosniff;

    root /var/www/beispiel;
}
```

46.5. Reverse-Proxy

Niveau: Fortgeschritten

Ein **Reverse-Proxy** nimmt Anfragen entgegen und leitet sie an ein Backend weiter (Anwendungsserver, Container, andere Hosts). Vorteile: zentrales TLS, mehrere Anwendungen hinter einer Adresse, Caching, Lastverteilung.

```
upstream app {
    server 127.0.0.1:8000;
    server 127.0.0.1:8001 backup;
}

server {
    listen 443 ssl;
    server_name app.example.org;
    # ssl_certificate ... (wie oben)

    location / {
        proxy_pass http://app;
        proxy_set_header Host              $host;
        proxy_set_header X-Real-IP         $remote_addr;
        proxy_set_header X-Forwarded-For   $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_read_timeout 60s;
    }

    location /ws/ {                       # WebSocket
        proxy_pass http://app;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
    }
}
```

Mehr zu Lastverteilung und Hochverfügbarkeit: Kapitel 47. Meldet der Proxy `502 Bad Gateway`, ist meist das Backend nicht erreichbar – zuerst direkt mit `curl http://127.0.0.1:8000/` prüfen. Läuft SELinux, muss ggf. `setsebool -P httpd_can_network_connect 1` gesetzt werden (Kapitel 23).

46.6. Zugriffsschutz, Protokolle und Leistung

Niveau: Experte

Zugriffsschutz mit Basic Authentication und IP-Einschränkung:

```
$ sudo apt install apache2-utils
$ sudo htpasswd -c /etc/nginx/.htpasswd anna

location /intern/ {
    auth_basic "Interner Bereich";
    auth_basic_user_file /etc/nginx/.htpasswd;
    allow 192.168.10.0/24;
    deny  all;
    satisfy any;
}
```

Protokolle: Das Zugriffsprotokoll enthält je Anfrage eine Zeile. Mit den Werkzeugen aus Kapitel 8 lässt es sich auswerten:

```
$ tail -f /var/log/nginx/access.log
$ awk '{print $9}' /var/log/nginx/access.log | sort | uniq -c | sort -rn   # Statuscodes
$ awk '{print $1}' /var/log/nginx/access.log | sort | uniq -c | sort -rn | head   # Top-IPs
$ grep ' 404 ' /var/log/nginx/access.log | awk '{print $7}' | sort | uniq -c | sort -rn |
    head
$ sudo tail -50 /var/log/nginx/error.log
```

Leistung und Härtung:

- `gzip on;` sowie `worker_processes auto;` und passende `worker_connections`.
- Versionsangabe verbergen: `server_tokens off;`.
- Anfragegröße und Zeitlimits begrenzen: `client_max_body_size`, `client_body_timeout`.
- Ratenbegrenzung: `limit_req_zone` und `limit_req`.
- Dienst nicht als Root: Die Worker laufen unter `www-data`/`nginx`; Inhalte nicht schreibbar für den Webserver machen.
- `Options -Indexes` (Apache) bzw. `autoindex off;` (nginx): kein Verzeichnislisting.
- Updates, `fail2ban` für Anmeldeversuche und eine Web Application Firewall (ModSecurity, …) ergänzen die Härtung.

```
# Lasttest (Vorsicht, nur eigene Server)
$ sudo apt install apache2-utils
$ ab -n 1000 -c 20 http://localhost/
```

Übungen und Kontrollfragen

1. Wie testen Sie eine nginx-Konfiguration, ohne den Dienst zu stören?

   Lösung: `sudo nginx -t` prüft die Syntax; `systemctl reload nginx` übernimmt sie ohne Verbindungsabbruch.

2. Wodurch wählt nginx den passenden virtuellen Host?

   Lösung: Durch `listen` (Port/Adresse) und den `Host`-Header im Vergleich mit `server_name`.

3. Was bedeutet der Statuscode 502?

   Lösung: Der Proxy erhielt keine gültige Antwort vom Backend (nicht erreichbar, abgestürzt, falscher Port).

4. Wie leiten Sie HTTP dauerhaft auf HTTPS um?

   Lösung: In einem Server-Block auf Port 80: `return 301 https://$host$request_uri;`.

5. Welche Voraussetzungen hat die HTTP-Challenge von Let's Encrypt?

   Lösung: Der Domainname zeigt per DNS auf den Server und Port 80 ist aus dem Internet erreichbar.

6. Wie ermitteln Sie die zehn häufigsten Client-Adressen aus dem Zugriffsprotokoll?

   Lösung: `awk '{print $1}' access.log | sort | uniq -c | sort -rn | head`.

47. Proxy und Lastverteilung

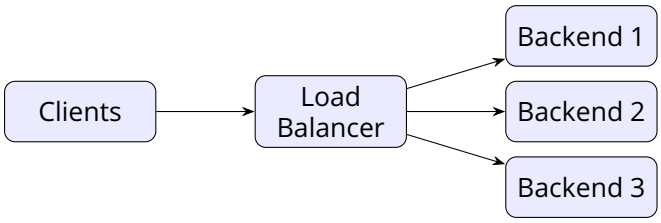
Inhalt: Forward- und Reverse-Proxy, Lastverteilung mit HAProxy, Gesundheitsprüfungen, Sitzungsbindung, Squid als Forward-Proxy.
Stufe: [F] Fortgeschritten bis [X] Experte
Lernziele: Nach diesem Kapitel können Sie
- die Rollen von Forward-Proxy, Reverse-Proxy und Load Balancer zu unterscheiden,
- mit HAProxy Verkehr auf mehrere Backends zu verteilen und deren Zustand zu überwachen,
- Squid als Forward-Proxy einzurichten und zu nutzen.

Vorkenntnisse: Kapitel 46 (Webserver und Reverse-Proxy), Kapitel 38 (TCP/IP) und Kapitel 43 (Firewall).

47.1. Proxy-Arten

Niveau: Fortgeschritten

- **Forward-Proxy**: Stellvertreter für *Clients*. Clients im internen Netz greifen über ihn ins Internet zu (Filterung, Caching, Protokollierung). Beispiel: Squid.
- **Reverse-Proxy**: Stellvertreter für *Server*. Er nimmt Anfragen aus dem Internet an und reicht sie an interne Server weiter (TLS-Terminierung, Schutz, Caching). Beispiele: nginx, HAProxy.
- **Load Balancer** (Lastverteiler): Reverse-Proxy, der Anfragen auf *mehrere gleichartige Backends* verteilt und ausgefallene Server meidet.
- **Layer 4 vs. Layer 7**: Layer 4 (TCP/UDP) verteilt Verbindungen ohne den Inhalt zu kennen (schnell, allgemein); Layer 7 (HTTP) kann nach Pfad, Host oder Cookie entscheiden.



47.2. HAProxy als Load Balancer

Niveau: Fortgeschritten

`HAProxy` ist ein sehr leistungsfähiger TCP- und HTTP-Lastverteiler mit Gesundheitsprüfungen, Statistikseite und feiner Steuerung.

```
$ sudo apt install haproxy
$ sudo haproxy -c -f /etc/haproxy/haproxy.cfg      # Konfiguration pruefen
$ sudo systemctl reload haproxy
$ echo 'show stat' | sudo socat stdio /run/haproxy/admin.sock | cut -d, -f1,2,18 | column -s,
    -t
```

```
# /etc/haproxy/haproxy.cfg
global
    log /dev/log local0
    maxconn 4000
    user haproxy
    group haproxy
    stats socket /run/haproxy/admin.sock mode 660 level admin

defaults
    mode http
    log global
    option httplog
    option forwardfor
    timeout connect 5s
    timeout client  30s
    timeout server  30s

frontend web
    bind *:80
    bind *:443 ssl crt /etc/haproxy/certs/example.pem
    http-request redirect scheme https unless { ssl_fc }
    acl ist_api path_beg /api
    use_backend api if ist_api
    default_backend webfarm

backend webfarm
    balance roundrobin
    option httpchk GET /health
    http-check expect status 200
    server web1 192.168.10.11:80 check inter 3s fall 3 rise 2
    server web2 192.168.10.12:80 check inter 3s fall 3 rise 2
    server web3 192.168.10.13:80 check backup

backend api
    balance leastconn
    cookie SERVERID insert indirect nocache
    server api1 192.168.10.21:8000 check cookie a1
    server api2 192.168.10.22:8000 check cookie a2

listen stats
    bind 127.0.0.1:8404
    stats enable
    stats uri /stats
    stats refresh 10s
```

Verfahren	**Wirkung**
`roundrobin`	Reihum; einfach und gleichmäßig
`leastconn`	Server mit den wenigsten aktiven Verbindungen; gut bei langen Verbindungen
`source`	Aus der Client-Adresse abgeleitet; gleiche Clients landen beim gleichen Server
`uri` / Cookie	Sitzungsbindung (*sticky sessions*) für zustandsbehaftete Anwendungen

- **Gesundheitsprüfung** (`check`): Fällt ein Backend aus (`fall 3`), erhält es keine Anfragen mehr; kehrt es zurück (`rise 2`), wird es wieder aufgenommen.
- **backup**: Server wird nur benutzt, wenn alle anderen ausfallen.
- **Sitzungsbindung** vermeiden, wenn möglich: Besser Sitzungsdaten zentral speichern (Redis, Datenbank), damit jeder Server jede Anfrage bedienen kann.

- **Wartung**: Server mit `disable server webfarm/web1` über den Admin-Socket aus der Rotation nehmen; laufende Verbindungen enden normal (*drain*).

47.3. TCP-Lastverteilung

Niveau: Fortgeschritten

Nicht nur HTTP: Auch Datenbanken, SMTP oder beliebige TCP-Dienste lassen sich verteilen (Modus `tcp`):

```
frontend pg_front
    mode tcp
    bind *:5432
    default_backend pg_back

backend pg_back
    mode tcp
    option tcp-check
    server db1 192.168.10.31:5432 check
    server db2 192.168.10.32:5432 check backup
```

47.4. Ausfallsicherheit des Load Balancers

Niveau: Fortgeschritten

Ein Load Balancer darf selbst kein Single Point of Failure sein. Üblich sind zwei HAProxy-Server, die sich eine *virtuelle IP-Adresse* teilen und per `keepalived` (VRRP) umgeschaltet werden. Die Einrichtung (inklusive Gesundheitsprüfung für HAProxy, `ip_nonlocal_bind`, Split-Brain und Test) beschreibt Kapitel 60.

47.5. Squid als Forward-Proxy

Niveau: Fortgeschritten

```
$ sudo apt install squid
$ sudo squid -k parse                        # Konfiguration pruefen
$ sudo systemctl reload squid

# /etc/squid/squid.conf  (Ausschnitt)
acl lokal src 192.168.10.0/24
acl sicherePorts port 80 443
acl blockiert dstdomain .beispiel-werbung.example
http_access deny !sicherePorts
http_access deny blockiert
http_access allow lokal
http_access deny all
http_port 3128
cache_dir ufs /var/spool/squid 2000 16 256

$ export https_proxy=http://proxy.example.lan:3128 http_proxy=http://proxy.example.lan:3128
$ curl -I https://example.org
$ sudo tail -f /var/log/squid/access.log
```

Für apt, dnf und Docker trägt man den Proxy in deren Konfigurationsdateien ein (bei apt in `/etc/apt/apt.conf.d/`). Transparente Proxys und TLS-Inspektion sind rechtlich und technisch anspruchsvoll (Datenschutz, Zertifikatsverteilung).

Übungen und Kontrollfragen

1. Worin unterscheiden sich Forward- und Reverse-Proxy?

   Lösung: Der Forward-Proxy vertritt Clients beim Zugriff nach außen; der Reverse-Proxy vertritt Server und nimmt Anfragen von außen entgegen.

2. Wozu dienen Gesundheitsprüfungen im Load Balancer?

   Lösung: Sie erkennen ausgefallene Backends, damit diese keine Anfragen mehr erhalten, und nehmen sie nach Rückkehr wieder auf.

3. Wann ist `leastconn` besser als `roundrobin`?

   Lösung: Bei langlebigen Verbindungen oder sehr unterschiedlich langen Anfragen, da die Last nach aktiven Verbindungen verteilt wird.

4. Warum sollte man Sitzungsbindung möglichst vermeiden?

   Lösung: Sie verteilt Last ungleichmäßig und erschwert Wartung/Ausfall; zentral gespeicherte Sitzungen erlauben freie Verteilung.

48. VPN: WireGuard und OpenVPN

Inhalt: Zweck und Arten von VPNs, WireGuard-Schlüssel und -Konfiguration, Site-to-Site und Roadwarrior, Routing und Firewall, OpenVPN im Überblick, Fehlersuche und Sicherheitsaspekte.

Stufe: [F] Fortgeschritten bis [X] Experte

Lernziele: Nach diesem Kapitel können Sie

- den Zweck eines VPN und die Unterschiede zwischen Site-to-Site und Roadwarrior zu erklären,
- ein WireGuard-VPN zwischen zwei oder mehr Rechnern aufzubauen,
- Routing, Firewall und typische Fehler (MTU, Schlüssel, Ports) zu prüfen.

Vorkenntnisse: Kapitel 38 (IP und Routing), Kapitel 39 (Netzwerk) und Kapitel 43 (Firewall).

48.1. Wozu ein VPN?

Niveau: Einsteiger

Ein **Virtual Private Network** verbindet Rechner oder Netze über ein unsicheres Netz (Internet) so, als lägen sie im selben privaten Netz. Der Verkehr wird verschlüsselt durch einen *Tunnel* geleitet.

- **Roadwarrior**: Einzelne Geräte (Laptop, Smartphone) greifen von unterwegs auf das Firmennetz zu.
- **Site-to-Site**: Zwei Standorte werden dauerhaft verbunden; die Router tauschen Netze aus.
- **Mesh**: Viele Rechner sprechen direkt miteinander (z. B. mit Tailscale oder Netbird, die auf WireGuard aufbauen).
- **Full tunnel** sendet den gesamten Verkehr über das VPN; **Split tunnel** nur den für das Zielnetz bestimmten Verkehr.

	WireGuard	**OpenVPN**
Code-Umfang	Klein (wenige tausend Zeilen), im Kernel	Groß, im Benutzerbereich
Konfiguration	Schlüsselpaare, wenige Zeilen	Zertifikate (PKI), viele Optionen
Transport	UDP	UDP oder TCP (auch Port 443)
Leistung	Sehr hoch	Gut, meist langsamer
Besonderheit	Peers sind über öffentliche Schlüssel identifiziert	Flexibel, Benutzer-Authentifizierung, Skripte

48.2. WireGuard: Grundprinzip

Niveau: Fortgeschritten

WireGuard ist seit Linux 5.6 Teil des Kernels. Jede Seite besitzt ein Schlüsselpaar (privat/öffentlich); man tauscht nur die öffentlichen Schlüssel aus. Jeder **Peer** hat eine Liste erlaubter IP-Adressen (`AllowedIPs`), die zugleich als Routing-Tabelle *und* Zugriffsfilter dient: Pakete von einem Peer werden nur akzeptiert, wenn die Quelladresse in dessen `AllowedIPs` liegt.

```
$ sudo apt install wireguard                  # Pakete: wireguard, wireguard-tools
$ umask 077
$ wg genkey | tee privatekey | wg pubkey > publickey
$ cat publickey                               # diesen Schluessel an die Gegenseite geben
$ wg genpsk > preshared                       # optional: zusaetzlicher symmetrischer
    Schluessel
```

Der private Schlüssel verlässt den Rechner nie und darf nur für den Besitzer lesbar sein.

48.3. Beispiel: Roadwarrior und Server

Niveau: Fortgeschritten

Das VPN-Netz ist `10.8.0.0/24`; der Server hat `10.8.0.1`, der Laptop `10.8.0.2`.

```
# Server: /etc/wireguard/wg0.conf
[Interface]
Address = 10.8.0.1/24
ListenPort = 51820
PrivateKey = <privater Schluessel des Servers>
PostUp   = nft add rule inet filter forward iif wg0 accept
# Weitere Optionen: SaveConfig = false

[Peer]
# Laptop von Anna
PublicKey = <oeffentlicher Schluessel des Laptops>
AllowedIPs = 10.8.0.2/32
```

```
# Laptop: /etc/wireguard/wg0.conf
[Interface]
Address = 10.8.0.2/24
PrivateKey = <privater Schluessel des Laptops>
DNS = 10.8.0.1

[Peer]
PublicKey = <oeffentlicher Schluessel des Servers>
Endpoint = vpn.example.org:51820
AllowedIPs = 10.8.0.0/24, 192.168.10.0/24     # Split Tunnel: VPN- und Firmennetz
# AllowedIPs = 0.0.0.0/0, ::/0                # Full Tunnel
PersistentKeepalive = 25                      # haelt NAT-Eintraege offen
```

```
$ sudo chmod 600 /etc/wireguard/wg0.conf
$ sudo wg-quick up wg0                        # Schnittstelle starten
$ sudo wg-quick down wg0
$ sudo systemctl enable --now wg-quick@wg0    # beim Booten
$ sudo wg show                                # Peers, letzter Handshake, Datenmenge
$ ip addr show wg0
$ ip route | grep wg0
$ ping 10.8.0.1
```

Auf dem Server muss UDP 51820 in der Firewall erlaubt sein. Soll der Laptop über den Server ins Firmennetz oder Internet, braucht der Server `net.ipv4.ip_forward=1` und eine Weiterleitungs- bzw. NAT-Regel (Kapitel 43):

```
$ echo 'net.ipv4.ip_forward=1' | sudo tee /etc/sysctl.d/90-wg.conf && sudo sysctl --system
$ sudo nft add rule inet filter input udp dport 51820 accept
$ sudo nft add rule ip nat postrouting oif eth0 ip saddr 10.8.0.0/24 masquerade
```

Handshake prüfen

In der Ausgabe von `wg show` zeigt `latest handshake`, ob die Verbindung steht. Fehlt er, liegt es fast immer an: falschem Schlüssel, falschem `Endpoint`/Port, blockierter UDP-Verbindung oder vertauschten `AllowedIPs`. WireGuard antwortet auf ungültige Pakete gar nicht – Fehler sieht man deshalb selten in Logs.

48.4. Site-to-Site mit WireGuard

Niveau: Fortgeschritten

Zwei Standorte A (LAN `192.168.10.0/24`) und B (LAN `192.168.20.0/24`) verbinden ihre Gateways:

```
# Gateway A
[Interface]
Address = 10.8.0.1/30
ListenPort = 51820
PrivateKey = <Schluessel A>
[Peer]
PublicKey = <oeffentlicher Schluessel B>
Endpoint = gw-b.example.org:51820
AllowedIPs = 10.8.0.2/32, 192.168.20.0/24
PersistentKeepalive = 25

# Gateway B: analog, AllowedIPs = 10.8.0.1/32, 192.168.10.0/24
```

Beide Gateways brauchen IP-Forwarding, und die Rechner im LAN müssen als Route für das jeweils andere Netz das Gateway benutzen (oder der Standard-Router verweist darauf). Prüfen mit `traceroute 192.168.20.5` von einem Rechner im Netz A.

48.5. OpenVPN im Überblick

Niveau: Fortgeschritten

OpenVPN verwendet TLS und eine eigene PKI (Kapitel 58): Server und Clients haben Zertifikate, die von einer eigenen Zertifizierungsstelle signiert sind. Vorteile: Benutzerauthentifizierung (auch mit Passwort/2FA), Betrieb über TCP 443 durch restriktive Firewalls, breite Client-Unterstützung.

```
# Server (Ausschnitt): /etc/openvpn/server/server.conf
port 1194
proto udp
dev tun
ca   ca.crt
cert server.crt
key  server.key
dh   none
server 10.9.0.0 255.255.255.0
push "route 192.168.10.0 255.255.255.0"
keepalive 10 120
cipher AES-256-GCM
user nobody
group nogroup
persist-key
persist-tun
status /var/log/openvpn/status.log
verb 3

$ sudo systemctl enable --now openvpn-server@server
$ sudo journalctl -u openvpn-server@server -f
$ sudo openvpn --config client.ovpn          # Client starten (zum Testen)
```

Zum Erstellen der Zertifikate dienen `easy-rsa` oder die Werkzeuge aus Kapitel 58. Jeder Client erhält ein eigenes Zertifikat, das bei Verlust des Geräts widerrufen werden kann (*CRL*).

48.6. Fehlersuche und Sicherheit

Niveau: Experte

- **MTU**: Der Tunnel verbraucht Platz für Kopfdaten; bei einer Standard-MTU von 1500 liegt die WireGuard-MTU typischerweise bei 1420. Symptom bei Problemen: Verbindung steht, kleine Pakete (Ping) gehen, große (TLS, Downloads) hängen. Abhilfe: `MTU = 1380` im `[Interface]`; testen mit `ping -M do -s 1372 ziel`.
- **Routing**: `ip route get 192.168.10.5` zeigt, welchen Weg ein Paket nimmt.
- **Mitschnitt**: `sudo tcpdump -ni eth0 udp port 51820` (verschlüsselter Verkehr) und `tcpdump -ni wg0` (entschlüsselt).
- **Schlüssel rotieren** und widerrufen: Ein Peer wird entfernt, indem man seinen Abschnitt löscht (`wg set wg0 peer <Schlüssel> remove`).
- **Prinzip der geringsten Rechte**: `AllowedIPs` und Firewallregeln auf das Nötige begrenzen; ein VPN-Zugang ist kein Freibrief für das ganze interne Netz.
- **DNS-Lecks**: Bei Full Tunnel muss auch DNS durch den Tunnel laufen (`DNS=` in der Konfiguration).
- **Überwachung**: Letzter Handshake und Datenvolumen pro Peer (`wg show`) lassen sich mit Monitoring-Werkzeugen erfassen (Kapitel 57).

```
$ sudo wg show wg0 latest-handshakes
$ sudo wg show wg0 transfer
$ sudo tcpdump -ni wg0 icmp
$ ip -s link show wg0
```

Übungen und Kontrollfragen

1. Worin unterscheiden sich Roadwarrior- und Site-to-Site-VPN?

   Lösung: Roadwarrior: einzelne Geräte wählen sich ins Netz ein; Site-to-Site: zwei Netze werden dauerhaft über Gateways verbunden.

2. Welche Aufgabe haben `AllowedIPs` bei WireGuard?

   Lösung: Sie wirken als Routing-Tabelle (welche Ziele gehen zu diesem Peer) und als Filter (von diesem Peer werden nur Pakete dieser Quelladressen akzeptiert).

3. Welcher Schlüssel muss unbedingt geheim bleiben?

   Lösung: Der private Schlüssel jedes Peers; nur die öffentlichen Schlüssel werden ausgetauscht.

4. Der Tunnel steht, aber Webseiten laden nicht. Welche Ursache liegt nahe?

   Lösung: Ein MTU-Problem; große Pakete werden verworfen. Kleinere MTU setzen und mit `ping -M do -s` testen.

5. Wie leiten Sie den gesamten Verkehr eines Clients durch das VPN?

   Lösung: `AllowedIPs = 0.0.0.0/0, ::/0` auf dem Client; auf dem Server Weiterleitung und NAT aktivieren; DNS durch den Tunnel setzen.

6. Woran erkennen Sie, ob ein WireGuard-Peer verbunden ist?

   Lösung: An `latest handshake` in `wg show` (nicht älter als wenige Minuten).

49. Dateidienste: NFS und Samba

Inhalt: Netzwerkdateisysteme im Überblick, NFS-Exporte und -Mounts, Berechtigungen und Squash, Samba-Freigaben für Windows und Linux, Benutzerverwaltung, Mounten per CIFS, Leistung und Sicherheit.

Stufe: [F] Fortgeschritten

Lernziele: Nach diesem Kapitel können Sie

- Verzeichnisse per NFS freizugeben und auf Clients einzubinden,
- mit Samba Freigaben für Windows-Clients bereitzustellen und Zugriffsrechte zu setzen,
- dauerhafte Mounts über `/etc/fstab` oder systemd einzurichten.

Vorkenntnisse: Kapitel 22 (Rechte), Kapitel 20 (Benutzer) und Kapitel 29 (Dateisysteme und fstab).

49.1. Überblick

Niveau: Einsteiger

Dateidienste machen Verzeichnisse eines Servers auf anderen Rechnern wie lokale Ordner verfügbar.

	NFS	SMB/CIFS (Samba)
Typische Clients	Linux, Unix	Windows, macOS, Linux
Authentifizierung	Host- bzw. Kerberos-basiert (v4)	Benutzername/Passwort, Active Directory
Rechte	Unix-UID/GID	Windows-ACLs, Unix-Rechte
Port	TCP 2049 (NFSv4)	TCP 445
Einsatz	Home-Verzeichnisse, Cluster, VM-Speicher	Büro-Freigaben, Drucker, AD-Mitglied

49.2. NFS: Server

Niveau: Fortgeschritten

```
$ sudo apt install nfs-kernel-server          # RHEL: nfs-utils
$ sudo mkdir -p /srv/nfs/projekte
$ sudo chown nobody:nogroup /srv/nfs/projekte
$ sudo nano /etc/exports
/srv/nfs/projekte  192.168.10.0/24(rw,sync,no_subtree_check)
/srv/nfs/readonly  192.168.10.0/24(ro,sync,no_subtree_check,root_squash)
$ sudo exportfs -ra                            # Exporte neu einlesen
$ sudo exportfs -v                             # aktive Exporte
$ sudo systemctl enable --now nfs-server
$ sudo showmount -e localhost
```

Option	Bedeutung
`rw` / `ro`	Lesen und Schreiben bzw. nur Lesen
`sync` / `async`	Schreibvorgänge erst nach dem Sichern auf Platte bestätigen (sicher) bzw. vorher (schneller, riskanter)
`root_squash`	Remote-Root wird auf einen unprivilegierten Benutzer (`nobody`) abgebildet – Standard und empfohlen
`no_root_squash`	Remote-Root bleibt Root (gefährlich)
`all_squash`	Alle Benutzer werden auf `anonuid`/`anongid` abgebildet
`no_subtree_check`	Vermeidet Probleme beim Umbenennen und ist üblich

Wichtig: NFS vertraut den UIDs und GIDs der Clients. Ein Benutzer `anna` (UID 1000) auf dem Client greift als UID 1000 auf dem Server zu. Deshalb müssen UIDs über alle Rechner einheitlich sein (zentrale

Anmeldung, Kapitel 21). Ohne Kerberos (`sec=krb5p`) sollte NFS nur in vertrauenswürdigen Netzen laufen.

49.3. NFS: Client

Niveau: Fortgeschritten

```
$ sudo apt install nfs-common                  # RHEL: nfs-utils
$ showmount -e fileserver                       # Exporte des Servers
$ sudo mkdir -p /mnt/projekte
$ sudo mount -t nfs4 fileserver:/srv/nfs/projekte /mnt/projekte
$ df -hT /mnt/projekte
$ mount | grep nfs
$ sudo umount /mnt/projekte

# dauerhaft in /etc/fstab
fileserver:/srv/nfs/projekte  /mnt/projekte  nfs4  defaults,_netdev,nofail  0  0
```

Mit `_netdev` wartet das System, bis das Netz verfügbar ist; `nofail` verhindert, dass ein nicht erreichbarer Server den Start blockiert. Für selten genutzte Freigaben ist der **Automounter** sinnvoll: mit `x-systemd.automount,x-systemd.idle-timeout=60` in den Optionen (oder `autofs`) wird erst beim Zugriff eingehängt und später wieder getrennt.

```
# systemd-Automount per fstab
fileserver:/srv/nfs/projekte  /mnt/projekte  nfs4  _netdev,noauto,x-systemd.automount,x-
    systemd.idle-timeout=60  0  0
$ sudo systemctl daemon-reload && sudo systemctl restart mnt-projekte.automount
```

> **Hängende Mounts**
>
> Fällt ein NFS-Server aus, blockieren Prozesse, die darauf zugreifen (Zustand `D`). Mit der Option `soft` wird abgebrochen statt gewartet – das kann aber zu Datenverlust führen. Besser: Hochverfügbarkeit des Servers und `umount -f` oder `umount -l` im Notfall.

49.4. Samba: Freigaben für Windows

Niveau: Fortgeschritten

Samba implementiert das SMB-Protokoll. Es kann als Dateiserver, Druckserver und sogar als Active-Directory-Domänencontroller dienen. Hier: eigenständiger Dateiserver.

```
$ sudo apt install samba                        # RHEL: samba samba-client
$ sudo mkdir -p /srv/samba/team
$ sudo groupadd team
$ sudo chgrp team /srv/samba/team
$ sudo chmod 2770 /srv/samba/team               # setgid: Dateien erben die Gruppe

# /etc/samba/smb.conf
[global]
    workgroup = ARBEIT
    server string = Dateiserver
    server role = standalone server
    map to guest = never
    server min protocol = SMB2
    log file = /var/log/samba/log.%m

[team]
    path = /srv/samba/team
    valid users = @team
    read only = no
    create mask = 0660
    directory mask = 2770

[public]
    path = /srv/samba/public
    guest ok = yes
    read only = yes
```

Samba führt eine *eigene Passwortdatenbank*: Jeder Benutzer braucht ein Systemkonto und ein Samba-Passwort.

```
$ sudo useradd -m -G team anna
$ sudo smbpasswd -a anna                        # Samba-Passwort setzen
$ sudo pdbedit -L                               # Samba-Benutzer
$ testparm                                      # smb.conf pruefen
$ sudo systemctl enable --now smbd nmbd         # RHEL: smb nmb
$ sudo smbstatus                                # Verbindungen und gesperrte Dateien
$ smbclient -L //fileserver -U anna             # Freigaben auflisten
$ smbclient //fileserver/team -U anna           # interaktiver Client
```

49.5. Samba-Freigaben unter Linux einhängen

Niveau: Fortgeschritten

```
$ sudo apt install cifs-utils
$ sudo mkdir -p /mnt/team
$ sudo mount -t cifs //fileserver/team /mnt/team -o username=anna,uid=1000,gid=1000,vers=3.0
# Passwort nicht in fstab! Besser eine Credentials-Datei:
$ sudo nano /root/.smbcred
username=anna
password=geheim
domain=ARBEIT
$ sudo chmod 600 /root/.smbcred
# /etc/fstab
//fileserver/team  /mnt/team  cifs  credentials=/root/.smbcred,uid=1000,gid=1000,_netdev,
    nofail  0  0
```

Unter Windows verbindet man sich im Explorer mit `\\fileserver\team`. Die Namensauflösung im Windows-Netz funktioniert über DNS (Kapitel 44) oder, in einfachen Netzen, über `nmbd`/NetBIOS.

49.6. Active Directory, ACLs und Sicherheit

Niveau: Experte

- **Domänenmitglied**: Mit `realmd` und `sssd` (Kapitel 21) tritt ein Linux-Server einer AD-Domäne bei (`realm join example.lan`); Samba nutzt `winbind`. Anmeldung und Rechte kommen dann aus dem Verzeichnis.
- **ACLs**: Für feinere Rechte als Besitzer/Gruppe/Andere: `setfacl -m u:bob:rwx verzeichnis`; Samba kann Windows-ACLs auf POSIX-ACLs abbilden (`vfs objects = acl_xattr`).
- **Sicherheit**: SMB1 deaktivieren (`server min protocol = SMB2`), Signierung und Verschlüsselung (`server signing = mandatory`, `smb encrypt = required`), Zugriff per Firewall (Kapitel 43) auf TCP 445 beschränken, keine Gast-Freigaben ohne Grund.
- **SELinux**: Freigegebene Verzeichnisse brauchen den Kontext `samba_share_t` und ggf. Booleans wie `samba_enable_home_dirs` (Kapitel 23).
- **Papierkorb**, Schattenkopien (`shadow_copy2` mit Btrfs/ZFS-Snapshots, Kapitel 30) und Quotas ergänzen den Betrieb.

