Informatik 7bi Schuljahr 2018/2019 als PDF exportieren

Informatik 7. Klasse - Schuljahr 2018/19

Lehrinhalte

- Lehrplaninhalte

Remote-Zugriff auf Schulserver

Kapitel

- 1) Virtualisierung
- 2) CMS-Systeme
- 3) Netzwerke
- 4) Betriebssysteme
- 5) Datenbanken
- 6) PHP & MySQL
- 7) C++
- 8) Visual C++

Leistungsbeurteilung

- **Schularbeiten (SA)**
  - 2x SA (2h) pro Semester
- **Mitarbeit (MA)**
  - Aktive Mitarbeit im Unterricht (aMA)
  - Mündliche Stundenwiederholungen (mMA)
  - Schriftliche Stundenwiederholungen (sMA)
- **Praktische Arbeiten (PA)**
  - 1x praktischer Arbeitsauftrag pro Woche

- Aktueller Leistungsstand

Stoff für die 2. Schularbeit in Informatik – 7BI – 26.04.2019

Stoff für die 1. Schularbeit in Informatik – 7BI – 09.11.2018

- Tag der offenen Tür

Was wird in der 7. Klasse gemacht?

5. Semester

Sicherung der Nachhaltigkeit

- Notwendiges Vorwissens für die Kompetenzbereiche dieses Moduls wiederholen und aktivieren
- Grundlagen für die Kompetenzbereiche dieses Moduls ergänzen und bereitstellen

Gesellschaftliche Aspekte der Informationstechnologie

Verantwortung, Datenschutz und Datensicherheit

- Die Bedeutung von Informatik in der Gesellschaft beschreiben, die Auswirkungen auf die Einzelnen und die Gesellschaft einschätzen und Vor- und Nachteile an konkreten Beispielen abwägen können.
- Maßnahmen und rechtliche Grundlagen im Zusammenhang mit Datensicherheit, Datenschutz und Urheberrecht kennen und anwenden können.

Informatiksysteme – Hardware, Betriebssysteme und Vernetzung

Technische Grundlagen und Funktionsweisen (Hardware)

- Aktuelle Entwicklungen auf dem Hardwaresektor kennen
- Mit Hardware-Komponenten praktisch umgehen können

Virtualisierung

- Virtualisierungs-Software für Betriebssysteme und Server einsetzen können
- Betriebssysteme und Software virtualisieren können

Betriebssysteme (Linux)

- Die Arbeitsumgebung (Oberflächen, Tools, Script-Programmierung) von Linux einrichten können
- Anwenderprogramme unter Linux installieren und verwenden können
- Die Bedeutung von Linux als Betriebssystem für Internetdienst kennen
- Im LAN und im Internet mit Linux arbeiten können

Netzwerke

- Grundlagen der Vernetzung von Computern beschreiben und lokale und globale Computernetzwerke nutzen können.
- Netz-Topologien kennen
- Den Datenverkehr in Netzen simulieren und analysieren können
- Netzhardware kennen und praktisch damit umgehen können
- Protokolle und Adressierung kennen und anwenden können
- Sicherheitskonzepte für Netzwerke kennen
- Netztypen (Peer-to-Peer, Server-Client) kennen
- Peer-to-Peer und Server-Client-Netz installieren können
- Einfache Netzadministration durchführen können
- Programme für die Verwendung im Netz installieren können

Angewandte Informatik, Datenbanksysteme und Internet

Suche, Auswahl und Organisation von Information

- Informationsquellen erschließen, Inhalte systematisieren, strukturieren, bewerten, verarbeiten und unterschiedliche Informationsdarstellungen verwenden können.

Datenmodelle und Datenbanksysteme

- Datenbanken durch ein Programm ansteuern können
- Datenbanken benutzen und einfache Datenmodelle entwerfen können.
- Die Charakteristika von SQL kennen
- PHP-Scripts zur Anbindung einer Webseite an eine MySQL-Datenbank erstellen können

6. Semester

Sicherung der Nachhaltigkeit

- Notwendiges Vorwissens für die Kompetenzbereiche dieses Moduls wiederholen und aktivieren
- Grundlagen für die Kompetenzbereiche dieses Moduls ergänzen und bereitstellen

Angewandte Informatik, Datenbanksysteme und Internet

Web-Techniken

- PHP-Scripts zur Anbindung einer Webseite an eine MySQL-Datenbank erstellen können
- MySQL-Datenbank mittels PHP verwalten können

Gesellschaftliche Aspekte der Informationstechnologie

Verantwortung, Datenschutz und Datensicherheit

- Kommunikation mit einem externen Server herstellen können
- Die Bedeutung der Absicherung extern verwalteter Daten einschätzen können

Algorithmik und Programmierung

Algorithmen und Datenstrukturen

- Algorithmen erklären, entwerfen, darstellen können.
- Klassen und Objekte kennen und verwenden können
- Files kennen und einsetzen können
- Datenstrukturen Zeiger und Listen kennen und visualisieren können
- Algorithmen mit Zeigern und Listen erstellen können

Programmierung (Objektorientierte visuelle Programmiersprache)

- Weitere Komponenten der visuellen Programmierung kennen und anwenden können
- Mit einer visuellen Programmiersprache auf fortgeschrittenem Niveau arbeiten können
- Mit Files arbeiten können
- Graphikelemente einsetzen können
- Algorithmen in der visuellen Programmiersprache implementieren können
- Algorithmen mit Zeigern und Listen programmieren können

```
7. Klasse (3 Stunden,  je eine 2h Schularbeit pro Semester)
```

Virtualisierung

Definition

Bei der Virtualisierung handelt es sich um den Prozess der Erstellung einer **softwarebasierten (virtuellen) Komponente anstatt einer hardwarebasierten (physischen)**.

Virtualisieren lassen sich sowohl Anwendungen als auch Server, Storage und Netzwerke. Die Virtualisierung ist unabhängig von der Unternehmensgröße bei Weitem der effektivste Weg zur Reduzierung der IT-Ausgaben bei gleichzeitiger Steigerung der Effizienz und Agilität.

Gründe für Virtualisierung

Durch Virtualisierung kann die Agilität, Flexibilität und Skalierbarkeit der IT erhöht werden. Gleichzeitig werden deutliche Kosteneinsparungen ermöglicht. IT-Komponenten lassen sich einfacher verwalten und kostengünstiger betreiben, da Workloads schneller bereitgestellt, Performance und Verfügbarkeit optimiert sowie Betriebsabläufe automatisiert werden. Weitere Vorteile:

- Senkung der Investitions- und Betriebskosten
- Minimierung oder Vermeidung von Ausfallzeit
- Erhöhung der Produktivität, Effizienz, Agilität und Reaktionsfähigkeit der IT
- Beschleunigung und Vereinfachung des Anwendungs-Provisioning (Bereitstellung von Ressourcen)
- Nachhaltige Planung und einfache Wiederherstellung
- Vereinfachung des Managements von Rechenzentren
- Einfacher Austausch/Einfache Erweiterung von Software (da plattformunabhängig)

Virtualisierungsarten

Eine Einteilung in Gruppen bzw. Kategorien fällt nicht immer ganz leicht, da manche Virtualisierungstechnologien mehreren zugeordnet werden können. Hier findest du eine mögliche Einteilung:

1) Servervirtualisierung (auch Betriebssystemvirtualisierung genannt)

Die Virtualisierung von Servern bedient sich der Virtualisierung von Computern, die auch Hardware-Virtualisierung oder Betriebssystemvirtualisierung genannt wird. Hierbei wird auf physikalischer Computerhardware dem VM Host ein als sog. Hypervisor dienendes Betriebssystem installiert das vorhandene Hardwareressourcen intelligent auf virtuelle Maschinen verteilt. Dadurch ermöglicht diese Konstellation einem Serverbetriebssystem Teile der Vorhandenen Hardware, also Prozessor, Speicher und Anschlüsse, zu nutzen. Dabei wird einem Betriebssystem ein vorhandener Hardwarecomputer vorgespiegelt. Die gängigsten, kommerziellen Lösungen die von Herstellern verfügbar sind VMWare ESX, Citrix XEN, Microsoft Hyper-V sowie IBM MPAR bei professionellen Mainframe

Großrechnersystemen. Freie kostenlose Virtualisierungslösungen für Serverbetriebssysteme sind ebenfalls in großer Anzahl verfügbar. Die bekanntesten wären VMWare ESXi und VMWare Player, XEN, KVM und Oracle VirtualBox.

Man unterscheidet mehrere Arten der Servervirtualisierung:

1.1) Hardwarevirtualisierung

Die wohl am einfachsten zu verstehende Art ist die reine Hardwarevirtualisierung. Hierbei sind nicht die aktuelle Eigenschaften von Intel oder AMD-CPUs gemeint, sondern sogenannte Hardware-Partitions wie sie z. B. in IBMs System p oder Fujitsus Sparc Enterprise Serie realisiert sind. Vereinfacht ausgedrückt kauft der Kunde einen Schrank voller CPUs, Arbeitsspeicherriegel etc. Im Gegensatz zu normalen Servern muss jedoch bei der Erweiterung des Arbeitsspeichers nicht ein neues RAMModul aus dem Keller geholt und explizit in einen Server eingebaut werden. Vielmehr kann dies bei Hardware-Partitions „per Mausklick“ erfolgen, da die vorhandenen Ressourcen innerhalb des „Schranks“ frei auf die logischen Partitions verteilt werden können. Auf jeder so erstellten Partition kann dann ein entsprechendes Betriebssystem installiert werden. Die Virtualisierung erfolgt hierbei durch die Aufteilung der vorhandenen Hardware in kleinere Rechner (die erwähnten Partitions).

1.2) Vollvirtualisierung

Vollvirtualisierung bezeichnet eine in Software verwirklichte Technologie. Sie kann sowohl als Hypervisor realisiert werden, das heißt die Virtualisierungsschicht läuft direkt auf der Hardware, oder als Hosted-Lösung (siehe unten) In diesen Abschnitt wird erst einmal nur die HypervisorLösung betrachtet. Die Gast-Betriebssysteme müssen bei der Vollvirtualisierung nicht speziell angepasst werden, da diese in der virtuellen Umgebung bekannte Hardware vorfinden. Während das Gastbetriebssystem denkt, auf physikalische Geräte (wie zum Beispiel Netzwerkkarten) zuzugreifen, sind diese jedoch nur virtuell vorhanden. Wenn nun ein Gastbetriebssystem mit einem anderen Rechner kommunizieren will, so sendet es beispielsweise über seine virtuelle Intel EPro1000-Netzwerkkarte Daten ins LAN. Diese Daten müssen nun von der Virtualisierungsschicht über die physikalische Netzwerkkarte weitergeleitet werden („Virtualisierungsoverhead“ für Ein- und Ausgabeoperationen). Reine Rechenoperationen werden direkt auf den physikalischen CPUs ausgeführt. Die Problematik besteht hierbei darin, dass spezielle Befehle unterbunden werden müssen. Als Beispiel sei das Ausschalten einer virtuellen Maschine genannt. Um bei einem einfachen Modell zu bleiben, wird im Folgendem angenommen, dass hierfür das virtualisierte Betriebssystem den CPU-Befehl „Power-Off-Hardware“ ausführt. Würde dieser Befehl jedoch eins-zu-eins vom Hypervisor an den physikalischen Rechner weitergeleitet werden, würde der komplette Server mit allen VMs ausgeschaltet werden – anstatt einer einzelnen VM. Deshalb muss dieser Befehl abgefangen und anders verarbeitet werden – hier wird er durch den Befehl „Power-OffVM“ ersetzt. Dies bedeutet, dass grundsätzlich sämtliche von virtuellen Maschinen ausgeführte Befehle auf solche kritischen Kommandos hin untersucht und bei Bedarf umgeschrieben werden müssen. Dieser Vorgang wirkt sich entsprechend auf die Performance aus („Virtualisierungsoverhead“ für CPU-Befehle).

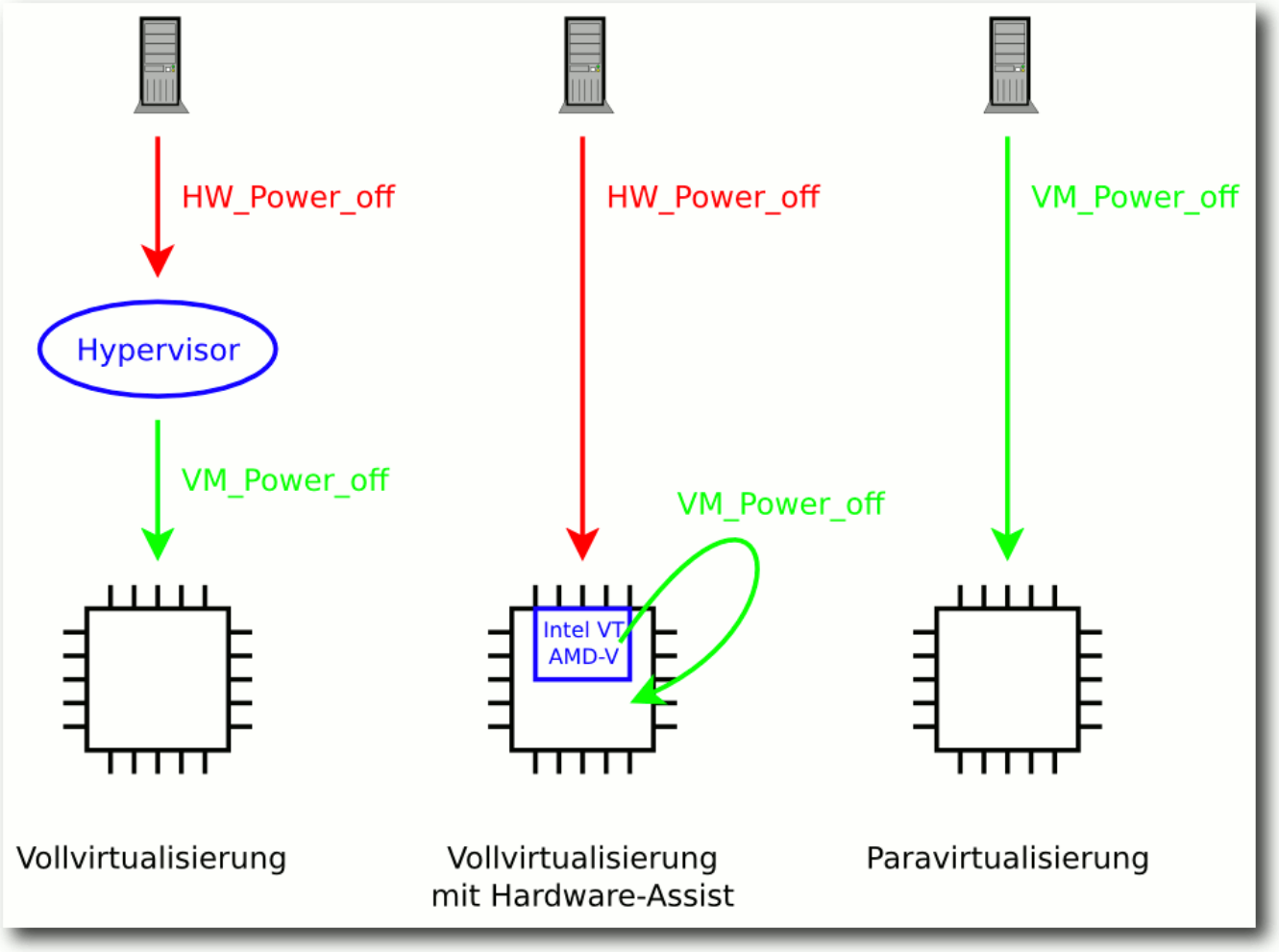
1.3) Hardware-Assisted-Virtualisierungen

Genau hier setzen Hardware-Assisted Virtualisierungen an. Bei diesen Lösungen übernimmt die Hardware das Umsetzen der Befehle von „Power-Off-Hardware“ zu „Power-Off-VM“. Dies hat zur

Folge, dass die Virtualisierungsschicht sich nicht mehr so intensiv um die Befehle der virtuellen Maschinen kümmern muss. Die Technologie wird bei Intel-CPUs als Vanderpool Technology „Intel VT-x“ 4 bzw. bei AMD als AMD-Virtualization „AMD-V“ 5 (früher Codename Pacifica) bezeichnet. In neueren Rechnergenerationen wird dieses Prinzip nicht mehr nur für CPUs sondern auch für Ein- und AusgabeBefehle verwendet (z. B. „Intel VT-d“).

