

corewire

Linux

File System

# Folien-Hinweis

- Space, Page down: Nächste Folie
- Page up: Vorherige Folie
- ESC, o: Übersicht

[Zur Kapitelübersicht](#)

# Filesystem Hierarchy Standard (FHS)

- **Standardisierte Verzeichnisstruktur** für Unix-ähnliche Betriebssysteme
- **Einheitliche Organisation** der Dateien und Verzeichnisse
- **Vorhersagbare Struktur** für Benutzer und Software
- **Kompatibilität** zwischen verschiedenen Linux-Distributionen

# Wichtige Verzeichnisse im Root-Filesystem

```
/
├─ bin/      # Grundlegende Befehle für alle Benutzer
├─ sbin/     # Systembefehle für Administratoren
├─ usr/      # Benutzeranwendungen und -daten
├─ etc/      # Systemkonfigurationsdateien
├─ var/      # Variable Daten (Logs, Caches)
├─ home/     # Benutzerverzeichnisse
├─ opt/      # Zusätzliche Software
├─ tmp/      # Temporäre Dateien
└─ ...
```

# /bin - Binary Executables

- **Grundlegende Befehle** für alle Benutzer
- Verfügbar auch im Single-User-Modus
- Beispiele: `ls`, `cat`, `cp`, `mv`, `bash`

# /sbin - System Binary Executables

- **Systembefehle** primär für Administratoren
- Beispiele: `mount`, `ifconfig`, `iptables`, `fsck`

# /usr - User System Resources

## Hierarchische Struktur von `/usr`

```
/usr/  
├── bin/      # Benutzeranwendungen  
├── sbin/     # Nicht-kritische Systembefehle  
├── lib/      # Bibliotheken für /usr/bin und /usr/sbin  
├── local/    # Lokal installierte Software  
├── share/    # Architektur-unabhängige Daten  
└── include/  # Header-Dateien für C/C++
```

### `/usr/local/bin` - Lokal installierte Software

- Software die **nicht** über den Paketmanager installiert wurde
- Getrennt von System-Software

# /usr/bin und /usr/sbin heute

- `/bin` und `/sbin` für kritische Befehle
- `/usr/bin` und `/usr/sbin` für unkritische Befehle
- Moderne Systeme binden häufig:
  - `/bin` als Symlink auf `/usr/bin`
  - `/sbin` als Symlink auf `/usr/sbin`



# /etc - Konfigurationsdateien

- Textbasierte Konfigurationsdateien
- Systemweite Einstellungen
- Keine Binärdateien

```
# Wichtige Konfigurationsdateien
/etc/passwd      # Benutzerinformationen
/etc/shadow      # Passwort-Hashes
/etc/fstab       # Dateisystem-Mount-Tabelle
/etc/hosts       # Hostname-zu-IP Zuordnungen
/etc/ssh/sshd_config # SSH-Daemon Konfiguration
```

# /var - Variable Daten

## Sich verändernde Systemdaten

```
/var/  
├─ log/      # Systemlogs  
├─ cache/    # Anwendungscaches  
├─ tmp/      # Temporäre Dateien (persistent)  
├─ spool/    # Warteschlangen (Mail, Druckaufträge)  
└─ lib/      # Zustandsinformationen von Programmen
```

## Beispiele:

```
# System-Logs anzeigen  
sudo ls -la /var/log/  
tail /var/log/syslog  
  
# Cache-Verzeichnisse  
ls /var/cache/
```

# /home - Benutzerverzeichnisse

## Persönliche Daten der Benutzer

- Jeder Benutzer hat ein **eigenes Unterverzeichnis**
- **Persönliche Konfigurationen (dotfiles)**
- **Benutzerdaten und Dokumente**

```
# Struktur eines Benutzerverzeichnisses
/home/username/
├── .bashrc          # Bash-Konfiguration
├── .ssh/            # SSH-Schlüssel und Konfiguration
├── Documents/       # Dokumente
├── Downloads/       # Downloads
└── Desktop/         # Desktop-Dateien
```

# PATH-Variable verstehen

# Was ist die PATH-Variable?

## Umgebungsvariable für ausführbare Dateien

- **Liste von Verzeichnissen** getrennt durch Doppelpunkt **:**
- Shell sucht in **dieser Reihenfolge** nach Befehlen
- Ermöglicht Ausführung von Befehlen **ohne vollständigen Pfad**

```
# PATH-Variable anzeigen
echo $PATH

# Typische Ausgabe:
/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin
```

# Wie funktioniert die PATH-Suche?

## Suchreihenfolge bei Befehlsausführung:

1. Shell prüft ob es ein **Built-in-Command** ist
2. Durchsucht **jeden Ordner in PATH** von links nach rechts
3. Führt die **erste gefundene** ausführbare Datei aus
4. **Fehler** wenn Befehl nicht gefunden wird

```
# Beispiel: 'python' wird gesucht
# 1. /usr/local/bin/python  <- wenn gefunden, wird ausgeführt
# 2. /usr/bin/python        <- wird nur geprüft wenn oben nicht gefunden
# 3. /bin/python            <- wird nur geprüft wenn oben nicht gefunden
```

# PATH-Variable anpassen

## Temporär für aktuelle Session:

```
# Verzeichnis zur PATH-Variable hinzufügen
export PATH="/opt/my-app/bin:$PATH"

# PATH für aktuelle Session anzeigen
echo $PATH
```

## Permanent in der Shell-Konfiguration:

```
# Für Bash: ~/.bashrc oder ~/.bash_profile
echo 'export PATH="/opt/my-app/bin:$PATH"' >> ~/.bashrc

# Konfiguration neu laden
source ~/.bashrc
```