```
$ getfacl /srv/samba/team
$ sudo setfacl -R -m g:team:rwx /srv/samba/team
$ sudo setfacl -R -d -m g:team:rwx /srv/samba/team     # Standard fuer neue Dateien
$ sudo ls -Zd /srv/samba/team                            # SELinux-Kontext
$ sudo chcon -R -t samba_share_t /srv/samba/team          # (besser: semanage fcontext)
$ sudo journalctl -u smbd --since "10 min ago"
$ sudo tail -f /var/log/samba/log.smbd
```

Übungen und Kontrollfragen

1. Wozu dient `root_squash`?

   Lösung: Es bildet den Root-Benutzer eines Clients auf einen unprivilegierten Benutzer ab, damit ein kompromittierter Client nicht als Root auf dem Server schreiben kann.

2. Wie machen Sie einen NFS-Export dauerhaft auf einem Client verfügbar, ohne den Start zu blockieren, wenn der Server fehlt?

   Lösung: Eintrag in `/etc/fstab` mit den Optionen `_netdev,nofail`, optional mit `x-systemd.automount`.

3. Warum brauchen Samba-Benutzer ein eigenes Passwort?

   Lösung: Samba verwaltet eigene Passwort-Hashes (NT-Hash), die sich nicht aus den Linux-Hashes ableiten lassen; Anlegen mit `smbpasswd -a`.

4. Wozu dient das setgid-Bit (`2770`) auf einem Gruppenverzeichnis?

   Lösung: Neue Dateien und Unterverzeichnisse erben die Gruppe des Verzeichnisses, sodass alle Teammitglieder Zugriff haben.

5. Wie prüfen Sie die Syntax von `smb.conf` und die aktuellen Verbindungen?

   Lösung: `testparm` bzw. `smbstatus`.

6. Warum sollte man Passwörter nicht direkt in `/etc/fstab` eintragen?

   Lösung: Die Datei ist für alle lesbar; stattdessen eine nur für Root lesbare Credentials-Datei (`chmod 600`) verwenden.

50. E-Mail-Server: Postfix und Dovecot

Inhalt: Wege einer E-Mail (MUA, MTA, MDA), SMTP und IMAP, Postfix-Grundkonfiguration, Dovecot, TLS, SPF, DKIM und DMARC, Warteschlange und Fehlersuche, Spam-Abwehr und Betriebsrisiken.

Stufe: [F] Fortgeschritten bis [X] Experte

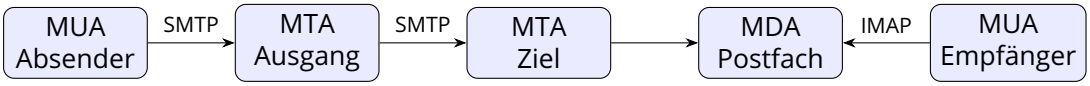
Lernziele: Nach diesem Kapitel können Sie

- den Weg einer Nachricht vom Absender zum Postfach zu beschreiben,
- Postfix als Mailserver und Dovecot für IMAP grundlegend zu konfigurieren,
- Warteschlange und Protokolle zu lesen und Zustellprobleme einzugrenzen.

Vorkenntnisse: Kapitel 44 (DNS, MX-Einträge), Kapitel 58 (TLS) und Kapitel 37 (Protokolle).

50.1. Der Weg einer E-Mail

Niveau: Einsteiger



- **MUA** (*Mail User Agent*): das Mailprogramm (Thunderbird, `mutt`, Webmail).
- **MTA** (*Mail Transfer Agent*): nimmt Nachrichten an und leitet sie weiter (Postfix, Exim, Sendmail). Findet den Zielserver über den `MX`-Eintrag im DNS.
- **MDA** (*Mail Delivery Agent*): legt die Nachricht im Postfach ab (Dovecot LDA, `procmail`).
- **Zugriffsprotokolle**: **IMAP** (Nachrichten bleiben auf dem Server, mehrere Geräte) und **POP3** (Abholen und Löschen).

Dienst	Port	Zweck
SMTP	25	Server-zu-Server-Zustellung
Submission	587 (STARTTLS)	Clients liefern ausgehende Mails beim eigenen Server ab (mit Anmeldung)
SMTPS	465	Submission mit implizitem TLS
IMAP / IMAPS	143 / 993	Postfachzugriff
POP3 / POP3S	110 / 995	Postfach abholen

50.2. Ein SMTP-Dialog von Hand

Niveau: Einsteiger

SMTP ist ein einfaches Textprotokoll – gut für das Verständnis und die Fehlersuche:

```
$ nc mail.example.lan 25
220 mail.example.lan ESMTP Postfix
EHLO client.example.lan
250-mail.example.lan
250-STARTTLS
250 SIZE 52428800
MAIL FROM:<anna@example.lan>
250 2.1.0 Ok
RCPT TO:<bob@example.lan>
250 2.1.5 Ok
DATA
354 End data with <CR><LF>.<CR><LF>
Subject: Test

Hallo Bob!
.
250 2.0.0 Ok: queued as 4F2A91234
QUIT
```

Die Statuscodes sind dreistellig: `2xx` Erfolg, `4xx` vorübergehender Fehler (später erneut versuchen), `5xx` dauerhafter Fehler (Rückläufer). Mit `openssl s_client -starttls smtp -connect mail.example.lan:25` testet man zusätzlich die TLS-Aushandlung.

50.3. Postfix einrichten

Niveau: Fortgeschritten

```
$ sudo apt install postfix mailutils          # "Internet Site" waehlen
$ sudo postconf -n                             # nur geaenderte Einstellungen
$ sudo postfix check
$ sudo systemctl reload postfix
```

Die Konfiguration liegt in `/etc/postfix/main.cf` (Parameter) und `master.cf` (Prozesse und Ports). Änderungen setzt man mit `postconf -e`:

```
$ sudo postconf -e 'myhostname = mail.example.lan'
$ sudo postconf -e 'mydomain = example.lan'
$ sudo postconf -e 'myorigin = $mydomain'
$ sudo postconf -e 'mydestination = $myhostname, localhost.$mydomain, $mydomain'
$ sudo postconf -e 'inet_interfaces = all'
$ sudo postconf -e 'mynetworks = 127.0.0.0/8 [::1]/128'
$ sudo postconf -e 'home_mailbox = Maildir/'
$ sudo postconf -e 'smtpd_tls_cert_file = /etc/letsencrypt/live/mail.example.lan/fullchain.
    pem'
$ sudo postconf -e 'smtpd_tls_key_file  = /etc/letsencrypt/live/mail.example.lan/privkey.pem'
$ sudo postconf -e 'smtpd_tls_security_level = may'
$ sudo postconf -e 'smtp_tls_security_level  = may'
```

- `mydestination`: für welche Domains der Server lokal zustellt.
- `mynetworks`: Rechner, die *ohne Anmeldung* über diesen Server senden dürfen. **Nie** zu großzügig setzen – sonst entsteht ein offenes Relay, das Spammer missbrauchen.
- `home_mailbox = Maildir/`: Ein Verzeichnis je Postfach, eine Datei je Nachricht (robuster als eine große `mbox`-Datei).

Aliase und Weiterleitungen in `/etc/aliases`:

```
# /etc/aliases
postmaster: root
root:       anna
info:       anna, bob
$ sudo newaliases                          # Datenbank neu erzeugen
$ echo "Testnachricht" | mail -s "Test" anna@example.lan
$ sudo tail -f /var/log/mail.log           # RHEL: /var/log/maillog, oder journalctl -u
    postfix
```

50.4. Warteschlange und Fehlersuche

Niveau: Fortgeschritten

Kann eine Nachricht nicht sofort zugestellt werden, verbleibt sie in der Warteschlange (*queue*) und wird mehrfach erneut versucht.

```
$ mailq                                    # Warteschlange anzeigen (= postqueue -p)
$ sudo postqueue -f                        # alles erneut versuchen
$ sudo postcat -q 4F2A91234                # Inhalt einer Nachricht
$ sudo postsuper -d 4F2A91234              # einzelne Nachricht loeschen
$ sudo postsuper -d ALL deferred           # ALLE zurueckgestellten loeschen (Vorsicht!)
$ sudo grep 4F2A91234 /var/log/mail.log    # Lebenslauf der Nachricht
```

In den Logzeilen entscheidet das Feld `status=`: `sent` (zugestellt), `deferred` (vorübergehend gescheitert, wird wiederholt), `bounced` (dauerhaft gescheitert). Der Zusatz in Klammern nennt die Ursache.

Meldung	Typische Ursache
`Connection timed out`	Zielserver nicht erreichbar; Firewall/Port 25 gesperrt (viele Provider sperren ausgehend Port 25)
`Host or domain name not found`	DNS-Problem, kein `MX`/`A`-Eintrag
`Relay access denied`	Absender darf nicht relayen; Anmeldung fehlt oder falsches `mynetworks`
`User unknown`	Empfänger existiert nicht
`Mailbox full`	Quota überschritten
`550 ... blocked / blacklisted`	IP-Adresse oder Domain steht auf einer Sperrliste

50.5. Dovecot: IMAP-Zugriff

Niveau: Fortgeschritten

```
$ sudo apt install dovecot-imapd dovecot-lmtpd
$ sudo doveconf -n                          # wirksame Konfiguration
$ sudo systemctl enable --now dovecot

# /etc/dovecot/conf.d/10-mail.conf
mail_location = maildir:~/Maildir

# /etc/dovecot/conf.d/10-ssl.conf
ssl = required
ssl_cert = </etc/letsencrypt/live/mail.example.lan/fullchain.pem
ssl_key  = </etc/letsencrypt/live/mail.example.lan/privkey.pem

# /etc/dovecot/conf.d/10-auth.conf
disable_plaintext_auth = yes
auth_mechanisms = plain login
```

Damit Clients sich beim Postfix (Submission) anmelden können, nutzt Postfix die Anmeldung über Dovecot:

```
# /etc/dovecot/conf.d/10-master.conf  (Abschnitt service auth)
unix_listener /var/spool/postfix/private/auth {
    mode = 0660
    user = postfix
    group = postfix
}

$ sudo postconf -e 'smtpd_sasl_type = dovecot'
$ sudo postconf -e 'smtpd_sasl_path = private/auth'
$ sudo postconf -e 'smtpd_sasl_auth_enable = yes'
$ sudo postconf -e 'smtpd_recipient_restrictions = permit_mynetworks,
    permit_sasl_authenticated, reject_unauth_destination'
```

Mit `openssl s_client -connect mail.example.lan:993` und `doveadm auth test anna` prüft man TLS und Benutzeranmeldung. Die letzte Regel (`reject_unauth_destination`) verhindert das offene Relay.

50.6. SPF, DKIM, DMARC und Betrieb

Niveau: Experte

Wer E-Mail an große Provider zustellen will, muss seine Absenderdomain absichern. Die Verfahren werden über DNS-`TXT`-Einträge veröffentlicht:

- **SPF** (*Sender Policy Framework*): Welche Server dürfen für die Domain senden? Beispiel: `v=spf1 mx -all`.
- **DKIM** (*DomainKeys Identified Mail*): Der Server signiert ausgehende Mails mit einem privaten Schlüssel (z. B. `opendkim`); der öffentliche Schlüssel steht im DNS.
- **DMARC**: Legt fest, was bei fehlgeschlagener SPF/DKIM-Prüfung geschehen soll (`none`, `quarantine`, `reject`), und fordert Berichte an. Der Eintrag ist ein `TXT`-Record unter `_dmarc` (siehe Listing).
- **Reverse-DNS (PTR)** der Server-IP muss zum Mailserver-Namen passen; ohne ihn lehnen viele Server die Mail ab.

```
$ dig +short TXT example.lan                        # SPF, z. B.: v=spf1 mx -all
$ dig +short TXT default._domainkey.example.lan     # DKIM-Schluessel
$ dig +short TXT _dmarc.example.lan                 # DMARC
"v=DMARC1; p=quarantine; rua=mailto:dmarc@example.lan"
$ dig +short -x 203.0.113.25                        # PTR der Server-IP
```

Spam- und Virenschutz: `rspamd` oder `SpamAssassin` bewerten Nachrichten; `ClamAV` sucht nach Schadsoftware; *Greylisting* und DNS-Sperrlisten (RBL) filtern vor der Annahme. `fail2ban` sperrt Angreifer bei wiederholten Anmeldefehlern.

> **Betriebsrisiko**
>
> Einen Mailserver zuverlässig zu betreiben ist anspruchsvoll: Ruf der IP-Adresse, Sperrlisten, Spam-Abwehr, Backups und Sicherheitsupdates erfordern laufende Pflege. Viele Organisationen nutzen deshalb gehostete Dienste oder fertige Pakete (z. B. Mailcow, Mail-in-a-Box). Für Systemmeldungen genügt oft ein schlankes Relay (`msmtp`, Postfix “satellite”) über einen Smarthost.

```
# Postfix als Satellit: alles ueber einen Smarthost senden
$ sudo postconf -e 'relayhost = [smtp.example.org]:587'
$ sudo postconf -e 'smtp_sasl_auth_enable = yes'
$ sudo postconf -e 'smtp_sasl_password_maps = hash:/etc/postfix/sasl_passwd'
$ sudo postconf -e 'smtp_tls_security_level = encrypt'
$ echo '[smtp.example.org]:587 benutzer:passwort' | sudo tee /etc/postfix/sasl_passwd
$ sudo chmod 600 /etc/postfix/sasl_passwd && sudo postmap /etc/postfix/sasl_passwd
$ sudo systemctl reload postfix
```

Übungen und Kontrollfragen

1. Welche Aufgaben haben MUA, MTA und MDA?

   Lösung: MUA: Mailprogramm des Benutzers; MTA: nimmt Mails an und leitet sie per SMTP weiter; MDA: legt sie im Postfach ab.

2. Wie findet ein MTA den Zielserver einer Domain?

   Lösung: Über den `MX`-Eintrag der Domain im DNS (mit Priorität), danach über dessen `A`/`AAAA`-Eintrag.

3. Was bedeutet `status=deferred` im Mail-Log?

   Lösung: Die Zustellung ist vorübergehend gescheitert; Postfix versucht es später erneut.

4. Was ist ein offenes Relay und wie vermeiden Sie es?

   Lösung: Ein Server, der Mails beliebiger Absender an beliebige Empfänger weiterleitet. Vermeidung: `mynetworks` eng halten und `reject_unauth_destination` verwenden, Authentifizierung für Clients.

5. Wofür stehen SPF, DKIM und DMARC?

   Lösung: SPF: erlaubte sendende Server; DKIM: Signatur der Nachrichten; DMARC: Richtlinie und Berichte bei Fehlschlägen beider Verfahren.

6. Wie sehen Sie, warum eine bestimmte Nachricht noch in der Warteschlange hängt?

   Lösung: `mailq` zeigt die Queue-ID und den Grund; mit `grep ID /var/log/mail.log` und `postcat -q ID` lässt sich der Verlauf untersuchen.

51. Datenbanken: MariaDB und PostgreSQL

Inhalt: Relationale Datenbanken und SQL-Grundlagen, Installation und Absicherung von MariaDB und PostgreSQL, Benutzer und Rechte, Datensicherung mit Dump, Leistungs- und Betriebsaspekte, Replikation im Überblick.

Stufe: [F] Fortgeschritten bis [X] Experte

Lernziele: Nach diesem Kapitel können Sie

- grundlegende SQL-Befehle auszuführen und eine Datenbank samt Benutzer anzulegen,
- MariaDB und PostgreSQL zu installieren, abzusichern und zu sichern,
- Verbindungen, Rechte und Zustand der Datenbankserver zu prüfen.

Vorkenntnisse: Kapitel 20 (Benutzer), Kapitel 32 (Dienste) und Kapitel 53 (Datensicherung).

51.1. Relationale Datenbanken und SQL

Niveau: Einsteiger

Eine relationale Datenbank speichert Daten in **Tabellen** mit Zeilen und Spalten. Die Abfragesprache heißt **SQL**. Anwendungen (Webshops, Wikis, Monitoring) nutzen sie als Speicher; Administratoren richten Server, Benutzer und Sicherungen ein.

```
-- Tabelle anlegen und fuellen
CREATE TABLE kunden (
    id      INT PRIMARY KEY AUTO_INCREMENT,   -- PostgreSQL: SERIAL
    name    VARCHAR(100) NOT NULL,
    stadt   VARCHAR(60),
    angelegt DATE DEFAULT CURRENT_DATE
);
INSERT INTO kunden (name, stadt) VALUES ('Anna Muster', 'Bonn'), ('Bob Beispiel', 'Berlin');

-- Abfragen
SELECT * FROM kunden;
SELECT name FROM kunden WHERE stadt = 'Bonn' ORDER BY name;
SELECT stadt, COUNT(*) AS anzahl FROM kunden GROUP BY stadt;
UPDATE kunden SET stadt = 'Koeln' WHERE id = 1;
DELETE FROM kunden WHERE id = 2;

-- Zwei Tabellen verbinden
SELECT k.name, b.betrag
FROM kunden k JOIN bestellungen b ON b.kunde_id = k.id;
```

Begriff	Bedeutung
Primärschlüssel	Eindeutige Kennung einer Zeile
Fremdschlüssel	Verweis auf den Primärschlüssel einer anderen Tabelle (Beziehung)
Index	Zusatzstruktur für schnelle Suche; kostet Speicher und Schreibzeit
Transaktion	Zusammengehörige Änderungen, die ganz oder gar nicht ausgeführt werden (ACID)
View	Gespeicherte Abfrage, die wie eine Tabelle benutzt wird

51.2. MariaDB

Niveau: Fortgeschritten

```
$ sudo apt install mariadb-server              # RHEL: mariadb-server
$ sudo systemctl enable --now mariadb
$ sudo mysql_secure_installation               # Root-Zugang absichern, Testdaten entfernen
$ sudo mariadb                                 # als Root per Unix-Socket anmelden
$ mariadb --version
$ ls /var/lib/mysql /etc/mysql/
```

Im Client legt man Datenbank und Benutzer an und vergibt gezielt Rechte:

```
MariaDB> CREATE DATABASE shop CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
MariaDB> CREATE USER 'shopuser'@'localhost' IDENTIFIED BY 'sicheres-passwort';
MariaDB> GRANT ALL PRIVILEGES ON shop.* TO 'shopuser'@'localhost';
MariaDB> FLUSH PRIVILEGES;
MariaDB> SHOW DATABASES;
MariaDB> SHOW GRANTS FOR 'shopuser'@'localhost';
MariaDB> SELECT user, host FROM mysql.user;
MariaDB> SHOW PROCESSLIST;
MariaDB> SHOW STATUS LIKE 'Threads_connected';
```

```
# /etc/mysql/mariadb.conf.d/50-server.cnf  (Ausschnitt)
[mysqld]
bind-address = 127.0.0.1           # nur lokal; fuer Netzzugriff gezielt aendern + Firewall
character-set-server = utf8mb4
innodb_buffer_pool_size = 1G       # bei dediziertem DB-Server ca. 50-70 % des RAM
max_connections = 150
slow_query_log = 1
slow_query_log_file = /var/log/mysql/slow.log
long_query_time = 2
```

51.3. PostgreSQL

Niveau: Fortgeschritten

```
$ sudo apt install postgresql                  # RHEL: postgresql-server, danach postgresql-
    setup --initdb
$ sudo systemctl enable --now postgresql
$ sudo -u postgres psql                        # als Superuser "postgres"
$ psql --version
$ ls /etc/postgresql/*/main/                   # Debian: postgresql.conf, pg_hba.conf
```

```
postgres=# CREATE ROLE shopuser LOGIN PASSWORD 'sicheres-passwort';
postgres=# CREATE DATABASE shop OWNER shopuser;
postgres=# \l                  -- Datenbanken
postgres=# \c shop             -- Datenbank wechseln
postgres=# \dt                 -- Tabellen
postgres=# \d kunden           -- Struktur einer Tabelle
postgres=# \du                 -- Rollen
postgres=# \q
$ psql -h localhost -U shopuser -d shop
```

PostgreSQL steuert den Zugang in `pg_hba.conf` (*host-based authentication*): Welcher Benutzer darf von welcher Adresse auf welche Datenbank mit welcher Methode?

```
# TYPE  DATABASE  USER       ADDRESS           METHOD
local   all       postgres                     peer
local   all       all                          scram-sha-256
host    shop      shopuser   192.168.10.0/24   scram-sha-256
host    all       all        0.0.0.0/0         reject
$ sudo -u postgres psql -c 'SELECT pg_reload_conf();'     # ohne Neustart uebernehmen
```

Netzzugriff erfordert außerdem `listen_addresses = '192.168.10.5'` in `postgresql.conf` und eine Firewallregel für TCP 5432 (MariaDB: 3306).

51.4. Sicherung und Wiederherstellung

Niveau: Fortgeschritten

Laufende Datenbanken dürfen **nicht** einfach per Dateikopie gesichert werden (inkonsistente Dateien). Üblich ist ein **logischer Dump**:

```
# MariaDB
$ mariadb-dump --single-transaction --routines --events shop | gzip > shop-$(date +%F).sql.gz
$ mariadb-dump --all-databases --single-transaction | gzip > alles.sql.gz
$ gunzip -c shop-2025-01-15.sql.gz | mariadb shop            # Wiederherstellen

# PostgreSQL
$ sudo -u postgres pg_dump -Fc shop > shop.dump               # komprimiertes Custom-Format
$ sudo -u postgres pg_dumpall > cluster.sql                   # inkl. Rollen
$ sudo -u postgres pg_restore -d shop --clean shop.dump
$ sudo -u postgres psql -d shop < shop.sql                    # fuer Klartext-Dumps
```

- Dumps per `cron` oder systemd-Timer erzeugen (Kapitel 36), mit `borg` oder `restic` offsite sichern (Kapitel 53).
- **Wiederherstellung regelmäßig testen** – eine nie getestete Sicherung ist keine.
- Für große Datenbanken: physische Backups (`mariabackup`, `pg_basebackup`) und kontinuierliche Archivierung der Transaktionslogs (WAL) für *Point-in-Time-Recovery*.

51.5. Betrieb, Leistung und Replikation

Niveau: Experte

```
# Zustand pruefen
$ sudo mariadb -e 'SHOW GLOBAL STATUS LIKE "Uptime"; SHOW VARIABLES LIKE "max_connections";'
$ sudo -u postgres psql -c 'SELECT datname, numbackends FROM pg_stat_database;'
$ sudo -u postgres psql -c 'SELECT pid, state, query FROM pg_stat_activity;'
$ sudo -u postgres psql -c "SELECT pg_size_pretty(pg_database_size('shop'));"
$ du -sh /var/lib/mysql /var/lib/postgresql

# Langsame Abfragen analysieren
MariaDB> EXPLAIN SELECT * FROM kunden WHERE stadt = 'Bonn';
postgres=# EXPLAIN ANALYZE SELECT * FROM kunden WHERE stadt = 'Bonn';
postgres=# VACUUM ANALYZE;                  -- Statistik und Aufraeumen (laeuft meist
    automatisch)
```

- **Speicher**: Datenbanken profitieren von viel RAM (Pufferpool, `shared_buffers`) und schnellen Platten (SSD/NVMe, eigenes Dateisystem, Kapitel 28).

- **Replikation**: Ein Primärserver sendet Änderungen an Replikate (Lesezugriff, Ausfallsicherheit). PostgreSQL: Streaming Replication; MariaDB: asynchrone Replikation oder Galera-Cluster. Die Replikation ersetzt *kein* Backup (fehlerhafte Löschungen werden mitrepliziert).
- **Hochverfügbarkeit**: Werkzeuge wie Patroni (PostgreSQL) oder Galera-Cluster (MariaDB); siehe Kapitel 60.
- **Sicherheit**: Datenbank nur im internen Netz erreichbar machen, eigene Benutzer pro Anwendung mit minimalen Rechten, TLS für Verbindungen, Standardkonten absichern.
- **Updates**: Hauptversionen erfordern Migration (`pg_upgrade`, `mariadb-upgrade`) – vorher sichern und testen.

SQL-Injection

Auf der Anwendungsseite ist *SQL-Injection* eine der häufigsten Schwachstellen: Eingaben des Benutzers dürfen nie direkt in SQL-Text eingefügt werden, sondern müssen über parametrisierte Abfragen (*prepared statements*) übergeben werden. Datenbankbenutzer mit minimalen Rechten begrenzen den Schaden.

Übungen und Kontrollfragen

1. Wie legen Sie in MariaDB einen Benutzer an, der nur auf die Datenbank `shop` zugreifen darf?

   Lösung: `CREATE USER 'u'@'localhost' IDENTIFIED BY '...'; GRANT ALL ON shop.* TO 'u'@'localhost';`

2. Wofür dient `pg_hba.conf`?

   Lösung: Sie legt fest, welche Benutzer von welchen Adressen auf welche Datenbanken mit welcher Authentifizierungsmethode zugreifen dürfen.

3. Warum ist das Kopieren des Datenverzeichnisses keine zuverlässige Sicherung einer laufenden Datenbank?

   Lösung: Die Dateien können sich während des Kopierens ändern und sind dann inkonsistent; stattdessen Dump oder ein konsistentes physisches Backup verwenden.

4. Ersetzt die Replikation das Backup?

   Lösung: Nein; Fehler wie versehentliches Löschen oder Verschlüsselung durch Schadsoftware werden an die Replikate weitergegeben.

5. Wie finden Sie langsame Abfragen?

   Lösung: Slow-Query-Log aktivieren (`long_query_time`), `EXPLAIN`/`EXPLAIN ANALYZE` verwenden und Indizes prüfen.

6. Welche Ports verwenden MariaDB und PostgreSQL standardmäßig?

   Lösung: 3306 (MariaDB) und 5432 (PostgreSQL).

Teil IV.

Sicherheit und Betrieb

52. Sicherheit und Härtung

Inhalt: Sicherheitsprinzipien, Updates, Angriffsfläche, Benutzer-Härtung, fail2ban, SELinux und AppArmor, Prüf- und Verschlüsselungswerkzeuge.
Stufe: [E] Einsteiger bis [X] Experte
Lernziele: Nach diesem Kapitel können Sie
- die Angriffsfläche eines Systems zu bewerten,
- Zugänge und Dienste zu härten,
- die Grundlagen von SELinux und AppArmor anzuwenden.

Vorkenntnisse: Kapitel 20, 22, 32 und 40.

52.1. Grundprinzipien

Niveau: Einsteiger

Sicherheit ist kein Produkt, sondern eine Folge vieler kleiner Maßnahmen. Für Linux-Systeme gelten fünf Grundregeln:

- **Minimalprinzip**: Nur installieren und starten, was gebraucht wird; jeder offene Port ist Angriffsfläche.
- **Geringste Rechte**: Benutzer und Dienste erhalten nur die nötigen Rechte (Kapitel 22); administrative Aufgaben per `sudo`.
- **Aktualität**: Sicherheitsupdates zeitnah einspielen.
- **Mehrschichtigkeit**: Firewall, Rechte, Zugriffskontrolle (MAC), Protokollierung und Backups ergänzen sich.
- **Nachvollziehbarkeit**: Ereignisse protokollieren und die Logs auch lesen (Kapitel 37).

52.2. Updates einspielen

Niveau: Einsteiger

```
# apt update && apt upgrade                    # Debian/Ubuntu
# apt install unattended-upgrades              # automatische Sicherheitsupdates
# dpkg-reconfigure -plow unattended-upgrades
# dnf upgrade --security                       # RHEL: nur Sicherheitsupdates
# dnf install dnf-automatic                    # automatische Updates (Timer aktivieren)
$ needs-restart                                # RHEL: Dienste/System neu starten? (dnf-utils
    )
$ [ -f /var/run/reboot-required ] && echo "Neustart noetig"   # Debian/Ubuntu
```

Nach Kernel-Updates ist ein Neustart nötig, um den neuen Kernel zu laden; bei Bibliotheksupdates müssen betroffene Dienste neu gestartet werden (`needrestart` unter Debian/Ubuntu).

52.3. Angriffsfläche prüfen

Niveau: Einsteiger

```
$ ss -tulpn                                    # welche Dienste lauschen von aussen?
$ systemctl list-unit-files --state=enabled    # welche Dienste starten automatisch?
$ find / -perm -4000 -type f 2>/dev/null       # SUID-Programme auflisten
$ find / -xdev -type f -perm -0002 2>/dev/null # fuer alle schreibbare Dateien
$ find / -xdev -nouser -o -nogroup 2>/dev/null # Dateien ohne gueltigen Besitzer
$ awk -F: '$3==0 {print $1}' /etc/passwd       # alle Konten mit UID 0 (nur root!)
$ awk -F: '$2=="" {print $1}' /etc/shadow      # Konten ohne Passwort (als root)
```

52.4. Benutzer und Anmeldung absichern

Niveau: Fortgeschritten

- Direkte `root`-Anmeldung über SSH sperren, Schlüssel statt Passwörter verwenden (Kapitel 42).
- Passwortrichtlinien: Mindestlänge und Ablauf über `/etc/login.defs`, `chage` und das PAM-Modul `pam_pwquality`.
- Nicht benötigte Konten sperren: `usermod -L -s /usr/sbin/nologin konto`.
- `sudo` fein steuern: Mit `visudo` nur bestimmte Befehle erlauben statt `ALL`.
- Mehrfaktor-Anmeldung (TOTP) per `libpam-google-authenticator` für SSH ist möglich.

```
# chage -l anna                                # Ablaufinformationen
# chage -M 90 -W 7 anna                        # Passwort max. 90 Tage, Warnung 7 Tage vorher
# visudo -f /etc/sudoers.d/webadmin
%webadmin ALL=(root) /usr/bin/systemctl restart nginx, /usr/bin/systemctl reload nginx
```

52.5. Brute-Force-Schutz mit fail2ban

Niveau: Fortgeschritten

`fail2ban` wertet Logdateien aus und sperrt Adressen mit wiederholten Fehlversuchen temporär in der Firewall.