1.4) Paravirtualisierung

Als letzte Technologie in diesem Abschnitt ist die Paravirtualisierung zu nennen. Der Unterschied zu den bisher genannten Techniken ist, dass hierbei das Gastbetriebssystem angepasst werden muss. Die Anpassung sorgt dafür, dass die virtuelle Maschine weiß, dass sie nicht auf physikalischer Hardware läuft und somit direkt den Befehl „Power-Off-VM“ ausführt. Deshalb erzeugt Paravirtualisierung weniger Overhead mit dem Nachteil der nötigen Anpassung des Gastes. Dadurch war diese Technologie zuerst nur Linux-Kernel basierten VMs vorenthalten. Microsoft hat mittlerweile auch für Windows Server 2008 solche Anpassungen entwickelt („Enlightenments“), welche jedoch ausschließlich Vereinfachtes Schema der Betriebssystemvirtualisierung. von Microsoft-Virtualisierungslösungen (sprich von Microsofts Hyper-V) verwendet werden dürfen, wenn nicht explizit ein Vertrag zusätzlich mit Microsoft ausgehandelt wurde. Es ist davon auszugehen, dass Microsoft sich dies entsprechend honorieren lässt. Um dieses Problem etwas zu mildern, wurden spezielle Treiber entwickelt, welche bei einem vollvirtualisierten Gast zumindest Teile paravirtualisiert betreiben können (z. B. der paravirtualisierte SCSI-Treiber derVMware Tools).



2) Desktop Virtualisierung

Die Virtualisierung von Desktop PCs entspricht technisch im weitest gehenden der Virtualisierungstechnik für Server. In der neuesten Generation der Desktop Virtualisierungen kann ein Virtualisierter PC auf beliebige Endgeräte der Anwender wie Laptops, Pads, Thin Clients oder PCs plattformübergreifend gestreamt werden. Für Desktopbetriebssysteme können die professionellen Virtualisierungslösungen für Server ebenfalls verwendet werden. Spezielle Softwarelösungen für Desktop Virtualisierung sind Microsoft Virtual PC; VMWare Workstation und Apple‘s Parallels Workstation. Konkret heißt das ein Windows 10 läuft nicht mehr am Arbeitsplatz-Rechner sondern innerhalb einer virtuellen Server-Infrastruktur (ESX, XenServer, Hyper-V). Die Benutzer verwenden entweder einen Terminal-Services-Client oder direkt einen ThinClient mit Hilfe dessen sie die virtuellen Clients steuern können.

Vorteile sind hierbei, dass sämtliche Server und Client-Betriebssysteme an einem zentralen Ort betrieben werden können (z. B. VMware vSphere-Umgebung im Serveraum). Somit ist eine zentrale Datenhaltung mit verbesserten Backup-, Performance- oder KostenersparnisMöglichkeiten gegeben. Für die Verwaltung der virtuellen Desktop-VMs wurden VMware View und Citrix XenDesktop entwickelt. Einen entscheidenden Nachteil teilen sich jedoch alle Virtualisierungslösungen, die für die (Client-)Benutzer-Interaktion ausgelegt sind: die Grafikkartenperformance. Bei einem physikalischen Client-PC ist die Grafikkarte direkt mit dem Monitor verbunden, während bei einer Client-VM die Daten von der virtuellen Grafikkarte im Serverraum erst über das Netzwerk zur lokalen Grafikkarte und letztlich zum Monitor übertragen werden müssen.

3) Applikations/Anwendungs-Virtualisierung

Die Virtualisierung von Applikationen erlaubt es Softwarepakete in komplexer Konfiguration zeitnah und unabhängig von Benutzerprofilen für Desktop PCs oder Notebooks bereitzustellen. Hierbei wird eine benötigte Anwendung inklusive aller konfigurierten Einstellungen in eine ausführbare .exe Datei umgewandelt. Durch die Bereitstellung der Datei im Netzwerk wird die Installation und Administration auf einzelnen Clients überflüssig was den administrativen Zeitaufwand erheblich verringert. Die entsprechende Software kann für den Außendienst oder das Homeoffice als portable Anwendungsdatei auf Computern gespeichert werden. Verfügbare Lösungen für die Software Virtualisierung sind VMWare ThinApp und Spoon Virtual Application Studio.

Man unterscheidet mehrere Arten der Anwendungsvirtualisierung:

3.1) Gehostete Vollvirtualisierung

Die gehostete Vollvirtualisierung wird in diesem Artikel der Gruppe der Anwendungsvirtualisierungen zugeordnet, während sie ebenfalls der Betriebssystemvirtualisierung zugeordnet werden könnte. Da jedoch die Virtualisierungsschicht nicht direkt auf der Hardware läuft, sondern vielmehr ein Basisbetriebssystem erforderlich ist, innerhalb dessen die Virtualisierung als normale Anwendung läuft, wird sie diesen Abschnitt zugeordnet. Als bekannte Vertreter dieser Gattung wären unter anderem VirtualBox, Microsoft VirtualPC und VMware Server/Workstation zu nennen. Die Vorteile liegen auf der Hand: Es ist nicht mehr die Virtualisierungsschicht für das Ansprechen der physikalischen Hardware verantwortlich, sondern das zu Grunde liegende Betriebssystem (z. B. Linux, Windows, Mac OS X). Somit müssen keine eigenen Treiber entwickelt werden. Beispielsweise unterstützt VMware ESX-Server 4 nur eine begrenzte Anzahl an Netzwerkkarten, während die Lösung

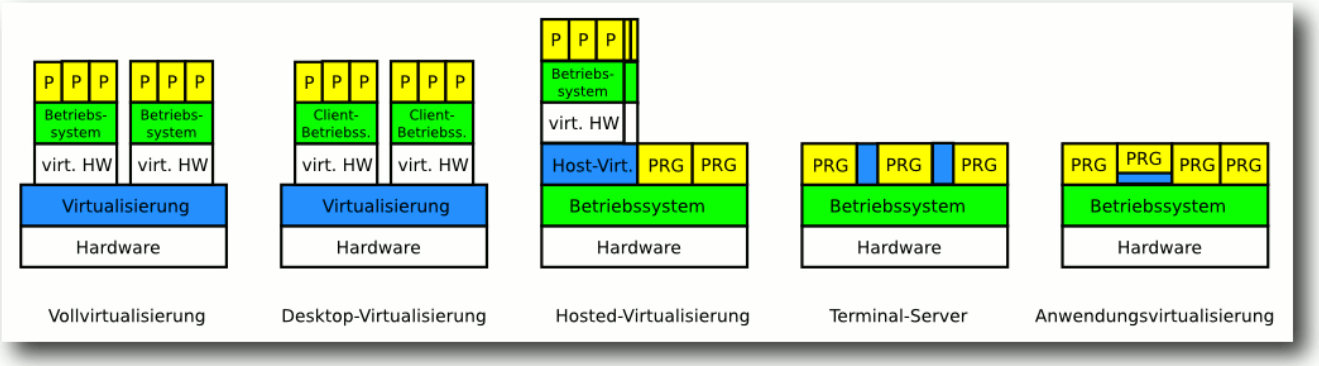
mit VMware Server sozusagen alle von Linux und Windows unterstützten Geräte verwenden kann. Auch Citrix XenServer, welches auf einem Linux-Kernel basiert und somit theoretisch sämtliche Linux-Treiber verwenden könnte, unterstützt nur einen Teil derer. Der Hauptnachteil von Hosted-Lösungen ist jedoch, dass das Virtualisierungsprogramm nur noch eines von vielen im Host-Betriebssystem ist und entsprechend auch als solches behandelt wird. Des Weiteren ist nicht garantiert, dass ein für Windows zertifizierter Treiber auch mit sämtlichen auf Windows ausführbaren Virtualisierungslösungen fehlerfrei funktioniert, während die Hersteller von Hypervisor-Produkten ihre Treiber genau für diesen Anwendungsfall testen.

3.2) Terminal-Server

Die wohl bekannteste Art der Anwendungsvirtualisierungen sind Terminal-Server. Hierbei werden nicht verschiedene Betriebssystem-Instanzen parallel auf einem Rechner ausgeführt, sondern nur parallele Sitzungen mehrere Nutzer innerhalb eines Betriebssystems. Unter Windows wären Microsofts Terminal Services und Citrix XenApp zu nennen, während Linux mit einer Vielzahl von Lösungen aufwarten kann. Dies liegt höchstwahrscheinlich daran, dass Linux schon immer als Mehrbenutzer-System entwickelt wurde. So können bereits die StandardDesktop-Manager (kdm, gdm) über XDMCP Sitzungen für Remote-Benutzer bereitstellen. Um Anwendungen in einer Mehrbenutzer-Umgebung von einander abzuschotten, wird normalerweise auf Home-Verzeichnissen und Variablen gesetzt. Dabei liegen die Programmdaten nur einmal vor (z. B. C:\Program Files\... bei Windows bzw. /usr/bin bei Linux), während für das Speichern der Dokumente eine Variable %HOMEPATH% (z. B. C:\Users\admin\) bzw. $HOME (z. B. /home/admin/) verwendet wird.

3.3) Anwendungsvirtualisierungen

Eine verwandte Art sind die (für viele Anwender missverständlich) direkt als Anwendungsvirtualisierungen betitelten Lösungen. Linux-Benutzer kennen ein ähnliches Verfahren seit Jahren. Gemeint ist eine chroot-Umgebung. Hierbei wird eine Anwendung in einer vom eigentlichen Betriebssystem abgekoppelten Umgebung („Sandbox“) ausgeführt. Die Technologien gehen hierbei heutzutage jedoch weiter und spiegeln eher das Verhalten von UnionFS wieder. Die Produkte der führenden Hersteller nennen sich VMware ThinApp, Citrix XenApp Streaming und Microsoft App-V . Alle beruhen auf dem Prinzip, dass ein Programm denkt, es laufe ungehindert im Betriebssystem, während in Wirklichkeit jedoch sämtliche Lese- und Schreiboperationen abgefangen und bei Bedarf umgebogen werden können. Einem Programm kann somit simuliert werden, dass es z. B. Daten von C:\temp nach D:\temp kopieren kann. In Wirklichkeit existieren jedoch die Ordner nicht einmal im Dateisystem sondern nur innerhalb der virtualisierten Anwendungsumgebung.



4) Storage Virtualisierung

Die Virtualisierung von Speichersystemen stellt die Zusammenfassung vorhandener Speicherressourcen wie Festplatten, NAS und SAN Systeme dar. Hierbei werden die vorhandenen Datenspeicher unterschiedlicher Technologien und Hersteller in einem gemeinsamen großen Speicherpool zusammengefügt. Einzelne, vorhandene Rechnersysteme können benötigten Speicherplatz in flexibler Größe aus dem virtuellen Speichersystem in Anspruch nehmen. Durch Hinzufügen weiterer Speicherhardware kann bei anwachsenden Datenmengen der Speicherpool im laufenden Betrieb beliebig vergrößert werden. Zusätzlich minimiert Storage-Virtualisierung den Aufwand zur Schaffung von Redundanz als Schutzmaßnahme gegen Datenverlust oder Betriebsausfälle. Man unterscheidet zwischen drei Arten der Storage-Virtualisierung. Die Hostbasierte Lösungen wie Datacore SANSymphony, Symantec Storage Foundation for Windows oder HPs LeftHand Virtual SAN laufen auf eigenen Rechnersystemen. Die zweite Möglichkeit findet direkt in Speicherhardware wie SAN Systemen statt bei denen weitere Geräte und Switches als zusätzliche vermittelnde Schicht ins SAN integriert werden. Der Storage Controller erkennt diese als Hostsystem während zugreifende Hosts diese als Storage erkennen. Die Produkte solcher kombinierter Systeme sind u.A. EMC Invista, LSI Logic StoreAge und HPs SVSP. Die dritte Möglichkeit ist Storage Virtualisierung die direkt in einem dedizierten Controller, der Anschlussmöglichkeiten für verschiedene Speichersysteme anbietet, stattfindet. Hierzu zählen IBM Systems SVC (Storage Virtualisierungs Controller) oder VMWare vSphere Storage Appliance.

5) Netzwerk Virtualisierung

Die Virtualisierung von Netzwerken bietet die Möglichkeit ein Firmennetzwerk über bauliche Abgrenzungen hinweg in mehrere Teilnetze aufzuteilen. Bei einer klassischen physikalischen Netzwerkinfrastruktur gilt ein Adressbereich für das ganze Unternehmen. Zugriffsberechtigungen und Netzwerkkontrolle kann bei dieser Konfiguration nur durch Dateiberechtigungen und komplizierte Firewall Lösungen ausgeübt werden. Eine physikalische Trennung einzelner Abteilungen ist bei dieser Konfiguration beinahe unmöglich. Bei einem virtualisierten Netzwerk können einzelne Clients sogar über Stockwerksgrenzen hinweg einer einzelnen Netzwerkgruppe zugeordnet werden. Jedes der so erstellten virtuellen Netzwerke hat einen eigenen Adressbereich obwohl sich diese im gleichen physikalischen Netzwerk befinden. Ressourcen der Entwicklungsabteilung z.B. sind somit vor ungewollten oder unautorisierten Zugriffen durch eine andere Abteilung auf unterster Ebene abgesichert. Mittlerweile implementiert fast jeder Hersteller von Netzwerkhardware die Möglichkeit zur Virtualisierung von Netzwerken in seine Produkte. Voraussetzung sind Layer2 oder Layer3 fähige Netzwerkswitche. Die besten Erfahrungen bei der Netzwerkvirtualisierung hatten wir mit Produkten der Hersteller Cisco, Extreme Networks, HP und Juniper.

VMNetwerkverbindungsarten

VMware Workstation bietet mehrere Möglichkeiten, die Netzwerkressourcen des Hosts zu nutzen. Je nach Anforderungen wird man eine dieser Möglichkeiten auswählen. Hier sind die wichtigsten erklärt:

Bridge

Hier benutzt der Gast die Netzwerkverbindung des Hosts mit einer eigenen IP in dessen lokalem Netz.