# PATH-Best Practices

## Reihenfolge beachten:

- **Sicherheitskritisch:** `/usr/local/bin` vor `/usr/bin`
- Lokale Installationen haben **Vorrang** vor System-Paketen
- Benutzerdefinierte Tools vor System-Tools

## Sicherheitsaspekte:

```
# NIEMALS aktuelle Verzeichnis in PATH!  
export PATH=".: $PATH" # ❌ GEFÄHRLICH!  
  
# Immer absolute Pfade verwenden  
export PATH="/home/user/bin: $PATH" # ✅ SICHER
```



# which-Command

## Lokalisiert ausführbare Dateien in PATH

- Zeigt den **vollständigen Pfad** einer ausführbaren Datei
- Folgt der **gleichen Suchreihenfolge** wie die Shell
- Hilfreich für **Troubleshooting** und **Debugging**

```
# Einfache Verwendung
which python3
# Ausgabe: /usr/bin/python3

which ls
# Ausgabe: /bin/ls
```

# which vs whereis vs type

## Verschiedene Tools für verschiedene Zwecke:

```
# which - nur PATH-Suche
which python3

# whereis - sucht Binärdateien, Quellen und Manpages
whereis python3

# type - zeigt wie Shell den Befehl interpretiert
type python3
type cd      # Built-in command
type ls      # aliased oder binary
```

# Linux File Typen

# Übersicht der Dateitypen in Linux

Linux kennt verschiedene Dateitypen:

- **Regular Files** (normale Dateien)
- **Directories** (Verzeichnisse)
- **Symbolic Links** (symbolische Links)
- **Sockets** (IPC-Kommunikation)
- **Named Pipes** (FIFOs)
- **Block Devices** (Festplatten)
- **Character Devices** (Terminals, serielle Schnittstellen)

```
# Dateitypen mit ls -l anzeigen
ls -l /dev/ | head -5
# Erstes Zeichen zeigt den Dateityp an
```

# Verzeichnisse (Directories)

## Spezielle Dateien für Ordnerstruktur

- Enthalten **Referenzen** auf andere Dateien und Verzeichnisse
- **Hierarchische Organisation** des Dateisystems
- Kennzeichen: `d` am Anfang der `ls -l` Ausgabe

```
# Verzeichnis erstellen
mkdir mein-verzeichnis

# Verzeichnisinhalt anzeigen
ls -la mein-verzeichnis/
```

# Sockets - Inter-Process Communication

- **Lokale Kommunikation** zwischen Prozessen
- **Schneller** als Netzwerk-Sockets
- Kennzeichen: `s` am Anfang der `ls -l` Ausgabe
- Häufig in `/run/` oder `/tmp/`

```
# Socket-Dateien finden
find /run -type s 2>/dev/null | head -5

# Docker Socket (häufig verwendetes Beispiel)
ls -la /var/run/docker.sock

# Systemd Sockets
ls -la /run/systemd/
```

# Soft und Hard Links



# Was sind Links?

## Zwei Arten von Links in Linux:

- **Hard Links:** Direkter Verweis auf Inode
- **Soft Links (Symbolic Links):** Verweis auf Dateipfad

## Gemeinsame Eigenschaften:

- Ermöglichen **mehrere Namen** für dieselbe Datei
- **Platzsparend** - keine Duplizierung der Daten
- Verschiedene **Einsatzzwecke** und **Limitierungen**

# Hard Links verstehen

## Eigenschaften von Hard Links:

- **Direkter Verweis** auf die Inode der Datei
- **Gleiche Berechtigung** und **Metadaten** wie Original
- **Link Counter** verwaltet die Anzahl der Verweise
- Datei wird erst gelöscht wenn **alle Hard Links** entfernt wurden

```
# Hard Link erstellen
ln original-file hard-link

# Link-Anzahl anzeigen (zweite Spalte in ls -l)
ls -l original-file hard-link

# Inode-Nummer anzeigen
ls -i original-file hard-link # Gleiche Inode-Nummer
```

# Hard Link Limitierungen

## Was Hard Links NICHT können:

- **Keine Verzeichnisse verlinken** (mit Ausnahmen)
- **Keine Filesystembegrenzung überschreiten**
- **Keine unterschiedlichen Partitionen verknüpfen**
- **Nur auf lokalen Dateisystemen** (nicht über NFS)

```
# Diese Befehle schlagen fehl:
ln /tmp/file /home/user/link          # Verschiedene Partitionen
ln my-directory hard-dir-link         # Verzeichnis-Link nicht erlaubt

# Hard Links finden
find . -samefile original-file        # Alle Hard Links zu einer Datei
```

# Soft Links (Symbolic Links)

## Eigenschaften von Soft Links:

- **Verweist auf Dateipfad** (nicht auf Inode)
- **Eigene Inode** und Metadaten
- **Flexibler** als Hard Links
- **Kennzeichen:** `1` am Anfang der `ls -l` Ausgabe

```
# Soft Link erstellen
ln -s /path/to/original-file soft-link

# Soft Link zu Verzeichnis
ln -s /path/to/directory dir-link

# Link-Ziel anzeigen
ls -l soft-link
readlink soft-link
```

# Soft Link - Vorteile

- **Verzeichnisse** können verlinkt werden
- **Filesystemgrenzen** überschreitbar
- **Relative** und **absolute** Pfade möglich
- **Remote Filesysteme** unterstützt

# Soft Link - Nachteile

- **Broken Links** wenn Ziel gelöscht wird
- **Langsamerer Zugriff** (doppelte Auflösung)
- **Separate Berechtigungen** für Link und Ziel

## Zur Kapitelübersicht

- Vorheriges Kapitel: [Paketmanager](#)
- Nächstes Kapitel: [Benutzer, Gruppen und Rechte](#)