```
# apt install fail2ban
# nano /etc/fail2ban/jail.local
[DEFAULT]
bantime  = 1h
findtime = 10m
maxretry = 5
ignoreip = 127.0.0.1/8 192.168.1.0/24

[sshd]
enabled = true
```

```
# systemctl enable --now fail2ban
# fail2ban-client status sshd                  # gesperrte Adressen
# fail2ban-client set sshd unbanip 203.0.113.7
```

52.6. Zwingende Zugriffskontrolle: SELinux und AppArmor

Niveau: Experte

Die normalen Zugriffsrechte sind *diskretionär*: Der Besitzer entscheidet. **Mandatory Access Control** (MAC) legt zusätzlich zentral fest, was ein Prozess tun darf – auch wenn er als `root` läuft. Wird ein Dienst kompromittiert, bleibt der Schaden begrenzt.

	SELinux	**AppArmor**
Verbreitung	RHEL, Fedora, CentOS	Ubuntu, Debian, openSUSE
Prinzip	Labels (Kontexte) an Dateien und Prozessen	Pfadbasierte Profile je Programm
Modi	`enforcing`, `permissive`, `disabled`	`enforce`, `complain`
Status	`getenforce`, `sestatus`	`aa-status`

```
$ getenforce                                    # SELinux-Modus
Enforcing
# setenforce 0                                  # vorlaeufig permissive (zur Fehlersuche)
$ ls -Z /var/www/html                           # Sicherheitskontext von Dateien
$ ps -eZ | grep nginx
# semanage fcontext -a -t httpd_sys_content_t "/srv/web(/.*)?"
# restorecon -Rv /srv/web                       # Kontexte anwenden
# setsebool -P httpd_can_network_connect on     # Schalter dauerhaft setzen
$ ausearch -m avc -ts recent                    # blockierte Zugriffe (Denials)
$ sealert -a /var/log/audit/audit.log           # Erklaerung und Loesungsvorschlag
```

> **Hinweis**
>
> SELinux nicht abschalten, wenn ein Dienst nicht funktioniert. Meist fehlt nur der richtige Kontext oder Schalter. Mit `setenforce 0` testen, ob SELinux die Ursache ist, dann die Meldung in `audit.log` auswerten und gezielt beheben.

52.7. Weitere Werkzeuge

Niveau: Experte

Werkzeug	**Zweck**
`auditd`, `auditctl`	Kernel-Audit: Dateizugriffe und Systemaufrufe protokollieren
`lynis audit system`	Prüfung des Systems gegen Härtungsempfehlungen
`aide`, `tripwire`	Integritätsprüfung von Dateien (Prüfsummen)
`rkhunter`, `chkrootkit`	Suche nach Rootkits
`cryptsetup` / LUKS	Festplattenverschlüsselung
`gpg`	Dateien verschlüsseln und signieren
`sysctl`-Härtung	z. B. `kernel.dmesg_restrict=1`, `net.ipv4.conf.all.rp_filter=1`

```
$ gpg --full-generate-key                       # Schluesselpaar
$ gpg -c geheim.txt                             # symmetrisch verschluesseln (-> geheim.txt.
    gpg)
$ gpg -d geheim.txt.gpg > geheim.txt            # entschluesseln
$ sha256sum image.iso                           # Pruefsumme bilden
$ sha256sum -c SHA256SUMS                       # Pruefsummen verifizieren
```

53. Datensicherung

Inhalt: 3-2-1-Regel, Sicherungsarten, `tar` und `rsync`, Skripte und Zeitsteuerung, `borg`, Snapshots, Wiederherstellung.
Stufe: [E] Einsteiger bis [X] Experte
Lernziele: Nach diesem Kapitel können Sie
- eine Backup-Strategie zu planen,
- Sicherungen mit `tar`, `rsync` und `borg` durchzuführen,
- die Wiederherstellung zu testen.

Vorkenntnisse: Kapitel 15, 36 und 40.

53.1. Warum und was sichern?

Niveau: Einsteiger

Datenverlust entsteht durch Hardwaredefekte, Bedienfehler, Schadsoftware oder Diebstahl. Ein Backup ist nur dann eines, wenn die **Wiederherstellung getestet** wurde.

Tipp
3-2-1-Regel: Mindestens *drei* Kopien der Daten, auf *zwei* verschiedenen Medien, davon *eine* an einem anderen Ort (Offsite oder Cloud, möglichst unveränderbar).

Typische Sicherungsziele:

- `/home` – Benutzerdaten
- `/etc` – Konfiguration
- `/var/lib/...` – Anwendungsdaten, Datenbanken (vorher konsistent exportieren!)
- `/srv`, `/opt` – Webinhalte und installierte Software
- Liste installierter Pakete: `dpkg --get-selections` bzw. `rpm -qa`

Nicht sichern muss man `/proc`, `/sys`, `/dev`, `/run`, `/tmp` und Caches.

53.2. Sicherungsarten

Niveau: Einsteiger

Art	Inhalt	Merkmal
Vollsicherung	Alle Daten	Einfache Wiederherstellung, viel Platz und Zeit
Differenziell	Änderungen seit der letzten Vollsicherung	Wachsend; Wiederherstellung: Voll + letzte Differenz
Inkrementell	Änderungen seit der letzten Sicherung (beliebiger Art)	Wenig Platz; Wiederherstellung braucht die ganze Kette
Snapshot	Zustand eines Dateisystems (LVM, Btrfs, ZFS)	Sofort, platzsparend, Grundlage für konsistente Sicherungen

53.3. Sichern mit tar und rsync

Niveau: Einsteiger

```
# tar -czpf /backup/etc-$(date +%F).tar.gz /etc        # /etc komprimiert, Rechte erhalten
# tar -tzf /backup/etc-2025-03-01.tar.gz | head         # Inhalt ansehen
# tar -xzpf /backup/etc-2025-03-01.tar.gz -C /tmp/test  # in Testordner entpacken

$ rsync -aHAX --delete /home/ /backup/home/             # Spiegelung, Rechte/ACLs/xattrs
$ rsync -a --link-dest=/backup/last /home/ /backup/$(date +%F)/   # Hardlink-Snapshots
```

Die letzte Variante legt für jeden Tag ein vollständig aussehendes Verzeichnis an; unveränderte Dateien sind per *Hardlink* nur einmal gespeichert. So entsteht ein platzsparendes, leicht durchsuchbares Backup ohne Spezialwerkzeug.

53.4. Skript und Zeitsteuerung

Niveau: Fortgeschritten

```
#!/bin/bash
# /usr/local/bin/backup-home.sh
set -euo pipefail
QUELLE=/home
ZIEL=/backup/home
HEUTE=$(date +%F)
mkdir -p "$ZIEL"

rsync -a --delete --link-dest="$ZIEL/latest" "$QUELLE/" "$ZIEL/$HEUTE/"
ln -sfn "$ZIEL/$HEUTE" "$ZIEL/latest"

# Aelter als 30 Tage entfernen
find "$ZIEL" -maxdepth 1 -type d -name '20*' -mtime +30 -exec rm -rf {} +
logger -t backup "Sicherung $HEUTE abgeschlossen"
```

```
# chmod 750 /usr/local/bin/backup-home.sh
# echo '30 2 * * * root /usr/local/bin/backup-home.sh' > /etc/cron.d/backup-home
```

53.5. Spezialwerkzeuge

Niveau: Fortgeschritten

Werkzeug	Eigenschaften
`borg`	Deduplizierung, Kompression, Verschlüsselung, Mounten von Archiven
`restic`	Ähnlich wie `borg`; viele Ziele (SFTP, S3, Cloud), ein einziges Programm
`duplicity`	Verschlüsselte, inkrementelle Sicherungen per GnuPG
`rear`	Komplette Systemwiederherstellung (Disaster Recovery)
`dump`, `xfsdump`	Dateisystemorientiert (ext4 / XFS)
`pg_dump`, `mysqldump`	Konsistente Datenbankexporte

```
$ borg init --encryption=repokey /backup/borg                    # Repository anlegen
$ borg create --stats --compression zstd /backup/borg::{hostname}-{now} /home /etc
$ borg list /backup/borg                                         # vorhandene Archive
$ borg extract /backup/borg::srv01-2025-03-01T02:30 home/anna    # einzelne Dateien
$ borg prune --keep-daily 7 --keep-weekly 4 --keep-monthly 6 /backup/borg
```

53.6. Konsistente Sicherungen und Abbilder

Niveau: Experte

Laufende Dienste ändern Dateien während der Sicherung. Konsistent wird sie mit *Snapshots*:

```
# lvcreate -L 5G -s -n snap_root /dev/vg0/lv_root       # LVM-Snapshot anlegen
# mount -o ro /dev/vg0/snap_root /mnt/snap
# rsync -a /mnt/snap/ /backup/root/
# umount /mnt/snap && lvremove -y /dev/vg0/snap_root    # aufraeumen

# mysqldump --single-transaction --all-databases | gzip > /backup/db-$(date +%F).sql.gz
# pg_dumpall | gzip > /backup/pg-$(date +%F).sql.gz
# btrfs subvolume snapshot -r /home /home/.snap-$(date +%F)     # Btrfs
```

Ein Komplettabbild einer Platte (`dd`, `partclone`, `clonezilla`) sichert Partitionstabelle, Bootloader und Daten, ist aber groß und unflexibel. Für Server verwendet man besser Konfigurationsmanagement (`ansible`) plus Datensicherung: Das System ist dann in Minuten neu aufgebaut.

Wiederherstellung üben

1. Regelmäßig eine *Stichprobe* in ein Testverzeichnis zurückspielen und mit `diff -r` bzw. `sha256sum` vergleichen.
2. Einmal komplett einen leeren Rechner/eine VM aus dem Backup aufbauen und die benötigte Zeit messen.
3. Zugangsdaten und Schlüssel des Backups getrennt aufbewahren: Ohne den Schlüssel ist ein verschlüsseltes Backup wertlos.
4. Das Backup selbst überwachen: Fehlgeschlagene Läufe müssen auffallen (Mail, Monitoring).

54. Leistung und Fehlersuche

Inhalt: Systemlast, systematische Fehlersuche, Leistungswerkzeuge, offene Dateien, Limits, `strace`.
Stufe: [E] Einsteiger bis [X] Experte
Lernziele: Nach diesem Kapitel können Sie

- Last, Speicher und Ein-/Ausgabe zu analysieren,
- Probleme systematisch einzugrenzen,
- Systemaufrufe und offene Dateien zu untersuchen.

Vorkenntnisse: Kapitel 24, 37 und 39.

54.1. Systemlast auf einen Blick

Niveau: Einsteiger

```
$ uptime                                  # Laufzeit und Load Average
 14:05:12 up 12 days,  3:41,  2 users,  load average: 0.52, 0.48, 0.45
$ nproc                                   # Anzahl CPU-Kerne
$ top                                     # interaktiv; Taste 1: einzelne Kerne, M: nach
    Speicher, P: nach CPU
$ htop                                    # komfortabler (Paket htop)
$ free -h                                 # Arbeitsspeicher und Swap
$ df -h                                   # Plattenbelegung
```

Der **Load Average** zeigt die mittlere Anzahl lauffähiger bzw. auf Platten-E/A wartender Prozesse für 1, 5 und 15 Minuten. Als Faustregel ist die Last unkritisch, solange sie unter der Zahl der Kerne liegt. Ein langsam steigender 15-Minuten-Wert deutet auf ein dauerhaftes Problem, ein hoher 1-Minuten-Wert auf eine Spitze.
In der Speicheranzeige zählt nicht die Spalte `free`, sondern `available`: Linux nutzt freien RAM als Dateicache und gibt ihn bei Bedarf sofort wieder her. Ein “voller” Speicher ist daher normal.

54.2. Systematisches Vorgehen

Niveau: Einsteiger

1. **Symptom genau beschreiben**: Was geht nicht? Seit wann? Was wurde zuletzt geändert?
2. **Reproduzierbar machen**: Tritt es immer oder nur manchmal auf, bei allen Benutzern?
3. **Logs lesen**: `journalctl -xe`, `journalctl -b -p err`, Anwendungslogs.
4. **Ressourcen prüfen**: CPU, Speicher, Platte, Inodes (`df -i`), offene Dateien, Netzwerk.
5. **Hypothese testen**: Immer *eine* Änderung auf einmal; Ergebnis notieren.
6. **Dokumentieren**: Lösung festhalten, damit sie beim nächsten Mal schneller gefunden wird.

Symptom	Erste Schritte
System langsam	`top`: CPU-, Speicher- oder E/A-Wartezeit (`wa`)? `iostat`, `free`, Swap-Aktivität prüfen
"No space left on device"	`df -h` (Platz), `df -i` (Inodes), `du -sh /var/*`, gelöschte, aber offene Dateien: `lsof +L1`
Dienst startet nicht	`systemctl status`, `journalctl -u dienst -b`, Konfigurationstest (`nginx -t`, `sshd -t`), Port belegt? (`ss -tlpn`)
"Permission denied"	`ls -l`, `id`, Rechte *aller* Pfadelemente (`namei -l pfad`), ACLs, SELinux/AppArmor
Kein Netz	Schicht für Schicht: Link, Adresse, Gateway, DNS (Kapitel 39)
Prozess hängt	`ps`, `strace -p PID`, `kill -TERM`, im Notfall `kill -KILL`
Speicher knapp (OOM)	`journalctl -k | grep -i oom`; Prozess mit hohem Verbrauch suchen

54.3. Leistungsanalyse

Niveau: Fortgeschritten

```
$ vmstat 2 5                              # alle 2 s, 5 Messungen: CPU, Speicher, Swap, E/A
$ iostat -xz 2                            # Platten-Auslastung (%util, await)   (Paket
    sysstat)
$ mpstat -P ALL 2                         # CPU je Kern
$ sar -u 1 3                              # CPU-Historie; sar -r Speicher, sar -n DEV Netz
$ pidstat -d 2                            # E/A je Prozess
$ iotop -o                                # Prozesse mit Plattenzugriffen
$ ps aux --sort=-%mem | head              # groesste Speicherverbraucher
$ ps aux --sort=-%cpu | head              # groesste CPU-Verbraucher
$ ss -s                                   # Netzwerkverbindungen
$ iftop -i eth0                           # Bandbreite je Verbindung
```

Kennzahl	Bedeutung
`us`, `sy`	CPU-Zeit in Anwendungen / im Kernel
`wa`	CPU wartet auf Platten-E/A; dauerhaft hoch $\Rightarrow$ Plattenproblem
`si`, `so` (`vmstat`)	Swap-in/-out; dauerhaft $> 0 \Rightarrow$ zu wenig RAM
`%util` (`iostat`)	Auslastung der Platte; nahe 100 % $\Rightarrow$ Engpass
`await`	Mittlere Antwortzeit der Platte in ms

54.4. Offene Dateien und Verbindungen

Niveau: Fortgeschritten

```
$ lsof -p 1234                            # Dateien eines Prozesses
$ lsof -i :80                             # wer nutzt Port 80?
$ lsof -u anna                            # Dateien eines Benutzers
$ lsof +D /var/log                        # wer hat Dateien in diesem Verzeichnis offen?
$ lsof +L1                                # geloeschte, aber noch offene Dateien (belegen
    Platz!)
$ fuser -v /var/log/syslog
$ ls -l /proc/1234/fd | head              # offene Dateideskriptoren
$ cat /proc/1234/status | grep -E 'VmRSS|Threads'
```

54.4.1. Ressourcenlimits [F]

```
$ ulimit -a                                # Limits der aktuellen Shell
$ ulimit -n                                # max. offene Dateien (Standard 1024)
# nano /etc/security/limits.d/90-app.conf
meinapp  soft  nofile  65536
meinapp  hard  nofile  65536
# (fuer systemd-Dienste stattdessen LimitNOFILE= in der Unit setzen)
```

54.5. Systemaufrufe und Kernel-Werkzeuge

Niveau: Experte

```
$ strace -f -o trace.txt ./programm       # alle Systemaufrufe aufzeichnen
$ strace -p 1234 -e trace=network         # laufenden Prozess beobachten, nur Netzwerkaufrufe
$ strace -c ./programm                    # Zusammenfassung: wie oft, wie lange
$ ltrace ./programm                       # Bibliotheksaufrufe
$ perf top                                # CPU-Profil des Gesamtsystems
$ perf record -g -p 1234 -- sleep 30 && perf report
$ dmesg -T | tail -50                     # Kernelmeldungen
$ sysctl -a | grep vm.                    # Kernelparameter zum Speichermanagement
$ cat /proc/meminfo | head -5
$ cat /proc/cpuinfo | grep 'model name' | head -1
$ lspci; lsusb; lsmod                     # Hardware und geladene Kernelmodule
$ lshw -short                             # Hardwareuebersicht (root)
```

> **Hinweis**
>
> `strace` verlangsamt Programme stark und zeigt vor allem *Dateizugriffe, Netzwerkaufrufe und Fehler* (`ENOENT`: Datei nicht gefunden, `EACCES`: Zugriff verweigert). Bei der Frage "Warum findet das Programm seine Konfiguration nicht?" ist es unschlagbar.

Der Weg zur Hilfe

Hilfe finden: `man`, `... --help`, Dokumentation unter `/usr/share/doc`, die Wikis der Distributionen (ArchWiki, Ubuntu Wiki, Red Hat Dokumentation), Foren und Mailinglisten. Eine gute Frage enthält Distribution und Version, den genauen Befehl, die vollständige Fehlermeldung (kein Screenshot), relevante Logauszüge und was bereits versucht wurde.

55. Fehlersuche an Praxisszenarien

Inhalt: Methodisches Vorgehen bei Störungen und acht typische Szenarien: System bootet nicht, Platte voll, Dienst startet nicht, Netzwerk nicht erreichbar, hohe Last, Speicher knapp, Login nicht möglich, Zertifikat abgelaufen.

Stufe: [F] Fortgeschritten bis [X] Experte

Lernziele: Nach diesem Kapitel können Sie

- ein strukturiertes Vorgehen bei Störungen anzuwenden (beobachten, eingrenzen, beheben, dokumentieren),
- typische Ursachen in den Szenarien zu erkennen und mit den passenden Werkzeugen zu bestätigen,
- Notfallzugänge und Rettungsmaßnahmen sicher einzusetzen.

Vorkenntnisse: Kapitel 54 (Leistung und Fehlersuche), Kapitel 32 (systemd), Kapitel 37 (Protokolle) und Kapitel 41 (Netzwerkwerkzeuge).

55.1. Methodik

Niveau: Einsteiger

1. **Symptom genau erfassen**: Was geht nicht? Seit wann? Wer ist betroffen? Was wurde zuletzt geändert (Update, Konfiguration, Hardware)?
2. **Hypothesen bilden** und mit dem *einfachsten* Test überprüfen – von den wahrscheinlichsten, billigsten Ursachen zu den seltenen.
3. **Eingrenzen**: Ebene (Hardware, Betriebssystem, Netz, Dienst, Anwendung), Ort (ein Host, alle?), Zeitpunkt (seit wann, periodisch?).
4. **Nur eine Sache ändern** und die Wirkung beobachten; Änderungen notieren, Konfigurationen vorher sichern.
5. **Beheben und verifizieren**: Funktioniert es wirklich aus Sicht des Benutzers?
6. **Ursache dokumentieren** (Post-Mortem): Wie lässt sich das Problem künftig vermeiden oder früher erkennen?

```
# Erste Orientierung auf einem gestoerten Server (ca. 60 Sekunden)
$ uptime                                    # Last, Laufzeit
$ dmesg -T | tail -30                       # Kernel: Hardware-, Dateisystem-, OOM-Fehler
$ journalctl -p err -b --no-pager | tail -30   # Fehler seit dem Boot
$ systemctl --failed                        # fehlgeschlagene Dienste
$ df -hT; df -i                             # Platz und Inodes
$ free -h                                   # Speicher
$ ss -tlnp                                  # lauschende Dienste
$ top -bn1 | head -15                       # Prozesse
```

55.2. Szenario 1: Das System bootet nicht

Niveau: Fortgeschritten

Symptom: Nach einem Update bleibt der Bildschirm schwarz, es erscheint ein `grub>`-Prompt, oder das System landet in der Notfall-Shell (`emergency mode`).

- **GRUB-Prompt**: GRUB findet seine Konfiguration nicht. Von einem Live-System booten, Partitionen einhängen (`mount`, `chroot`) und GRUB neu installieren.
- **Emergency Mode wegen `fstab`**: Ein nicht vorhandenes Laufwerk blockiert den Start. Meldung `Dependency failed for /data`. Mit `journalctl -xb` prüfen, in `/etc/fstab` Option `nofail` setzen oder den Eintrag auskommentieren.
- **Neuer Kernel startet nicht**: Im GRUB-Menü unter "Erweiterte Optionen" den vorigen Kernel wählen.
- **Dateisystemfehler**: `fsck` im Rettungsmodus (Dateisystem *nicht* eingehängt).

```
# Vom Live-System (oder Rescue-Modus) aus reparieren
$ sudo mount /dev/vg0/root /mnt
$ sudo mount /dev/sda1 /mnt/boot/efi        # UEFI: EFI-Partition
$ for d in dev proc sys run; do sudo mount --rbind /$d /mnt/$d; done
$ sudo chroot /mnt
# update-grub && grub-install /dev/sda      # Debian; RHEL: grub2-mkconfig -o /boot/grub2/
    grub.cfg
# dpkg --configure -a                       # unterbrochene Paket-Updates abschliessen
# exit
# Kernelparameter einmalig im GRUB-Menue (Taste e): ... systemd.unit=rescue.target  |  init=/
    bin/bash
```

Details zum Bootvorgang: Kapitel 54. Bei vergessenem Root-Passwort: Kernelparameter `rd.break` (RHEL) bzw. `init=/bin/bash` (Debian), Wurzel schreibbar einhängen, `passwd`, bei SELinux `touch /.autorelabel`.

55.3. Szenario 2: Die Platte ist voll

Niveau: Fortgeschritten

Symptom: Anwendungen melden `No space left on device`, Dienste stürzen ab, Anmeldung ist träge.

```
$ df -h                                     # welches Dateisystem?
$ df -i                                     # Inodes erschoepft? (viele Mini-Dateien)
$ sudo du -xh --max-depth=1 / 2>/dev/null | sort -rh | head      # grosse Verzeichnisse
$ sudo du -xh /var --max-depth=2 | sort -rh | head -15
$ sudo ncdu -x /                            # interaktiv (Paket ncdu)
$ sudo find / -xdev -type f -size +500M -exec ls -lh {} + 2>/dev/null
$ sudo journalctl --disk-usage
$ sudo journalctl --vacuum-size=200M        # Journal verkleinern
$ sudo apt clean                            # Paket-Cache
$ sudo lsof +L1                             # geloeschte, aber noch geoeffnete Dateien
```

Wenn `df` voll anzeigt, `du` aber weniger findet: Eine Datei wurde gelöscht, ist aber noch von einem Prozess geöffnet (`lsof +L1`). Der Platz wird erst frei, wenn der Prozess endet oder die Datei schließt (`systemctl restart dienst`). Typische Verursacher: nicht rotierte Logs, Core-Dumps, Docker-Images (`docker system df`), alte Kernel, Datenbank-Binlogs, Mail-Warteschlangen. Langfristig: Partitionen trennen (`/var`), Monitoring (Kapitel 57), Logrotation prüfen.

55.4. Szenario 3: Ein Dienst startet nicht

Niveau: Fortgeschritten

```
$ systemctl status nginx                        # Zustand und letzte Logzeilen
$ journalctl -u nginx -b --no-pager | tail -30
$ journalctl -xeu nginx                         # mit Erlaeuterungen
$ sudo nginx -t                                 # dienstspezifischer Konfigurationstest
$ ss -tlnp | grep ':80'                         # Port bereits belegt?
$ systemctl cat nginx                           # wirksame Unit-Datei
$ systemd-analyze verify /etc/systemd/system/mein.service
$ sudo -u www-data ls /var/www/                 # Rechte des Dienstbenutzers pruefen
$ sudo ausearch -m avc -ts recent               # SELinux-Ablehnungen
$ getenforce
```

Meldung	Wahrscheinliche Ursache
`Address already in use`	Anderer Prozess belegt den Port (`ss -tlnp`)
`Permission denied`	Dateirechte, falscher Benutzer, SELinux/AppArmor
`No such file or directory`	Pfad, fehlende Datei oder Bibliothek (`ldd`)
`Failed to start ... start request repeated too quickly`	Dienst stürzt wiederholt ab; `journalctl -u` zeigt den Grund
`Exit code=203/EXEC`	Programm nicht gefunden oder nicht ausführbar
`Exit code=217/USER`	Benutzer der Unit existiert nicht

55.5. Szenario 4: Netzwerk nicht erreichbar

Niveau: Fortgeschritten

Vorgehen von unten nach oben (Kapitel 41):

```
$ ip -br link; ip -br addr                  # Link und Adresse
$ ip route                                  # Standardroute?
$ ping -c2 192.168.10.1                     # Gateway
$ ping -c2 9.9.9.9                          # Internet per IP
$ getent hosts example.org                  # Namensaufloesung
$ resolvectl status | head -20              # Welcher DNS-Server?
$ sudo nft list ruleset | head -50          # Firewall blockiert?
$ ss -tlnp | grep :22                       # lauscht der Dienst?
$ sudo tcpdump -ni eth0 host 192.168.10.50 and port 22    # kommt etwas an?
```

- Ping zum Gateway klappt, ins Internet nicht: Routing oder NAT am Gateway.
- Ping per IP klappt, Namen nicht: DNS (Resolver, `/etc/resolv.conf`, Kapitel 44).
- Dienst lokal erreichbar (`curl localhost`), von außen nicht: Firewall, falsche Bindeadresse (`127.0.0.1` statt `0.0.0.0`), Security Group.
- Plötzlich langsam oder abbrechend: Duplex-/Kabelfehler (`ethtool`, `ip -s link`), MTU, DNS-Timeouts.

55.6. Szenario 5: Hohe Last

Niveau: Fortgeschritten

```
$ uptime                                    # load average: 18.2, 12.4, 6.1  (bei 4 Kernen!)
$ top -bn1 | head -15                       # wer verbraucht CPU?
$ vmstat 1 5                                # r (laufbereit), b (blockiert), wa (I/O-
    Wartezeit), si/so (Swap)
$ iostat -xz 1 3                            # Platten: %util, await
$ sudo iotop -oP                            # wer verursacht I/O?
$ pidstat 1 5                               # pro Prozess
$ ps -eo pid,stat,wchan:20,cmd | awk '$2 ~ /D/'     # Prozesse im Zustand D (wartend auf I/O)
```

Die **Last** (*load average*) zählt laufbereite und auf I/O wartende Prozesse. Sie muss relativ zur Kernanzahl (`nproc`) gelesen werden. Entscheidend ist die Ursache:

- **Hohe CPU-Nutzung** (`us`/`sy` hoch): ein Prozess rechnet, Endlosschleife, zu viele Anfragen. `perf top` oder `strace -p PID` zeigen, was er tut.
- **Hohe I/O-Wartezeit** (`wa` hoch): Platte überlastet, defekt oder NFS hängt (Kapitel 49). `smartctl -a`, `dmesg`.
- **Swapping** (`si`/`so` $>$ 0 dauerhaft): Speicher knapp; die Maschine wird extrem langsam.
- **Viele Verbindungen**: Angriff oder Lastspitze; `ss -s`, Zugriffslogs prüfen, Rate-Limits (Kapitel 43).

55.7. Szenario 6: Speicher knapp und OOM-Killer

Niveau: Fortgeschritten

```
$ free -h
$ ps aux --sort=-%mem | head -10            # groesste Verbraucher
$ sudo dmesg -T | grep -iE 'out of memory|oom-kill|killed process'
$ sudo journalctl -k | grep -i oom
$ cat /proc/meminfo | head -5
$ sudo smem -rtk | head                     # anteilig (PSS), Paket smem
$ systemctl status mein-dienst              # wurde er vom OOM-Killer beendet?  (Result: oom
    -kill)
```

Wenn der Kernel keinen Speicher mehr vergeben kann, beendet der **OOM-Killer** einen Prozess (häufig den größten, z. B. die Datenbank). Abhilfe: Speicherleck beheben, Limits setzen (`MemoryMax=` in systemd-Units, Kapitel 33), mehr RAM/Swap, Anwendung korrekt dimensionieren (z. B. Pufferpool der Datenbank, Kapitel 51). `available` in `free` ist maßgeblich, nicht `free` – Linux nutzt freien Speicher für den Cache.

55.8. Szenario 7: Anmeldung nicht möglich

Niveau: Fortgeschritten

- **SSH-Schlüssel wird abgelehnt**: Rechte von `~/.ssh` (700) und `authorized_keys` (600), Besitzer, Heimverzeichnis nicht gruppenschreibbar. Auf dem Server: `journalctl -u ssh`; auf dem Client: `ssh -vvv`.
- **Konto gesperrt/abgelaufen**: `passwd -S anna`, `chage -l anna`, `faillock --user anna` (bzw. `pam_tally2`); entsperren mit `faillock --user anna --reset`.
- **Shell ungültig**: Eintrag in `/etc/passwd` verweist auf nicht vorhandene oder nicht in `/etc/shells` stehende Shell.
- **Zeit abweichend**: Bei Kerberos/LDAP oder TOTP führt eine falsche Uhr zu Anmeldefehlern (Kapitel 45).
- **Zentrale Anmeldung**: `sssctl domain-status`, `getent passwd anna`; Verzeichnisserver erreichbar? (Kapitel 21).
- **Aus dem System ausgesperrt**: Notfallzugang über Konsole des Hypervisors/Cloud-Anbieters oder Rescue-Modus; deshalb Notfallzugänge vor Änderungen testen.

55.9. Szenario 8: Zertifikat abgelaufen und Weiteres

Niveau: Experte

Symptom: Browser zeigt `NET::ERR_CERT_DATE_INVALID`, `curl` meldet `certificate has expired`, Dienst-zu-Dienst-Verbindungen schlagen plötzlich fehl.

```
$ echo | openssl s_client -connect www.example.org:443 -servername www.example.org 2>/dev/
    null \
    | openssl x509 -noout -dates -subject
$ sudo certbot certificates                  # lokale Let's-Encrypt-Zertifikate
$ sudo certbot renew --dry-run               # Erneuerung funktioniert?
$ systemctl list-timers | grep -i certbot
$ sudo journalctl -u certbot --since "7 days ago"
$ sudo systemctl reload nginx                # Zertifikat neu laden (Dienst liest nur beim
    Start/Reload)
```

Häufige Ursachen: Erneuerung scheitert (Port 80 gesperrt, DNS geändert, Timer fehlt), der Dienst wurde nach der Erneuerung nicht neu geladen, falsche Systemzeit. Vorbeugen: Ablauf überwachen (Kapitel 58, Kapitel 57), Deploy-Hooks nutzen.

> **Immer dokumentieren**
>
> Schreiben Sie während der Störung mit: Zeitpunkt, Beobachtung, ausgeführte Befehle, Ergebnis. Das hilft bei Übergaben, bei der Ursachenanalyse und beim Abschlussbericht. Ändern Sie nie mehrere Dinge gleichzeitig, und notieren Sie Ausgangszustände, damit Sie zurückrollen können.

Übungen und Kontrollfragen

1. Df zeigt 100 % belegt, `du` summiert aber nur 60 %. Woran kann das liegen?

   Lösung: Gelöschte, aber noch von Prozessen geöffnete Dateien; `lsof +L1` findet sie, ein Neustart des Prozesses gibt den Platz frei. Auch reservierte Blöcke oder andere Mounts verdeckt durch Mountpoints sind möglich.

2. Ein Dienst meldet `Address already in use`. Wie gehen Sie vor?

   Lösung: Mit `ss -tlnp | grep :PORT` den belegenden Prozess ermitteln und diesen beenden oder den Port ändern.

3. Welche drei Werte aus `vmstat` zeigen, ob die Ursache hoher Last CPU, I/O oder Speicher ist?

   Lösung: `us`/`sy` (CPU), `wa` (I/O-Wartezeit) und `si`/`so` (Swapping).

4. Woran erkennen Sie einen OOM-Kill?

   Lösung: Meldungen `Out of memory: Killed process` in `dmesg` bzw. `journalctl -k`, bei systemd `Result: oom-kill`.

5. Das System landet nach dem Booten im Emergency Mode. Welche Datei ist häufig die Ursache?

   Lösung: `/etc/fstab`: Ein Eintrag für ein fehlendes Laufwerk oder falsche UUID; Optionen wie `nofail` oder korrigierter Eintrag helfen.

6. Warum soll man bei der Fehlersuche nur eine Änderung gleichzeitig vornehmen?

   Lösung: Nur so lässt sich die Wirkung einer Änderung eindeutig zuordnen und notfalls rückgängig machen.

56. Automatisierung mit Ansible

Inhalt: Konfigurationsmanagement und Infrastruktur als Code, Inventar, Ad-hoc-Befehle, Playbooks, Variablen, Handler, Templates, Rollen, Vault für Geheimnisse, Idempotenz und Best Practices.

Stufe: [F] Fortgeschritten bis [X] Experte

Lernziele: Nach diesem Kapitel können Sie

- den Nutzen von Idempotenz und deklarativer Konfiguration zu erklären,
- ein Inventar und einfache Playbooks mit Variablen, Handlern und Templates zu schreiben,
- wiederverwendbare Rollen aufzubauen und Geheimnisse mit Vault zu schützen.

Vorkenntnisse: Kapitel 42 (SSH), Kapitel 19 (Paketverwaltung), Kapitel 32 (systemd) und Kapitel 16 (Git).

56.1. Warum Automatisierung?

Niveau: Einsteiger

Wer mehr als einen Server betreut, wiederholt dieselben Schritte: Pakete installieren, Konfigurationsdateien kopieren, Dienste starten. Von Hand führt das zu Abweichungen (“Schneeflocken-Server”), Fehlern und fehlender Dokumentation. **Konfigurationsmanagement** beschreibt den gewünschten Zustand in Textdateien (*Infrastructure as Code*), die man versioniert (Kapitel 16), prüft und beliebig oft anwenden kann.

- **Ansible** arbeitet *agentenlos*: Es verbindet sich per SSH, führt Module aus und trennt wieder. Auf den Zielsystemen ist nur Python nötig.
- **Deklarativ**: Man beschreibt *was* gelten soll (“Paket nginx ist installiert”), nicht *wie* man dorthin kommt.
- **Idempotent**: Mehrfaches Ausführen ergibt dasselbe Ergebnis; ist der Zustand schon erreicht, passiert nichts (`ok` statt `changed`).
- Alternativen: Puppet, Chef, Salt (mit Agent), Terraform (für Infrastruktur, Kapitel 61).

56.2. Installation, Inventar und Ad-hoc-Befehle

Niveau: Einsteiger

```
$ sudo apt install ansible              # oder: pipx install ansible-core
$ ansible --version
$ mkdir -p ~/ansible && cd ~/ansible
```

Das **Inventar** listet die verwalteten Rechner, gruppiert nach Rolle:

```
# inventory.ini
[webserver]
web1.example.lan
web2.example.lan ansible_port=2222

[datenbank]
db1.example.lan ansible_user=admin

[produktion:children]
webserver
datenbank

[all:vars]
ansible_user=anna
ansible_python_interpreter=/usr/bin/python3
```

```
$ ansible all -i inventory.ini -m ping                      # Erreichbarkeit und Python
$ ansible webserver -i inventory.ini -m command -a 'uptime'
$ ansible webserver -i inventory.ini -b -m apt -a 'name=htop state=present update_cache=yes'
$ ansible all -i inventory.ini -b -m service -a 'name=chrony state=started enabled=yes'
$ ansible-inventory -i inventory.ini --list
$ ansible all -i inventory.ini -m setup -a 'filter=ansible_distribution*'   # Fakten
```

Mit `-b` (*become*) führt Ansible Befehle mit `sudo` aus. Ad-hoc-Befehle eignen sich für Einzelaktionen; Wiederholbares gehört in Playbooks.

56.3. Playbooks

Niveau: Fortgeschritten

Ein **Playbook** ist eine YAML-Datei mit einem oder mehreren *Plays*. Jeder Play wählt Hosts und führt eine Liste von *Tasks* aus; jeder Task ruft ein *Modul* auf.