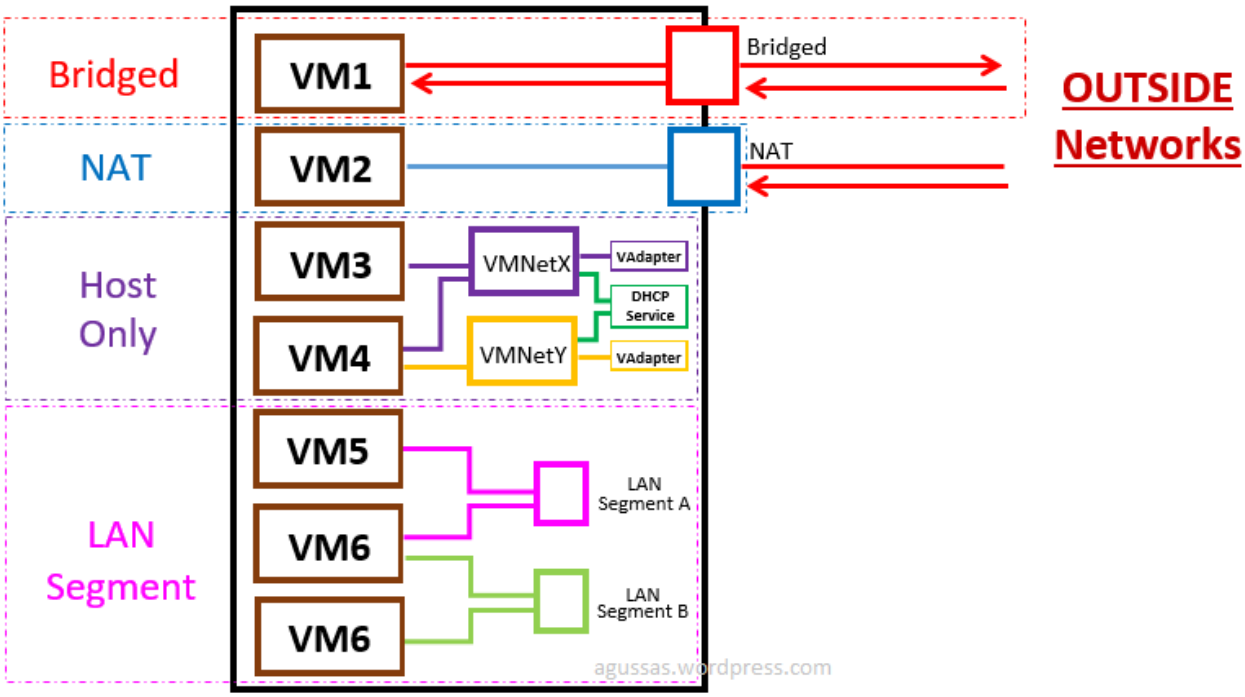
Das kommt der Installation eines separaten Rechners gleich – auch von außen her gesehen.

NAT

Der Gast bekommt eine IP in einem von VMware dafür eingerichteten privaten Netz, das vom physischen Netz des Hostrechners verschieden ist. Es kommt ein virtueller Netzwerkadapter zum Einsatz, den VMWare im Host einrichtet und mit einer weiteren IP des privaten Netzes konfiguriert; der Host wird auf Seite des Clients als Default-Gateway eingetragen. Via Adressübersetzung (NAT) kann der Gast nun das Host-Netz erreichen. Ressourcen des Gasts, z. B. Windows-Freigaben, sind nur vom Host aus unter der privaten IP des Gasts erreichbar. Von außen her ist nur eine IP sichtbar; dass sich dahinter mehrere Systeme befinden, kann nur durch Analyse des Datenverkehrs erkannt werden.

Host only

Auch hier richtet VMware ein privates Netz ein. Es werden jedoch keine Regeln definiert, die Datenpaketen des Gastes erlauben, dieses private Netz zu verlassen. Wenn zusätzliche Verbindungen gewünscht sind, müssen diese auf dem Host durch Routing (Forwarding) explizit hergestellt oder als Serverdienst (z. B. Proxy) realisiert werden. Diese Methode eignet sich vorzüglich, um einen dedizierten Server im lokalen Netz zu betreiben. Beispielsweise würde man für einen Terminalserver nur den Port für RDP freischalten. Damit wäre die Maschine für ihren eigentlichen Bestimmungszweck im Netz erreichbar, während z. B. Viren, die sich über andere Ports verbreiten, beim Host enden würden. Auch für private Zwecke eignet sich diese Methode, da damit verhindert werden kann, dass der Gast unerwünschte IP-Verbindungen (z. B. für Spamversand) aufbaut. Anmerkung: Mehrere Gastsysteme können separate private Netze nutzen, die nur miteinander kommunizieren können, wenn es jeweils im Hostsystem explizit konfiguriert ist.



Content Management System (CMS-System)

Ein CMS, ein **Content-Management-System**, dient zur **Verwaltung der Inhalte (Einpflege von Texten, Bildern, Videos und Daten)**. Ein WCMS (Web Content Management System) verwaltet Inhalte von einer oder auch mehrerer **Websites**. WCMS sind **mandantenfähig**, d.h. sie können mehrere Kunden (Mandanten) bedienen, ohne dass diese Einblick in die Benutzerverwaltung des anderen haben. In weiterer Folge, wird immer von einem WCMS ausgegangen, jedoch der Einfachheit halber wird immer der Begriff CMS verwendet.

Vorteile

Dezentralisierte Wartung

Ein gutes webbasiertes Content Management System funktioniert mit jedem gewöhnlichen Webbrowser (zum Beispiel Internet Explorer, Mozilla Firefox oder Opera). Sie benötigen also keine zusätzliche Software und können ohne Engpässe Ihre Internet-Präsentation warten und erweitern, wann und wo Sie wollen.

Trennung Inhalt & Gestaltung

Zum einen werden bei einem CMS die G**estaltung der Webseite (Layout & Design)** und die **Verwaltung der Inhalte getrennt** verwaltet. Dadurch muss man sich beim Einsatz eines CMS nicht bei Inhalten um das Layout kümmern und umgekehrt. Außerdem kann so **mit wenigen Klicks** im CMS das **Layout der kompletten Webseite geändert** werden.

Für Autoren ohne technischen Background

Wer Textverarbeitung beherrscht, kann auch in einem modernen CMS System online Inhalte erstellen, Bilder veröffentlichen oder neue Produkte anlegen. Über ein modernes Content Management System können Sie ohne HTML-Kenntnisse Ihre Homepage pflegen.

Konfigurierbare Zugriffsbeschränkung

Ein CMS System beinhaltet in der Regel eine eigene Benutzerverwaltung. Benutzern können Rollen und Berechtigungen zugewiesen werden, wodurch die unautorisierte Veränderung von Inhalten effektiv verhindert wird.

Einhalten von Design-Vorgaben

In einem CMS können Inhalte vom Design getrennt gespeichert werden. Folglich wird der gesamte

Content aller Autoren in einem Design zentral ausgegeben.

Automatische Navigations-Generierung

Menüs können in CMS Lösungen aus den Datenbankinhalten automatisch erzeugt bzw. generiert werden. Hyperlinks werden nur dargestellt, wenn sie auf gültige Seiten verweisen. Wenn beispielsweise diese Seiten gelöscht werden, können auch die Hyperlinks automatisch ausgeblendet werden.

Speicherung der Inhalte in einer Datenbank

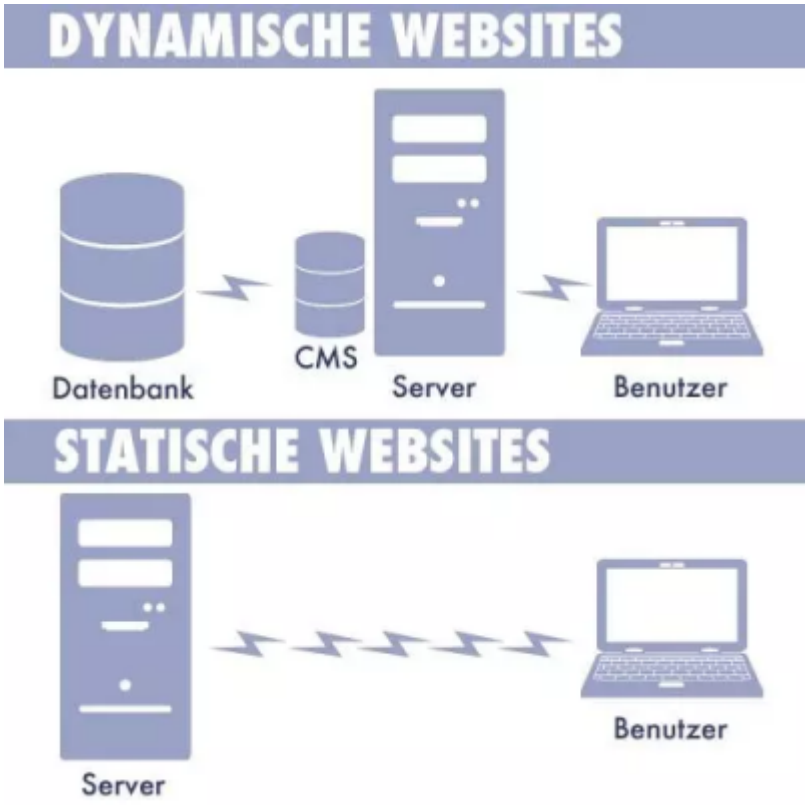
In einem guten CMS System werden die Inhalte in der zentraler Datenbank abgelegt. Eine zentrale Content-Speicherung bedeutet, dass Inhalte an verschiedenen Stellen wiederverwendet werden und für das jeweilige Medium (Webbrowser, Mobiltelefon/Wap, PDA, Print) adäquat formatiert werden können.

Dynamische Inhalte

Modulare Erweiterungen wie z.B. Foren, Umfragen, Shops, Anwendungen, Suchfunktionen und News-Management stehen als Module zum Einsatz bereit.

Content just in time

Die Publikation von Inhalten ist zeitgenau steuerbar. Inhalte können im Hintergrund vorbereitet und nur von berechtigten Usern in der sog. „hidden preview“ vorab eingesehen werden.



Nachteile

Eingeschränkte Gestaltung der Website

Durch die festgelegte Struktur eines CMS bestehen nicht vielen Freiräume was Widgets oder Features zur Integration in die Website betrifft. Es können nur verfügbare Features in die Website integriert werden.

Abhängigkeit vom Dienstleister

Entscheidet man sich für ein CMS, so ist man von der Aktualisierung und Service der Software vom jeweiligen Hersteller des Website-CMS abhängig.

Sicherheitsaspekt

Gerade **weitverbreitete Systeme (TYPO3, Joomla, Wordpress,...)** werden (genau wie bei den Betriebssystemen auch) natürlich **mit Vorliebe gehackt**. **Sicherheitslücken verbreiten sich schnell** in der Szene. Die vollständige Absicherung des Systems - zum Beispiel durch Ändern aller üblichen Installationspfade - kann sehr schnell sehr aufwändig werden. Allerdings sind bei den oben genannten Systemen auch die Entwickler sehr schnell mit der Erstellung von Patches. Es bedarf allerdings dann auch der entsprechenden Beobachtung der bekannten Foren und Mailinglisten.

Geringere Performance

Da die Inhalte dynamisch beim Aufruf der Seite generiert werden, kann der Auslieferung der kompletten Seite zumeist nur langsamer als bei einer statischen HTML-Seite erfolgen. Gute CMS-Systeme bieten hier allerdings geeignete Caching-Lösungen (Zwischenspeicher) an, die diesen Nachteil wieder ausgleichen.

Mehr Funktionalität als benötigt

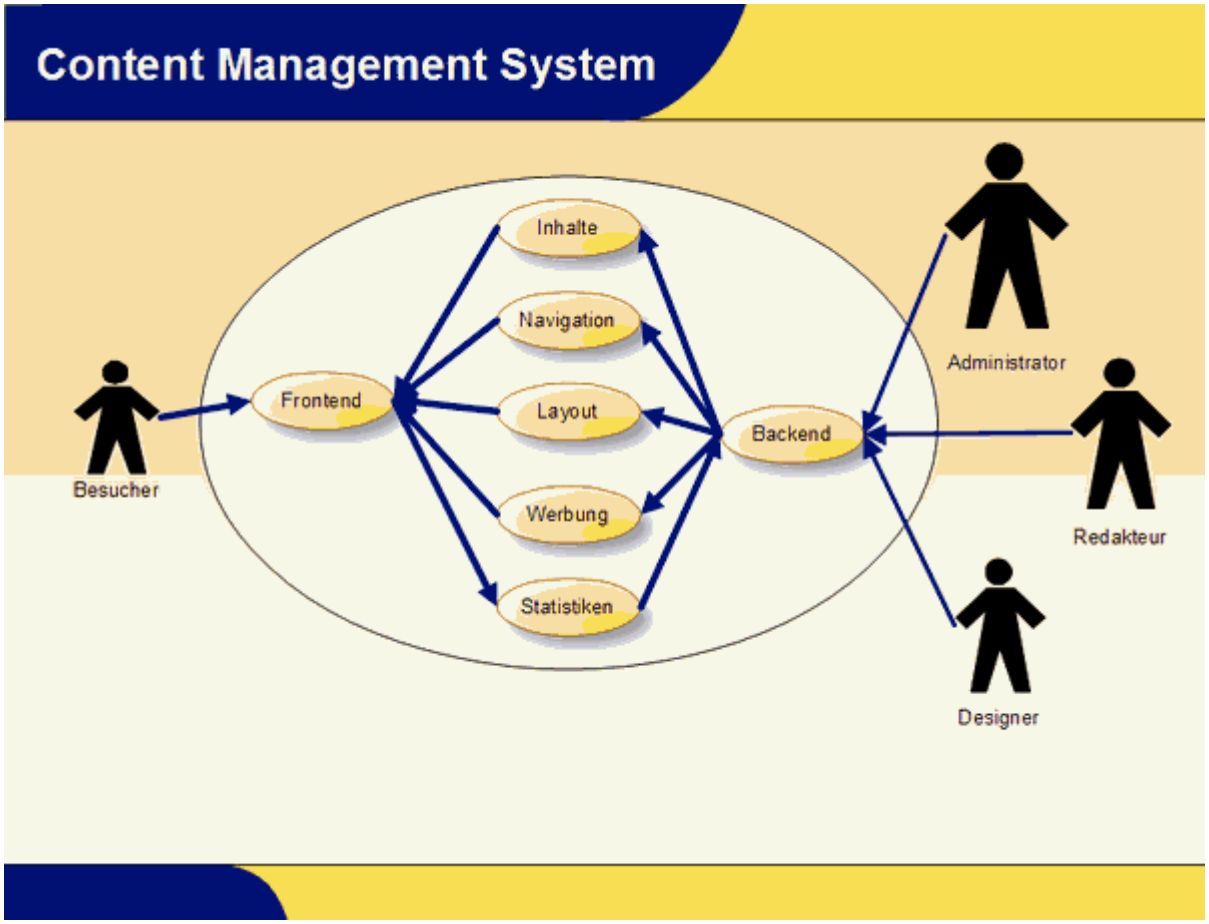
Ein CMS muss als Gesamtsystem angesehen werden, in dem sich viele Programmteile bedingen oder voneinander abhängen. Will man das System um nicht benötigte Funktionionen reduzieren kann es häufig zu unerwünschten Nebenwirkungen kommen.

Aufwendigere Backups

Genügt bei statischen Seiten meist ein einfaches Kopieren der Inhalte per FTP, so ist ein Backup eines CMS aufwändiger. Hier muss zusätzlich die Datenbank gesichert werden, sowie eventuell auch Systemeinstellungen des Servers. Auch das Rückspielen von Backups im Falles eines Datenverlustes sollte in regelmäßigen Abständen getestet werden.

Frontend & Backend eines CMS

Ein CMS wird meist in ein Frontend- und ein Backend-Bereich unterteilt. Das Frontend eines CMS (wörtlich übersetzt „Vorder-Ansicht“) ist die Webseite, die ein Besucher angezeigt bekommt.