```
# webserver.yml
---
- name: Webserver einrichten
  hosts: webserver
  become: true
  vars:
    web_port: 8080
    web_root: /var/www/html
  tasks:
    - name: Pakete installieren
      ansible.builtin.apt:
        name:
          - nginx
          - curl
        state: present
        update_cache: true

    - name: Konfiguration aus Vorlage erzeugen
      ansible.builtin.template:
        src: nginx-site.conf.j2
        dest: /etc/nginx/conf.d/site.conf
        owner: root
        group: root
        mode: "0644"
        validate: nginx -t -c /etc/nginx/nginx.conf
      notify: Nginx neu laden

    - name: Startseite kopieren
      ansible.builtin.copy:
        content: "Hallo von {{ inventory_hostname }}\n"
        dest: "{{ web_root }}/index.html"

    - name: Dienst aktivieren und starten
      ansible.builtin.service:
        name: nginx
        state: started
        enabled: true

  handlers:
    - name: Nginx neu laden
      ansible.builtin.service:
        name: nginx
        state: reloaded
```

```
$ ansible-playbook -i inventory.ini webserver.yml --syntax-check
$ ansible-playbook -i inventory.ini webserver.yml --check --diff    # Probelauf mit
    Unterschieden
$ ansible-playbook -i inventory.ini webserver.yml                   # ausfuehren
$ ansible-playbook -i inventory.ini webserver.yml --limit web1.example.lan
$ ansible-playbook -i inventory.ini webserver.yml --tags config -v
```

- **Module** erledigen die Arbeit idempotent: `apt`, `dnf`, `package`, `copy`, `template`, `file`, `lineinfile`, `user`, `group`, `service`, `systemd`, `cron`, `git`, `command`, `shell`, `uri`, `ufw`, `sysctl`, `mount` u. v. m. Die Dokumentation steht lokal: `ansible-doc ansible.builtin.template`.
- **Handler** laufen nur, wenn ein Task mit `notify` etwas geändert hat, und erst am Ende des Plays – ideal zum Neuladen von Diensten.

- **`command/shell`** sind *nicht* idempotent; mit `creates:`, `when:` oder `changed_when:` absichern, oder besser ein passendes Modul nutzen.
- **`--check --diff`** zeigt vorab, was sich ändern würde.
- **YAML**: Einrückung mit Leerzeichen (keine Tabs); Doppelpunkte oder `{{` am Wertanfang in Anführungszeichen setzen.

56.4. Variablen, Bedingungen, Schleifen und Templates

Niveau: Fortgeschritten

```
# Variablen: group_vars/webserver.yml, host_vars/web1.example.lan.yml
web_port: 8080
benutzer:
  - name: anna
    gruppen: [sudo, www-data]
  - name: bob
    gruppen: [www-data]

# Im Playbook
- name: Benutzer anlegen
  ansible.builtin.user:
    name: "{{ item.name }}"
    groups: "{{ item.gruppen | join(',') }}"
    append: true
    shell: /bin/bash
  loop: "{{ benutzer }}"

- name: Nur auf Debian-Systemen
  ansible.builtin.apt:
    name: unattended-upgrades
  when: ansible_facts['os_family'] == 'Debian'

- name: Ergebnis merken
  ansible.builtin.command: systemctl is-active nginx
  register: nginx_status
  changed_when: false
  failed_when: false

- ansible.builtin.debug:
    msg: "nginx ist {{ nginx_status.stdout }}"
```

Templates (Jinja2) erzeugen Konfigurationsdateien aus Variablen und Fakten:

```
# templates/nginx-site.conf.j2
# Verwaltet durch Ansible - Aenderungen gehen verloren!
server {
    listen {{ web_port }};
    server_name {{ inventory_hostname }};
    root {{ web_root }};
{% if ssl_aktiv | default(false) %}
    ssl_certificate /etc/ssl/{{ inventory_hostname }}.pem;
{% endif %}
}
```

56.5. Rollen

Niveau: Fortgeschritten

Eine **Rolle** bündelt Aufgaben, Vorlagen, Variablen und Handler wiederverwendbar in einer festen Verzeichnisstruktur:

```
$ ansible-galaxy init roles/nginx
roles/nginx/
  tasks/main.yml        # Aufgaben
  handlers/main.yml     # Handler
  templates/            # Jinja2-Vorlagen
  files/                # statische Dateien
  defaults/main.yml     # Standardwerte (leicht ueberschreibbar)
  vars/main.yml         # feste Variablen
  meta/main.yml         # Abhaengigkeiten, Metadaten

# site.yml
- hosts: webserver
  become: true
  roles:
    - common
    - role: nginx
      vars:
        web_port: 8080

$ ansible-galaxy collection install community.general     # Sammlungen aus Galaxy
$ ansible-galaxy install -r requirements.yml
```

Empfohlene Projektstruktur:

```
projekt/
  ansible.cfg
  inventory/
    produktion/hosts.ini
    test/hosts.ini
  group_vars/
  host_vars/
  roles/
  site.yml
  README.md
```

```
# ansible.cfg
[defaults]
inventory = inventory/test/hosts.ini
roles_path = roles
host_key_checking = True
forks = 10
[privilege_escalation]
become = True
```

56.6. Vault, Tests und Betrieb

Niveau: Experte

Ansible Vault verschlüsselt Geheimnisse (Passwörter, Schlüssel), damit sie im Git-Repository liegen dürfen:

```
$ ansible-vault create group_vars/all/vault.yml           # neue verschluesselte Datei
$ ansible-vault edit group_vars/all/vault.yml
$ ansible-vault encrypt_string 'geheim' --name 'db_passwort'
$ ansible-playbook site.yml --ask-vault-pass
$ ansible-playbook site.yml --vault-password-file ~/.vault_pass     # Datei mit chmod 600,
    nicht ins Git!
```

- **Test**: `ansible-lint` prüft Stil und Fehler; `molecule` testet Rollen in Containern; zuerst auf Testsystemen, dann produktiv. Ein zweiter Lauf muss `changed=0` melden – sonst ist etwas nicht idempotent.
- **Ausführung in Etappen**: `serial: 1` oder `serial: "25%"` bei Rolling Updates, damit nie alle Server gleichzeitig neu starten.
- **Geschwindigkeit**: `forks`, SSH-Multiplexing (`ControlPersist`, Kapitel 42), Fakten nur bei Bedarf sammeln (`gather_facts: false`).
- **Sicherheit**: Eigener Benutzer für Ansible mit `sudo`, SSH-Schlüssel mit Passphrase und Agent, keine Passwörter im Klartext, Zugriff auf das Repository beschränken.
- **Grenzen**: Ansible bringt Server in einen Sollzustand, erkennt aber manuelle Änderungen erst beim nächsten Lauf. Regelmäßige Läufe (`cron`, `ansible-pull`, AWX/Semaphore) beheben Abweichungen.
- **Anwendungsbeispiele**: Basis-Härtung aller Server, Benutzer und SSH-Schlüssel verteilen, Updates ausrollen, Monitoring-Agenten (Kapitel 57) installieren, Backups einrichten.

```
$ ansible-lint site.yml
$ ansible-playbook site.yml --check --diff       # Drift-Erkennung
$ ansible-pull -U https://git.example.lan/ops/config.git site.yml   # Pull-Modus auf dem
    Zielhost
```

Übungen und Kontrollfragen

1. Was bedeutet Idempotenz bei Ansible?

   Lösung: Ein Playbook kann mehrfach ausgeführt werden; es ändert nur, was vom Sollzustand abweicht. Ein zweiter Lauf meldet `changed=0`.

2. Wozu dienen Handler?

   Lösung: Sie führen Aktionen (z. B. Dienst neu laden) nur aus, wenn ein Task mit `notify` etwas geändert hat, und zwar am Ende des Plays.

3. Wie prüfen Sie ein Playbook, ohne etwas zu ändern?

   Lösung: `ansible-playbook play.yml --check --diff` (zusätzlich `--syntax-check`).

4. Wo legen Sie Variablen für eine Gruppe von Servern ab?

   Lösung: In `group_vars/gruppenname.yml` (bzw. Rolle: `defaults/main.yml`); hostspezifisch in `host_vars/`.

5. Wie schützen Sie ein Datenbankpasswort im Repository?

   Lösung: Mit Ansible Vault (`ansible-vault encrypt_string`); das Vault-Passwort liegt nicht im Repository.

6. Warum ist `shell` oft problematisch?

   Lösung: Es ist nicht idempotent und meldet stets `changed`; besser ein Modul nutzen oder mit `creates`/`when`/`changed_when` absichern.

57. Monitoring und Alarmierung

Inhalt: Wozu überwachen, Metriken, Logs und Traces, Black-Box- und White-Box-Monitoring, Prometheus mit node_exporter, Grafana, Alertmanager, Nagios-Ansatz mit Plugins, zentrale Protokolle und Alarmregeln.

Stufe: [F] Fortgeschritten bis [X] Experte

Lernziele: Nach diesem Kapitel können Sie

- Metriken, Logs und Alarme voneinander abzugrenzen und sinnvolle Messgrößen zu wählen,
- Prometheus mit node_exporter und Grafana aufzubauen und einfache Abfragen zu schreiben,
- Alarmregeln zu definieren, die wenige, aber aussagekräftige Meldungen erzeugen.

Vorkenntnisse: Kapitel 37 (Protokollierung), Kapitel 54 (Leistung und Fehlersuche) und Kapitel 32 (systemd).

57.1. Wozu Monitoring?

Niveau: Einsteiger

Ein System, das niemand beobachtet, fällt irgendwann aus – und der Benutzer merkt es vor dem Administrator. Monitoring beantwortet drei Fragen: *Läuft es?* (Verfügbarkeit), *Läuft es gut?* (Leistung, Trends) und *Wird es bald eng?* (Kapazitätsplanung).

Säule	Inhalt	Beispiele und Werkzeuge
Metriken	Zahlenreihen über die Zeit (CPU, Speicher, Anfragen pro Sekunde)	Prometheus, InfluxDB, Zabbix
Logs	Ereignisse als Text mit Zeitstempel	journald, rsyslog, Loki, Elastic Stack (ELK)
Traces	Weg einer Anfrage durch viele Dienste	OpenTelemetry, Jaeger, Tempo
Alarme	Benachrichtigung bei Abweichungen	Alertmanager, Nagios, Zabbix

- **Black-Box-Monitoring** prüft von außen, was der Benutzer sieht: Antwortet die Website? Ist das Zertifikat gültig? (Blackbox-Exporter, `curl`, Statusseiten).
- **White-Box-Monitoring** nutzt interne Messwerte: Auslastung, Warteschlangen, Fehlerraten.
- **Die vier goldenen Signale** (Google SRE): *Latenz*, *Verkehr*, *Fehler*, *Sättigung*. Für Server klassisch: USE – *Utilization*, *Saturation*, *Errors*.

Weniger ist mehr

Alarme müssen *handlungsrelevant* sein. Wer täglich Dutzende irrelevanter Meldungen erhält, ignoriert irgendwann auch die wichtige (*Alert Fatigue*). Alarmieren sollte man bei Auswirkungen auf Benutzer oder in absehbarer Zeit (Platte voll in 4 Stunden), nicht bei jeder kurzen CPU-Spitze.

57.2. Prometheus und node_exporter

Niveau: Fortgeschritten

Prometheus holt (*scrape*) in festen Abständen Metriken von HTTP-Endpunkten (`/metrics`) und speichert sie in einer Zeitreihendatenbank. Der **node_exporter** stellt Betriebssystem-Metriken eines Linux-Rechners bereit.

```
$ sudo apt install prometheus prometheus-node-exporter prometheus-alertmanager
$ sudo systemctl enable --now prometheus-node-exporter
$ curl -s localhost:9100/metrics | head -20          # Rohdaten des Exporters
$ curl -s localhost:9100/metrics | grep ^node_memory_MemAvailable
```

```
# /etc/prometheus/prometheus.yml
global:
  scrape_interval: 15s
  evaluation_interval: 15s

rule_files:
  - /etc/prometheus/rules/*.yml

alerting:
  alertmanagers:
    - static_configs:
        - targets: ['localhost:9093']

scrape_configs:
  - job_name: prometheus
    static_configs:
      - targets: ['localhost:9090']
  - job_name: nodes
    static_configs:
      - targets:
          - web1.example.lan:9100
          - web2.example.lan:9100
          - db1.example.lan:9100
        labels:
          umgebung: produktion
  - job_name: nginx-blackbox
    metrics_path: /probe
    params:
      module: [http_2xx]
    static_configs:
      - targets: ['https://www.example.org']
    relabel_configs:
      - source_labels: [__address__]
        target_label: __param_target
      - target_label: __address__
        replacement: localhost:9115

$ promtool check config /etc/prometheus/prometheus.yml
$ sudo systemctl reload prometheus
```

Die Weboberfläche unter Port 9090 erlaubt Abfragen in der Sprache **PromQL**:

```
up                                                    # 1 = Ziel erreichbar
node_load1                                            # Last (1 min)
100 - (avg by (instance) (rate(node_cpu_seconds_total{mode="idle"}[5m])) * 100)   # CPU-
    Auslastung %
(1 - node_memory_MemAvailable_bytes / node_memory_MemTotal_bytes) * 100           # Speicher
    belegt %
100 - (node_filesystem_avail_bytes{mountpoint="/"} / node_filesystem_size_bytes{mountpoint
    ="/"} * 100)   # Platte %
rate(node_network_receive_bytes_total{device="eth0"}[5m]) * 8                      # Empfang
    in Bit/s
predict_linear(node_filesystem_avail_bytes{mountpoint="/"}[6h], 4*3600) < 0       # voll in 4
     Stunden?
```

Metrik-Typen: **Counter** (steigt nur, z. B. übertragene Bytes; Auswertung mit `rate()`), **Gauge** (Messwert, steigt/fällt, z. B. Speicher), **Histogram** (Verteilung, z. B. Antwortzeiten).

57.3. Alarmregeln und Alertmanager

Niveau: Fortgeschritten

```
# /etc/prometheus/rules/server.yml
groups:
  - name: server
    rules:
      - alert: HostDown
        expr: up == 0
        for: 2m
        labels:
          severity: critical
        annotations:
          summary: "{{ $labels.instance }} ist nicht erreichbar"

      - alert: DiskWillFillIn4h
        expr: predict_linear(node_filesystem_avail_bytes{fstype!="tmpfs"}[6h], 4*3600) < 0
        for: 30m
        labels:
          severity: warning
        annotations:
          summary: "Platte {{ $labels.mountpoint }} auf {{ $labels.instance }} wird voll"

      - alert: HighMemory
        expr: (1 - node_memory_MemAvailable_bytes / node_memory_MemTotal_bytes) > 0.9
        for: 10m
        labels:
          severity: warning

$ promtool check rules /etc/prometheus/rules/server.yml
```

`for: 2m` verhindert Fehlalarme durch kurze Ausschläge: Die Bedingung muss zwei Minuten ununterbrochen gelten. Der **Alertmanager** bündelt, dedupliziert und leitet Alarme weiter (E-Mail, Chat, Telefon-Bereitschaft); er unterstützt Ruhezeiten (*silences*) für Wartungsarbeiten.

```
# /etc/prometheus/alertmanager.yml
route:
  receiver: team-mail
  group_by: [alertname, instance]
  group_wait: 30s
  repeat_interval: 4h
  routes:
    - matchers: [ 'severity="critical"' ]
      receiver: bereitschaft
receivers:
  - name: team-mail
    email_configs:
      - to: admin@example.lan
        from: alertmanager@example.lan
        smarthost: mail.example.lan:587
  - name: bereitschaft
    webhook_configs:
      - url: http://localhost:5001/pager
$ amtool check-config /etc/prometheus/alertmanager.yml
$ amtool silence add alertname=HostDown instance=web1.example.lan:9100 -d 2h -c "Wartung"
```

57.4. Grafana

Niveau: Fortgeschritten

Grafana stellt Metriken als Dashboards dar. Prometheus wird als Datenquelle eingebunden; fertige Dashboards (z. B. “Node Exporter Full”) lassen sich importieren.

```
$ sudo apt install grafana                      # aus dem Grafana-Repository
$ sudo systemctl enable --now grafana-server
# Weboberflaeche: http://server:3000 (Erstanmeldung admin/admin, Passwort sofort aendern)
# Verbindung: Connections -> Data sources -> Prometheus -> URL http://localhost:9090
```

Ein gutes Dashboard zeigt oben die *Gesamtlage* (Ampel, Verfügbarkeit), darunter Details; nicht zu viele Panels, klare Einheiten, sinnvolle Zeitbereiche. Grafana kann auch Logs (Loki) und Alarmierung übernehmen.

57.5. Klassisch: Nagios, Icinga und Zabbix

Niveau: Fortgeschritten

- **Nagios/Icinga** führen *Checks* über Plugins aus (Exit-Code: 0 = OK, 1 = Warnung, 2 = kritisch, 3 = unbekannt). Stark bei Verfügbarkeitsprüfungen und Abhängigkeiten.
- **Zabbix** bietet Agenten, Auto-Discovery, Vorlagen und Weboberfläche aus einer Hand.
- **Checkmk** kombiniert Auto-Erkennung mit vorgefertigten Regelwerken.

Ein Nagios-Plugin ist ein einfaches Programm – auch ein Shellskript genügt:

```
#!/bin/bash
# check_diskfree.sh - Warnung ab 80 %, kritisch ab 90 %
nutzung=$(df --output=pcent / | tail -1 | tr -dc '0-9')
if   [ "$nutzung" -ge 90 ]; then echo "CRITICAL - / zu ${nutzung}% belegt"; exit 2
elif [ "$nutzung" -ge 80 ]; then echo "WARNING - / zu ${nutzung}% belegt";  exit 1
else echo "OK - / zu ${nutzung}% belegt"; exit 0
fi
```

57.6. Zentrale Logs und Betrieb

Niveau: Experte

- **Zentrale Protokollierung**: `rsyslog` oder `systemd-journal-remote` sammeln Logs auf einem Server; **Loki/Promtail** oder der **Elastic Stack** bieten Suche und Auswertung. Vorteile: Logs überleben den Ausfall des Servers, Angreifer können sie nicht lokal löschen, Korrelation über mehrere Hosts.
- **Retention**: Aufbewahrungsdauer und Speicherbedarf planen (Prometheus: Startoption `--storage.tsdb.retention.time=30d`).
- **Absicherung**: Exporter nur im internen Netz erreichbar machen (Firewall, Kapitel 43) oder hinter einem Reverse-Proxy mit TLS und Authentifizierung.
- **Eigene Metriken**: Der node_exporter liest Textdateien aus einem Verzeichnis (*textfile collector*); so lassen sich Skriptergebnisse (z. B. Alter der letzten Sicherung) veröffentlichen.
- **Überwache den Überwacher**: Ein ausgefallener Monitoring-Server alarmiert nicht. Externe Prüfung (*Dead Man's Switch*, Heartbeat) einplanen.
- **SLOs**: Ziele wie "99,9 % der Anfragen unter 300 ms" machen Qualität messbar; Alarme leiten sich daraus ab.

```
# Beispiel: Alter der letzten Sicherung als Metrik (cron-Skript)
$ stat -c %Y /backup/latest.tar.zst | awk '{print "backup_last_success_timestamp " $1}' \
    | sudo tee /var/lib/prometheus/node-exporter/backup.prom
# Alarm: time() - backup_last_success_timestamp > 86400 * 2
```

Übungen und Kontrollfragen

1. Worin unterscheiden sich Metriken und Logs?

   Lösung: Metriken sind numerische Zeitreihen (kompakt, gut für Trends/Alarme); Logs sind einzelne Ereignisse als Text (detailreich, für Ursachenanalyse).

2. Was bedeutet `for: 5m` in einer Prometheus-Alarmregel?

   Lösung: Die Bedingung muss fünf Minuten lang ununterbrochen erfüllt sein, bevor der Alarm ausgelöst wird; das vermeidet Fehlalarme durch kurze Spitzen.

3. Wie finden Sie heraus, ob ein Ziel von Prometheus erreicht wird?

   Lösung: Mit der Abfrage `up` (1/0) oder auf der Seite "Targets" der Weboberfläche.

4. Wozu dient der Alertmanager?

   Lösung: Er gruppiert, dedupliziert und routet Alarme an Empfänger und unterstützt Ruhezeiten (Silences).

5. Welche Exit-Codes verwenden Nagios-Plugins?

   Lösung: 0 = OK, 1 = Warnung, 2 = kritisch, 3 = unbekannt.

6. Warum sollte man vor einer voll laufenden Platte alarmieren, statt erst bei 100 %?

   Lösung: Mit `predict_linear` oder Schwellwerten (z. B. 85 %) bleibt Zeit zum Eingreifen, bevor der Dienst ausfällt.

58. TLS, Zertifikate und eigene PKI

Inhalt: Verschlüsselung und Vertrauen im Netz, symmetrische und asymmetrische Verfahren, X.509-Zertifikate und Vertrauensketten, `openssl`, eigene CA, Let's Encrypt, Zertifikatsprüfung und Fehlersuche, Dateiverschlüsselung mit GnuPG.

Stufe: [F] Fortgeschritten bis [X] Experte

Lernziele: Nach diesem Kapitel können Sie

- den TLS-Handshake und die Rolle von Zertifikaten und Zertifizierungsstellen zu erklären,
- mit `openssl` Schlüssel, Anforderungen und selbstsignierte Zertifikate zu erzeugen und Zertifikate zu prüfen,
- eine kleine eigene CA zu betreiben sowie Zertifikatsablauf zu überwachen.

Vorkenntnisse: Kapitel 38 (TCP/IP), Kapitel 46 (Webserver) und Kapitel 52 (Sicherheit).

58.1. Grundlagen der Kryptografie

Niveau: Einsteiger

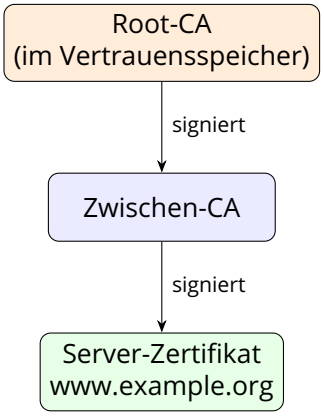
- **Symmetrische Verfahren** (AES, ChaCha20): ein gemeinsamer Schlüssel zum Ver- und Entschlüsseln. Schnell, aber der Schlüssel muss sicher vereinbart werden.
- **Asymmetrische Verfahren** (RSA, ECDSA, Ed25519): Schlüsselpaar aus *öffentlichem* und *privatem* Schlüssel. Was mit dem einen verschlüsselt wird, kann nur mit dem anderen entschlüsselt werden; Signaturen belegen Urheberschaft.
- **Hashfunktionen** (SHA-256, SHA-3): berechnen einen Fingerabdruck fester Länge; ändert sich ein Bit der Daten, ändert sich der Hash völlig.
- **Hybrid**: TLS tauscht per asymmetrischem Verfahren (Schlüsselaustausch) einen symmetrischen Sitzungsschlüssel aus und verschlüsselt damit die Daten.

```
$ sha256sum datei.iso                          # Pruefsumme
$ echo -n 'hallo' | sha256sum
$ openssl dgst -sha256 datei.iso
$ openssl rand -base64 32                       # Zufallswert, z. B. als Passwort
```

58.2. TLS und X.509-Zertifikate

Niveau: Einsteiger

TLS (*Transport Layer Security*, Nachfolger von SSL) schützt Verbindungen vor Mitlesen und Manipulation und beweist die Identität des Servers. Ein **Zertifikat** bindet einen öffentlichen Schlüssel an einen Namen (*Subject*, *Subject Alternative Name*) und ist von einer **Zertifizierungsstelle** (CA) signiert.



Der Client akzeptiert ein Zertifikat, wenn (1) die Kette bis zu einer *vertrauten Wurzel* führt, (2) der Name zum aufgerufenen Host passt, (3) es zeitlich gültig ist, und (4) es nicht widerrufen wurde. Die Vertrauensspeicher liegen unter `/etc/ssl/certs/` (Debian) bzw. `/etc/pki/` (RHEL).
Vereinfachter TLS-1.3-Handshake: Der Client sendet `ClientHello` (unterstützte Verfahren, Schlüsselanteil); der Server antwortet mit `ServerHello`, Schlüsselanteil, Zertifikat und Signatur; beide leiten daraus den Sitzungsschlüssel ab, der Rest ist verschlüsselt. Seit TLS 1.3 ist das in einem Roundtrip erledigt; TLS 1.0/1.1 und SSL gelten als unsicher.

```
$ openssl s_client -connect example.org:443 -servername example.org </dev/null   # Handshake
    ansehen
$ openssl s_client -connect example.org:443 -servername example.org </dev/null 2>/dev/null \
    | openssl x509 -noout -subject -issuer -dates -ext subjectAltName
$ echo | openssl s_client -connect example.org:443 2>/dev/null | openssl x509 -noout -enddate
$ curl -vI https://example.org 2>&1 | grep -E 'SSL|subject|expire|issuer'
```

58.3. Schlüssel, Zertifikatsanforderung und selbstsigniertes Zertifikat

Niveau: Fortgeschritten

```
# Privaten Schluessel erzeugen (nur Besitzer darf lesen!)
$ umask 077
$ openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256 -out server.key
$ openssl genrsa -out server-rsa.key 3072                # Alternative: RSA

# Zertifikatsanforderung (CSR) mit Namen
$ openssl req -new -key server.key -out server.csr \
    -subj "/C=DE/O=Beispiel GmbH/CN=www.example.lan" \
    -addext "subjectAltName=DNS:www.example.lan,DNS:example.lan"

# Selbstsigniertes Zertifikat fuer Tests (365 Tage)
$ openssl req -x509 -newkey ec -pkeyopt ec_paramgen_curve:P-256 -nodes \
    -keyout test.key -out test.crt -days 365 \
    -subj "/CN=test.lan" -addext "subjectAltName=DNS:test.lan"

# Inhalte pruefen
$ openssl req  -in server.csr -noout -text | head -30
$ openssl x509 -in test.crt -noout -text | grep -E 'Subject:|Not (Before|After)|DNS:'
$ openssl x509 -in test.crt -noout -fingerprint -sha256
```

Ein selbstsigniertes Zertifikat ist für den Browser nicht vertrauenswürdig (keine Kette zu einer bekannten CA) – gut für Tests, nicht für öffentliche Dienste.

58.4. Eigene Zertifizierungsstelle

Niveau: Fortgeschritten

Für interne Dienste lohnt sich eine eigene kleine CA: Man verteilt nur ihr Wurzelzertifikat an die Clients und stellt dann beliebig viele Zertifikate aus.

```
# 1. CA-Schluessel und Wurzelzertifikat (10 Jahre) - Schluessel gut schuetzen, am besten
    offline!
$ mkdir -p ~/ca && cd ~/ca && umask 077
$ openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-384 -aes256 -out ca.key
$ openssl req -x509 -new -key ca.key -sha384 -days 3650 -out ca.crt -subj "/O=Beispiel GmbH/
    CN=Beispiel Interne CA"

# 2. Server-Zertifikat signieren (90 Tage)
$ cat > ext.cnf <<'EOF'
basicConstraints = CA:FALSE
keyUsage = digitalSignature
extendedKeyUsage = serverAuth
subjectAltName = DNS:www.example.lan, DNS:example.lan, IP:192.168.10.10
EOF
$ openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -CAcreateserial \
    -out server.crt -days 90 -sha256 -extfile ext.cnf

# 3. Kette pruefen
$ openssl verify -CAfile ca.crt server.crt
server.crt: OK

# 4. Wurzelzertifikat auf Clients vertrauen (Debian/Ubuntu)
$ sudo cp ca.crt /usr/local/share/ca-certificates/beispiel-ca.crt
$ sudo update-ca-certificates
# RHEL/Fedora:
$ sudo cp ca.crt /etc/pki/ca-trust/source/anchors/ && sudo update-ca-trust
```

Für größere Umgebungen eignen sich `easy-rsa`, `step-ca` (smallstep, mit ACME-Unterstützung) oder die Zertifikatsdienste von Verzeichnisdiensten (FreeIPA, Active Directory). Wichtig: Laufzeit kurz halten, Erneuerung automatisieren, Widerruf (CRL/OCSP) planen.

58.5. Let's Encrypt und ACME

Niveau: Fortgeschritten

Das **ACME**-Protokoll automatisiert Ausstellung und Erneuerung. Der Client beweist die Kontrolle über die Domain:

- **HTTP-01**: eine Datei unter `/.well-known/acme-challenge/` abrufbar machen (Port 80).
- **DNS-01**: einen `TXT`-Eintrag `_acme-challenge` setzen. Nötig für Wildcard-Zertifikate und Dienste ohne öffentlichen Webserver.
- **TLS-ALPN-01**: Nachweis über Port 443.

```
$ sudo apt install certbot
$ sudo certbot certonly --webroot -w /var/www/html -d example.org -d www.example.org
$ sudo certbot certonly --standalone -d mail.example.org                # Port 80 kurz
    belegen
$ sudo certbot certonly --manual --preferred-challenges dns -d '*.example.org'
$ sudo certbot certificates                                             # uebersicht,
    Ablaufdaten
$ sudo certbot renew --dry-run
$ ls -l /etc/letsencrypt/live/example.org/
cert.pem  chain.pem  fullchain.pem  privkey.pem
```

Die Zertifikate sind 90 Tage gültig; `certbot` erneuert nach 60 Tagen automatisch über einen systemd-Timer oder Cron-Eintrag. Mit `--deploy-hook "systemctl reload nginx"` (oder Skripten in `/etc/letsencrypt/renewal-hooks/deploy/`) nehmen Dienste die neue Datei auf. Alternativen: `acme.sh`, `lego`, `Caddy` (automatisch eingebaut).

58.6. Betrieb: Konfiguration, Überwachung und Fehlersuche

Niveau: Experte

- **Moderne Einstellungen**: TLS 1.2 und 1.3 erlauben, Forward Secrecy (ECDHE) bevorzugen, HSTS setzen. Empfehlungen liefert der Mozilla SSL Configuration Generator; `testssl.sh` und `nmap --script ssl-enum-ciphers` prüfen Server.
- **Zertifikatsablauf überwachen**: häufigste Ursache von Ausfällen. Monitoring (Kapitel 57) mit `ssl_expiry`-Checks oder Blackbox-Exporter.
- **Private Schlüssel schützen**: Modus 600, eigener Benutzer, nicht in Backups ohne Verschlüsselung, nach Verdacht auf Kompromittierung sofort neu ausstellen und das alte Zertifikat widerrufen.
- **Mutual TLS (mTLS)**: Auch der Client weist sich mit einem Zertifikat aus – üblich bei Dienst-zu-Dienst-Verbindungen, VPNs (Kapitel 48) und Zero-Trust-Konzepten.

```
# Zertifikat laeuft bald ab? (Exit-Code 1, wenn weniger als 14 Tage)
$ echo | openssl s_client -connect example.org:443 -servername example.org 2>/dev/null \
    | openssl x509 -noout -checkend $((14*24*3600)) && echo OK || echo "LAEUFT BALD AB"

# Passt Schluessel zu Zertifikat? (Hashes muessen uebereinstimmen)
$ openssl x509 -in server.crt -noout -pubkey | sha256sum
$ openssl pkey -in server.key -pubout        | sha256sum

# Format-Umwandlung
$ openssl x509 -in cert.der -inform DER -out cert.pem -outform PEM
$ openssl pkcs12 -export -in server.crt -inkey server.key -out server.p12
```

Typische Fehlermeldungen: `certificate has expired` (abgelaufen, Uhrzeit prüfen!), `unable to get local issuer certificate` (Zwischenzertifikat fehlt – `fullchain.pem` statt `cert.pem` verwenden), `hostname mismatch` (Name nicht im SAN), `self-signed certificate` (CA nicht vertraut).

58.7. Dateien verschlüsseln und signieren mit GnuPG

Niveau: Fortgeschritten

```
$ gpg --full-generate-key                          # Schluesselpaar (ed25519/cv25519)
$ gpg --list-keys; gpg --list-secret-keys
$ gpg --armor --export anna@example.lan > anna.pub  # oeffentlichen Schluessel weitergeben
$ gpg --import bob.pub
$ gpg -e -r bob@example.lan geheim.txt              # fuer Bob verschluesseln -> geheim.txt.
    gpg
$ gpg -d geheim.txt.gpg                             # entschluesseln
$ gpg -c datei.tar                                  # nur mit Passphrase (symmetrisch)
$ gpg --sign datei.txt; gpg --detach-sign datei.iso  # signieren
$ gpg --verify datei.iso.sig datei.iso              # Signatur pruefen
$ gpg --fingerprint bob@example.lan                 # Fingerabdruck vergleichen!
```

Echtheit eines Schlüssels beweist nur der Vergleich des Fingerabdrucks über einen zweiten Kanal. Paketverwaltungen nutzen dasselbe Prinzip, um Repositories zu signieren (Kapitel 19).

Übungen und Kontrollfragen

1. Warum nutzt TLS sowohl asymmetrische als auch symmetrische Verfahren?

   Lösung: Asymmetrische Verfahren ermöglichen Schlüsselaustausch und Authentisierung ohne vorab geteiltes Geheimnis, sind aber langsam; die Daten selbst werden mit schnellen symmetrischen Verfahren verschlüsselt.

2. Was prüft ein Client bei einem Server-Zertifikat?

   Lösung: Kette zu vertrauter Wurzel, passender Name (SAN), zeitliche Gültigkeit und ggf. Widerruf.

3. Welcher Schlüssel muss unbedingt geheim bleiben, welcher darf veröffentlicht werden?

   Lösung: Der private Schlüssel bleibt geheim; Zertifikat und öffentlicher Schlüssel sind öffentlich.

4. Wie prüfen Sie das Ablaufdatum eines Zertifikats eines entfernten Servers?

   Lösung: `echo | openssl s_client -connect host:443 2>/dev/null | openssl x509 -noout -enddate`.

5. Wann brauchen Sie die DNS-01-Challenge?

   Lösung: Für Wildcard-Zertifikate oder wenn der Server von außen nicht über Port 80/443 erreichbar ist.

6. Ein Client meldet `unable to get local issuer certificate`. Was ist wahrscheinlich?

   Lösung: Der Server liefert das Zwischenzertifikat nicht mit (`fullchain` verwenden) oder die CA ist nicht im Vertrauensspeicher.

59. Performance-Tuning

Inhalt: Vorgehen bei der Leistungsanalyse, CPU, Speicher, Plattenzugriffe und Netzwerk messen, Kernelparameter mit `sysctl`, I/O-Scheduler, Dateisystemoptionen, tuned, Prozess-Prioritäten, Benchmarks und Grenzen der Optimierung.

Stufe: [X] Experte

Lernziele: Nach diesem Kapitel können Sie

- Engpässe mit der USE-Methode systematisch zu finden,
- die Werkzeuge `vmstat`, `iostat`, `pidstat`, `perf` und `strace` gezielt einzusetzen,
- Kernel- und Systemparameter sinnvoll anzupassen und die Wirkung zu messen.

Vorkenntnisse: Kapitel 24 (Prozesse), Kapitel 54 (Leistung und Fehlersuche), Kapitel 25 (Kernel) und Kapitel 55 (Fehlersuche an Szenarien).

59.1. Grundsätze

Niveau: Fortgeschritten

1. **Erst messen, dann ändern.** Ohne Ausgangswerte (*Baseline*) lässt sich keine Verbesserung belegen.
2. **Den Engpass finden.** Optimierung außerhalb des Engpasses bringt nichts (Amdahl). Meist begrenzt genau eine Ressource: CPU, Speicher, I/O oder Netz.
3. **Eine Änderung nach der anderen**, mit reproduzierbarem Test unter realistischer Last.
4. **Standardwerte sind meist gut.** Viele "Tuning-Tipps" aus dem Internet sind veraltet oder passen nicht zur Last. Jede Änderung dokumentieren und versionieren.
5. **Oft ist die Anwendung das Problem** (fehlender Datenbankindex, ineffiziente Abfragen, zu viele Prozesse) – nicht das Betriebssystem.

USE-Methode (Brendan Gregg): Für jede Ressource prüfen: *Utilization* (Auslastung), *Saturation* (Warteschlangen, Wartezeit), *Errors* (Fehler).

Ressource	Auslastung	Sättigung	Fehler
CPU	`mpstat -P ALL 1`	`vmstat` (Spalte `r`), Load > Kerne	`dmesg` (Hitze, MCE)
Speicher	`free -h`	Swap-Aktivität (`si`/`so`), `PSI`	OOM-Meldungen
Platte	`iostat -x` (`%util`)	`await`, Queue-Länge `aqu-sz`	`smartctl`, `dmesg`
Netzwerk	`sar -n DEV`, `ip -s link`	Drops, Retransmits (`ss -ti`)	`ip -s link`, `ethtool -S`

59.2. CPU und Prozesse

Niveau: Fortgeschritten

```
$ nproc; lscpu | head -20
$ mpstat -P ALL 1 3                          # Auslastung je Kern (Paket sysstat)
$ pidstat -u 1 5                             # CPU je Prozess
$ pidstat -d 1 5                             # I/O je Prozess
$ pidstat -r 1 5                             # Speicher je Prozess
$ top -H -p PID                              # Threads eines Prozesses
$ cat /proc/pressure/cpu                     # PSI: Wartezeit auf CPU (some/full)
$ sudo perf top                              # Hotspots live (Funktionen)
$ sudo perf record -g -p PID -- sleep 30 && sudo perf report
$ sudo strace -c -p PID                      # Systemaufrufe zaehlen (Ctrl-C beendet)
$ sudo strace -f -tt -e trace=openat,read -p PID
$ sudo ltrace -p PID                         # Bibliotheksaufrufe
```

- **Priorität**: `nice -n 10 befehl`, `renice 10 -p PID`; I/O-Priorität mit `ionice -c3 befehl` (Leerlauf-Klasse für Hintergrundarbeit wie Backups).
- **Systemd**: `CPUWeight=`, `CPUQuota=`, `MemoryMax=`, `IOWeight=` in Units begrenzen Ressourcen (cgroups, Kapitel 33).
- **CPU-Frequenz**: `cpupower frequency-info`; Governor `performance` für latenzkritische Server, `powersave`/`schedutil` für Energiesparen.
- **CPU-Zuordnung**: `taskset -c 0,1 befehl`; auf NUMA-Systemen `numactl` und `numastat`.

59.3. Speicher

Niveau: Fortgeschritten

```
$ free -h
$ vmstat -s | head -10
$ cat /proc/meminfo | grep -E 'MemAvailable|Dirty|Writeback|Slab|SwapCached'
$ cat /proc/pressure/memory
$ sysctl vm.swappiness vm.dirty_ratio vm.dirty_background_ratio vm.vfs_cache_pressure
$ cat /sys/kernel/mm/transparent_hugepage/enabled
$ sudo slabtop -o | head
```

Parameter	Wirkung
`vm.swappiness`	Neigung, Anwendungsspeicher auszulagern (0–100; Standard 60). Datenbankserver oft 1–10
`vm.dirty_ratio`	Anteil des Speichers, ab dem schreibende Prozesse blockieren, bis Daten auf Platte sind
`vm.dirty_background_ratio`	Ab diesem Anteil beginnt das Zurückschreiben im Hintergrund
`vm.vfs_cache_pressure`	Wie stark Dateisystem-Caches zurückgegeben werden
`vm.overcommit_memory`	Verhalten bei Speicherzusagen (0 heuristisch, 1 immer, 2 strikt)
`vm.max_map_count`	Maximale Speicherbereiche je Prozess (z. B. für Elasticsearch erhöhen)

```
# dauerhaft: /etc/sysctl.d/90-tuning.conf
vm.swappiness = 10
vm.dirty_ratio = 15
vm.dirty_background_ratio = 5
net.core.somaxconn = 4096
fs.file-max = 2097152
$ sudo sysctl --system                          # alle Dateien einlesen
$ sudo sysctl -w vm.swappiness=10               # sofort, nicht dauerhaft
$ sysctl -a | grep -c .                         # Anzahl verfuegbarer Parameter
```

Ein **zu großer Cache** ist kein Problem: Linux nutzt freien Speicher für den Seitencache und gibt ihn bei Bedarf sofort her. **Swap** ist nicht per se schlecht (lagert selten genutzte Seiten aus); schlecht ist *aktives* Swappen (`si`/`so` dauerhaft hoch). Moderne Alternativen: `zram` (komprimierter Swap im RAM) und `zswap`.

59.4. Datenträger und Dateisysteme

Niveau: Fortgeschritten

```
$ iostat -xz 1 5                              # %util, await, r/s, w/s, aqu-sz
$ sudo iotop -oPa
$ cat /sys/block/nvme0n1/queue/scheduler      # I/O-Scheduler: [none] mq-deadline kyber bfq
$ cat /sys/block/sda/queue/rotational         # 1 = HDD, 0 = SSD
$ sudo hdparm -tT /dev/sda                    # einfache Lesegeschwindigkeit
$ lsblk -o NAME,ROTA,SCHED,SIZE,MOUNTPOINT
$ findmnt -o TARGET,SOURCE,FSTYPE,OPTIONS /var/lib/mysql
```

- **I/O-Scheduler**: `none` für NVMe, `mq-deadline` für SSD/RAID, `bfq` für Desktop und HDD (Fairness). Ändern: `echo mq-deadline | sudo tee /sys/block/sda/queue/scheduler`; dauerhaft per udev-Regel.
- **Mount-Optionen**: `noatime` (oder `relatime`, Standard) vermeidet Schreibzugriffe bei jedem Lesen; `discard` oder besser regelmäßig `fstrim.timer` für SSD.
- **Dateisystemwahl und -layout**: XFS für große Dateien und viel Parallelität, ext4 als Allrounder; Datenbanken auf eigene Volumes (Kapitel 28); Journal auf schnellem Medium nur bei Bedarf.
- **RAID und Caches**: Schreibcache und Batterie/Kondensator beachten; Software-RAID-Parameter wie `stripe_cache_size`.
- **Schreiblast**: `fio` misst realistisch (Beispielaufruf siehe unten).

```
$ sudo systemctl enable --now fstrim.timer
$ sudo fio --name=randread --rw=randread --bs=4k --size=1G --iodepth=32 --runtime=20 --
    time_based --filename=/data/fio.tmp
$ rm /data/fio.tmp
```

59.5. Netzwerk

Niveau: Fortgeschritten

```
$ ss -s                                     # Verbindungsuebersicht
$ ss -ti dst 192.168.10.5                   # TCP-Details: rtt, cwnd, retrans
$ nstat -az | grep -iE 'retrans|drop|overflow'
$ ethtool -k eth0 | grep -E 'offload'
$ ethtool -g eth0                           # Ring-Puffer
$ sysctl net.ipv4.tcp_congestion_control net.core.default_qdisc
$ cat /proc/net/softnet_stat | head -3
```

```
# typische Parameter fuer stark belastete Server: /etc/sysctl.d/91-netz.conf
net.core.somaxconn = 4096                   # Warteschlange fuer accept()
net.ipv4.tcp_max_syn_backlog = 8192
net.ipv4.ip_local_port_range = 10240 65000
net.ipv4.tcp_tw_reuse = 1
net.core.rmem_max = 16777216
net.core.wmem_max = 16777216
net.ipv4.tcp_congestion_control = bbr
net.core.default_qdisc = fq
```

Für schnelle Verbindungen mit hoher Latenz (Fernverbindungen) sind größere TCP-Puffer und `bbr` hilfreich; im LAN bringen sie oft nichts. Auch die Anzahl offener Dateien (`ulimit -n`, `LimitNOFILE=` in systemd) begrenzt Server mit vielen Verbindungen.

59.6. Werkzeuge zur Gesamtbetrachtung

Niveau: Experte

- **tuned**: Profile für typische Lasten (`tuned-adm list`), z. B. `throughput-performance`, `latency-performance` oder `virtual-guest`. Standard unter RHEL; auch auf Debian verfügbar.
- **sar** (`sysstat`): sammelt Verlaufsdaten (z. B. `sar -u`, `sar -r`, `sar -d`) und beantwortet "Was war gestern um 14 Uhr los?"
- **BPF-Werkzeuge** (`bcc-tools`, `bpftrace`): Latenz-Histogramme, Dateizugriffe, Schedulerverzögerungen mit geringem Aufwand. Beispiele: `biolatency`, `opensnoop`, `execsnoop`.
- **Flame Graphs** visualisieren `perf`-Messungen und zeigen, wo CPU-Zeit verbraucht wird.
- **Anwendungsebene**: Datenbank-Indizes und Abfragepläne (Kapitel 51), Caching (Redis, nginx), Verbindungspools, Komprimierung – oft größerer Hebel als jeder Kernelparameter.
- **Kapazitätsplanung**: Trends (Monitoring, Kapitel 57) zeigen, wann Hardware nachgerüstet werden muss. Manchmal ist der richtige Schritt, mehr Ressourcen bereitzustellen, statt zu optimieren.

```
$ sudo apt install sysstat tuned
$ sudo systemctl enable --now sysstat
$ sar -u 1 3; sar -r; sar -d -p; sar -n DEV 1 3
$ sar -u -f /var/log/sysstat/sa$(date -d yesterday +%d)     # gestern
$ tuned-adm active
$ sudo tuned-adm profile virtual-guest
$ sudo biolatency 10 1                                       # BPF: Platten-Latenz-Histogramm
```

Nicht blind übernehmen

Tuning-Parameter hängen von Hardware, Kernelversion und Last ab. Wer Werte aus Anleitungen kopiert, ohne zu messen, verschlechtert das System gelegentlich. Änderungen immer in einer Testumgebung prüfen, Standardwerte als Rückfallposition dokumentieren und die Wirkung mit denselben Messungen vorher und nachher vergleichen.

Übungen und Kontrollfragen

1. Was besagt die USE-Methode?

   Lösung: Für jede Ressource Auslastung (Utilization), Sättigung (Saturation) und Fehler (Errors) prüfen, um den Engpass zu finden.

2. Wie erkennen Sie mit `vmstat`, dass das System swappt?

   Lösung: Dauerhaft Werte größer 0 in den Spalten `si` (swap in) und `so` (swap out).

3. Welche Aussage macht `%util` bei `iostat -x`?

   Lösung: Anteil der Zeit, in der das Gerät beschäftigt war; nahe 100 % deutet bei HDDs auf Sättigung hin (bei NVMe mit hoher Parallelität weniger aussagekräftig; `await` und `aqu-sz` ergänzend beachten).

4. Wie machen Sie einen sysctl-Wert dauerhaft?

   Lösung: In einer Datei unter `/etc/sysctl.d/` eintragen und mit `sysctl --system` laden.

5. Wie führen Sie ein Backup mit möglichst geringem Einfluss auf andere Prozesse aus?

   Lösung: Mit `nice -n 19 ionice -c3 befehl` bzw. `Nice=19` und `IOSchedulingClass=idle` in der systemd-Unit.

6. Warum ist viel "belegter" Speicher in `free` meist kein Problem?

   Lösung: Linux nutzt ungenutzten Speicher als Cache; `available` zeigt, wie viel Speicher Anwendungen tatsächlich bekommen können.

60. Hochverfügbarkeit und Cluster

Inhalt: Verfügbarkeit und Ausfallsicherheit, Redundanz auf allen Ebenen, Keepalived mit virtueller IP, Pacemaker und Corosync, Quorum und Fencing, DRBD, Datenbank-Replikation, Wiederherstellungsziele und Tests.

Stufe: [X] Experte

Lernziele: Nach diesem Kapitel können Sie

- Verfügbarkeit in Prozent und Ausfallzeiten zu berechnen und Single Points of Failure zu erkennen,
- eine virtuelle IP-Adresse mit Keepalived und einen Cluster mit Pacemaker/Corosync im Prinzip zu planen,
- Quorum, Split-Brain und Fencing zu erklären und die Rolle von Replikation (DRBD, Datenbank) einzuordnen.

Vorkenntnisse: Kapitel 38 (TCP/IP), Kapitel 47 (Proxy und Lastverteilung), Kapitel 32 (systemd) und Kapitel 51 (Datenbanken).

60.1. Verfügbarkeit und Redundanz

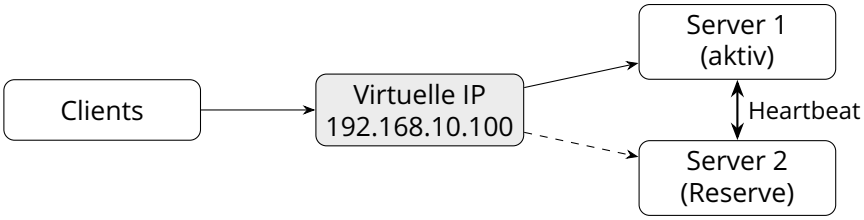
Niveau: Einsteiger

Die **Verfügbarkeit** gibt an, welcher Anteil der Zeit ein Dienst nutzbar ist.

Verfügbarkeit	Ausfall pro Jahr	Ausfall pro Monat
99 %	3,65 Tage	7,3 Stunden
99,9 % (“drei Neunen”)	8,8 Stunden	44 Minuten
99,99 %	53 Minuten	4,4 Minuten
99,999 %	5,3 Minuten	26 Sekunden

Jede zusätzliche Neun macht die Lösung deutlich teurer und komplexer. Die Zielverfügbarkeit ergibt sich aus dem Geschäftsbedarf und wird als *Service Level Objective* (SLO) vereinbart.

- **Single Point of Failure (SPOF)**: Komponente, deren Ausfall den Dienst stoppt: ein Server, ein Switch, ein Netzteil, die einzige Datenbank, der Mensch, der das Passwort kennt.
- **Redundanz** auf allen Ebenen: doppelte Netzteile, RAID (Kapitel 27), Bonding, zwei Switches, zwei Server, zwei Standorte.
- **Serienschaltung vs. Parallelschaltung**: Hängen n Komponenten mit je Verfügbarkeit v in Reihe zusammen, ist die Gesamtverfügbarkeit v^n; bei Parallelbetrieb zweier Komponenten $1-(1-v)^2$. Zwei Server mit je 99 % ergeben parallel 99,99 %.
- **Kennzahlen**: *MTBF* (mittlere Zeit zwischen Ausfällen), *MTTR* (mittlere Reparaturzeit), *RTO* (maximal tolerierbare Wiederherstellungszeit), *RPO* (maximal tolerierbarer Datenverlust).



60.2. Einfache Lösung: virtuelle IP mit Keepalived

Niveau: Fortgeschritten

Keepalived nutzt das Protokoll **VRRP**: Zwei Server teilen sich eine virtuelle IP-Adresse; fällt der aktive aus, übernimmt der andere die Adresse innerhalb von Sekunden. Es eignet sich für zustandslose Dienste (Webserver, Proxys, DNS).

```
# /etc/keepalived/keepalived.conf auf server1 (MASTER)
global_defs {
    router_id web01
}
vrrp_script chk_nginx {
    script "/usr/bin/systemctl is-active --quiet nginx"
    interval 2
    fall 2
    rise 2
}
vrrp_instance VI_1 {
    state MASTER                # server2: BACKUP
    interface eth0
    virtual_router_id 51
    priority 150                # server2: 100
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass geheim12
    }
    virtual_ipaddress {
        192.168.10.100/24 dev eth0
    }
    track_script {
        chk_nginx
    }
}
```

```
$ sudo apt install keepalived
$ sudo sysctl -w net.ipv4.ip_nonlocal_bind=1     # Dienste duerfen an VIP binden, die noch
    nicht lokal ist
$ sudo systemctl enable --now keepalived
$ ip -br addr show eth0                          # VIP auf dem aktiven Server sichtbar
$ sudo journalctl -u keepalived -f
$ sudo systemctl stop nginx                      # Test: VIP wandert zum anderen Server
```

Die Firewall muss VRRP (IP-Protokoll 112) zwischen den Servern zulassen. Die Konfigurationen beider Server müssen übereinstimmen (`virtual_router_id`, Passwort, VIP), nur Priorität und Zustand unterscheiden sich.

Load Balancer absichern: Auch ein Lastverteiler (HAProxy, Kapitel 47) darf kein SPOF sein. Zwei HAProxy-Server teilen sich dann die VIP; als Gesundheitsprüfung dient statt `nginx` z. B. `script "/usr/bin/killall -0 haproxy"`. Die VIP muss beim Master liegen, der HAProxy-Dienst auf beiden laufen; `net.ipv4.ip_nonlocal_bind=1` (siehe oben) erlaubt HAProxy auf dem Reserveserver, an die VIP zu binden.

60.3. Cluster mit Pacemaker und Corosync

Niveau: Experte

Für Dienste mit Zustand (Datenbank, Dateisystem, mehrere voneinander abhängige Ressourcen) verwendet man einen Cluster-Manager:

- **Corosync**: Kommunikationsschicht; sorgt für Mitgliedschaft, Nachrichtenaustausch und Quorum.
- **Pacemaker**: Ressourcenmanager; startet, überwacht und verschiebt Ressourcen (IP, Dateisystem, Dienst), beachtet Abhängigkeiten und Reihenfolgen.
- **Resource Agents**: Skripte, die Ressourcen steuern (`ocf:heartbeat:IPaddr2`, `Filesystem`, `systemd:nginx`).
- **Fencing (STONITH)**: Ein ausgefallener oder nicht erreichbarer Knoten wird zwangsweise abgeschaltet (Strom, IPMI, Hypervisor), bevor seine Ressourcen anderswo starten.

```
# Zwei-Knoten-Cluster einrichten (pcs, Debian/RHEL aehnlich)
$ sudo apt install pacemaker corosync pcs fence-agents
$ sudo systemctl enable --now pcsd
$ sudo passwd hacluster                          # auf beiden Knoten gleiches Passwort
$ sudo pcs host auth node1 node2 -u hacluster
$ sudo pcs cluster setup mycluster node1 node2
$ sudo pcs cluster start --all
$ sudo pcs cluster enable --all

# Ressourcen
$ sudo pcs resource create vip ocf:heartbeat:IPaddr2 ip=192.168.10.100 cidr_netmask=24 op
    monitor interval=10s
$ sudo pcs resource create web systemd:nginx op monitor interval=15s
$ sudo pcs constraint colocation add web with vip INFINITY    # zusammen auf einem Knoten
$ sudo pcs constraint order vip then web                      # erst VIP, dann Webserver

# Fencing (Beispiel: IPMI) - in Produktion zwingend
$ sudo pcs stonith create fence-node1 fence_ipmilan ip=10.0.0.11 username=admin password=...
    pcmk_host_list=node1

# Betrieb
$ sudo pcs status
$ sudo crm_mon -1
$ sudo pcs resource move web node2                            # manuell verschieben
$ sudo pcs resource clear web                                 # Verschiebe-Regel entfernen
$ sudo pcs node standby node1                                 # Knoten fuer Wartung leeren
$ sudo pcs node unstandby node1
```

Quorum und Split-Brain

Fällt die Verbindung zwischen den Knoten aus, glauben beide, der andere sei tot (*Split-Brain*) – beide starten die Ressourcen und beschreiben dieselben Daten. Dagegen helfen **Quorum** (nur die Gruppe mit Mehrheit der Stimmen darf arbeiten; deshalb drei oder mehr Stimmen, z. B. ein dritter Knoten oder ein *Quorum-Device*) und **Fencing** (der unterlegene Knoten wird abgeschaltet). Ein Cluster ohne Fencing ist bei Zustandsdiensten nicht verlässlich.

60.4. Replikation: DRBD und Datenbanken

Niveau: Experte

DRBD (*Distributed Replicated Block Device*) spiegelt ein Blockgerät über das Netz – eine Art "RAID 1 über Rechner". Der aktive Knoten (*Primary*) schreibt, der andere (*Secondary*) erhält alle Änderungen.

```
# /etc/drbd.d/r0.res
resource r0 {
    protocol C;                      # synchron: Schreiben erst nach Bestaetigung vom
    Partner
    device    /dev/drbd0;
    disk      /dev/vg0/daten;
    meta-disk internal;
    on node1 { address 10.0.0.1:7789; }
    on node2 { address 10.0.0.2:7789; }
}

$ sudo drbdadm create-md r0                        # auf beiden Knoten
$ sudo drbdadm up r0
$ sudo drbdadm primary --force r0                  # nur einmal, auf node1
$ sudo mkfs.ext4 /dev/drbd0
$ cat /proc/drbd; sudo drbdadm status r0
```

Im Cluster steuert Pacemaker, welcher Knoten Primary ist und das Dateisystem einhängt. Für Datenbanken ist häufig die **anwendungseigene Replikation** besser: PostgreSQL (Streaming-Replikation, Patroni), MariaDB/MySQL (Replikation, Galera), Redis (Sentinel), Dateien (`rsync`, GlusterFS, Ceph).

Verfahren	Geeignet für	Anmerkung
Keepalived/VRRP	Zustandslose Dienste	Einfach; kein Datenabgleich
HAProxy + mehrere Server	Webanwendungen, APIs	Lastverteilung plus Gesundheitsprüfung; Proxy selbst redundant (VIP)
Pacemaker + DRBD	Klassische Dienste mit Daten	Mächtig, aber komplex; Fencing nötig
DB-Replikation	Datenbanken	Anwendungsspezifisch; Failover oft über Patroni o. ä.
Orchestrierung (Kubernetes)	Container-Anwendungen	Selbstheilung und Verteilung eingebaut

60.5. Planung, Tests und Grenzen

Niveau: Fortgeschritten

- **Hochverfügbarkeit ersetzt kein Backup**: Löschfehler, Verschlüsselungstrojaner und Fehlkonfigurationen werden sofort auf alle Knoten repliziert (Kapitel 53).
- **Failover testen**: Dienst stoppen, Netzwerkkabel ziehen, Knoten hart abschalten, Datenbank-Primary stoppen. Ungetestete Redundanz funktioniert im Ernstfall meist nicht.
- **Wartung ohne Ausfall**: *Rolling Updates*, Knoten einzeln in den Standby (`pcs node standby`), Verbindungen abbauen lassen (Draining).
- **Überwachung**: Auch der *Ausfall der Redundanz* (ein Knoten weg) muss Alarme auslösen (Kapitel 57), sonst bemerkt man erst den zweiten Ausfall.
- **Komplexität**: Jede zusätzliche Komponente kann selbst ausfallen. Oft ist ein einfacher Server mit gutem Backup, schneller Wiederherstellung und Automatisierung (Kapitel 56) zuverlässiger als ein schlecht verstandener Cluster.
- **Georedundanz**: Mehrere Standorte schützen vor Standortausfall, bringen aber Latenz und Konsistenzfragen (Replikation asynchron ⇒ möglicher Datenverlust, RPO beachten).

Übungen und Kontrollfragen

1. Wie hoch ist die Gesamtverfügbarkeit zweier parallel betriebener Server mit je 99 %?

   Lösung: $1 - (1 - 0{,}99)^2 = 0{,}9999$, also 99,99 %.

2. Was ist ein Single Point of Failure? Nennen Sie zwei Beispiele.

   Lösung: Eine Komponente, deren Ausfall den ganzen Dienst stoppt, z. B. ein einzelner Switch, eine einzelne Datenbank oder ein einzelnes Netzteil.

3. Wozu dient die virtuelle IP bei Keepalived?

   Lösung: Clients sprechen immer dieselbe Adresse an; fällt der aktive Server aus, übernimmt der Reserveserver die Adresse über VRRP.

4. Was ist Split-Brain und wie verhindert man es?

   Lösung: Beide Knoten glauben, aktiv zu sein, weil die Verbindung gestört ist, und schreiben konkurrierend. Verhinderung durch Quorum (Mehrheitsentscheid) und Fencing (STONITH).

5. Was bedeuten RTO und RPO?

   Lösung: RTO: maximal tolerierbare Wiederherstellungszeit; RPO: maximal tolerierbarer Datenverlust (Zeitraum).

6. Warum ersetzt ein Cluster kein Backup?

   Lösung: Fehler wie versehentliches Löschen oder Schadsoftware werden sofort auf alle Knoten repliziert; nur ein getrenntes Backup erlaubt die Rückkehr zu einem früheren Zustand.

61. Cloud, cloud-init und Infrastruktur als Code

Inhalt: Cloud-Modelle und Begriffe, virtuelle Server und Images, cloud-init für die Erstkonfiguration, Terraform als Beschreibung von Infrastruktur, Kubernetes im Überblick, Kosten, Sicherheit und Verantwortung.

Stufe: [X] Experte

Lernziele: Nach diesem Kapitel können Sie

- die Cloud-Modelle IaaS, PaaS und SaaS sowie das Modell der geteilten Verantwortung einzuordnen,
- einen Server mit cloud-init beim ersten Start automatisch einzurichten,
- den Ablauf von Terraform (init, plan, apply) zu erklären und eine einfache Beschreibung zu lesen.

Vorkenntnisse: Kapitel 42 (SSH), Kapitel 56 (Ansible), Kapitel 35 (Virtualisierung) und Kapitel 34 (Container).

61.1. Cloud-Grundlagen

Niveau: Fortgeschritten

Cloud Computing stellt IT-Ressourcen bei Bedarf über das Netz bereit: schnell, skalierbar, nach Verbrauch abgerechnet. Technisch beruht es auf Virtualisierung (Kapitel 35), Containern und Automatisierung.

Modell	Was erhält der Kunde?	Beispiele
IaaS	Virtuelle Server, Speicher, Netze; Betriebssystem und alles darüber verantwortet der Kunde	AWS EC2, Azure VMs, Hetzner Cloud, OpenStack
PaaS	Laufzeitumgebung und Dienste (Datenbank, Container-Plattform); Anbieter betreibt Unterbau	Managed Kubernetes, Managed PostgreSQL
SaaS	Fertige Anwendung	Webmail, Nextcloud-Hosting, Büro-Suiten

- **Public Cloud**: Ressourcen von einem Anbieter, mit anderen Kunden geteilt. **Private Cloud**: eigene Plattform (z. B. OpenStack, Proxmox). **Hybrid**: Kombination.
- **Elastizität**: Ressourcen wachsen und schrumpfen mit dem Bedarf (*Autoscaling*).
- **Regionen und Zonen**: Rechenzentren an verschiedenen Orten; Verteilung auf mehrere Zonen erhöht die Ausfallsicherheit.
- **Instanzen sind flüchtig**: Server werden nicht liebevoll gepflegt (“Haustiere”), sondern automatisiert erzeugt und ersetzt (“Vieh”). Daten gehören auf separate Volumes oder Dienste.

Geteilte Verantwortung

Der Anbieter sichert die Plattform (Gebäude, Hardware, Hypervisor); der Kunde ist für das Betriebssystem, Updates, Konfiguration, Zugriffe und Daten zuständig. Die meisten Cloud-Vorfälle entstehen durch Fehlkonfiguration (offene Speicher, zu weite Rechte, schwache Zugangsdaten), nicht durch Fehler des Anbieters.

61.2. Images und Erstkonfiguration mit cloud-init

Niveau: Fortgeschritten

Cloud-Anbieter stellen fertige **Images** (Debian, Ubuntu, Rocky, …) bereit. Damit aus dem generischen Image *dein* Server wird, liest das Programm **cloud-init** beim ersten Start die sogenannten *User-Data* und richtet das System ein: Benutzer, SSH-Schlüssel, Pakete, Dateien, Befehle.