Das Backend eines CMS (wörtlich übersetzt „Hinter-Ansicht“) ist ein geschützter Bereich, in dem die Administration des CMS stattfindet. Hier stellt das CMS nach einer Anmeldung alle Funktionen zur Verfügung, die der jeweilige Benutzer zum Verwalten der Webseite benötigt. Häufig können für unterschiedliche CMS-Benutzer auch unterschiedliche Zugriffsrechte festgelegt werden, z.B. der Designer auf die Templates des CMS, der Redakteur auf die Inhalte des CMS.

Verbreitete CMS-Systeme

WordPress

Ursprünglich ein reines Blogsystem ist es heute mit über 70 Mio. Installationen das am weitesten verbreitete CMS.

Vorteile

- Installations- und Einrichtungsaufwand überschaubar
- Riesige Auswahl an kostenlosen bzw. -günstigen Designs
- Vielfältige Erweiterungsmöglichkeiten durch Plugins
- Ideal für SEO-Maßnahmen (sehr gute Tools: z.B. YOAST)

Nachteile

- WordPress braucht jede Menge Systemressourcen
- Ladegeschwindigkeit bei hohem Traffic oft langsam
- Viele Updates, teilweise mit Sicherheitsrisiken

TYPO3

Content-Management-Systeme: Typo3-LogoDas „Open Source Flaggschiff“ mit mehr als 9 Mio. Installationen und 500.000+ Websites.

Vorteile

- Weitverbreitet, viele Experten und Entwickler
- Viele Funktionen
- Erheblich schneller als WordPress, auch bei hoher Besucherfrequenz

Nachteile

- Das Setup ist komplex und für den Laien ungeeignet
- Auch bei Backend-Anpassungen sind profunde Kenntnisse im Administrationsbereich erforderlich

---

Joomla

Content-Management-Systeme: Joomla-LogoIst schon lange auf dem Markt und hat mit 1,2 Mio.+ Downloads eine eingeschworene Fangemeinde. Durch einen Entwicklungsstopp vor ein paar Jahren ist Joomla ein wenig ins Hintertreffen geraten. Derzeit holt dieses CMS allerdings wieder mächtig auf.

Vorteile

- Installation und Einrichtung einfach und gut dokumentiert
- Viele Erweiterungen und vorgefertigte Designs (ähnlich wie bei WordPress)

Nachteile

- Beliebtes Ziel von Hackern, nicht zuletzt aufgrund der vielen Erweiterungsmöglichkeiten (gilt übrigens auch für WordPress)
- Beim Rollen- und Rechtemanagement gibt’s Luft nach oben

⇒ 2.01) Joomla Installation

Drupal

Content-Management-Systeme: Drupal-LogoDrupal ist ein Baukaustensystem mit einer riesigen Auswahl an Features. Obwohl es unter die Content-Management-Systeme fällt, ist es eigentlich ein Content-Management-Framework (CMF). Drupal funktioniert im Grunde wie Lego – man kann alles damit bauen.

Vorteile

- Drupal verfügt über ein differenziertes Rollen- und Rechtesystem
- Es hat jede Menge Funktionen, die als Baustein in das System integriert werden können
- Das Backend ist voll individualisierbar

Nachteile

- Das Drupal-Setup ist vergleichsweise kompliziert
- Eingriffe im Backend und am Server erfordern Routine

JOOMLA Installation unter Ubuntu

Voraussetzungen

Einzige Voraussetzung ist ein Webserver (bspw. Apache) mit geladenem PHP5-Modul sowie MySQL als Datenbank. Eine Schnellanleitung zur Installation ist dem Artikel LAMP zu entnehmen. Detail-Informationen finden sich im Wiki in den folgenden Artikeln.

- Webserver (z.B. Apache)
- PHP
- MySQL

siehe https://wiki.ubuntuusers.de/Joomla%21/

Installation von Apache

- sudo apt-get install apache2
- Anleitung: https://wiki.ubuntuusers.de/Apache_2.4/
- Webverzeichnis liegt auf `/var/www/html`
- Server-Ip-Adresse anzeigen lassen mit `ifconfig`
- Überprüfung des Webservers durch Aufruf der IP-Adresse mit Browser

Installation von PHP

- sudo apt-get install php
- sudo apt-get install php-cli
- sudo apt-get install libapache2-mod-php7.0
- sudo apt-get install php-mysql
- Anleitung: https://wiki.ubuntuusers.de/PHP/
- Eventuell müssen Updates eingespielt werden: sudo apt-get update
- Überprüfung des Webserver-PHP-Moduls durch eine index.php Seite mit dem Befehl **phpinfo();**
- PHP-Version ermitteln –> php -v

Installation von MySQL

- sudo apt-get install mysql-server
- PW: schule
- Anleitung: https://wiki.ubuntuusers.de/MySQL/

Installation vom Modul PHP-MySQL

- sudo apt-get install php-mysql

Installation von Joomla

- Download von Joomla unter https://downloads.joomla.org/de/cms/joomla3/3-8-12/Joomla_3-8-12-Stable-Full_Package.zip?format=zip
- Entweder am Host-System downloaden und in die VM kopieren mittels WinScp über SSH (Dazu muss man vorher den Dienst ssh installieren) oder direkt mittels dem Befehl wget https://downloads.joomla.org/de/cms/joomla3/3-8-12/Joomla_3-8-12-Stable-Full_Package.zip?format=zip am Gastsystem downloaden
- Entpacken mittels unzip
- Rechte setzen für den Webuser www-data (chown www-data:www-data /var/www/html -R)

Mögliche Probleme

- sudo apt-get update
- sudo apt-get install php-xml
- Neustart des Servers: `init 6`

MySQL Root Passwort zurücksetzen

- Passwort des mysql-Users debian-sys-maint auslesen in der Datei /etc/mysql/debian.cnf
  - sudo less /etc/mysql/debian.cnf
    - Man „merkt“ sich die Buchstaben-Zahlenkolonne
- mysql -u debian-sys-maint -p
  - Passwort eingeben (gemerkte Buchstaben-Zahlenkolonne)
- Jetzt befindet man sich auf der mysql-Konsole, folgenden Befehl eingeben:
  - `UPDATE mysql.user SET authentication_string=PASSWORD('schule'), plugin='mysql_native_password' WHERE User='root' AND Host='localhost';`
  - `flush privileges;`
  - `exit;`
- mysql-Server neu starten:
  - `/etc/init.d/mysql restart`
- Jetzt kann man mit dem User root einsteigen:
  - `mysql -u root -p`
  - Passwort schule eingeben
- Jetzt kann Joomla installiert werden.

NETZWERKE

Allgemeine Netzwerkgrundlagen

Netzwerk-Grundlagen

Was ist ein Netzwerk?

Ein Netzwerk ist die **physikalische und logische Verbindung von Computersystemen**. Ein einfaches Netzwerk besteht aus zwei Computersystemen. Sie sind über ein Kabel miteinander verbunden und somit in der Lage ihre Ressourcen gemeinsam zu nutzen. Wie zum Beispiel Rechenleistung, Speicher, Programme, Daten, Drucker und andere Peripherie-Geräte. Ein netzwerkfähiges Betriebssystem stellt den Benutzern auf der Anwendungsebene diese Ressourcen zur Verfügung.

Notwendigkeit für ein Netzwerk

Als es die ersten Computer gab, waren diese sehr teuer. Peripherie-Geräte und Speicher waren fast unbezahlbar. Zudem war es erforderlich zwischen mehreren Computern Daten auszutauschen. Aus diesen Gründen wurden Computer miteinander verbunden bzw. vernetzt.

Daraus ergaben sich einige Vorteile gegenüber unvernetzten Computern:

- zentrale Steuerung von Programmen und Daten
- Nutzung gemeinsamer Datenbeständen
- erhöhter Datenschutz und Datensicherheit
- größere Leistungsfähigkeit
- gemeinsame Nutzung der Ressourcen

Die erste Möglichkeit, Peripherie-Geräte gemeinsam zu nutzen, waren manuelle Umschaltboxen. So konnte man von mehreren Computern aus einen Drucker nutzen. An welchem Computer der Drucker angeschlossen war, wurde über die Umschaltbox bestimmt. Leider haben Umschaltboxen den Nachteil, dass Computer und Peripherie beieinander stehen müssen, weil die Kabellänge begrenzt ist.

Größenordnung von Netzwerken

Jedes Netzwerk basiert auf Übertragungstechniken, Protokollen und Systemen, die eine Kommunikation zwischen den Netzwerk-Teilnehmern ermöglichen. Bestimmte Netzwerktechniken unterliegen dabei Beschränkungen, die insbesondere deren Reichweite und Ausdehnung begrenzt. Hierbei haben sich verschiedene Netzwerk-Dimensionen durchgesetzt für die es unterschiedliche Netzwerktechniken gibt.

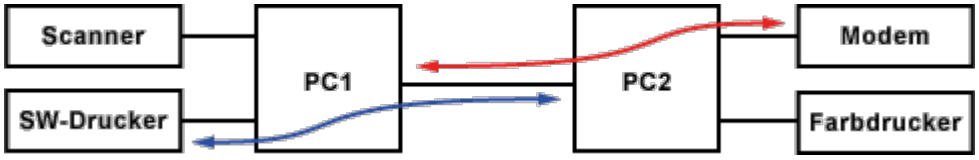
- PAN - Personal Area Network: personenbezogenes Netz, z. B. Bluetooth
- LAN - Local Area Network: lokales Netz, z. B. Ethernet
- MAN - Metropolitan Area Network: regionales Netz
- WAN - Wide Area Network: öffentliches Netz, z. B. ISDN
- GAN - Global Area Network: globales Netz, z. B. das Internet

In der Regel findet ein Austausch zwischen den Netzen statt. Das heißt, dass Netzwerk-Teilnehmer eines LANs auch ein Teilnehmer eines WANs oder eines GANs ist. Eine 100%ig klare Abgrenzung zwischen diesen Dimensionen ist nicht immer möglich, weshalb man meist nur eine grobe Einteilung vornimmt. So unterscheidet man in der Regel zwischen LAN und WAN, wobei es auch Techniken und Protokolle gibt, die sowohl im LAN, als auch im WAN zum Einsatz kommen.

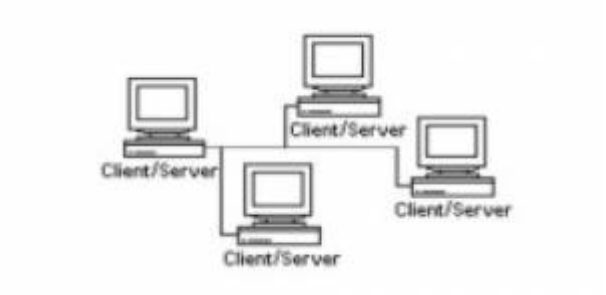
Peer-to-Peer-Netze und Client-Server-Architekturen

Man unterscheidet zwei „Philosophien“:

Peer-to-Peer-Netzwerke



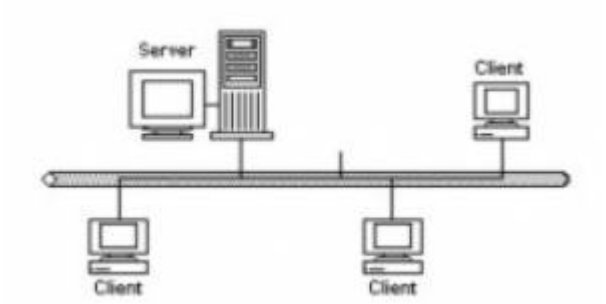
In einem Peer-to-Peer-Netzwerk ist jeder angeschlossene Computer zu den anderen gleichberechtigt. Jeder Computer stellt den anderen Computern seine Ressourcen zur Verfügung. Ein Peer-to-Peer-Netzwerk eignet sich für bis zu 10 Stationen. Bei mehr Stationen wird es schnell unübersichtlich. Diese Art von Netzwerk ist relativ schnell und kostengünstig aufgebaut. Die Teilnehmer sollten möglichst dicht beieinander stehen. Einen Netzwerk-Verwalter gibt es nicht. Jeder Netzwerk-Teilnehmer ist für seinen Computer selber verantwortlich. Deshalb muss jeder Netzwerk-Teilnehmer selber bestimmen, welche Ressourcen er freigeben will. Auch die Datensicherung muss von jedem Netzwerk-Teilnehmer selber vorgenommen werden.



Peer-to-Peer-Netze brauchen keinen eigenen Server-Rechner, da jeder PC Server-Funktionen übernehmen kann.

Client/Server-Architekturen

In einem serverbasierten Netzwerk werden die Daten auf einem zentralen Computer gespeichert und verwaltet. Man spricht von einem dedizierten Server, auf dem keine Anwendungsprogramme ausgeführt werden, sondern nur eine Server-Software und Dienste ausgeführt werden. Diese Architektur unterscheidet zwischen der Anwender- bzw. Benutzerseite und der Anbieter- bzw. Dienstleisterseite. Der Anwender betreibt auf seinem Computer Anwendungsprogramme (Client), die die Ressourcen des Servers auf der Anbieterseite zugreifen. Hier werden die Ressourcen zentral verwaltet, aufgeteilt und zur Verfügung gestellt. Für den Zugriff auf den Server (Anfrage/Antwort) ist ein Protokoll verantwortlich, dass sich eine geregelte Abfolge der Kommunikation zwischen Client und Server kümmert. Die Client-Server-Architektur ist die Basis für viele Internet-Protokolle, wie HTTP für das World Wide Web oder SMTP/POP3 für E-Mail. Der Client stellt eine Anfrage. Der Server wertet die Anfrage aus und liefert eine Antwort bzw. die Daten zurück.



Netzwerk-Topologien

Die Struktur eines Netzwerks bezeichnet man als Topologie. Wie wichtig die Struktur eines Netzwerks ist, merkt man bei einem Leitungsausfall: ein gutes Netzwerk findet bei einem Leitungsausfall selbstständig einen neuen Pfad zum Empfänger.

Physikalische und logische Topologie

Interessant ist, dass sich die **sichtbare Topologie** (also die physische Verkabelungsstruktur) vom tatsächlichen Datenfluss unterscheiden kann. Deshalb verwendet man für die hardwaremäßige Realisierung den Begriff **physikalische Topologie** während man für den tatsächlichen Datenfluss den Begriff „logische Topologie“ verwendet.

Die wichtigsten Netzwerktopologien sind:

Bus-Topologie

ALLE GERÄTE nutzen DASSELBE KABEL

Bei einem Bussystem sind alle Rechner hintereinander geschaltet und über Abzweige (T-Stücke) an das Netzwerkkabel angeschlossen. Problem: Eine Verbindungsunterbrechung betrifft den ganzen Bus!

Vorteile

- Relativ niedrige Kosten, da geringe Kabelkosten
- Ausfall einer Station führt zu keinem Netzausfall

Nachteile

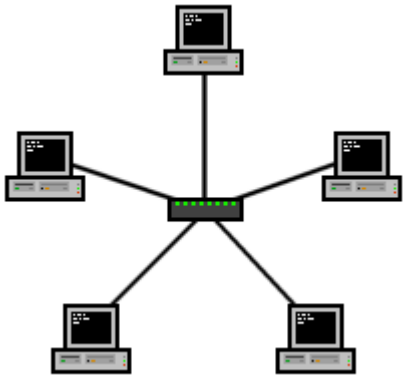
- Alle Daten über ein Kabel
- Nur eine Station kann senden. Alle anderen sind blockiert.
- Eine Störung an einer Stelle (z.B.: Defektes Kabel) führt zu einem Netzausfall (⇒ aufwendige Fehlersuche)
- Unverschlüsselter Netzwerkverkehr kann direkt am Bus (=Kabel) mitgelesen werden

Einsatzgebiet

Früher aufgrund der niedrigen Kosten häufig verwendet, heute spielt die Bus-Topologie keine Rolle mehr und wurde von der Stern-Topologie verdrängt.

Bus-Topologie - Details

Stern-Topologie



JEDES GERÄT nutzt EIN KABEL.
Damit ist es zu einem Verteiler verbunden. Es existiert eine Punkt-zu-Punkt Verbindung zwischen Verteiler und Gerät. Als Verteiler kann ein HUB oder ein SWITCH dienen.

Vorteile

- Ausfall einer Station oder eines Defekts an einem Kabel führt zu keinem Netzausfall
- Aktive Verteiler (Switch, Hub) dienen gleichzeitig als Signalverstärker
- Bei richtiger Konfiguration können zwei Stationen die volle Bandbreite des Übertragungsmediums für ihre Kommunikation nutzen, ohne andere Stationen dabei zu behindern. Diese physikalische Topologie erlaubt somit sehr hohe Datendurchsatzraten.
- Weitere Stationen oder Verteiler können einfach hinzugefügt werden. Sehr leicht skalierbar.

Nachteile

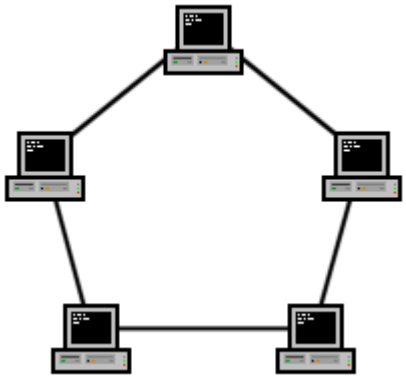
- Große Kabelmengen
- Beim Ausfall des Verteilers ist kein Netzverkehr mehr möglich

Einsatzgebiet

Im praktischen Einsatz bei lokalen Netzwerken findet die Stern-Topologie Verwendung. Häufigste Form der Verkabelung.

Stern-Topologie - Details

Ring-Topologie



JEDES GERÄT ist mit ZWEI NACHBARN verbunden.

Die Ring-Topologie ist eine geschlossene Form, es gibt keinen Kabelanfang und kein Kabelende.
Es handelt sich jeweils um eine Punkt-zu-Punkt Verbindung zwischen den Rechnern.
Jede Station hat genau einen Vorgänger und einen Nachfolger.
Datenverkehr findet immer nur in eine Richtung statt.

Vorteile

- Vorgänger und Nachfolger sind festgelegt
- Alle Stationen verstärken das Signal
- Alle Stationen haben gleiche Zugriffsmöglichkeit
- Leicht umsetzbar

Nachteile

- Ausfall einer Station oder eines Kabelteils führt zu einen Netzausfall
- Hoher Aufwand bei der Verkabelung (Jede Station braucht 2 Netzwerkkarten)
- Leicht abhörbar
- langsame Datenübertragung bei vielen Stationen

Einsatzgebiet

Physikalische Anwendung gibt es heute keine mehr.
Logische Anwendung findet sie im Token Ring.

W Ring-Topologie - Details

Mischformen

Sind zumeist **Kombinationen aus Bus, Stern und Ring**.

Backbone

Unter einem Backbone („Rückgrat“) wird die physikalische Verbindung zwischen einzelner Teilnetze verstanden. Es wird auch oft als Hintergrundnetz betitelt und verbindet z.B. mehrere Gebäude.

Stern-Bus-Netz

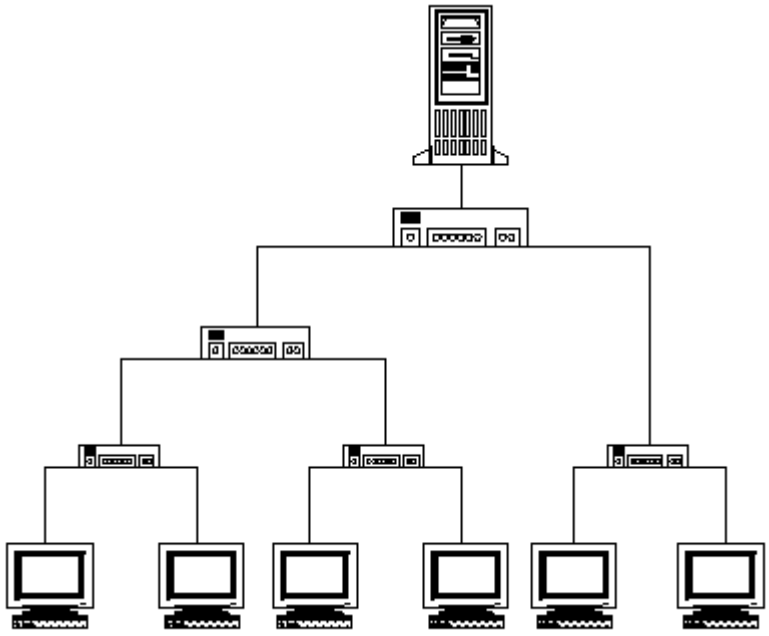
Ein Stern-Bus-Netz entsteht, wenn mehrere Verteiler über einen Bus miteinander verbunden sind. Häufig sind so mehrere Stockwerke miteinander verbunden.

Stern-Stern-Netz

Ein Stern-Stern-Netz entsteht, wenn mehrere Verteiler wiederum über einen Verteiler verbunden sind. Häufigste Anwendung ist wiederum das Verbinden von mehreren Subnetzen (z.B. Netze in verschiedenen Stockwerken).
Fällt der Hauptverteiler aus, so kann zwischen den Stockwerken nicht mehr kommuniziert werden. Jedoch kann man auch die Hauptverteiler redundant auslegen.

Baum

Eine Baum-Topologie wird so aufgebaut, dass, ausgehend von der Wurzel, eine Menge von Verzweigungen zu weiteren Verteilungsstellen existiert.
Es handelt sich somit um eine Erweiterung der Stern-Stern-Topologie auf mehrere Ebenen.



Maschen-Topologie

Vorherrschende Netzstruktur in großflächigen Netzen (z.B. öffentliche Telekommunikationsnetze).
Maschen-Topologie - Details

Zell-Topologie

Die Zell-Topologie kommt hauptsächlich bei drahtlosen Netzen zum Einsatz. Eine Zelle ist der Bereich um eine Basisstation (z.B. Wireless Access Point), in dem eine Kommunikation zwischen den Endgeräten und der Basisstation möglich ist.
Zell-Topologie - Details

Übertragungsmedien

Die Übertragungsmedien sind die Straßen der Daten. Der Aufbau dieser Straßen muss sehr gut geplant werden, um alle aktuellen Anforderungen bzw. eventuelle zukünftige Anforderungen ohne großartige Veränderungen zu erfüllen.

Als **Maßeinheit für die Übertragungsgeschwindigkeit** werden die Werte in **bit/s, b/s bps, -> also Bit pro Sekunde** angegeben. Achtung: Nicht zu verwechseln mit **Byte/s -> Byte pro Sekunde**!!

`C= D/t` `(bits)/(s)`

1) Rechenbeispiel:

Es werden 100MB in 10s übertragen. Wie hoch ist die Übertragungsgeschwindigkeit?

- D=100MByte
- t=10s

Rechenschritt	**Berechnung**
D umwandeln in bits	`100*1024*1024*8 = 838860800` `bits`
C berechnen	`C=D/t=(838 860 800)/10=83886080` `(bit)/(s)`
C umwandeln in Mbit/s	`(83886080)/(1024)/(1024)=80` `(Mbit)/(s)`

2) Rechenbeispiel:

Max hat eine Datentransferrate von 10Mbit/s Download und 2Mbit/s Upload.

a) Wie lange braucht er, um 10MB runterzuladen?

- D=10MB
- C=10Mbit/s

Rechenschritt	**Berechnung**
D umwandeln in bits	`10*1024*1024*8 = 83886080` `bits`
C umwandeln in bit/s	`10*1024*1024=10485760` `(bit)/(s)`
Formel umformeln	`t=(D)/(C)` `(bits)/((bit)/s)`
In Formel einsetzen	`t=83886080/10485760=8` `s`

b) Wie lange braucht er, um 10MB hochzuladen?

Rechenschritt	**Berechnung**
D umwandeln in bits	`10*1024*1024*8 = 83886080` `bits`
C umwandeln in bit/s	`2*1024*1024=2097152` `(bit)/(s)`
Formel umformeln	`t=(D)/(C)` `(bits)/((bit)/s)`
In Formel einsetzen	`t=83886080/2097152=40` `s`

Leitergebundene Übertragung

Bei einer leitergebundenen Übertragung werden Medien in Form von Kabeln benötigt (Metallische Leiter, Glasfaser).
Ein Kabel besteht zumindest aus einer Ader (=Faser).
Mehrere Adern sind durch entsprechende Isolationssschichten getrennt.
Alle Adern wiederum werden von einem Mantel als Schutz umgeben.
Die Übertragung selbst erfolgt durch elektromagnetische Schwingungen.

Koaxialkabel

Das früher verwendete Koaxialkabel ist in modernen Netzen praktisch vollständig von Twisted Pair-Kabeln (TP) und Lichtwellenleiter (LWL) abgelöst worden.

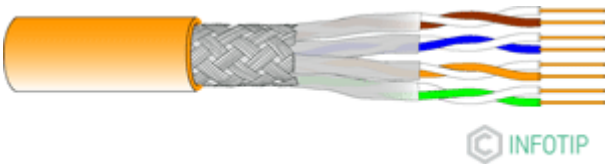
Es besteht aus

- einem Innenleiter (Kupfer, Stahlkupfer)
- einer Isolation (Dielektrikum)
- einer Abschirmung (Metallgeflecht schützt vor magnetischen Störungen –> Rauschen & Übersprechen)
- einem Mantel

Es waren bis zu 10Mbps möglich:

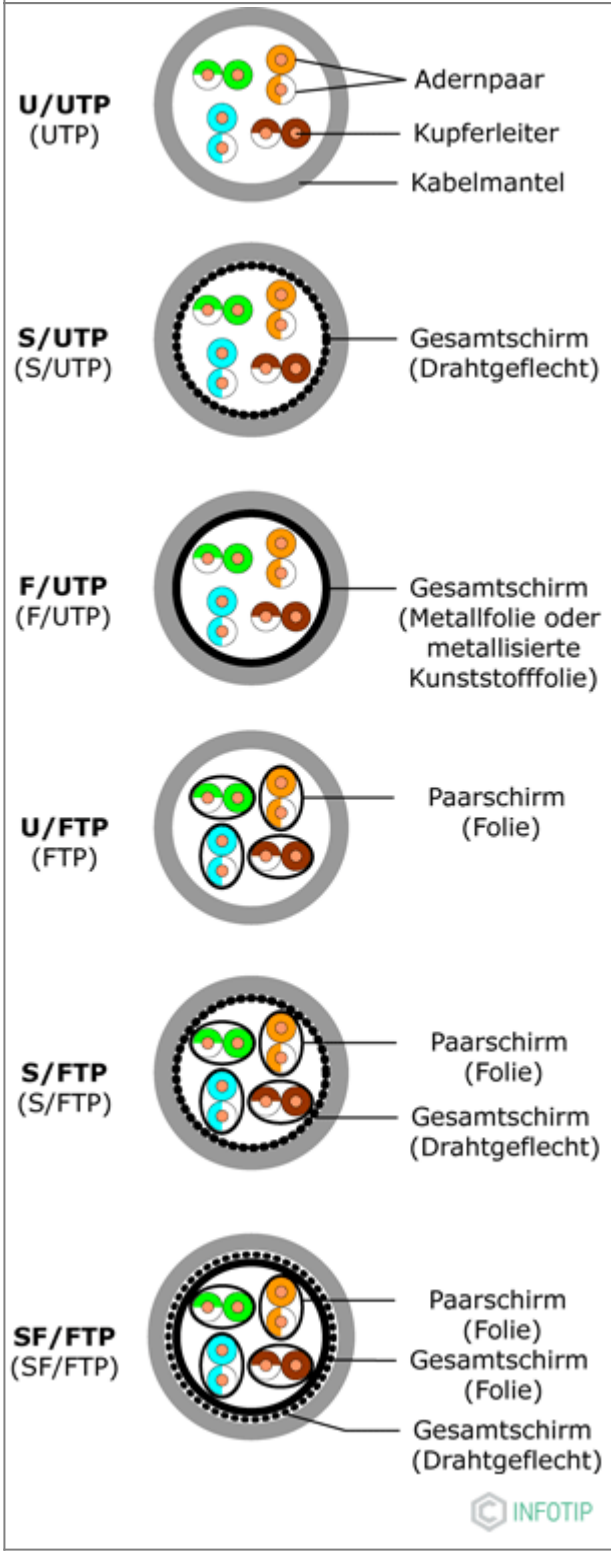
- Thicknet (10Base5)
- Thinnet (10Base2) - Heute noch bei Satellitenempfang im Einsatz

Twisted-Pair Kabel



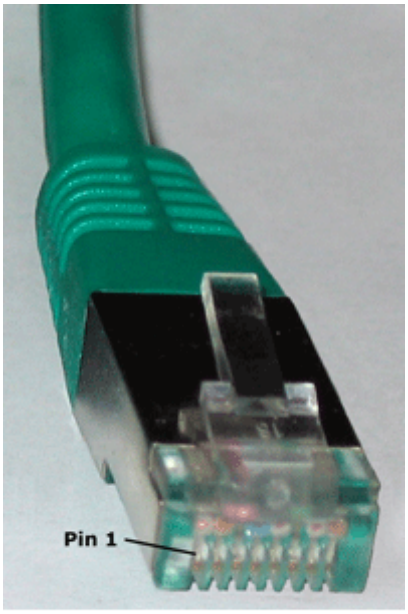
Twisted Pair ist ein vier-, acht- oder mehr-adriges Kupferkabel, bei dem jeweils zwei Adern miteinander verdrillt sind. Durch die Verdrillung kompensieren sich Leitungskapazität und -induktivität. Dadurch steigt die Übertragungsbandbreite und die mögliche Übertragungsreichweite wird praktisch nur durch die Dämpfung des Wirkwiderstandes begrenzt. Die Verwendung von symmetrischen Signalen (Differentialspannungen) erhöht die Festigkeit gegen elektromagnetische Störstrahlung.