```
#cloud-config
hostname: web01
timezone: Europe/Berlin
package_update: true
package_upgrade: true
packages:
  - nginx
  - chrony
  - fail2ban
users:
  - name: anna
    groups: [sudo]
    shell: /bin/bash
    sudo: ALL=(ALL) NOPASSWD:ALL
    ssh_authorized_keys:
      - ssh-ed25519 AAAAC3... anna@laptop
disable_root: true
ssh_pwauth: false
write_files:
  - path: /var/www/html/index.html
    content: |
      <h1>Server web01 laeuft</h1>
runcmd:
  - systemctl enable --now nginx
  - systemctl enable --now chrony
final_message: "Fertig nach $UPTIME Sekunden"
```

```
$ cloud-init status --wait                          # auf dem neuen Server: fertig?
$ cloud-init status --long
$ sudo cat /var/log/cloud-init-output.log           # Ausgabe der Befehle
$ sudo cat /var/log/cloud-init.log | less
$ cloud-init schema --config-file user-data.yml     # Syntax pruefen
$ sudo cloud-init clean --logs                      # Zustand zuruecksetzen (z. B. vor Image-
    Erstellung)
```

cloud-init läuft nur beim *ersten* Start (Instanz-ID). Auf einem lokalen KVM-Host (Kapitel 35) kann man dasselbe ausprobieren:

```
$ sudo apt install cloud-image-utils
$ wget https://cloud.debian.org/images/cloud/bookworm/latest/debian-12-genericcloud-amd64.
    qcow2
$ cloud-localds seed.iso user-data.yml
$ qemu-img create -f qcow2 -F qcow2 -b debian-12-genericcloud-amd64.qcow2 test.qcow2 10G
$ virt-install --name cloudtest --memory 2048 --vcpus 2 --import \
    --disk test.qcow2 --disk seed.iso,device=cdrom \
    --os-variant debian12 --network network=default --noautoconsole
```

61.3. Infrastruktur als Code mit Terraform

Niveau: Fortgeschritten

Terraform (bzw. das freie Fork **OpenTofu**) beschreibt Infrastruktur deklarativ: Server, Netze, Firewalls, DNS-Einträge. Es vergleicht den beschriebenen mit dem tatsächlichen Zustand und führt nur die nötigen Änderungen durch. Der Zustand wird in einer *State-Datei* gespeichert.

```
# main.tf  (Beispiel mit Hetzner Cloud; Provider-Syntax je Anbieter verschieden)
terraform {
  required_providers {
    hcloud = {
      source  = "hetznercloud/hcloud"
      version = "~> 1.45"
    }
  }
}

variable "hcloud_token" {
  type      = string
  sensitive = true
}

provider "hcloud" {
  token = var.hcloud_token
}

resource "hcloud_ssh_key" "anna" {
  name       = "anna"
  public_key = file("~/.ssh/id_ed25519.pub")
}

resource "hcloud_firewall" "web" {
  name = "web"
  rule {
    direction  = "in"
    protocol   = "tcp"
    port       = "443"
    source_ips = ["0.0.0.0/0", "::/0"]
  }
  rule {
    direction  = "in"
    protocol   = "tcp"
    port       = "22"
    source_ips = ["203.0.113.0/24"]
  }
}

resource "hcloud_server" "web" {
  count        = 2
  name         = "web-${count.index + 1}"
  server_type  = "cx22"
  image        = "debian-12"
  location     = "nbg1"
  ssh_keys     = [hcloud_ssh_key.anna.id]
  firewall_ids = [hcloud_firewall.web.id]
  user_data    = file("user-data.yml")
}

output "ips" {
  value = hcloud_server.web[*].ipv4_address
}
```

```
$ terraform init                        # Provider herunterladen
$ terraform fmt; terraform validate     # Format und Syntax
$ export TF_VAR_hcloud_token='...'      # Geheimnisse nie im Code oder im Git!
$ terraform plan                        # Was wuerde sich aendern?
$ terraform apply                       # ausfuehren (Rueckfrage)
$ terraform output ips
$ terraform state list
$ terraform destroy                     # alles wieder loeschen
```

- **Immer erst `plan`** lesen: Er zeigt, was hinzukommt (+), sich ändert (~) oder *zerstört* wird (–). Server, die "ersetzt" werden, verlieren ihre lokalen Daten.
- **State-Datei** enthält oft Geheimnisse: nicht ins Git, sondern in ein gesichertes *Backend* (verschlüsselter Objektspeicher mit Sperre).
- **Module** bündeln wiederverwendbare Bausteine; **Workspaces** oder getrennte Verzeichnisse trennen Test und Produktion.
- **Zusammenspiel**: Terraform erzeugt die Server, cloud-init oder Ansible (Kapitel 56) konfiguriert sie.

61.4. Kubernetes im Überblick

Niveau: Experte

Kubernetes (K8s) verteilt Container (Kapitel 34) auf viele Rechner und hält den gewünschten Zustand ein: fällt ein Container aus, wird er neu gestartet; steigt die Last, werden weitere Instanzen gestartet.

Begriff	Bedeutung
Cluster, Node	Gruppe von Rechnern; *Control Plane* steuert, *Worker Nodes* führen aus
Pod	Kleinste Einheit: ein oder mehrere gemeinsam laufende Container
Deployment	Beschreibt, wie viele Replikate eines Pods laufen sollen, inkl. Rolling Updates
Service	Stabile Adresse und Lastverteilung vor mehreren Pods
Ingress	HTTP-Routing von außen zu Services (ähnlich Reverse-Proxy)
ConfigMap, Secret	Konfiguration und Geheimnisse, getrennt vom Image
Namespace	Logische Trennung innerhalb eines Clusters

```
# deployment.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web
spec:
  replicas: 3
  selector:
    matchLabels: { app: web }
  template:
    metadata:
      labels: { app: web }
    spec:
      containers:
        - name: nginx
          image: nginx:1.27
          ports: [ { containerPort: 80 } ]
          resources:
            requests: { cpu: 100m, memory: 64Mi }
            limits:   { cpu: 500m, memory: 256Mi }
---
apiVersion: v1
kind: Service
metadata:
  name: web
spec:
  selector: { app: web }
  ports: [ { port: 80, targetPort: 80 } ]

$ kubectl apply -f deployment.yml
$ kubectl get pods,svc -o wide
$ kubectl describe pod web-xyz
$ kubectl logs -f deploy/web
$ kubectl scale deploy/web --replicas=5
$ kubectl rollout undo deploy/web
$ kubectl exec -it web-xyz -- sh
```

Für den Einstieg eignen sich `minikube`, `kind` oder `k3s` (leichtgewichtig, auch für einzelne Server). Kubernetes ist mächtig, aber komplex: Für wenige Server genügt oft Compose (Kapitel 34) oder systemd.

61.5. Kosten, Sicherheit und Betrieb

Niveau: Fortgeschritten

- **Kosten**: Abrechnung nach Nutzung. Vergessene Ressourcen (Server, Volumes, Snapshots, öffentliche IP-Adressen) kosten weiter. Budgets und Alarme einrichten, Ressourcen markieren (*Tags*), Test-Umgebungen automatisch löschen.
- **Zugriff**: Identitäts- und Rechteverwaltung (IAM) nach dem Prinzip der geringsten Rechte; Mehrfaktor-Authentifizierung; keine Zugriffsschlüssel im Code oder Git; kurzlebige Zugangsdaten.
- **Netz**: Standardmäßig alles schließen (Security Groups/Firewalls), Verwaltung nur über Bastion oder VPN (Kapitel 48), Datenbanken nie öffentlich.
- **Backups und Wiederherstellung**: Snapshots der Anbieter ersetzen keine Strategie (Kapitel 53); Backups in anderer Region/anderem Konto.

- **Lock-in und Datenschutz**: Proprietäre Dienste binden an einen Anbieter. Standort der Daten, Auftragsverarbeitung und Verschlüsselung prüfen (DSGVO).
- **Unveränderliche Server**: Statt auf Servern zu patchen, neue Images bauen (z. B. mit Packer) und Server ersetzen.

Übungen und Kontrollfragen

1. Was unterscheidet IaaS von PaaS?

   Lösung: Bei IaaS verantwortet der Kunde Betriebssystem und alles darüber; bei PaaS betreibt der Anbieter auch die Plattform (Laufzeit, Datenbank), der Kunde liefert Anwendung und Daten.

2. Wofür wird cloud-init verwendet und wann läuft es?

   Lösung: Zur automatischen Erstkonfiguration (Benutzer, Schlüssel, Pakete, Dateien, Befehle) eines Cloud-Servers beim ersten Start anhand der User-Data.

3. Was zeigt `terraform plan`?

   Lösung: Welche Ressourcen angelegt, geändert oder zerstört würden, ohne etwas auszuführen.

4. Warum gehört die Terraform-State-Datei nicht ins öffentliche Git?

   Lösung: Sie kann Geheimnisse (Passwörter, Tokens) und Infrastrukturdetails enthalten; Ablage in einem geschützten Backend mit Sperre.

5. Was ist ein Pod in Kubernetes?

   Lösung: Die kleinste Einheit: ein oder mehrere gemeinsam geplante Container mit gemeinsamem Netzwerk.

6. Nennen Sie drei typische Cloud-Sicherheitsfehler.

   Lösung: Öffentlich zugängliche Speicher oder Datenbanken, zu weite Rechte (kein Prinzip der geringsten Rechte), Zugangsschlüssel im Code/Repository, fehlende Mehrfaktor-Authentifizierung.

62. Skriptprojekte aus der Praxis

Inhalt: Wiederverwendbare Skripte für typische Administrationsaufgaben: Strukturvorlage, Parameter, Fehlerbehandlung, Protokollierung, Sicherung, Aufräumen, Benutzeranlage, Systemprüfung, Log-Auswertung, Sperren und Tests mit ShellCheck.

Stufe: [F] Fortgeschritten bis [X] Experte

Lernziele: Nach diesem Kapitel können Sie

- robuste Skripte mit Strukturvorlage, Parameterprüfung und Fehlerbehandlung zu schreiben,
- typische Aufgaben (Sicherung, Aufräumen, Auswertung, Gesundheitsprüfung) zu automatisieren,
- Skripte mit `shellcheck` zu prüfen und zeitgesteuert auszuführen.

Vorkenntnisse: Kapitel 13 (Shellskripte), Kapitel 9 (reguläre Ausdrücke) und Kapitel 36 (Zeitsteuerung).

62.1. Strukturvorlage für robuste Skripte

Niveau: Fortgeschritten

Gute Administrationsskripte folgen einem festen Gerüst: strenger Modus, Parameterprüfung, Aufräumen bei Abbruch, Protokollierung.

```
#!/usr/bin/env bash
# skript.sh - Kurzbeschreibung
# Aufruf: skript.sh [-v] [-n] ZIELVERZEICHNIS
set -euo pipefail
IFS=$'\n\t'

PROG=${0##*/}
VERBOSE=0
DRYRUN=0
LOCKFILE=/var/lock/${PROG}.lock

usage() {
    cat <<EOF
Verwendung: $PROG [-v] [-n] ZIELVERZEICHNIS
  -v   ausfuehrliche Ausgabe
  -n   Probelauf, nichts aendern
  -h   diese Hilfe
EOF
}

log()  { printf '%s %s [%s] %s\n' "$(date '+%F %T')" "$PROG" "$1" "$2" >&2; }
info() { log INFO "$*"; }
fehler() { log ERROR "$*"; }
run()  { if (( DRYRUN )); then info "(Probelauf) $*"; else "$@"; fi; }

tmpdir=$(mktemp -d)
cleanup() { rm -rf "$tmpdir"; rm -f "$LOCKFILE"; }
trap cleanup EXIT
trap 'fehler "Abbruch durch Signal"; exit 130' INT TERM

while getopts ':vnh' opt; do
    case $opt in
        v) VERBOSE=1 ;;
        n) DRYRUN=1 ;;
        h) usage; exit 0 ;;
        \?) fehler "Unbekannte Option: -$OPTARG"; usage; exit 2 ;;
```

```
    esac
done
shift $((OPTIND - 1))

[[ $# -eq 1 ]] || { usage; exit 2; }
ZIEL=$1
[[ -d $ZIEL ]] || { fehler "Kein Verzeichnis: $ZIEL"; exit 1; }

# Nur eine Instanz gleichzeitig
exec 9>"$LOCKFILE"
flock -n 9 || { fehler "Laeuft bereits"; exit 1; }

info "Start"
# ... eigentliche Arbeit ...
info "Fertig"
```

- `set -euo pipefail`: Abbruch bei Fehlern (`-e`), bei undefinierten Variablen (`-u`) und bei Fehlern innerhalb einer Pipe.
- `trap ... EXIT`: Aufräumen, egal wie das Skript endet.
- `flock`: Verhindert parallele Läufe (z. B. wenn der Cron-Job länger dauert als der Abstand).
- `getopts`: Standardisierte Optionsverarbeitung. Variablen immer in Anführungszeichen setzen.
- Ein **Probelaufmodus** (`-n`) macht gefährliche Skripte beherrschbar.
- Meldungen auf `stderr` (`>&2`), Ergebnisse auf `stdout` – so lassen sich Skripte in Pipes verwenden.

62.2. Projekt 1: Verzeichnis sichern mit Rotation

Niveau: Fortgeschritten

```
#!/usr/bin/env bash
# sichern.sh - Quelle als tar.zst sichern, alte Sicherungen loeschen
set -euo pipefail

QUELLE=${1:?Quellverzeichnis fehlt}
ZIEL=${2:-/backup}
BEHALTEN=${3:-7}                        # Anzahl Sicherungen
NAME=$(basename "$QUELLE")
DATEI="$ZIEL/${NAME}-$(date +%F_%H%M).tar.zst"

mkdir -p "$ZIEL"
tar --create --zstd --file "$DATEI" --exclude='*.tmp' --exclude='cache' -C "$(dirname "
    $QUELLE")" "$NAME"
sha256sum "$DATEI" > "$DATEI.sha256"
echo "Sicherung: $DATEI ($(du -h "$DATEI" | cut -f1))"

# Rotation: nur die neuesten $BEHALTEN Sicherungen behalten
ls -1t "$ZIEL/${NAME}"-*.tar.zst | tail -n +$((BEHALTEN + 1)) | while read -r alt; do
    echo "Loesche $alt"
    rm -f -- "$alt" "$alt.sha256"
done
```

Ergänzungen für den Produktivbetrieb: Ergebnis prüfen (`tar --test-label` bzw. `tar -tf`), Wiederherstellung testen und die Datei auf einen zweiten Rechner kopieren (`rsync`, Kapitel 53).

62.3. Projekt 2: Platten- und Dienstwächter

Niveau: Fortgeschritten

```
#!/usr/bin/env bash
# gesundheit.sh - einfache Systempruefung mit Exit-Code
set -uo pipefail
SCHWELLE=${1:-85}
STATUS=0

# Plattenbelegung
while read -r nutzung mp; do
    pct=${nutzung%\%}
    if (( pct >= SCHWELLE )); then
        echo "WARNUNG: $mp zu ${pct}% belegt"
        STATUS=1
    fi
done < <(df --output=pcent,target -x tmpfs -x devtmpfs | tail -n +2)

# Dienste
for dienst in sshd chronyd nginx; do
    if systemctl list-unit-files "$dienst.service" >/dev/null 2>&1 \
       && ! systemctl is-active --quiet "$dienst"; then
        echo "WARNUNG: Dienst $dienst laeuft nicht"
        STATUS=1
    fi
done

# Speicher und Last
frei=$(awk '/MemAvailable/ {printf "%d", $2/1024}' /proc/meminfo)
(( frei < 200 )) && { echo "WARNUNG: nur $frei MiB Speicher verfuegbar"; STATUS=1; }
last=$(cut -d' ' -f1 /proc/loadavg)
kerne=$(nproc)
if awk -v l="$last" -v k="$kerne" 'BEGIN{exit !(l > k*2)}'; then
    echo "WARNUNG: Last $last bei $kerne Kernen"; STATUS=1
fi

# Fehlgeschlagene systemd-Einheiten
failed=$(systemctl --failed --no-legend | wc -l)
(( failed > 0 )) && { echo "WARNUNG: $failed fehlgeschlagene Einheiten"; STATUS=1; }

exit $STATUS
```

Per `cron` oder systemd-Timer ausgeführt und mit `mail` oder einem Monitoring-System verbunden, wird daraus ein einfacher Wächter (Kapitel 57).

62.4. Projekt 3: Aufräumen und Benutzeranlage

Niveau: Fortgeschritten

```
# alte Dateien loeschen: erst anzeigen, dann loeschen
$ find /var/tmp/export -type f -mtime +30 -print
$ find /var/tmp/export -type f -mtime +30 -delete
# grosse Logdateien finden
$ find /var/log -type f -size +100M -exec ls -lh {} +
# leere Verzeichnisse entfernen
$ find /srv/projekte -type d -empty -delete
```

Benutzer aus einer CSV-Datei anlegen (`benutzer.csv`: `name;gruppe;kommentar`):

```
#!/usr/bin/env bash
# neue_benutzer.sh benutzer.csv  - als root ausfuehren
set -euo pipefail
(( EUID == 0 )) || { echo "Root erforderlich" >&2; exit 1; }

while IFS=';' read -r name gruppe kommentar; do
    [[ -z $name || $name == \#* ]] && continue
    if id "$name" &>/dev/null; then
        echo "Existiert bereits: $name"; continue
    fi
    getent group "$gruppe" >/dev/null || groupadd "$gruppe"
    useradd --create-home --shell /bin/bash --gid "$gruppe" --comment "$kommentar" "$name"
    passwort=$(openssl rand -base64 12)
    echo "$name:$passwort" | chpasswd
    chage --lastday 0 "$name"            # Passwort beim ersten Login aendern
    echo "$name  $passwort" >> /root/neue_passwoerter.txt
    echo "Angelegt: $name"
done < "${1:?CSV-Datei fehlt}"
chmod 600 /root/neue_passwoerter.txt
```

62.5. Projekt 4: Logdateien auswerten

Niveau: Fortgeschritten

```
#!/usr/bin/env bash
# fehlgeschlagene SSH-Anmeldungen der letzten 24 Stunden
journalctl -u ssh --since "24 hours ago" --no-pager \
  | grep -E 'Failed password|Invalid user' \
  | grep -oE '[0-9]{1,3}(\.[0-9]{1,3}){3}' \
  | sort | uniq -c | sort -rn | head -10 \
  | awk '{printf "%5d Versuche von %s\n", $1, $2}'

# Webserver: haeufigste URLs mit Status 404
awk '$9 == 404 {print $7}' /var/log/nginx/access.log | sort | uniq -c | sort -rn | head

# Antwortzeiten (Logformat mit $request_time als letztem Feld)
awk '{s+=$NF; n++} END {printf "Mittel: %.3f s ueber %d Anfragen\n", s/n, n}' /var/log/nginx/
    access.log
```

62.6. Qualität: ShellCheck, Tests und Zeitsteuerung

Niveau: Experte

```
$ sudo apt install shellcheck
$ shellcheck sichern.sh                    # findet typische Fehler und Stolperfallen
$ bash -n sichern.sh                       # nur Syntax pruefen
$ bash -x sichern.sh /srv/daten            # Ablauf mit Trace-Ausgabe
$ set -x; ...; set +x                      # Trace nur fuer einen Abschnitt
```

- **ShellCheck** warnt vor fehlenden Anführungszeichen, Wortzerlegung, unsicheren `rm`-Aufrufen, Portabilitätsproblemen und vielem mehr. Es gehört in jede Skript-Pipeline (Git-Hook, CI).
- **Tests**: Mit dem Framework `bats` lassen sich Skripte automatisiert prüfen; kritische Funktionen in kleine, testbare Einheiten aufteilen.
- **Fallstricke**: Dateinamen mit Leerzeichen (`find -print0 | xargs -0`), Variablen immer quoten, `cd` prüfen (`cd dir || exit 1`), nie `rm -rf "$VAR/"` mit möglicherweise leerer Variable (Schutz: `set -u` oder `${VAR:?}`).
- **Zeitsteuerung**: systemd-Timer sind der moderne Ersatz für Cron (Kapitel 36), inklusive Protokollierung im Journal und `Persistent=true` für verpasste Läufe.
- **Rechte**: Skripte, die als Root laufen, dürfen nur für Root schreibbar sein; Konfigurationsdateien mit Passwörtern `chmod 600`.
- **Wartbarkeit**: Kommentare für das *Warum*, Hilfetext, Versionierung in Git (Kapitel 16). Ab etwa 200 Zeilen oder komplexer Datenverarbeitung ist eine andere Sprache (Python) oft die bessere Wahl.

```
# systemd-Timer fuer das Sicherungsskript
# /etc/systemd/system/sichern.service
[Unit]
Description=Verzeichnis sichern
[Service]
Type=oneshot
ExecStart=/usr/local/bin/sichern.sh /srv/daten /backup 14

# /etc/systemd/system/sichern.timer
[Unit]
Description=Taegliche Sicherung
[Timer]
OnCalendar=*-*-* 02:30:00
Persistent=true
RandomizedDelaySec=10m
[Install]
WantedBy=timers.target

$ sudo systemctl daemon-reload
$ sudo systemctl enable --now sichern.timer
$ systemctl list-timers sichern.timer
$ sudo journalctl -u sichern.service -n 20
```

Übungen und Kontrollfragen

1. Was bewirkt `set -euo pipefail`?

   Lösung: Abbruch bei Fehlern, bei Verwendung nicht gesetzter Variablen und wenn ein Befehl in einer Pipe fehlschlägt.

2. Wie verhindern Sie, dass ein Skript mehrfach gleichzeitig läuft?

   Lösung: Mit einer Sperre über `flock` (oder Lock-Datei), z. B. `exec 9>lock; flock -n 9 || exit 1`.

3. Wozu dient `trap cleanup EXIT`?

   Lösung: Die Funktion wird beim Beenden des Skripts immer ausgeführt (auch bei Fehlern) und räumt temporäre Dateien auf.

4. Warum ist `rm -rf "$DIR/"` gefährlich und wie schützen Sie sich?

   Lösung: Ist `DIR` leer, wird `/` adressiert. Schutz: `set -u`, `${DIR:?}`, Prüfung der Variable vor dem Löschen.

5. Wie testen Sie ein Skript, ohne etwas zu verändern?

   Lösung: Mit einem Probelaufmodus (`-n`), `bash -n` für Syntax, `bash -x` für den Ablauf und `shellcheck` für statische Prüfung.

6. Warum sind systemd-Timer oft besser als Cron-Jobs?

   Lösung: Sie protokollieren im Journal, können verpasste Läufe nachholen (`Persistent`), haben Abhängigkeiten und Ressourcenlimits.

63. Zertifizierungen und Lernpfade

Inhalt: Überblick über die gängigen Linux-Zertifizierungen (LPIC, LFCS, RHCSA, Linux+), Zuordnung der Kapitel dieses Buchs zu den Prüfungsthemen, Lernstrategien, Übungsumgebungen und Hinweise zur Prüfungsvorbereitung.

Stufe: [E] Einsteiger bis [F] Fortgeschritten

Lernziele: Nach diesem Kapitel können Sie

- die wichtigsten Linux-Zertifizierungen zu unterscheiden und die passende Prüfung auszuwählen,
- die Themen der Prüfungen den Kapiteln dieses Buchs zuzuordnen,
- einen realistischen Lernplan und eine geeignete Übungsumgebung aufzubauen.

Vorkenntnisse: Keine besonderen; sinnvoll nach dem Durcharbeiten von Teil A und Teil B.

63.1. Warum zertifizieren?

Niveau: Einsteiger

Eine Zertifizierung belegt Kenntnisse gegenüber Arbeitgebern und strukturiert das eigene Lernen. Sie ersetzt keine Praxis, bietet aber einen klaren Lehrplan. Prüfungsinhalte und -formate ändern sich: Die Angaben hier sind eine Orientierung – aktuelle Themenlisten stehen immer auf den Webseiten der Anbieter.

Zertifikat	Anbieter	Charakter	Geeignet für
LPIC-1 (und -2, -3)	Linux Professional Institute	Herstellerneutral; Prüfungen mit Multiple-Choice und Eingabeaufgaben; in Stufen aufgebaut	Einstieg bis Fortgeschrittene, Debian- und RHEL-Welt
LFCS	Linux Foundation	Praktische Online-Prüfung an der Kommandozeile	Administratoren mit Praxiserfahrung
RHCSA (RHCE)	Red Hat	Praktische Prüfung auf RHEL; hohe Anerkennung in Unternehmen	Umgebungen mit RHEL, Rocky, Alma
CompTIA Linux+	CompTIA	Herstellerneutral; Multiple-Choice und praktische Aufgaben	Einstieg in Administration und Support
LPI Security Essentials, DevOps Tools	LPI	Spezialisierungen	Weiterbildung
CKA / CKAD	Linux Foundation (CNCF)	Praktische Kubernetes-Prüfungen	Container- und Cloud-Betrieb

63.2. Zuordnung der Prüfungsthemen zu den Kapiteln

Niveau: Fortgeschritten

Die folgende Tabelle ordnet typische Themenbereiche den Kapiteln dieses Buchs zu. Sie ist keine offizielle Prüfungsliste.

Themenbereich	Kapitel
Kommandozeile, Hilfesystem, Dateien	3, 6, 8, 9, 10
Editoren (vi/vim)	5
Shell und Skripte	11, 13, 14, 62
Archive, Versionierung, Build	16, 17
Paketverwaltung	19
Benutzer, Gruppen, Rechte, PAM, SELinux	20, 22, 21, 23
Prozesse, Kernel, Leistung	24, 25, 59
Platten, Partitionen, LVM, RAID, Dateisysteme	26, 27, 28, 29, 30
Bootvorgang, systemd, Zeitsteuerung, Logging	32, 36, 37
Container, Virtualisierung	33, 34, 35, 61
Netzwerk-Grundlagen und -Konfiguration	38, 39, 41
Netzdienste (DNS, DHCP, NTP, Web, Mail, Samba/NFS)	44, 45, 46, 50, 49
Sicherheit, SSH, Firewall, VPN, TLS	52, 42, 43, 48, 58
Automatisierung, Überwachung	56, 57
Backup, Wiederherstellung, Fehlersuche	53, 54, 55
Hochverfügbarkeit	60

63.3. Lernstrategie und Übungsumgebung

Niveau: Fortgeschritten

- **Praxis vor Theorie**: Jeder Befehl wird selbst ausgeführt, verändert und bewusst falsch benutzt. Praktische Prüfungen verlangen Geschwindigkeit und Sicherheit an der Tastatur.
- **Eigene Laborumgebung**: Zwei bis drei virtuelle Maschinen (VirtualBox, KVM/libvirt, Multipass) mit Debian/Ubuntu und Rocky/Alma. Snapshots erlauben gefahrloses Experimentieren (Kapitel 35).
- **Alles aufschreiben**: Ein persönliches Notizbuch mit Befehlen, Dateien und Fehlerbildern; Konfigurationen in Git (Kapitel 16).
- **Nur mit `man` und `/usr/share/doc`**: In praktischen Prüfungen steht meist nur die lokale Dokumentation bereit. Üben Sie `man -k`, `apropos`, `--help` (Kapitel 3).
- **Szenarien statt Einzelbefehle**: "Richte einen Webserver mit TLS ein, der nach Neustart läuft, und erlaube ihn durch die Firewall." Kombinierte Aufgaben trainieren genau die Prüfungssituation.
- **Zeitplan**: Regelmäßig (z. B. vier mal pro Woche eine Stunde) schlägt Marathon-Sitzungen. Themen im Abstand wiederholen.
- **Dauerhaftigkeit prüfen**: Nach Konfigurationsänderungen neu starten – überlebt die Einstellung den Reboot (`systemctl enable`, `/etc/fstab`, `sysctl.d`, Firewall dauerhaft)?