Twisted Pair-Kabel gibt es in zahlreichen Varianten. Twisted Pair-Verbindungen werden außer in der Kommunikationstechnik (Netzwerkkabel, Telefonkabel) auch bei HDMI-, DVI- und LVDS-(in LCD/Plasma-TV zwischen Signalprozessor und Display) Verbindungen eingesetzt. Die Anzahl der Leiterpaare im Kabel hängt dabei von der benötigten Datenübertragungsrate ab. In Netzwerken wird für jede Übertragungsrichtung (senden, empfangen) wird jeweils ein Adernpaar (bei 100BaseT4 und 1000BaseT jeweils zwei) genutzt. Die Übertragungsreichweite ist abhängig vom Aufbau des Kabels, von der Dämpfung (=Länge) des Kabels und von den externen Störeinflüssen. Twisted Pair-Kabel für Netzwerke gibt es in zahlreichen Varianten:

Bild	Benennung	Beschreibung
	**U/UTP-Kabel**	**Unshielded/Unshielded Twisted Pair** sind nicht abgeschirmte verdrillte Leitungen und gehörten früher typischerweise der CAT3 an. UTP-Kabel sollten im industriellen Bereich oder in der Datentechnik mit hohen Datenraten nicht verwendet werden.
	S/UTP-Kabel	**Screened/Unshielded Twisted Pair** haben einen Gesamtschirm aus einem Kupfergeflecht zur Reduktion der äußeren Störeinflüsse
	F/UTP-Kabel	**Foilshielded/Unshielded Twisted Pair** besitzen zur Abschirmung einen Gesamtschirm, zumeist aus einer alukaschierten Kunststofffolie
	U/FTP-Kabel	**Unshielded/Foilshielded Twisted Pair** auch genannt **CAT5** oder **CAT5e**. Die Leitungsadern sind paarweise mit Folie abgeschirmt
	S/FTP-Kabel	**Screened/ Foilshielded Twisted Pair** auch genannt **CAT6** sollten in Bereichen mit hoher Störstrahlung (z.B. Büros mit mehreren PCs) eingesetzt werden.
	SF/FTP-Kabel	**Screened Foilshielded/Foilshielded Twisted Pair** auch genannt **CAT6e** oder **CAT7** besitzen eine Abschirmung für jedes Kabelpaar sowie eine doppelte Gesamtschirmung. Hierdurch kann eine optimale Störleistungsunterdrückung erreicht werden. Auch das Übersprechen zwischen den einzelnen Adernpaaren wird so wirksam unterdrückt

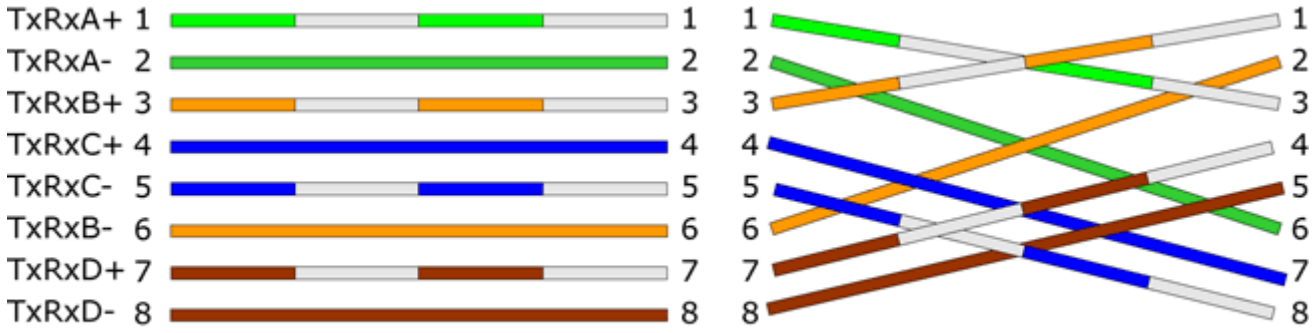
Die Preisunterschiede zwischen CAT-5e- Kabeln und CAT-7-Kabeln ist so gering, dass es sich bei Neuinstallation auf jeden Fall empfiehlt, CAT-7-Kabel einzusetzen. Dieses ist als einziges Kupfermedium in der Lage mit dem kommenden 10GBit-LAN verwendet zu werden.

Verbinder - RJ45



Der typische Standardverbinder für die Twisted-Pair-Verkabelung eines kupfergebundenen Ethernet-Netzwerkes ist der **8polige Western-Modularstecker RJ-45** (8P8C), auch RJ-48 oder RJ-49 genannt. RJ-45 Steckverbindungen können auf zwei Arten belegt sein, wobei die Belegung nach T568B am weitesten verbreitet zu sein scheint:

Belegung nach EIA/T-T568A		**Belegung nach EIA/T-T568B**	
Pin	**Farbe**	**Pin**	**Farbe**
1	weiß-grün	1	weiß-orange
2	grün	2	orange
3	weiß-orange	3	weiß-grün
4	blau	4	blau
5	weiß-blau	5	weiß-blau
6	orange	6	grün
7	weiß-braun	7	weiß-braun
8	braun	8	braun



Bei 1:1-Verbindungen sind beide Beschaltungen sind elektrisch zueinander kompatibel. Nur bei Erweiterungen von fest verdrahteten Netzen ist festzustellen, welche Belegung bereits vorgegeben ist. Normale Verbindungskabel („ Patchkabel“) mit RJ-45-Steckern sind 1:1 verschaltet, d.h. Pin 1 des einen Steckers geht auf den Pin 1 des anderen Steckers usw. Nur in besonderen Fällen, wenn z.B.

zwei Netzwerkkarten direkt miteinander verbunden werden sollen oder wenn Netzwerkkomponenten (z.B. Hubs älterer Bauart) über keinen dedizierten Uplink-Port verfügen, kann der Einsatz von Crossover-Kabeln notwendig werden.

LichtWellenLeiter (LWL)

Sind mit der Netzwerkverkabelung weite Strecken zu überwinden, z.B. zwischen einzelnen Gebäuden auf einem Fabrikgelände („Campusbereich“), sind sehr hohe Datenübertragungsraten (z.Zt. bis zu 170Gb/s) gefordert oder wenn sich die Datenübertragung per Kupferkabel aus technischen Gründen (z.B. bei extremer Störstrahlung) oder aus Gründen der Sicherheit verbietet, werden Lichtwellenleiter (LWL, Glasfasern) als Übertragungsmedium eingesetzt. Die Lieferprogramme der Hersteller erlauben mittlerweile die Übertragungsstrecken bis zum Einzelplatz komplett auf der Basis von LWL auszuführen.

Aufbau und Prinzip

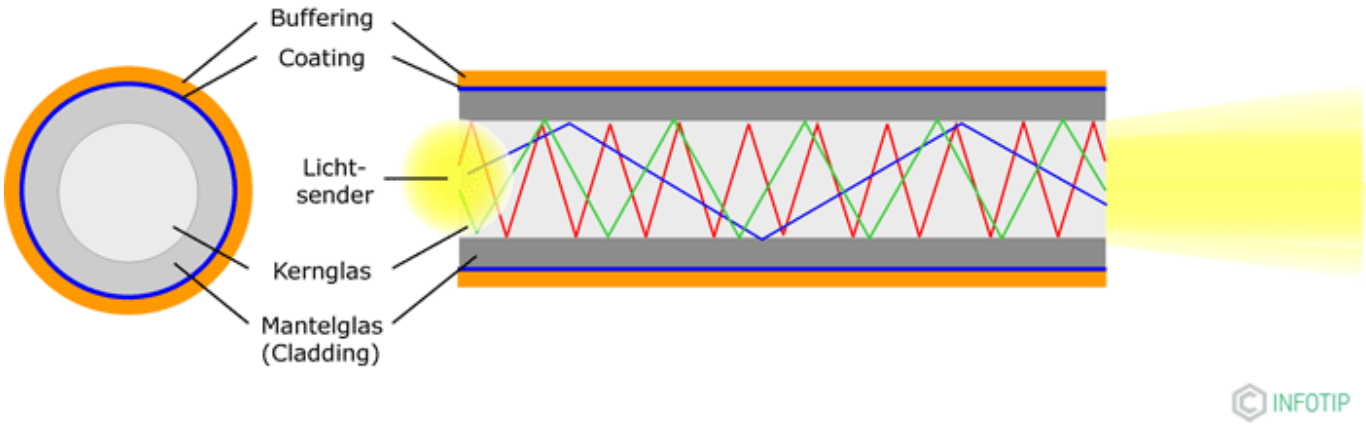
In einem LWL werden die Informationen nicht, wie in einem Kupferkabel, elektrisch übertragen, sondern mit **Licht**.
Der eigentliche LWL ist eine Faser aus Glas oder Kunststoff. Jede Faser besteht aus zwei Schichten. Der konzentrische Kern besteht aus einem optischen Material mit einem hohen Brechungsindex, das Mantelglas („ Cladding“) aus einem Material mit niedrigem. Licht, das in einem bestimmten Winkelbereich auf den Übergang von Kern zum Mantel trifft wird dort vollständig reflektiert. Über solche fortlaufenden Totalreflexionen pflanzt sich das Licht durch den LWL bis zum Ende der Faser fort.

Je steiler der Einfallswinkel des Lichts bei der Einspeisung in den LWL ist, desto häufiger wird die Lichtwelle reflektiert. Mit jeder Reflektion der Lichtwelle wird der Weg, des sogenannten Modes, länger.

Licht, das wenig häufig reflektiert wird, hat einen kürzeren Weg und durchläuft die Faser schneller. Es ist Licht niedrigen Modes.

Licht, das sehr häufig reflektiert wird, hat eine niedrige Ausbreitungsgeschwindigkeit in der Faser. Er ist Licht hohen Modes.

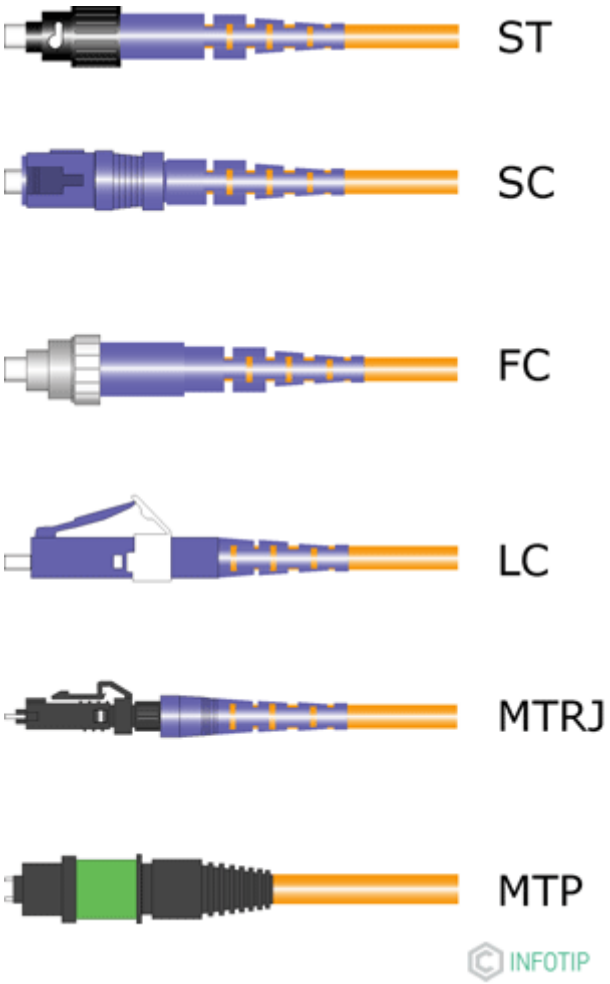
Erzeugt die Lichtquelle des Senders ein nicht-kohärentes Licht, tritt das Licht mit einer Vielzahl unterschiedlicher Winkel in die Faser ein. Dadurch entstehen natürlich durch die unterschiedlichen Moden Laufzeitunterschiede zwischen den Signalanteilen. Ein Eingangsimpuls mit steilen Flanken wird dadurch verschliffen und in seiner Breite gedehnt. Je länger ein Kabel ist, desto höher wird auch diese sog. Dispersion (Einheit: ns/km). Die Dispersion beeinflusst direkt die Übertragungsbandbreite der Glasfaserverbindung.



Da die Fasern sehr dünn und empfindlich sind, werden sie zum mechanischen Schutz mit einer Kunststoffbeschichtung („ Coating“) und einem Schutzüberzug versehen. In einem LWL-Kabel können mehrere Fasern, sogar in mehreren Bündeln, zusammen gefasst sein.

Verbinder

Die Hersteller von Netzwerkzubehör bieten konfektionierte Verbindungs- und Patchkabel mit einer Vielzahl von verschiedenen Steckerformen an. Meist sind die Kabel paarweise angelegt um beide Datenflussrichtungen (TX und RX) gleichzeitig herstellen zu können.



Vorteile

- hohe Reichweite
- hohe Übertragungsbandbreite
- Potentialfrei, daher auch für explosionsgefährdete Bereiche geeignet
- hohe Störfestigkeit, LWL können sogar zu Energieversorgungskabeln parallel verlegt werden
- hohe Abhörsicherheit

Nachteile

- Material für die Verkabelung ist teuer
- teure Verbindungstechnik
- Die Montagekosten sind wegen des höheren technischen Aufwandes höher
- komplexe und teure Messtechnik
- zusätzliche Kosten für Medienkonverter auf Kupfer-Ethernet

Leiterungebundene Übertragung

Als leiterungebundene Übertragung bezeichnet man eine Übertragung per

- Funk
- Ultraschall

- Infrarot
- Laser
- Licht

Drahtlose Übertragung (WLAN)

Die Übertragung von Informationen ohne Kabel ist mittlerweile in vielen Lebensbereichen als praktische Alternative eingezogen. Von der Fernbedienung eines Fernsehers über drahtlose Lautsprecher bis zum Smartphone gibt es viele Beispiele für die Umsetzung dieser Technik. Dabei werden Funksignale in frei verfügbaren Frequenzbändern anstelle von Kabeln für die Datenübertragung verwendet.

Vorteile

- Es sind keine baulichen Maßnahmen innerhalb eines Gebäudes nötig.
- Die baulichen Maßnahmen zwischen verschiedenen Gebäuden sind geringer als bei einer Verkabelung.
- Höhere Mobilität, da theoretisch jeder Punkt eines Firmengeländes drahtlos erreichbar ist.

Nachteile

- Oft geringere Datenübertragungsraten als bei Kabeln, die abhängig von Hindernissen sind.
- Anfällig für Störeinflüsse und Abhören durch Unbefugte.
- Probleme mit Ausleuchtung und Reflexionen.
- Bei vielen gleichzeitigen Nutzern an einem WLAN-Zugang bricht die Übertragungsrate ein (Shared Media).

Sicherheit als kritischer Bereich

Gerade im Bereich Sicherheit gibt es bei WLANs einige Punkte zu beachten, die sich auch in einem etwas größeren Konfigurationsaufwand äußern. Die sogenannte **SSID (Service Set Identifier)** kann eine eindeutige Identifikation (Firmenname etc.) enthalten, damit bei Problemen eine Kontaktaufnahme mit dem Betreiber möglich ist. Ein Verbergen bringt nicht viel, da die SSID in jedem Paket mitgeschickt wird und es Programme gibt, die auch verborgene SSIDs auslesen können. Eine versehentliche Verbindung durch Unbefugte ist bei verschlüsselten Zugängen nicht zu erwarten. Es gibt auch Empfehlungen, als SSID eine zufällige Zeichenfolge einzugeben, damit die SSID keine Rückschlüsse auf den Betreiber zulässt.

Letztendlich sollte die Übertragung im WLAN nur **verschlüsselt** erfolgen, wobei die Verschlüsselung mit **WEP (Wired Equivalent Privacy)** unsicher ist. Besser ist der Einsatz von **WPA (Wi-Fi Protected Access)** bzw. der Nachfolgetechnologie **WPA2**, da hier deutlich stärkere Verschlüsselungsmechanismen mit **AES (Advanced Encryption Standard)** verwendet werden. Zusammen mit **TKIP (Temporal Key Integrity Protocol)** sind allerdings nur max. 54 Mbit/s möglich! Höhere Raten erreicht z. B. WPA2 zusammen mit CCMP (Counter-Mode/CBC-Mac Protocol)

Grundlegende Beschreibung

Kommunikation über WLAN erfolgt entweder als Punkt-zu-Punkt- oder als Mehrpunkt-Kommunikation. Die erste Variante dient z. B. der Überwindung größerer Distanzen durch den Einsatz zweier Richtantennen.

Bei der Mehrpunkt-Kommunikation werden ein oder mehrere sogenannte Access Points eingesetzt, die im Prinzip jeweils wie Zentralen (Verteiler) fungieren und die Datenströme mehrerer Clients koordinieren.