```
# Lernlabor mit Multipass (Ubuntu-VMs)
$ sudo snap install multipass
$ multipass launch --name lab1 --cpus 2 --memory 2G --disk 15G 22.04
$ multipass shell lab1
$ multipass snapshot lab1 --name sauber
$ multipass restore lab1.sauber
$ multipass delete --purge lab1

# Alternative: libvirt mit Cloud-Image (siehe Kapitel Cloud)
$ virt-install --name rocky1 --memory 2048 --vcpus 2 --disk size=15 \
    --cdrom Rocky-9-minimal.iso --os-variant rocky9 --network network=default
```

63.4. Prüfungsvorbereitung

Niveau: Fortgeschritten

1. **Aktuelle Themenliste** beim Anbieter herunterladen und jeden Punkt abhaken: "Kann ich das ohne Hilfe?"
2. **Schwächen gezielt trainieren**: Typisch schwierig sind reguläre Ausdrücke, `sed`/`awk`, LVM/RAID, SELinux, Firewall-Regeln, Bootprobleme und Netzkonfiguration.
3. **Zeitdruck üben**: Aufgaben mit Zeitlimit lösen; Muster-Skripte und häufige Konfigurationsdateien kennen.
4. **Prüfungsumgebung kennen**: Tastaturlayout, Fernzugriff, erlaubte Hilfsmittel, Bildschirmaufteilung und Zeitablauf vorab klären.
5. **Am Prüfungstag**: Aufgaben zuerst überfliegen, Einfaches zuerst lösen, Ergebnisse jeweils prüfen (Neustart-fest, Rechte, Dienste), Zwischenstände sichern.

> **Zertifikat und Praxis**
>
> Prüfungen messen nur einen Ausschnitt. Wer sie als Etappenziel nutzt, lernt strukturiert; entscheidend bleibt, dass das Gelernte im echten Betrieb angewendet wird – im eigenen Heimnetz, in Test-Umgebungen, in freien Projekten oder im Beruf. Auch die Teilnahme an Open-Source-Projekten, Foren und Usergroups verbessert Praxiswissen und Fehlersuche.

63.5. Weiterführende Themen und Karrierepfade

Niveau: Experte

- **Systemadministration → Plattform/DevOps/SRE**: Automatisierung (Kapitel 56), Infrastruktur als Code (Kapitel 61), Kubernetes, CI/CD, Monitoring und Observability (Kapitel 57).
- **Sicherheit**: Härtung, Penetrationstests, SIEM, Forensik; Vertiefung in TLS/PKI (Kapitel 58) und SELinux (Kapitel 23).
- **Netzwerk**: Routing-Protokolle (BGP, OSPF), VLANs, SDN; Linux als Router und Firewall (Kapitel 43).
- **Speicher und Datenbanken**: Ceph, ZFS, Replikation und Hochverfügbarkeit (Kapitel 30, 51, 60).
- **Entwicklung nahe am System**: Kernelmodule, eBPF, Systemprogrammierung in C oder Rust.
- **Bücher und Quellen**: Handbuchseiten, Dokumentation der Distributionen (Debian Administrator's Handbook, Arch Wiki als Nachschlagewerk), RFCs, Projektwebseiten und Quellcode.

Übungen und Kontrollfragen

1. Worin unterscheiden sich LPIC und RHCSA grundsätzlich?

   Lösung: LPIC ist herstellerneutral und stufenweise aufgebaut; RHCSA ist eine praktische Prüfung auf Red Hat Enterprise Linux mit hoher Anerkennung im RHEL-Umfeld.

2. Warum sind virtuelle Maschinen mit Snapshots eine gute Übungsumgebung?

   Lösung: Experimente sind gefahrlos, weil man zum sauberen Ausgangszustand zurückkehren kann; mehrere Maschinen erlauben Netz- und Dienstszenarien.

3. Wie stellen Sie sicher, dass eine Konfiguration einen Neustart überlebt?

   Lösung: Dienste mit `systemctl enable` aktivieren, Mounts in `/etc/fstab`, Kernelparameter in `sysctl.d`, Firewall-Regeln dauerhaft speichern; anschließend neu starten und prüfen.

4. Welche Hilfsmittel stehen in praktischen Prüfungen typischerweise zur Verfügung?

   Lösung: Meist nur die lokale Dokumentation (`man`, `--help`, `/usr/share/doc`); deshalb das Hilfesystem üben.

A. Befehlsübersicht

Die Übersicht fasst die wichtigsten Befehle nach Themen zusammen. Die Spalte *N* zeigt das Niveau: [E] Einsteiger, [F] Fortgeschritten, [X] Experte. Details stehen im jeweiligen Kapitel.

Navigation und Dateien

Befehl	Zweck	N
`pwd`, `cd`, `ls -la`	Verzeichnis anzeigen, wechseln, auflisten	[E]
`cp`, `mv`, `rm`, `mkdir`	Kopieren, Verschieben, Löschen, Anlegen	[E]
`cat`, `less`, `head`, `tail -f`	Dateien anzeigen und verfolgen	[E]
`find`, `locate`	Dateien suchen	[E]
`ln -s`	Symbolischen Link anlegen	[E]
`man`, `--help`, `info`	Hilfe	[E]
`tar`, `gzip`, `zip`	Archive und Kompression	[E]
`file`, `stat`, `du`, `df`	Dateityp, Metadaten, Platzbedarf	[E]

Text verarbeiten

Befehl	Zweck	N
`grep`, `egrep`	Zeilen nach Muster filtern	[E]
`sort`, `uniq`, `wc`, `cut`, `tr`	Sortieren, Zählen, Spalten, Zeichen ersetzen	[E]
`nano`, `vim`	Texteditoren	[E]
`sed`, `awk`	Stream-Bearbeitung, spaltenweise Auswertung	[F]
`xargs`, `tee`	Argumente aus Eingabe bilden, Ausgabe aufteilen	[F]
`diff`, `patch`, `comm`	Dateien vergleichen und patchen	[F]

Benutzer, Rechte, Prozesse

Befehl	Zweck	N
`whoami`, `id`, `who`, `w`	Identität und angemeldete Benutzer	[E]
`sudo`, `su`	Befehle als anderer Benutzer	[E]
`chmod`, `chown`, `chgrp`, `umask`	Rechte und Besitzer	[E]
`useradd`, `usermod`, `userdel`, `passwd`	Benutzerverwaltung	[F]
`groupadd`, `gpasswd`, `groups`	Gruppenverwaltung	[F]
`ps`, `top`, `htop`, `pstree`	Prozesse anzeigen	[E]
`kill`, `killall`, `pkill`	Signale senden, Prozesse beenden	[E]
`jobs`, `bg`, `fg`, `nohup`	Jobsteuerung in der Shell	[F]
`nice`, `renice`	Priorität ändern	[F]
`setfacl`, `getfacl`, `chattr`	ACLs und Dateiattribute	[X]

Pakete und Systemverwaltung

Befehl	Zweck	N
`apt`, `dpkg`	Paketverwaltung Debian/Ubuntu	[E]
`dnf`, `rpm`	Paketverwaltung RHEL/Fedora	[E]
`systemctl`	Dienste und Targets steuern	[E]
`journalctl`	Journal lesen und filtern	[E]
`shutdown`, `reboot`, `poweroff`	Herunterfahren und Neustart	[E]
`crontab`, `at`	Zeitgesteuerte Aufgaben	[F]
`hostnamectl`, `timedatectl`	Hostname, Zeit und Zeitzone	[F]
`logrotate`, `logger`	Log-Rotation, Meldungen schreiben	[F]
`systemd-analyze`	Startzeit und Absicherung analysieren	[X]
`sysctl`	Kernelparameter zur Laufzeit	[X]

Platten und Dateisysteme

Befehl	Zweck	N
`lsblk`, `blkid`, `fdisk -l`	Laufwerke und Partitionen anzeigen	[E]
`mount`, `umount`, `findmnt`	Einhängen, Aushängen	[E]
`fdisk`, `gdisk`, `parted`	Partitionieren	[F]
`mkfs.*`, `fsck`, `tune2fs`	Dateisysteme anlegen, prüfen, einstellen	[F]
`mkswap`, `swapon`, `free`	Swap und Speicher	[F]
`pvcreate`, `vgcreate`, `lvcreate`, `lvextend`	LVM	[X]
`mdadm`, `cryptsetup`	Software-RAID, Verschlüsselung	[X]
`dd`, `ddrescue`	Abbilder und Datenrettung	[X]

Netzwerk

Befehl	Zweck	N
`ip a`, `ip r`, `ip link`	Adressen, Routen, Schnittstellen	[E]
`ping`, `traceroute`, `mtr`	Erreichbarkeit und Weg	[E]
`ss -tulpn`	Lauschende Ports und Verbindungen	[E]
`dig`, `host`, `getent hosts`	DNS-Abfragen	[E]
`curl`, `wget`	HTTP-Abrufe und Downloads	[E]
`ssh`, `scp`, `rsync`, `ssh-keygen`	Entfernter Zugriff, Dateiübertragung, Schlüssel	[E]
`nmcli`, `netplan`	Netzwerkkonfiguration	[F]
`ufw`, `firewall-cmd`, `nft`	Firewall	[F]
`nc`, `nmap`	Portprüfung, Scan (nur eigenes Netz)	[F]
`tcpdump`, `wireshark`	Pakete mitschneiden	[X]

B. Übungen mit Lösungen

Die Aufgaben vertiefen die Kapitel und sind nach Niveau markiert ([E] Einsteiger, [F] Fortgeschritten, [X] Experte). Alle Übungen sollten in einer virtuellen Maschine oder einem Testsystem bearbeitet werden, nicht auf einem produktiven Rechner. Die Lösungen zeigen jeweils einen möglichen Weg; es gibt meist mehrere.

Teil A: Grundlagen

Übung 1 – Dateien und Verzeichnisse [E]

Legen Sie in Ihrem Home-Verzeichnis die Struktur `uebung/projekt/{doku,quellen,tmp}` an, erzeugen Sie in `doku` drei Dateien `a.txt`, `b.txt`, `c.txt`, kopieren Sie diese nach `quellen`, benennen Sie in `quellen` die Datei `c.txt` in `z.txt` um und löschen Sie anschließend `tmp`.

```
$ mkdir -p ~/uebung/projekt/{doku,quellen,tmp}
$ cd ~/uebung/projekt
$ touch doku/{a,b,c}.txt
$ cp doku/*.txt quellen/
$ mv quellen/c.txt quellen/z.txt
$ rmdir tmp
$ tree ~/uebung            # Kontrolle (Paket tree)
```

Übung 2 – Suchen und Filtern [E]

Zählen Sie, wie viele Benutzerkonten in `/etc/passwd` eine Login-Shell `bash` haben, und listen Sie die Namen sortiert auf.

```
$ grep -c ':/bin/bash$' /etc/passwd
$ grep ':/bin/bash$' /etc/passwd | cut -d: -f1 | sort
```

Übung 3 – find [E]

Finden Sie in `/etc` alle Dateien, die größer als 100 KiB sind und innerhalb der letzten 30 Tage geändert wurden. Fehlermeldungen sollen unterdrückt werden.

```
$ find /etc -type f -size +100k -mtime -30 2>/dev/null
```

Übung 4 – Pipes und awk [F]

Bestimmen Sie die fünf größten Verzeichnisse unterhalb von `/var` und geben Sie die Anzahl der `bash`-Prozesse aus, die gerade laufen. Lösen Sie anschließend mit `awk`: (a) die Größen der fünf Verzeichnisse in MiB mit einer Nachkommastelle ausgeben, (b) die `bash`-Prozesse zählen, (c) die Anzahl der Prozesse je Benutzer ausgeben und (d) die Summe der Spalte `RSS` (Speicherbedarf in KiB) aller Prozesse in MiB berechnen.

```
$ du -s /var/* 2>/dev/null | sort -rn | head -5
$ ps -e -o comm= | grep -c '^bash$'
$ ps -e -o comm= | sort | uniq -c | sort -rn | head -3

# (a) Groesse in MiB (du -s liefert KiB), Pfad dahinter
$ du -s /var/* 2>/dev/null | sort -rn | head -5 \
    | awk '{printf "%8.1f MiB  %s\n", $1/1024, $2}'

# (b) bash-Prozesse zaehlen
$ ps -e -o comm= | awk '$1 == "bash" {n++} END {print n+0}'

# (c) Prozesse je Benutzer
$ ps -e -o user= | awk '{anz[$1]++} END {for (u in anz) print anz[u], u}' | sort -rn

# (d) Summe des Speicherbedarfs (RSS) in MiB
$ ps -e -o rss= | awk '{s += $1} END {printf "%.1f MiB\n", s/1024}'
```

Bei awk steht `$1` für die erste Spalte; der `END`-Block wird nach der letzten Eingabezeile einmal ausgeführt. Mit `n+0` wird auch bei null Treffern `0` statt einer leeren Ausgabe erzeugt.

Übung 5 – sed [F]

Ersetzen Sie in einer Datei `konfig.txt` alle Vorkommen von `Port=80` durch `Port=8080`, aber erst, nachdem Sie eine Sicherung `konfig.txt.bak` angelegt haben. Löschen Sie außerdem alle Kommentarzeilen und Leerzeilen in der Ausgabe.

```
$ sed -i.bak 's/Port=80$/Port=8080/' konfig.txt
$ sed -e '/^#/d' -e '/^$/d' konfig.txt
```

Übung 6 – Shellskript [F]

Schreiben Sie ein Skript `gross.sh`, das für jede als Argument übergebene Datei prüft, ob sie existiert und lesbar ist, und ihre Zeilenanzahl ausgibt. Bei einem Fehler soll es mit Status 1 enden.

```
#!/bin/bash
status=0
if [ $# -eq 0 ]; then
    echo "Aufruf: $0 datei..." >&2
    exit 1
fi
for datei in "$@"; do
    if [ -r "$datei" ]; then
        echo "$datei: $(wc -l < "$datei") Zeilen"
    else
        echo "Fehler: $datei nicht lesbar" >&2
        status=1
    fi
done
exit $status
```

Übung 7 – Archive [E]

Packen Sie das Verzeichnis `~/uebung` in ein gzip-komprimiertes Archiv, zeigen Sie den Inhalt an und entpacken Sie es in `/tmp/test`.

```
$ tar -czf uebung.tar.gz -C ~ uebung
$ tar -tzf uebung.tar.gz
$ mkdir -p /tmp/test && tar -xzf uebung.tar.gz -C /tmp/test
```

Teil B: Systemadministration

Übung 8 – Benutzer und Gruppen [E]

Legen Sie die Gruppe `projekt` und die Benutzer `anna` und `bob` an; beide sollen Mitglied der Gruppe sein. Erstellen Sie `/srv/projekt`, das der Gruppe `projekt` gehört, in dem beide schreiben dürfen und in dem neue Dateien automatisch die Gruppe erben.

```
# groupadd projekt
# useradd -m -s /bin/bash -G projekt anna
# useradd -m -s /bin/bash -G projekt bob
# passwd anna; passwd bob
# mkdir /srv/projekt
# chown root:projekt /srv/projekt
# chmod 2770 /srv/projekt          # setgid-Bit: Gruppe wird vererbt
$ ls -ld /srv/projekt
drwxrws--- 2 root projekt 4096 ... /srv/projekt
```

Übung 9 – Rechte verstehen [E]

Was bedeuten `-rwxr-x---`, `chmod 640 datei` und `chmod u+x,g-w,o= datei`? Wie sieht die Maske aus, wenn `umask 027` gilt?
Lösung: `rwxr-x---` = 750: Besitzer alles, Gruppe lesen und ausführen, andere nichts. `640`: Besitzer lesen/schreiben, Gruppe lesen, andere nichts. Das dritte Kommando fügt dem Besitzer `x` hinzu, entzieht der Gruppe `w` und entfernt alle Rechte der anderen. Bei `umask 027` erhalten neue Dateien `640` (666 minus 027) und neue Verzeichnisse `750` (777 minus 027).

Übung 10 – Prozesse [E]

Starten Sie `sleep 600` im Hintergrund, finden Sie seine PID und beenden Sie ihn sauber. Starten Sie danach `sleep 700` mit niedriger Priorität und prüfen Sie den Nice-Wert.

```
$ sleep 600 &
[1] 4711
$ pgrep -a sleep
$ kill 4711                        # SIGTERM
$ nice -n 15 sleep 700 &
$ ps -o pid,ni,cmd -C sleep
```

Übung 11 – Dateisystem und fstab [F]

Eine neue Platte `/dev/sdb` soll als eine ext4-Partition unter `/srv/daten` dauerhaft eingehängt werden.

```
# parted -s /dev/sdb mklabel gpt mkpart daten ext4 1MiB 100%
# mkfs.ext4 -L daten /dev/sdb1
# mkdir -p /srv/daten
# blkid /dev/sdb1                          # UUID notieren
# echo 'UUID=<uuid> /srv/daten ext4 defaults,nofail 0 2' >> /etc/fstab
# systemctl daemon-reload
# mount -a && findmnt /srv/daten
```

Übung 12 – LVM [X]

Fassen Sie `/dev/sdb1` und `/dev/sdc1` zu einer Volume Group `vg_daten` zusammen, legen Sie ein Logical Volume von 5 GiB an und vergrößern Sie es später im laufenden Betrieb um 2 GiB samt Dateisystem.

```
# pvcreate /dev/sdb1 /dev/sdc1
# vgcreate vg_daten /dev/sdb1 /dev/sdc1
# lvcreate -L 5G -n lv_web vg_daten
# mkfs.ext4 /dev/vg_daten/lv_web
# mount /dev/vg_daten/lv_web /srv/web
# lvextend -r -L +2G /dev/vg_daten/lv_web    # -r vergroessert das Dateisystem mit
```

Übung 13 – systemd-Dienst und Timer [F]

Erstellen Sie einen Dienst `zeit.service`, der mit `logger` die aktuelle Zeit ins Journal schreibt, und einen Timer, der ihn alle 5 Minuten startet.

```
# cat > /etc/systemd/system/zeit.service << 'EOF'
[Unit]
Description=Zeit ins Journal schreiben

[Service]
Type=oneshot
ExecStart=/bin/sh -c 'logger -t zeit "Jetzt: $(date)"'
EOF
# cat > /etc/systemd/system/zeit.timer << 'EOF'
[Unit]
Description=Alle 5 Minuten

[Timer]
OnCalendar=*:0/5
Persistent=true

[Install]
WantedBy=timers.target
EOF
# systemctl daemon-reload
# systemctl enable --now zeit.timer
$ systemctl list-timers zeit.timer
$ journalctl -t zeit -n 5
```

Übung 14 – Protokolle [F]

Zeigen Sie alle Fehlermeldungen (Priorität `err` oder schlimmer) des aktuellen Starts, dann nur die des `ssh`-Dienstes der letzten Stunde.

```
$ journalctl -b -p err
$ journalctl -u ssh --since "1 hour ago"
```

Teil C und D: Netzwerk, Sicherheit und Betrieb

Übung 15 – Subnetting [F]

Das Netz `192.168.10.0/24` soll in vier gleich große Subnetze geteilt werden. Geben Sie Maske, Netzadresse, Broadcast und Hostbereich jedes Subnetzes an.

Lösung: Zwei zusätzliche Netzbits ergeben `/26` (Maske `255.255.255.192`), je 64 Adressen, 62 Hosts.

Netz	Hosts	Broadcast
192.168.10.0/26	.1 – .62	192.168.10.63
192.168.10.64/26	.65 – .126	192.168.10.127
192.168.10.128/26	.129 – .190	192.168.10.191
192.168.10.192/26	.193 – .254	192.168.10.255

Übung 16 – Netzwerkdiagnose [E]

Ermitteln Sie IP-Adresse, Standard-Gateway und DNS-Server Ihres Rechners. Prüfen Sie dann Schritt für Schritt, ob `example.org` erreichbar ist.

```
$ ip -br addr
$ ip route | grep default
$ resolvectl status | grep -i 'DNS Servers'    # oder: cat /etc/resolv.conf
$ ping -c 2 192.168.1.1                         # Gateway (Adresse anpassen)
$ ping -c 2 1.1.1.1                             # Internet ohne DNS
$ dig +short example.org                        # Namensaufloesung
$ curl -I https://example.org                   # HTTP
```

Übung 17 – SSH-Schlüssel [F]

Richten Sie die Anmeldung per Schlüssel auf einem Testserver ein, legen Sie einen Alias `testsrv` an und kopieren Sie eine Datei dorthin.

```
$ ssh-keygen -t ed25519
$ ssh-copy-id anna@192.0.2.10
$ cat >> ~/.ssh/config << 'EOF'
Host testsrv
    HostName 192.0.2.10
    User anna
    IdentityFile ~/.ssh/id_ed25519
EOF
$ ssh testsrv hostname
$ scp bericht.pdf testsrv:/tmp/
```

Übung 18 – Backup [F]

Sichern Sie `~/uebung` täglich um 02:00 Uhr per `rsync` mit Hardlink-Snapshots nach `~/backup` (Hinweis: Skript aus Kapitel 53, Eintrag per `crontab -e`).

```
$ crontab -e
0 2 * * * /home/anna/bin/backup-uebung.sh >> /home/anna/backup.log 2>&1
```

Übung 19 – Absicherung [X]

Sichern Sie einen SSH-Server ab: kein Root-Login, nur Schlüsselanmeldung, nur der Benutzer `anna`, Firewall nur mit Port 22, Schutz vor Brute Force.

```
# cat > /etc/ssh/sshd_config.d/90-hardening.conf << 'EOF'
PermitRootLogin no
PasswordAuthentication no
AllowUsers anna
MaxAuthTries 3
EOF
# sshd -t && systemctl reload ssh
# ufw default deny incoming && ufw allow 22/tcp && ufw enable
# apt install fail2ban && systemctl enable --now fail2ban
$ ss -tlpn                                          # nur Port 22 sichtbar?
$ ssh -o PubkeyAuthentication=no anna@localhost    # muss abgelehnt werden
```

Übung 20 – Fehlersuche [X]

Ein Webserver (`nginx`) startet nicht. Wie gehen Sie vor?

1. `systemctl status nginx` – Zustand und letzte Zeilen.
2. `journalctl -u nginx -b --no-pager | tail -30` – genaue Fehlermeldung.
3. `nginx -t` – Syntax der Konfiguration prüfen, Fehlerzeile beheben.
4. `ss -tlpn | grep -E ':80|:443'` – Ist der Port bereits belegt (z. B. durch Apache)?
5. `ls -l` auf Zertifikate und Logverzeichnis; unter RHEL `ausearch -m avc -ts recent` für SELinux.
6. `df -h`, `df -i` – ist die Platte voll?
7. Änderung beheben, `systemctl restart nginx`, mit `curl -I http://localhost` prüfen.

C. Projektaufgaben mit Lösungswegen

Die folgenden Projekte verbinden mehrere Kapitel zu praxisnahen Szenarien. Sie sollten in einer virtuellen Maschine oder einem Testnetz durchgeführt werden. Die Lösungswege zeigen jeweils einen möglichen Ablauf; Befehle sind für Debian/Ubuntu geschrieben und lassen sich für die RHEL-Familie anpassen (`dnf` statt `apt`, `firewalld` statt `ufw`).

Projekt 1: Webserver mit TLS und Firewall [F]

Aufgabe. Richten Sie einen nginx-Webserver ein, der eine einfache Seite über HTTPS ausliefert (selbstsigniertes Zertifikat), HTTP auf HTTPS umleitet und durch eine Firewall geschützt ist, die nur SSH, HTTP und HTTPS zulässt. Der Dienst muss einen Neustart überstehen.
Querverweise: Kapitel 46, 58, 43, 32.

```
$ sudo apt install nginx ufw openssl
$ sudo openssl req -x509 -nodes -newkey rsa:3072 -days 365 \
    -keyout /etc/ssl/private/web.key -out /etc/ssl/certs/web.crt \
    -subj "/CN=web.lab.example"
$ sudo tee /etc/nginx/sites-available/lab >/dev/null <<'EOF'
server {
    listen 80;
    server_name web.lab.example;
    return 301 https://$host$request_uri;
}
server {
    listen 443 ssl;
    server_name web.lab.example;
    ssl_certificate     /etc/ssl/certs/web.crt;
    ssl_certificate_key /etc/ssl/private/web.key;
    root /var/www/lab;
    index index.html;
}
EOF
$ sudo mkdir -p /var/www/lab
$ echo '<h1>Labor laeuft</h1>' | sudo tee /var/www/lab/index.html
$ sudo ln -s /etc/nginx/sites-available/lab /etc/nginx/sites-enabled/lab
$ sudo rm -f /etc/nginx/sites-enabled/default
$ sudo nginx -t && sudo systemctl enable --now nginx && sudo systemctl reload nginx
$ sudo ufw default deny incoming
$ sudo ufw allow OpenSSH
$ sudo ufw allow 80/tcp
$ sudo ufw allow 443/tcp
$ sudo ufw enable
$ curl -k -I http://localhost                    # 301 auf https
$ curl -k https://localhost                      # Seite
$ sudo reboot                                    # danach erneut pruefen
```

Projekt 2: Plattenverwaltung mit LVM und Erweiterung im Betrieb [F]

Aufgabe. Fügen Sie der VM zwei virtuelle Platten (je 2 GiB) hinzu, erstellen Sie eine Volume Group mit einem Logical Volume (1 GiB, XFS), hängen Sie es dauerhaft unter `/daten` ein und vergrößern Sie es anschließend im laufenden Betrieb auf 3 GiB.
Querverweise: Kapitel 26, 27, 28, 29.

```
$ lsblk                                              # neue Platten finden, z. B. /dev/vdb, /dev
    /vdc
$ sudo pvcreate /dev/vdb /dev/vdc
$ sudo vgcreate vgdaten /dev/vdb /dev/vdc
$ sudo lvcreate -L 1G -n lvdaten vgdaten
$ sudo mkfs.xfs /dev/vgdaten/lvdaten
$ sudo mkdir /daten
$ echo "/dev/vgdaten/lvdaten /daten xfs defaults,nofail 0 2" | sudo tee -a /etc/fstab
$ sudo mount -a && df -hT /daten
$ sudo lvextend -L 3G -r /dev/vgdaten/lvdaten     # -r vergroessert das Dateisystem mit
$ df -hT /daten; sudo vgs; sudo lvs
```

Projekt 3: Zentrale Protokollierung und Auswertung [F]

Aufgabe. Konfigurieren Sie einen Rechner als Log-Server (rsyslog, TCP-Port 514), einen zweiten als Client, der alle Meldungen weiterleitet. Werten Sie anschließend fehlgeschlagene SSH-Anmeldungen aus.
Querverweise: Kapitel 37, 42, 9.

```
# --- Log-Server ---
$ sudo tee /etc/rsyslog.d/10-server.conf >/dev/null <<'EOF'
module(load="imtcp")
input(type="imtcp" port="514")
template(name="PerHost" type="string" string="/var/log/remote/%HOSTNAME%/%PROGRAMNAME%.log")
if $fromhost-ip != '127.0.0.1' then ?PerHost
EOF
$ sudo mkdir -p /var/log/remote && sudo systemctl restart rsyslog
$ sudo ufw allow from 192.168.10.0/24 to any port 514 proto tcp

# --- Client ---
$ echo '*.* action(type="omfwd" target="192.168.10.5" port="514" protocol="tcp")' \
    | sudo tee /etc/rsyslog.d/90-forward.conf
$ sudo systemctl restart rsyslog
$ logger -p local0.notice "Testmeldung vom Client"

# --- Auswertung auf dem Server ---
$ sudo grep -h "Failed password" /var/log/remote/*/sshd.log \
    | awk '{for(i=1;i<=NF;i++) if($i=="from") print $(i+1)}' | sort | uniq -c | sort -rn |
   head
```

Projekt 4: Automatisierte Sicherung mit Timer und Wiederherstellungstest [F]

Aufgabe. Sichern Sie `/etc` und `/home` täglich per systemd-Timer mit `rsync` auf einen zweiten Datenträger, mit Rotation (sieben Tage, harte Verknüpfungen). Stellen Sie anschließend eine gelöschte Datei wieder her.
Querverweise: Kapitel 53, 36, 62.

```
$ sudo tee /usr/local/bin/tagessicherung.sh >/dev/null <<'EOF'
#!/usr/bin/env bash
set -euo pipefail
ZIEL=/backup
HEUTE=$(date +%F)
mkdir -p "$ZIEL"
LETZTE=$(ls -1d "$ZIEL"/20* 2>/dev/null | tail -1 || true)
rsync -aH --delete ${LETZTE:+--link-dest="$LETZTE"} /etc /home "$ZIEL/$HEUTE.tmp/"
mv "$ZIEL/$HEUTE.tmp" "$ZIEL/$HEUTE"
ls -1d "$ZIEL"/20* | head -n -7 | xargs -r rm -rf
EOF
$ sudo chmod 750 /usr/local/bin/tagessicherung.sh
# Service und Timer wie im Kapitel "Skriptprojekte" anlegen, dann:
$ sudo systemctl enable --now tagessicherung.timer
$ sudo systemctl start tagessicherung.service
$ ls /backup

# Wiederherstellungstest
$ rm ~/wichtig.txt
$ cp -a /backup/$(ls /backup | tail -1)/home/$USER/wichtig.txt ~/
```

Projekt 5: Container-Anwendung mit Docker Compose [x]

Aufgabe. Betreiben Sie eine Anwendung aus zwei Containern (Webserver und Datenbank) mit Compose. Die Datenbank soll ihre Daten in einem benannten Volume speichern, der Webserver nur Port 8080 nach außen veröffentlichen. Prüfen Sie, dass die Daten einen Neustart der Container überstehen.
Querverweise: Kapitel 33, 34, 51.

```
# compose.yml
services:
  db:
    image: mariadb:11
    environment:
      MARIADB_ROOT_PASSWORD_FILE: /run/secrets/dbpass
      MARIADB_DATABASE: app
    secrets: [dbpass]
    volumes:
      - dbdaten:/var/lib/mysql
    restart: unless-stopped
  web:
    image: nginx:1.27
    ports:
      - "8080:80"
    depends_on: [db]
    restart: unless-stopped
volumes:
  dbdaten:
secrets:
  dbpass:
    file: ./dbpass.txt

$ openssl rand -base64 18 > dbpass.txt && chmod 600 dbpass.txt
$ docker compose up -d
$ docker compose ps
$ docker compose exec db mariadb -uroot -p"$(cat dbpass.txt)" -e 'CREATE TABLE app.t (i INT);
    INSERT INTO app.t VALUES (1);'
$ docker compose down                       # Container weg, Volume bleibt
$ docker compose up -d
$ docker compose exec db mariadb -uroot -p"$(cat dbpass.txt)" -e 'SELECT * FROM app.t;'
$ docker compose down -v                    # auch Volumes loeschen (Aufraeumen)
```

Projekt 6: Konfiguration mit Ansible automatisieren [X]

Aufgabe. Schreiben Sie ein Ansible-Playbook, das auf zwei Zielhosts einen Benutzer, die SSH-Härtung (Root-Login und Passwort-Anmeldung aus), nginx und die Firewall konfiguriert. Das Playbook muss idempotent sein.
Querverweise: Kapitel 56, 42, 43, 46.