Diese Access Points können bei größeren Installationen über Kabel und geeignete Managed Switches eine Verbindung zu einem sogenannten RADIUS (Remote Authentication Dial-In User Service)-Server erhalten, um zwischen berechtigten und nicht berechtigten Sendern zu unterscheiden (Authentifizierung).

Eine Unterscheidung über die MAC-Adresse sollte nur den zum Zugang berechtigten Geräten vorbehalten bleiben, die sich nicht per RADIUS authentifizieren können. Dabei wird in einer Zugangsliste (Access Control Table) vom Managed Switch die MAC-Adresse eingetragen. Auf einem Access-Point bringt dies keine höhere Sicherheit, da per Funk eine MAC-Adresse unverschlüsselt verschickt wird und somit gefälscht werden kann. Die kleinste Einheit ist eine sogenannte Funkzelle, womit der Bereich gemeint ist, der von einem Sender (=Access Point) abgedeckt werden kann. Er umfasst ca. 30m im Gebäude und bis zu 300m im Freien. Mit speziellen Antennen können auch mehrere Kilometer überbrückt werden.

ISM-Frequenzbänder

In den meisten Fällen wird von den Herstellern ein sogenanntes **ISM-Band (Industrial, Scientific and Medical)** verwendet. Manchmal finden Sie auch die Abkürzung ISMO, wobei der letzte Buchstabe für den Begriff „Office“ (Büro) steht. Der Einsatz dieser Frequenzbänder bietet zwei Vorteile. Sie sind

- **gebührenfrei**
- **genehmigungsfrei**

Hierin liegt aber auch gleichzeitig der Nachteil. Sie werden von sehr vielen Herstellern für die unterschiedlichsten Zwecke genutzt, wie z. B. drahtlose Lautsprecher, elektronische Türöffnung bei Autos oder Garagen, und so ist die Gefahr, dass sich Geräte gegenseitig stören, relativ hoch.

Die für WLAN wichtigsten ISM-Bänder sind

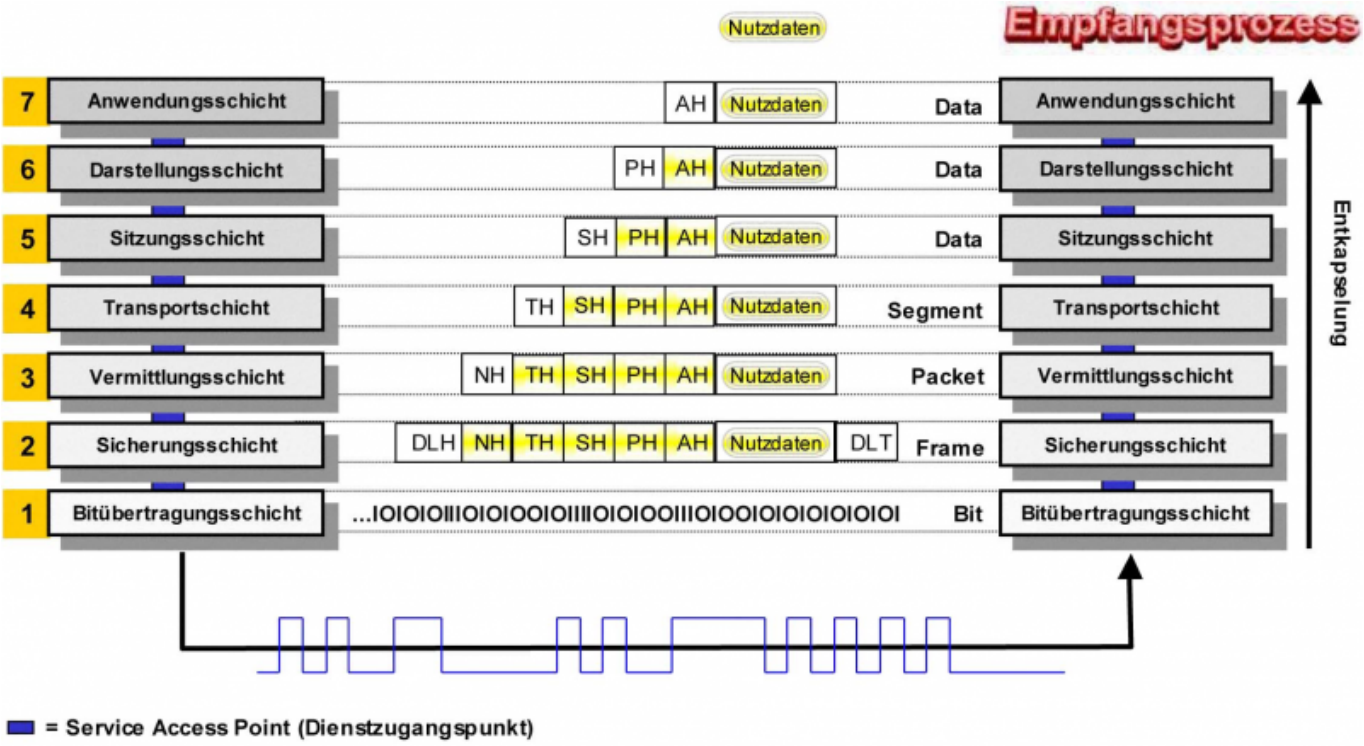
- das **2,4-GHz-Band** (2,3995 bis 2,4845 GHz) mit max. **13 überlappenden Kanälen** von 20 MHz Bandbreite; auch 40 MHz Bandbreite ist möglich, dann aber mit weit weniger nutzbaren Kanälen. Es sind Geschwindigkeiten bis zu 300MBit/s möglich.
- das **5-GHz-Band** (5,150 bis 5,350 GHz für Kanalnummer 36–64 und 5,470 bis 5,725 GHz für Kanalnummer 100–140) mit Kanälen von 20, 40, 80 oder 160 MHz Bandbreite, wobei **max. 19 Kanäle** bei 20 MHz Bandbreite nicht überlappend nutzbar sind. Beim Funken mit 40 MHz Bandbreite sind 2 dieser Kanäle gebündelt erforderlich, mit 80 MHz 4 Kanäle usw. Die **Reichweite ist geringer** als im 2,4 GHz-Band. Es sind Geschwindigkeiten bis zu 600MBit/s möglich.
- zukünftig das 60-GHz-Band (57 bis 66 GHz) mit vier 2000 MHz breiten Funkkanälen für kurze Distanzen

OSI - Schichtenmodell

Das OSI-7-Schichtenmodell ist ein Referenzmodell für herstellerunabhängige Kommunikationssysteme bzw. eine Design-Grundlage für Kommunikationsprotokolle und Computernetze. OSI steht für Open System Interconnection (Offenes System für Kommunikationsverbindungen) und wurde von der ISO (International Organization for Standardization), das ist die Internationale Organisation für Normung, als Grundlage für die Bildung von offenen Kommunikationsstandards entworfen. Bei allen ISO-Standards handelt es sich um Handlungsempfehlungen. Die Einhaltung einer ISO-Norm ist freiwillig. In der Regel wird die Einhaltung der ISO-Standards von verschiedenen Seiten, zum Beispiel Kooperationspartnern, Herstellern und Kunden, gefordert.

Das Modell

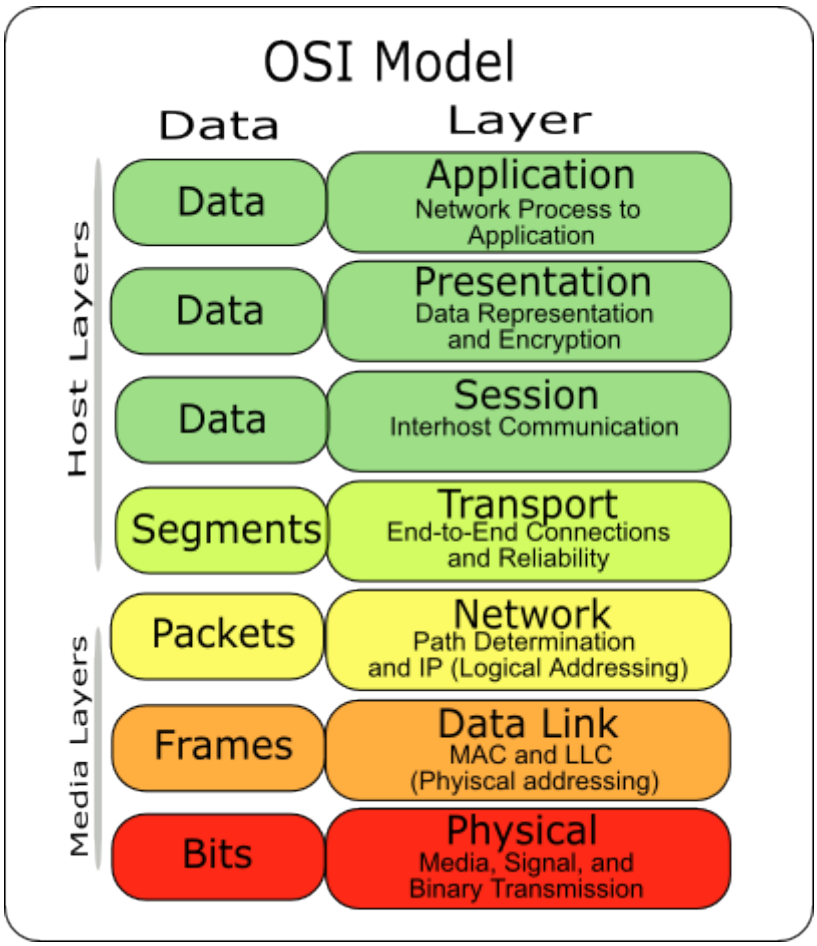
(Offenes System für Kommunikationsverbindungen) und wurde von der ISO (International Organization for Standardization), das ist die Internationale Organisation für Normung, als Grundlage für die Bildung von offenen Kommunikationsstandards entworfen. Bei allen ISO-Standards handelt es sich um Handlungsempfehlungen. Die Einhaltung einer ISO-Norm ist freiwillig. In der Regel wird die Einhaltung der ISO-Standards von verschiedenen Seiten, zum Beispiel Kooperationspartnern, Herstellern und Kunden, gefordert.



Die Schichten im Detail

Application Layer (Anwendungsschicht)	**Benutzerschnittstelle, Dienste, Anwendungen und Netzmanagement**
Schicht 7	Die Anwendungsschicht stellt Funktionen für die Anwendungen zur Verfügung. Diese Schicht stellt die Verbindung zu den unteren Schichten her. Auf dieser Ebene findet auch die Dateneingabe und -ausgabe statt.
Presentation Layer (Darstellungsschicht)	**Übersetzung, Verschlüsselung , Kompression in Standardformate**
Schicht 6	Die Darstellungsschicht setzt die Daten der Anwendungsebene in ein Zwischenformat um. Diese Schicht ist auch für Sicherheitsfragen zuständig. Durch sie werden Dienste zur Verschlüsselung von Daten bereitgestellt und gegebenenfalls Daten komprimiert.
Session Layer (Sitzungsschicht)	**Erstellung einer Verbindung, Freigabe von Verbindungen, Dialogsteuerung**
Schicht 5	Diese Schicht ermöglicht zwei Anwendungen auf verschiedenen Computern, eine gemeinsame Sitzung aufzubauen, damit zu arbeiten und sie zu beenden. Sie übernimmt ebenfalls die Dialogsteuerung zwischen den beiden Computern einer Sitzung und regelt, welcher der beiden wann und wie lange Daten überträgt.
Transport Layer (Transportschicht)	**Logische Ende-zu-Ende-Verbindungen (Transportkontrolle, Paketbildung)**
Schicht 4	Die Transportschicht stellt die zuverlässige Auslieferung der Nachrichten sicher und erkennt sowie behebt allfällige Fehler. Sie ordnet bei Bedarf auch die Nachrichten in Paketen neu, indem sie lange Nachrichten zur Datenübertragung in kleinere Pakete aufteilt. Am Ende des Weges stellt sie die kleinen Pakete wieder zur ursprünglichen Nachricht zusammen. Die empfangene Transportebene sendet auch eine Empfangs bestätigung.
Network Layer (Vermittlungsschicht)	**Routing (Internet), Datenflusskontrolle, Adressierung**

Application Layer (Anwendungsschicht)	**Benutzerschnittstelle, Dienste, Anwendungen und Netzmanagement**
Schicht 3	Die Vermittlungsschicht steuert die zeitliche und logische getrennte Kommunikation zwischen den Endgeräten, unabhängig vom Übertragungsmedium und der Topologie. Auf dieser Schicht erfolgt erstmals die logische Adressierung der Endgeräte. Die Adressierung ist eng mit dem Routing (Wegfindung vom Sender zum Empfänger) verbunden.
Data Link Layer (Sicherungsschicht)	**Logische Verbindungen mit Datenpaketen und elementare Fehlererkennungsmechanismen**
Schicht 2	Die Sicherungsschicht sorgt für eine zuverlässige und funktionierende Verbindung zwischen Endgerät und Übertragungsmedium. Zur Vermeidung von Übertragungsfehlern und Datenverlust enthält diese Schicht Funktionen zur Fehlererkennung, Fehlerbehebung und Datenflusskontrolle. Auf dieser Schicht findet auch die physikalische Adressierung von Datenpaketen statt.
Physical Layer (Physikalische Schicht)	**Maßnahmen und Verfahren zur Übertragung von Bitfolgen**
Schicht 1	Die Bitübertragungsschicht definiert die elektrische, mechanische und funktionale Schnittstelle zum Übertragungsmedium. Die Protokolle dieser Schicht unterscheiden sich nur nach dem eingesetzten Übertragungsmedium und -verfahren. Das Übertragungsmedium ist jedoch kein Bestandteil der Schicht 1.

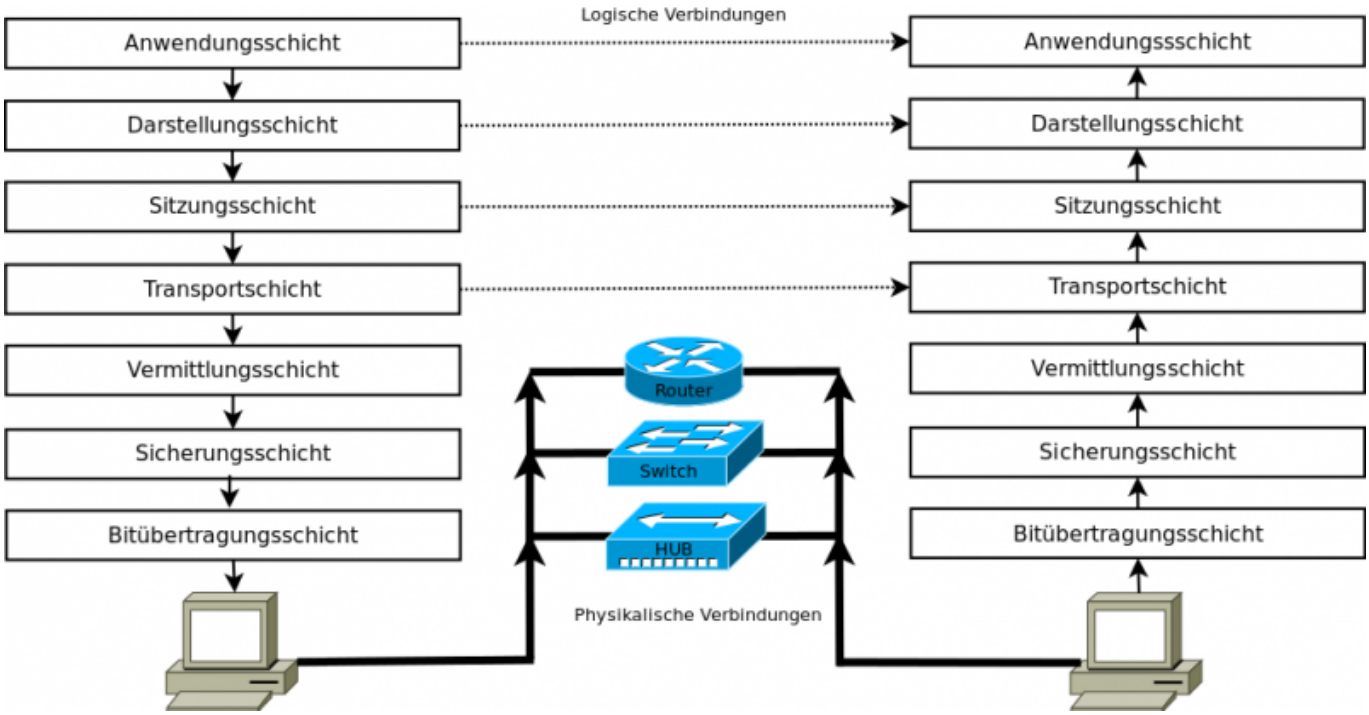


Protokolle

Protokolle sind eine **Sammlung von Regeln zur Kommunikation** auf einer bestimmten Schicht des OSI-Schichtenmodells. Die Protokolle einer Schicht sind zu den Protokollen der über- und untergeordneten Schichten weitestgehend transparent, so dass die Verhaltensweise eines Protokolls sich wie bei einer direkten Kommunikation mit dem Gegenstück auf der Gegenseite darstellt.