```
# inventar.ini
[web]
web1 ansible_host=192.168.10.11
web2 ansible_host=192.168.10.12

# site.yml
- hosts: web
  become: true
  vars:
    admin_user: anna
    admin_key: "ssh-ed25519 AAAA... anna@laptop"
  tasks:
    - name: Admin-Benutzer anlegen
      ansible.builtin.user:
        name: "{{ admin_user }}"
        groups: sudo
        shell: /bin/bash
    - name: SSH-Schluessel hinterlegen
```

```
      ansible.posix.authorized_key:
        user: "{{ admin_user }}"
        key: "{{ admin_key }}"
    - name: SSH haerten
      ansible.builtin.lineinfile:
        path: /etc/ssh/sshd_config
        regexp: "^#?{{ item.key }} "
        line: "{{ item.key }} {{ item.value }}"
        validate: "sshd -t -f %s"
      loop:
        - { key: PermitRootLogin, value: "no" }
        - { key: PasswordAuthentication, value: "no" }
      notify: SSH neu laden
    - name: nginx installieren
      ansible.builtin.apt:
        name: nginx
        state: present
        update_cache: true
    - name: nginx aktivieren
      ansible.builtin.service:
        name: nginx
        state: started
        enabled: true
    - name: Firewall - Ports freigeben
      community.general.ufw:
        rule: allow
        port: "{{ item }}"
        proto: tcp
      loop: ["22", "80", "443"]
    - name: Firewall aktivieren
      community.general.ufw:
        state: enabled
        policy: deny
  handlers:
    - name: SSH neu laden
      ansible.builtin.service:
        name: ssh
        state: reloaded

$ ansible-playbook -i inventar.ini site.yml --check --diff    # Probelauf
$ ansible-playbook -i inventar.ini site.yml
$ ansible-playbook -i inventar.ini site.yml                   # zweiter Lauf: changed=0
```

Projekt 7: Störungssuche im Team (Rollenspiel) [X]

Aufgabe. Eine Person bereitet drei Störungen auf einer VM vor (z. B. fehlerhafte `fstab`-Zeile, gestoppter Dienst mit falscher Konfiguration, Firewall-Regel, die Port 80 blockiert). Eine andere Person findet und behebt sie anhand der Methode aus Kapitel 55 und dokumentiert Symptom, Ursache, Behebung und Vorbeugung.
Auswertung: Wurde das Symptom exakt beschrieben? Wurden Hypothesen bewusst geprüft? Gab es nur eine Änderung zur Zeit? Ist die Ursache dokumentiert? Gibt es eine Maßnahme zur Vermeidung (Überwachung, Test, Prozess)?

D. Lernkontrolle

Die Fragen prüfen das Verständnis der vier Teile. Pro Frage ist genau eine Antwort richtig; die Lösungen stehen am Ende jedes Abschnitts.

Teil A: Grundlagen

1. Welcher Befehl zeigt das aktuelle Arbeitsverzeichnis an?
   (a) `ls` (b) `pwd` (c) `cd` (d) `where`

2. Was bewirkt `ls -la`?
   (a) Nur Verzeichnisse anzeigen (b) Alle Dateien einschließlich versteckter im Langformat anzeigen (c) Dateien löschen (d) Rekursiv suchen

3. Wie leitet man nur die Fehlermeldungen eines Befehls in eine Datei um?
   (a) `befehl > f` (b) `befehl 2> f` (c) `befehl | f` (d) `befehl < f`

4. Welcher Befehl zählt die Zeilen einer Datei?
   (a) `wc -l` (b) `wc -c` (c) `cat -n` (d) `sort -u`

5. Welcher reguläre Ausdruck passt auf eine Zeile, die mit `abc` beginnt?
   (a) `abc$` (b) `^abc` (c) `.abc` (d) `abc*`

6. Was gibt `echo $?` aus?
   (a) Die Prozess-ID (b) Den Exit-Code des letzten Befehls (c) Die Anzahl der Argumente (d) Den Benutzernamen

7. Wie macht man ein Skript ausführbar?
   (a) `chmod +x skript.sh` (b) `mkexec skript.sh` (c) `chown +x skript.sh` (d) `run skript.sh`

8. Was ist der Unterschied zwischen Hardlink und Symlink?
   (a) Kein Unterschied (b) Ein Hardlink ist ein weiterer Name für denselben Inode, ein Symlink verweist auf einen Pfad (c) Ein Symlink funktioniert nur innerhalb eines Dateisystems (d) Ein Hardlink kann auf Verzeichnisse zeigen

9. In welchem Vim-Modus tippt man Text?
   (a) Normalmodus (b) Einfügemodus (c) Kommandomodus (d) Visueller Modus

10. Welcher Befehl erstellt ein komprimiertes Archiv `a.tar.gz`?
    (a) `tar -czf a.tar.gz dir` (b) `tar -xzf a.tar.gz dir` (c) `gzip a.tar.gz` (d) `zip -t dir`

Lösungen: 1 b, 2 b, 3 b, 4 a, 5 b, 6 b, 7 a, 8 b, 9 b, 10 a.

Teil B: Systemadministration

1. Welche Datei enthält die Passwort-Hashes?
   (a) `/etc/passwd` (b) `/etc/shadow` (c) `/etc/group` (d) `/etc/hosts`

2. Was bedeutet die Rechtezeichenfolge `-rwxr-x---`?
   (a) Alle dürfen alles (b) Besitzer rwx, Gruppe r-x, andere nichts (c) Besitzer r-x, Gruppe rwx (d) Verzeichnis mit Sticky Bit

3. Welcher oktale Wert entspricht `rw-r-----`?
   (a) 640 (b) 604 (c) 750 (d) 460

4. Wie beendet man einen Prozess höflich?
   (a) `kill -9 PID` (b) `kill -15 PID` (c) `kill -STOP PID` (d) `rm PID`

5. Welcher Befehl zeigt Blockgeräte und Partitionen an?
   (a) `lsblk` (b) `lsmod` (c) `lspci` (d) `lsof`

6. Wofür steht LVM?
   (a) Linux Virtual Machine (b) Logical Volume Manager (c) Local Version Module (d) Large Volume Mount

7. Welcher Eintrag in `/etc/fstab` verhindert einen Abbruch beim Booten, wenn das Gerät fehlt?
   (a) `noauto` (b) `nofail` (c) `ro` (d) `sync`

8. Mit welchem Befehl aktiviert man einen Dienst beim Systemstart und startet ihn sofort?
   (a) `systemctl enable --now NAME` (b) `systemctl load NAME` (c) `service NAME persist` (d) `systemctl boot NAME`

9. Wie liest man das Journal einer Unit seit dem letzten Start?
   (a) `journalctl -u NAME -b` (b) `cat /var/log/NAME` (c) `dmesg -u NAME` (d) `systemctl log NAME`

10. Was bedeutet der Cron-Ausdruck `30 2 * * 1`?
    (a) Jeden Tag 2:30 Uhr (b) Jeden Montag um 2:30 Uhr (c) Alle 30 Minuten (d) Am 2. des Monats um 1:30 Uhr

Lösungen: 1 b, 2 b, 3 a, 4 b, 5 a, 6 b, 7 b, 8 a, 9 a, 10 b.

Teil C: Netzwerk

1. Wie viele nutzbare Hostadressen hat ein `/24`-Netz?
   (a) 256 (b) 254 (c) 255 (d) 24

2. Welcher Befehl zeigt die Routingtabelle?
   (a) `ip route` (b) `ip link` (c) `ip neigh` (d) `ss -r`

3. Welcher Befehl zeigt lauschende TCP-Ports mit Prozessen?
   (a) `ss -tlnp` (b) `ping -t` (c) `ip addr` (d) `route -p`

4. Welcher DNS-Eintragstyp ordnet einem Namen eine IPv4-Adresse zu?
   (a) MX (b) A (c) PTR (d) CNAME

5. Welche Port-Nummer nutzt SSH standardmäßig?
   (a) 21 (b) 22 (c) 23 (d) 25

6. Was bewirkt `ssh -L 8080:localhost:80 server`?
   (a) Öffnet lokal den Port 8080 und leitet ihn über den Server an dessen Port 80 weiter (b) Öffnet auf dem Server den Port 8080 (c) Kopiert Dateien (d) Startet einen VPN-Tunnel

7. Wofür dient ein Reverse-Proxy?
   (a) Anonymisierung der Clients (b) Entgegennahme von Anfragen für dahinterliegende Server (TLS-Terminierung, Lastverteilung) (c) Ersetzen des DNS (d) Verschlüsseln von Festplatten

8. Welche Firewall-Grundstrategie ist am sichersten?
   (a) Alles erlauben, Unerwünschtes sperren (b) Alles sperren, nur Benötigtes erlauben (c) Nur ausgehenden Verkehr filtern (d) Keine Regeln

9. Welche Aufgabe hat NTP?
   (a) Namensauflösung (b) Zeitsynchronisation (c) Dateifreigabe (d) Mailübertragung

10. Welches Protokoll verteilt IP-Adressen automatisch?
    (a) DNS (b) DHCP (c) ARP (d) ICMP

Lösungen: 1 b, 2 a, 3 a, 4 b, 5 b, 6 a, 7 b, 8 b, 9 b, 10 b.

Teil D: Sicherheit und Betrieb

1. Was bedeutet das Prinzip der geringsten Rechte?
   (a) Jeder erhält Root (b) Konten und Prozesse erhalten nur die Rechte, die sie benötigen (c) Passwörter müssen kurz sein (d) Dienste laufen ohne Netz

2. Warum ist `PermitRootLogin no` in `sshd_config` sinnvoll?
   (a) Spart Speicher (b) Verhindert direkte Root-Anmeldung per SSH und reduziert Angriffsfläche (c) Beschleunigt die Anmeldung (d) Erlaubt Passwörter

3. Was ist bei Backups die 3-2-1-Regel?
   (a) Drei Kopien, zwei Medientypen, eine Kopie extern (b) Drei Tage, zwei Wochen, ein Monat (c) Drei Server, zwei Netze, ein Router (d) Drei Passwörter, zwei Schlüssel, ein Zertifikat

4. Was ist ein Backup, das nie getestet wurde?
   (a) Ein sicheres Backup (b) Ein Backup mit unbekanntem Wert (c) Ein inkrementelles Backup (d) Ein Archiv

5. Welcher Wert aus `free -h` zeigt den für Anwendungen tatsächlich nutzbaren Speicher?
   (a) `free` (b) `available` (c) `buff/cache` (d) `shared`

6. Wofür dient Fencing im Cluster?
   (a) Verschlüsselung (b) Zwangsabschalten eines unklaren Knotens, um Split-Brain zu verhindern (c) Lastverteilung (d) Zeitsynchronisation

7. Was beschreibt Terraform?
   (a) Kernel-Module (b) Infrastruktur deklarativ als Code (c) Paketquellen (d) Firewall-Regeln eines Hosts

8. Wie prüft man das Ablaufdatum eines TLS-Zertifikats eines Servers?
   (a) `openssl s_client -connect host:443 | openssl x509 -noout -dates` (b) `ping host` (c) `ssh -v host` (d) `certbot ping`

9. Was bedeutet Idempotenz bei Ansible?
   (a) Der Befehl läuft nur einmal (b) Mehrfaches Ausführen führt zum selben Endzustand ohne unnötige Änderungen (c) Alle Hosts gleichzeitig (d) Verschlüsselte Übertragung

10. Was ist die erste Frage bei einer Störung?
    (a) Wer ist schuld? (b) Was genau funktioniert nicht, seit wann und was wurde zuletzt geändert? (c) Neustart des Servers (d) Neuinstallation

Lösungen: 1 b, 2 b, 3 a, 4 b, 5 b, 6 b, 7 b, 8 a, 9 b, 10 b.

E. Nachschlagetabellen

Wichtige Verzeichnisse (FHS)

Verzeichnis	Inhalt
`/`	Wurzel des Dateibaums
`/bin`, `/sbin`	Grundlegende Programme (heute meist Verweis auf `/usr/bin`, `/usr/sbin`)
`/boot`	Kernel, Initramfs, Boot-Loader-Dateien
`/dev`	Gerätedateien (`sda`, `null`, `tty`)
`/etc`	Systemweite Konfigurationsdateien
`/home`	Heimatverzeichnisse der Benutzer
`/lib`, `/lib64`	Bibliotheken und Kernelmodule
`/media`, `/mnt`	Einhängepunkte für Wechselmedien und temporäre Mounts
`/opt`	Zusätzliche, eigenständige Softwarepakete
`/proc`	Pseudo-Dateisystem für Prozess- und Kernelinformationen
`/root`	Heimatverzeichnis des Benutzers Root
`/run`	Laufzeitdaten seit dem Systemstart (PID-Dateien, Sockets)
`/srv`	Daten von Diensten (Web, FTP)
`/sys`	Pseudo-Dateisystem für Geräte- und Treiberinformationen
`/tmp`	Temporäre Dateien (oft beim Neustart gelöscht)
`/usr`	Programme, Bibliotheken, Dokumentation (`/usr/bin`, `/usr/lib`, `/usr/share`, `/usr/local`)
`/var`	Veränderliche Daten: Logs (`/var/log`), Spool, Caches, Datenbanken (`/var/lib`)

Wichtige Konfigurationsdateien

Datei	Zweck
`/etc/passwd`, `/etc/shadow`	Benutzerkonten; Passwort-Hashes (nur Root lesbar)
`/etc/group`, `/etc/gshadow`	Gruppen
`/etc/sudoers`, `/etc/sudoers.d/`	sudo-Regeln (mit `visudo` bearbeiten)
`/etc/fstab`	Dateisysteme, die beim Start eingehängt werden
`/etc/crypttab`	Verschlüsselte Datenträger
`/etc/hostname`, `/etc/hosts`	Rechnername; lokale Namensauflösung
`/etc/resolv.conf`	DNS-Server und Suchdomänen
`/etc/nsswitch.conf`	Reihenfolge der Namensquellen (Dateien, DNS, LDAP, …)
`/etc/network/interfaces`, `/etc/netplan/`	Netzwerkkonfiguration (Debian bzw. Ubuntu)
`/etc/NetworkManager/`	Konfiguration des NetworkManagers
`/etc/systemd/system/`	Eigene und angepasste systemd-Units
`/etc/default/grub`	Einstellungen für GRUB
`/etc/ssh/sshd_config`	Konfiguration des SSH-Servers
`~/.ssh/config`	Konfiguration des SSH-Clients
`~/.ssh/authorized_keys`	Erlaubte öffentliche Schlüssel

Datei	Zweck
`/etc/pam.d/`	PAM-Konfiguration
`/etc/security/limits.conf`	Ressourcengrenzen je Benutzer
`/etc/sysctl.conf`, `/etc/sysctl.d/`	Kernelparameter
`/etc/modprobe.d/`	Optionen und Sperren für Kernelmodule
`/etc/crontab`, `/etc/cron.d/`	Zeitgesteuerte Aufgaben
`/etc/rsyslog.conf`, `/etc/logrotate.d/`	Protokollierung und Rotation
`/etc/apt/sources.list`	Paketquellen (Debian)
`/etc/yum.repos.d/`	Paketquellen (RHEL-Familie)
`/etc/profile`, `~/.bashrc`	Shell-Konfiguration
`/etc/exports`	NFS-Freigaben
`/etc/samba/smb.conf`	Samba-Freigaben
`/etc/nginx/`, `/etc/apache2/`	Webserver-Konfiguration

Wichtige Ports

Port	Protokoll	Dienst
20, 21	TCP	FTP (Daten, Steuerung)
22	TCP	SSH, SFTP, SCP
25	TCP	SMTP (Mailübergabe)
53	TCP/UDP	DNS
67, 68	UDP	DHCP (Server, Client)
80	TCP	HTTP
110 / 995	TCP	POP3 / POP3 über TLS
123	UDP	NTP
137–139, 445	TCP/UDP	NetBIOS und SMB (Samba)
143 / 993	TCP	IMAP / IMAP über TLS
161	UDP	SNMP
389 / 636	TCP	LDAP / LDAP über TLS
443	TCP	HTTPS
514	UDP/TCP	Syslog
587	TCP	SMTP-Einlieferung mit Authentifizierung (Submission)
873	TCP	rsync-Daemon
2049	TCP/UDP	NFS
3306	TCP	MariaDB/MySQL
3389	TCP	Remote Desktop (RDP)
5432	TCP	PostgreSQL
6379	TCP	Redis
51820	UDP	WireGuard (Standard)
9090, 9100	TCP	Prometheus, node_exporter

Signale

Nr.	Name	Bedeutung
1	`SIGHUP`	Terminal geschlossen; viele Dienste lesen daraufhin die Konfiguration neu
2	`SIGINT`	Unterbrechung (Strg+C)
3	`SIGQUIT`	Beenden mit Speicherabbild (Strg+\)
9	`SIGKILL`	Sofort beenden; nicht abfangbar
15	`SIGTERM`	Höfliche Beendigung (Standard von `kill`)
18	`SIGCONT`	Angehaltenen Prozess fortsetzen
19	`SIGSTOP`	Prozess anhalten; nicht abfangbar
20	`SIGTSTP`	Anhalten per Tastatur (Strg+Z)
10, 12	`SIGUSR1`, `SIGUSR2`	Benutzerdefiniert (z. B. Logdatei neu öffnen)

Exit-Codes

Code	Bedeutung
0	Erfolg
1	Allgemeiner Fehler
2	Falsche Verwendung (Syntaxfehler, ungültige Option)
126	Befehl gefunden, aber nicht ausführbar
127	Befehl nicht gefunden
128+n	Beendet durch Signal n (z. B. 130 = Strg+C, 137 = `SIGKILL`, 143 = `SIGTERM`)

Tastenkombinationen der Shell (Bash/Readline)

Taste	Wirkung
Tab, Tab Tab	Vervollständigen; Möglichkeiten anzeigen
Strg+A / Strg+E	Zum Zeilenanfang / Zeilenende
Alt+B / Alt+F	Ein Wort zurück / vor
Strg+U / Strg+K	Bis Zeilenanfang / bis Zeilenende löschen
Strg+W	Wort vor dem Cursor löschen
Strg+Y	Zuletzt Gelöschtes einfügen
Strg+R	Rückwärtssuche im Verlauf
Strg+L	Bildschirm löschen
Strg+C / Strg+Z / Strg+D	Abbrechen / anhalten / Eingabe beenden (Abmelden)
`!!`, `!$`	Letzter Befehl; letztes Argument des letzten Befehls
`sudo !!`	Letzten Befehl mit sudo wiederholen

vi/vim: wichtige Befehle

Befehl	Wirkung
`i`, `a`, `o`	Einfügen vor dem Cursor, nach dem Cursor, neue Zeile darunter
`Esc`	Zurück in den Normalmodus
`h j k l`	Links, unten, oben, rechts

Befehl	**Wirkung**
`w`, `b`, `0`, `$`	Wort vor, Wort zurück, Zeilenanfang, Zeilenende
`gg`, `G`, `:42`	Dateianfang, Dateiende, Zeile 42
`x`, `dd`, `dw`	Zeichen, Zeile, Wort löschen
`yy`, `p`	Zeile kopieren, einfügen
`u`, `Strg+R`	Rückgängig, Wiederholen
`/muster`, `n`, `N`	Suchen, nächster, vorheriger Treffer
`:s/alt/neu/g`, `:%s/alt/neu/g`	Ersetzen in der Zeile bzw. in der ganzen Datei
`v`, `V`, `Strg+V`	Visuelle Auswahl: Zeichen, Zeilen, Block
`.`	Letzte Änderung wiederholen
`:w`, `:q`, `:wq`, `:q!`	Speichern, Beenden, Speichern und Beenden, Beenden ohne Speichern

systemd und Journal

Befehl	**Wirkung**
`systemctl status NAME`	Zustand und letzte Meldungen
`systemctl start|stop|restart NAME`	Dienst steuern
`systemctl reload NAME`	Konfiguration neu laden, ohne Neustart
`systemctl enable --now NAME`	Beim Start aktivieren und sofort starten
`systemctl --failed`	Fehlgeschlagene Units
`systemctl list-timers`	Timer anzeigen
`systemctl daemon-reload`	Unit-Dateien neu einlesen
`systemctl edit NAME`	Drop-in zur Anpassung erzeugen
`journalctl -u NAME -b`	Meldungen einer Unit seit dem Start
`journalctl -p err -b`	Fehler seit dem Start
`journalctl -f`	Live mitlesen
`systemd-analyze blame`	Startdauer der Dienste

F. Glossar

Das Glossar erklärt häufig verwendete Begriffe knapp. Verweise auf Kapitel stehen in Klammern.

Alias Kurzname für einen Befehl oder eine Befehlsfolge in der Shell (`alias ll='ls -l'`); Kapitel 11.

Argument Wert, der einem Befehl übergeben wird, z. B. ein Dateiname; Optionen beginnen meist mit `-`.

Bash *Bourne Again Shell*; die verbreitetste Shell unter Linux.

Bind Mount Einhängen eines Verzeichnisses an einer zweiten Stelle des Verzeichnisbaums (`mount --bind`).

Block-Gerät Gerät mit blockweisem, wahlfreiem Zugriff, z. B. Festplatte (`/dev/sda`).

Boot-Loader Programm, das den Kernel lädt und startet; meist GRUB; Kapitel 54.

Bridge Software-Switch, der Netzwerkschnittstellen auf Schicht 2 verbindet; wichtig für Virtualisierung und Container.

Btrfs Dateisystem mit Snapshots, Prüfsummen und Subvolumes; Kapitel 30.

cgroup *Control Group*; Kernelfunktion zur Begrenzung und Abrechnung von Ressourcen für Prozessgruppen; Basis von Containern.

chroot Ändert für einen Prozess das Wurzelverzeichnis; wird u. a. für Rettungseinsätze genutzt.

CIDR Schreibweise für Netze mit Präfixlänge, z. B. `192.168.10.0/24`.

Container Isolierte Prozessumgebung mit eigenem Dateisystem, Netz und Prozessbaum auf einem gemeinsamen Kernel; Kapitel 33.

cron Dienst für zeitgesteuerte Ausführung von Befehlen; Kapitel 36.

Daemon Dienst, der im Hintergrund läuft (z. B. `sshd`).

Device Mapper Kernel-Framework für virtuelle Blockgeräte (LVM, RAID, Verschlüsselung).

DHCP Protokoll zur automatischen Vergabe von IP-Adressen; Kapitel 45.

Distribution Zusammenstellung aus Kernel, Software, Paketverwaltung und Werkzeugen (Debian, Ubuntu, Fedora, Rocky …).

DNS Namensdienst, der Rechnernamen in IP-Adressen auflöst; Kapitel 44.

Environment Umgebungsvariablen eines Prozesses, z. B. `PATH`, `HOME`.

Exit-Code Rückgabewert eines Programms; 0 bedeutet Erfolg, alles andere Fehler (`$?`).

ext4 Standard-Dateisystem vieler Distributionen mit Journal; Kapitel 29.

Fencing Zwangsweises Abschalten eines Clusterknotens, um Datenbeschädigung zu verhindern (STONITH); Kapitel 60.

File Descriptor Nummer, mit der ein Prozess eine geöffnete Datei oder Verbindung anspricht (0 stdin, 1 stdout, 2 stderr).

Filter Programm, das von der Standardeingabe liest und auf die Standardausgabe schreibt (`sort`, `grep`, `awk`).

Firewall Filter für Netzwerkpakete (nftables, firewalld, ufw); Kapitel 43.

GID Gruppen-ID; numerische Kennung einer Gruppe.

Git Verteiltes Versionsverwaltungssystem; Kapitel 16.

Globbing Mustererweiterung der Shell für Dateinamen (`*`, `?`, `[a-z]`).

GPT *GUID Partition Table*; modernes Partitionsschema für große Datenträger und UEFI.

GRUB Verbreiteter Boot-Loader unter Linux.

Hardlink Zusätzlicher Verzeichniseintrag für dieselbe Datei (gleicher Inode); gilt nur innerhalb eines Dateisystems.

Heredoc Mehrzeilige Eingabe im Skript (`<<EOF`).

Home-Verzeichnis Persönliches Verzeichnis eines Benutzers, Kurzschreibweise `~`.

Hostname Name eines Rechners im Netz.

Hypervisor Software, die virtuelle Maschinen ausführt (KVM, Xen, VirtualBox); Kapitel 35.

Idempotenz Eigenschaft einer Operation, mehrfach ausgeführt dasselbe Ergebnis zu liefern; zentral bei Ansible und Terraform.

Image Unveränderliche Vorlage für Container (*Container-Image*) oder virtuelle Maschinen.

Init-System Erster Prozess (PID 1), der das System startet und Dienste verwaltet; heute meist systemd; Kapitel 32.

Inode Datenstruktur im Dateisystem mit Metadaten einer Datei (Besitzer, Rechte, Zeiten, Blockverweise), aber ohne Namen.

Interrupt Signal der Hardware an die CPU, die Arbeit kurz zu unterbrechen.

IPv4/IPv6 Versionen des Internet-Protokolls; Adressen mit 32 bzw. 128 Bit; Kapitel 38.

Journal Binäre Protokolldatenbank von systemd, abgefragt mit `journalctl`; Kapitel 37.

Kernel Kern des Betriebssystems: Hardwarezugriff, Speicher, Prozesse, Dateisysteme, Netz; Kapitel 25.

Kernelmodul Nachladbarer Teil des Kernels, meist Treiber (`lsmod`, `modprobe`).

Locale Spracheinstellung für Zeichensatz, Datum, Sortierung (`LANG=de_DE.UTF-8`).

Login-Shell Shell, die bei der Anmeldung gestartet wird und Profil-Dateien einliest.

Loopback Virtuelle Netzwerkschnittstelle `lo` (`127.0.0.1`), die nur den eigenen Rechner erreicht.

LVM *Logical Volume Manager*; flexible Datenträgerverwaltung mit PV, VG und LV; Kapitel 28.

MBR Älteres Partitionsschema mit maximal vier primären Partitionen und 2 TiB Grenze.

Mountpoint Verzeichnis, an dem ein Dateisystem in den Baum eingehängt wird.

MTU *Maximum Transmission Unit*; größtes Paket einer Netzwerkverbindung (Ethernet meist 1500 Byte).

Namespace Kernelfunktion, die Ressourcen (Prozesse, Netz, Mounts, Benutzer) voneinander abschirmt; Grundlage von Containern.

NAT Adressumsetzung, z. B. privater Adressen auf eine öffentliche (`masquerade`).

NFS Netzwerkdateisystem für Unix-Systeme; Kapitel 49.

Nice-Wert Prioritätsangabe für Prozesse von -20 (hoch) bis $+19$ (niedrig).

OOM-Killer Kernelfunktion, die bei Speichermangel Prozesse beendet; Kapitel 55.

Orchestrierung Automatisierte Verteilung und Verwaltung vieler Container (Kubernetes); Kapitel 61.

Paket Archiv mit Software, Metadaten und Abhängigkeiten (`.deb`, `.rpm`); Kapitel 19.

PAM *Pluggable Authentication Modules*; modulare Authentifizierung; Kapitel 21.

Partition Abschnitt eines Datenträgers.

PATH Umgebungsvariable mit den Verzeichnissen, in denen die Shell Programme sucht.

PID Prozess-ID; eindeutige Nummer eines laufenden Prozesses.

Pipe Verbindet die Ausgabe eines Befehls mit der Eingabe des nächsten (`|`); Kapitel 8.

Pseudo-Dateisystem Dateisystem ohne Datenträger, das Kernelinformationen anzeigt (`/proc`, `/sys`).

Quorum Mindestanzahl von Stimmen, die ein Cluster für Entscheidungen braucht; Kapitel 60.

RAID Zusammenfassung mehrerer Platten für Redundanz oder Geschwindigkeit; Kapitel 27 und 28.

Regulärer Ausdruck Muster zur Beschreibung von Zeichenketten; Kapitel 9.

Repository Quelle für Pakete (Paketverwaltung) oder Datenbestand der Versionsverwaltung (Git).

Reverse-Proxy Server, der Anfragen im Namen anderer Server entgegennimmt und weiterleitet; Kapitel 47.

Root Administratorkonto (UID 0) mit allen Rechten; auch Wurzel des Dateibaums (`/`).

Router Gerät oder Rechner, der Pakete zwischen Netzen weiterleitet.

rsync Programm zum effizienten Abgleichen von Verzeichnissen, lokal und über Netz; Kapitel 53.

Samba Implementierung des SMB-Protokolls für Datei- und Druckerfreigaben mit Windows; Kapitel 49.

SELinux Zugriffskontrolle auf Basis von Sicherheitsrichtlinien (Mandatory Access Control); Kapitel 23.

Shebang Erste Zeile eines Skripts (`#!/bin/bash`), die den Interpreter nennt.

Shell Kommandozeileninterpreter; Schnittstelle zwischen Benutzer und System; Kapitel 11.

Signal Nachricht an einen Prozess (`SIGTERM`, `SIGKILL`, `SIGHUP`).

Snapshot Momentaufnahme eines Dateisystems, Volumes oder einer virtuellen Maschine.

Socket Kommunikationsendpunkt für Netzwerk- oder lokale Verbindungen (IP-Adresse und Port).

Split-Brain Zustand eines Clusters, in dem mehrere Knoten gleichzeitig aktiv sind; Kapitel 60.

SSH Verschlüsselter Fernzugriff und Dateitransfer; Kapitel 42.

Standardausgabe/-eingabe/-fehler Datenströme `stdout`, `stdin`, `stderr`; Kapitel 8.

Sticky Bit Besonderes Rechtebit für Verzeichnisse: Dateien dürfen nur vom Besitzer gelöscht werden (`/tmp`); Kapitel 22.

sudo Ausführung einzelner Befehle mit Rechten eines anderen Benutzers, meist Root; Kapitel 18.

Swap Auslagerungsbereich auf Platte für wenig genutzte Speicherseiten.

Symlink Symbolische Verknüpfung; eine Datei, die auf einen Pfad verweist (`ln -s`).

systemd Init-System und Dienstverwaltung mit Units, Targets, Timern und Journal; Kapitel 32.

Target systemd-Zielzustand, gruppiert Units (`multi-user.target`).

TCP/UDP Transportprotokolle: zuverlässig und verbindungsorientiert bzw. verbindungslos; Kapitel 38.

TLS Verschlüsselung von Verbindungen (Nachfolger von SSL); Kapitel 58.

UID Benutzer-ID; numerische Kennung eines Benutzers.

Umask Maske, die festlegt, welche Rechte neu angelegte Dateien *nicht* erhalten.

Unit Von systemd verwaltete Einheit: Dienst, Mount, Socket, Timer …

UTC Koordinierte Weltzeit; Server laufen oft in UTC.

UUID Weltweit eindeutige Kennung, z. B. für Dateisysteme in `/etc/fstab`.

VLAN Virtuelles lokales Netz; trennt Netze logisch auf gemeinsamer Hardware.

VPN Verschlüsselter Tunnel zwischen Netzen oder Rechnern; Kapitel 48.

Wildcard Platzhalterzeichen in Dateimustern; siehe Globbing.

XFS Leistungsfähiges Dateisystem für große Dateien; Standard bei RHEL.

Zombie-Prozess Beendeter Prozess, dessen Rückgabewert der Elternprozess noch nicht abgeholt hat.

ZFS Dateisystem und Volume-Manager mit Prüfsummen, Snapshots und Pools; Kapitel 30.

G. Quellen und Lizenz

Herkunft der Inhalte

Dieses Dokument ist ein kompakter Workshop und baut auf den freien Schulungsunterlagen des Projekts **tuxcademy** auf. Ausgewertet wurden:

- *Linux-Grundlagen für Anwender und Administratoren*
- *Linux für Fortgeschrittene*
- *Linux Essentials*
- *Linux kompakt*

Autoren der Originalunterlagen sind **Tobias Elsner** und **Anselm Lingnau**; © Linup Front GmbH / tuxcademy. Weitere Informationen und die Originale: `https://www.tuxcademy.org`.

Bearbeitungshinweis

Die Inhalte wurden ausgewählt, gekürzt, neu strukturiert, mit eigenen Beispielen ergänzt und um Niveaustufen (Einsteiger, Fortgeschritten, Experte) erweitert. Kapitel zu Netzwerk, systemd und Protokollierung enthalten zusätzliche, nicht den Vorlagen entnommene Inhalte. Die Befehlsausgaben und Beispiele sind exemplarisch und wurden nicht auf jeder Distribution geprüft; Abweichungen je nach Version und Distribution sind möglich.

Lizenz

Die Originalunterlagen stehen unter der Lizenz **Creative Commons Namensnennung – Weitergabe unter gleichen Bedingungen 4.0 International** (CC BY-SA 4.0). Dieses Dokument ist eine Bearbeitung im Sinne der Lizenz und wird daher unter denselben Bedingungen weitergegeben:

`https://creativecommons.org/licenses/by-sa/4.0/deed.de`

Sie dürfen das Material teilen und bearbeiten, auch kommerziell, sofern Sie

- **angemessene Urheber- und Rechteangaben** machen (siehe oben), einen Link zur Lizenz angeben und kenntlich machen, ob Änderungen vorgenommen wurden,
- Ihre Beiträge **unter derselben Lizenz** (CC BY-SA 4.0) verbreiten und
- keine zusätzlichen Einschränkungen auferlegen, die die Nutzung durch andere beschränken.

Grafiken

Das Pinguin-Maskottchen *Tux* wurde von Larry Ewing mit GIMP erstellt (frei verwendbar bei Namensnennung). Weitere Grafiken (Diagramme) sind im Dokument mit TikZ selbst erstellt.

Haftungsausschluss

Die Inhalte werden ohne Gewähr bereitgestellt. Befehle mit Systemeingriff (Partitionieren, Formatieren, `dd`, Rechte- und Firewalländerungen) können zu Datenverlust oder Aussperrung führen. Vor dem Einsatz in produktiven Umgebungen sind sie in einer Testumgebung zu prüfen.

H. Abkürzungen

ACL Access Control Lists
BIOS Basic Input/Output System
CIDR Classless Inter-Domain Routing
CPU Central Processing Unit
DHCP Dynamic Host Configuration Protocol
DNS Domain Name System
DVD Digital Versatile Disc
EFI Extensible Firmware Interface
ELK Elasticsearch, Logstash, Kibana
EPEL Extra Packages for Enterprise Linux
ESP EFI System Partition
FHS Filesystem Hierarchy Standard
FTP File Transfer Protocol
GID Group Identifier
GNU GNU's Not Unix
GPL GNU General Public License
GRUB GRand Unified Bootloader
IP Internet Protocol
IPv6 Internet Protocol Version 6
LDAP Lightweight Directory Access Protocol
LGPL GNU Lesser General Public License
LSB Linux Standard Base
LTS Long Term Support
MAC Media Access Control
MBR Master Boot Record
NAT Network Address Translation
NFS Network File System
NTP Network Time Protocol
OSI Open Systems Interconnection
PID Process Identifier
POST Power-On Self-Test
PPID Parent Process Identifier
RAID Redundant Array of Independent Disks
RAM Random Access Memory
RFC Request for Comments
SLAAC Stateless Address Autoconfiguration
SSH Secure Shell
TCP Transmission Control Protocol
TTL Time to Live
UEFI Unified Extensible Firmware Interface
UID User Identifier
USB Universal Serial Bus
UTC Coordinated Universal Time