Die **Übergänge zwischen den Schichten sind Schnittstellen**, die von den Protokollen verstanden werden müssen. Weil manche Protokolle für ganz bestimmte Anwendungen entwickelt wurden, kommt es auch vor, dass sich **Protokolle über mehrere Schichten** erstrecken und mehrere Aufgaben abdecken. Dabei kommt es vor, dass in manchen Verbindungen einzelne Aufgaben in mehreren Schichten und somit mehrfach ausgeführt werden.

Anmerkung zur Skizze: End-zu-End-Verbindungen finden laut Definition erst ab Schicht 4 statt.



Das wichtigste Protokoll im Netzwerkverkehr, ist das **TCP & IP Protokoll** (Transmission Control Protocol & Internet Protocol), welche sich heute als Standard durchgesetzt haben.

Neben TCP und IP gibt es natürlich noch viele weitere Protokolle, wie in der obigen Abbildung zu sehen ist.

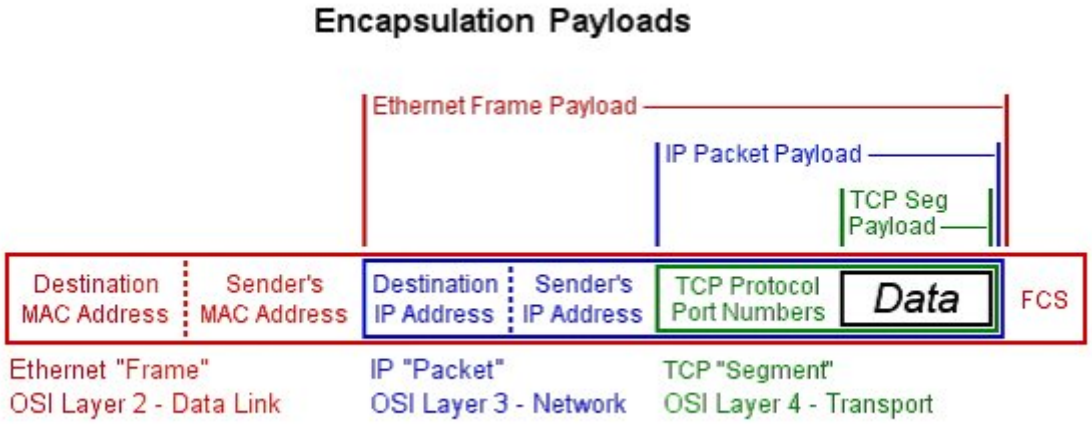
Die folgende Tabelle listet einige dieser Protokolle auf und ordnet sie ins obige Modell ein:

Protokoll	Schicht	Name	Beschreibung
FTP	5	File Transfer Protocol	Datenaustausch zwischen Rechnern
Telnet	5	Telecommunication Network Protocol	Terminalemulation zur Host-Kommunikation

Protokoll	Schicht	Name	Beschreibung
SMTP	5	Simple Mail Transfer Protocol	Versenden von EMails
HTTP	5	Hypertext Transfer Protocol	Übertragen von HTML-Seiten
POP	5	Post Office Protocol	Abrufen von E-Mails
TCP	4	Transmission Control Protocol	Aufbau logischer Verbindungen zwischen Applikationen
UDP	4	User Datagram Protocol	Verbindungsloses Übertragungsprotokoll. Es ist nicht so gesichert wie TCP dafür aber schneller
IP	3	Internet Protocol	Verbindungsloses Protokoll zur Paketlenkung und Paketvermittlung über IP-Adressen
IPSec	3	IP Secure	Erweitert das reguläre IP-Protokoll um ein Bündel von Sicherheitsmechanismen
ARP	3	Address Resolution Protocol	Dient dazu logische IP-Adressen physikalischen MAC-Adressen zuzuordnen

Datenkapselung

Unter der Datenkapselung versteht man den Prozess im OSI-Modell und im TCP/IP-Referenzmodell, der die zu versendenden Daten im Header (und ggf. Trailer) der jeweiligen Schichten ergänzt. Im OSI-Modell betrifft dies die Datagramme der Schichten 2 bis 4, die gekapselt (verpackt) werden.



OSI-Schichtenmodell

TCP/IP-Referenzmodell

Netzwerkkomponenten

In der Netzwerktechnik unterscheidet man zwischen aktiven und passiven Netzwerk-Komponenten. Während **aktive Netzwerk-Komponenten eine eigene Logik haben**, zählen die passiven Netzwerk-Komponenten zur fest installierten Netzwerk-Infrastruktur. In der Regel dienen Netzwerk-Komponenten zur Kopplung der Netzwerk-Stationen. Man spricht deshalb auch von Kopplungselementen.

Passive Netzwerk-Komponenten

- Patchkabel und Installationskabel
- Anschlussdose
- Steckverbinder
- Patchfeld / Patchpanel
- Netzwerk-Schrank / Patch-Schrank

Hinweis: Zu den passiven Netzwerk-Komponenten zählen die Bestandteile der Verkabelung. Diese ist im OSI-Schichtenmodell nicht definiert.

Aktive Netzwerk-Komponenten

In kleinen privaten Netzwerken, haben Netzwerk-Komponenten noch klare Bezeichnung, wie Switch oder Router. In großen Unternehmensnetzwerken ist die Benennung der Kopplungselemente nicht immer eindeutig.

- Netzwerkkarte
- Repeater
- Hub
- Bridge
- Switch
- Router
- Gateway
- Server

Netzwerkkarte

Eine Netzwerkkarte wird auch als Netzwerkadapter bezeichnet. Die englische Bezeichnung ist Network Interface Card (NIC). Eine Netzwerkkarte ermöglicht es, auf ein Netzwerk zuzugreifen und arbeitet auf der Bitübertragungsschicht (Schicht 1) und der Datensicherungsschicht (Schicht 2) des OSI-Schichtenmodells. Jede Netzwerkkarte hat eine Hardware-Adresse (Format: XX-XX-XX-XX-XX-XX), die es auf der Welt nur einmal gibt. Anhand dieser Adresse lässt sich eine Station auf der Bitübertragungsschicht adressieren.

Im Falle von Ethernet-Netzen besteht die **MAC-Adresse aus 48 Bit (sechs Bytes)**. Die Adressen werden in der Regel **hexadezimal** geschrieben. Üblich ist dabei eine **byteweise Schreibweise**,

wobei die einzelnen Bytes durch Bindestriche oder Doppelpunkte voneinander getrennt werden, z. B. 00-80-41-ae-fd-7e oder 00:80:41:ae:fd:7e. Seltener zu finden sind Angaben wie 008041aefd7e oder 0080.41ae.fd7

In den **ersten 24 Bits** (Bit 3 bis 24) wird eine von der IEEE vergebene **Herstellerkennung** (auch OUI – Organizationally Unique Identifier genannt) beschrieben, die weitgehend in einer Datenbank einsehbar sind[6]. Die **verbleibenden 24 Bits** (Bit 25 bis 48) werden **vom jeweiligen Hersteller** für jede Schnittstelle individuell **festgelegt**.

Repeater

Ein Repeater ist ein Kopplungselement, um die Übertragungsstrecke innerhalb von Netzwerken, zum Beispiel Ethernet, zu verlängern. Ein Repeater empfängt ein Signal und bereitet es neu auf. Danach sendet er es weiter. Auf diese Weise verlängert der Repeater die Übertragungsstrecke und räumliche Ausdehnung des Netzwerks. Im einfachsten Fall hat ein Repeater zwei Ports, die wechselweise als Ein- und Ausgang funktionieren (bidirektional).

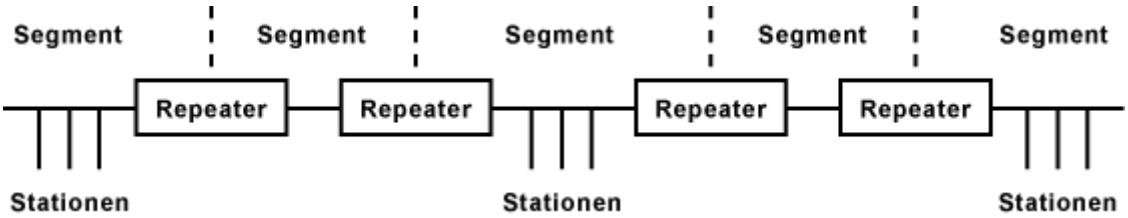
Repeater versteht man in der Regel als Verstärker von Übertragungsstrecken. Die weitere Beschreibung bezieht sich auf Repeater in kabelgebundenen Netzwerken, speziell in Ethernet-Netzwerken.

Ein Repeater arbeitet auf der Schicht 1, der Bitübertragungsschicht des OSI-Schichtenmodells. Der Repeater übernimmt keinerlei regulierende Funktion in einem Netzwerk. Er kann nur Signale empfangen und weiterleiten. Für angeschlossene Geräte ist nicht erkennbar, ob sie an einem Repeater angeschlossen sind. Er verhält sich völlig transparent.

Ein Repeater erweitert somit eine Kollisionsdomäne!!

Ein Repeater mit mehreren Ports wird auch als Hub (Multiport-Repeater) bezeichnet. Er kann mehrere Netzwerk-Segmente miteinander verbinden.

Die Repeater-Regel (5-4-3)



Um ein großes Netzwerk mit einer möglichst großen Reichweite aufzubauen, können mehrere Repeater hintereinandergeschaltet werden. Allerdings, nicht in beliebiger Anzahl. Der Grund liegt im Laufzeitverhalten und der Phasenverschiebung zwischen den Signalen an den Enden des Netzwerks. Deshalb gilt folgende Repeater-Regel:

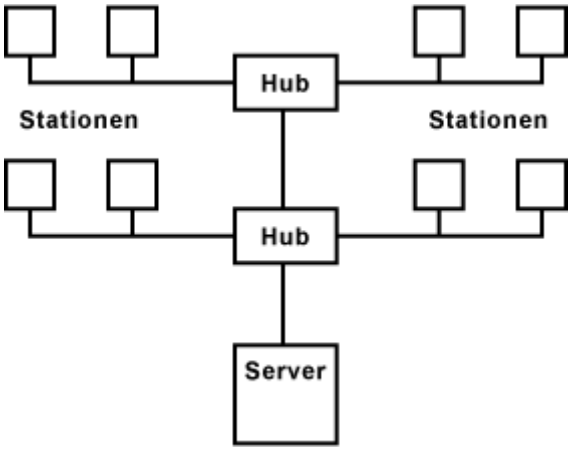
Es dürfen nicht mehr als fünf (5) Kabelsegmente verbunden werden. Dafür werden vier (4) Repeater eingesetzt. An nur drei (3) Segmenten dürfen Endstationen angeschlossen werden.

Diese **Repeater-Regel hat nur in den Ethernet-Netzwerken 10Base2 und 10BASE5** eine

Bedeutung. In Netzwerken, die mit Switches und Router aufgebaut sind, hat diese Repeater-Regel keine Bedeutung. Um die Nachteile von Repeatern in Ethernet-Netzwerken zu umgehen, werden generell Switches zur Kopplung der Hosts eingesetzt. In großen Netzwerken, insbesondere über unterschiedliche Übertragungssysteme hinweg, werden zusätzlich Router eingesetzt.

Hub

Ein Hub ist ein Kopplungselement, das mehrere Hosts in einem Netzwerk miteinander verbindet. In einem Ethernet-Netzwerk, das auf der Stern-Topologie basiert dient ein Hub als Verteiler für die Datenpakete. Hubs arbeiten auf der Bitübertragungsschicht (Schicht 1) des OSI-Schichtenmodells und sind damit auf die **reine Verteilfunktion** beschränkt. **Hubs erweitern somit die Kollisionsdomäne**. Ein Hub nimmt **ein Datenpaket entgegen und sendet es an alle anderen Ports** weiter. Das bedeutet, er **broadcastet**. Dadurch sind nicht nur **alle Ports**, sondern **auch alle Hosts belegt**. Sie bekommen alle Datenpakete zugeschickt, auch wenn sie nicht die Empfänger sind. Für die Hosts bedeutet das auch, dass sie nur dann senden können, wenn der Hub gerade keine Datenpakete sendet. Sonst kommt es zu Kollisionen.



Wenn die Anzahl der Anschlüsse an einem Hub für die Anzahl der Hosts nicht ausreicht, dann benötigt man noch einen zweiten Hub. Zwei Hubs werden über einen Uplink-Port eines der beiden Hubs oder mit einem Crossover-Kabel (Sende- und Empfangsleitungen sind gekreuzt) verbunden. Es gibt auch spezielle „stackable“ Hubs, die sich herstellerspezifisch mit Buskabeln kaskadieren lassen. Durch die Verbindung mehrerer Hubs lässt sich die Anzahl der möglichen Hosts im Netzwerk erhöhen. Allerdings ist die Anzahl der anschließbaren Hosts begrenzt. Hier gilt die Repeater-Regel.

Nachteile

- ineffizient
- unsicher (Jeder bekommt jede Nachricht)

Vorteile

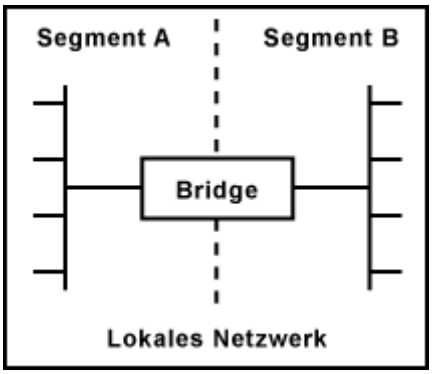
- zentraler Verteiler

Bridge

Eine Bridge ist ein Kopplungselement, das ein lokales Netzwerk in zwei Segmente aufteilt. Dabei

werden die Nachteile von Ethernet, die besonders bei großen Netzwerken auftreten ausgeglichen. Als Kopplungselement ist die Bridge eher untypisch. Man vermeidet die Einschränkungen durch Ethernet heute eher durch Switches.

Eine Bridge teilt eine Kollisionsdomäne!!

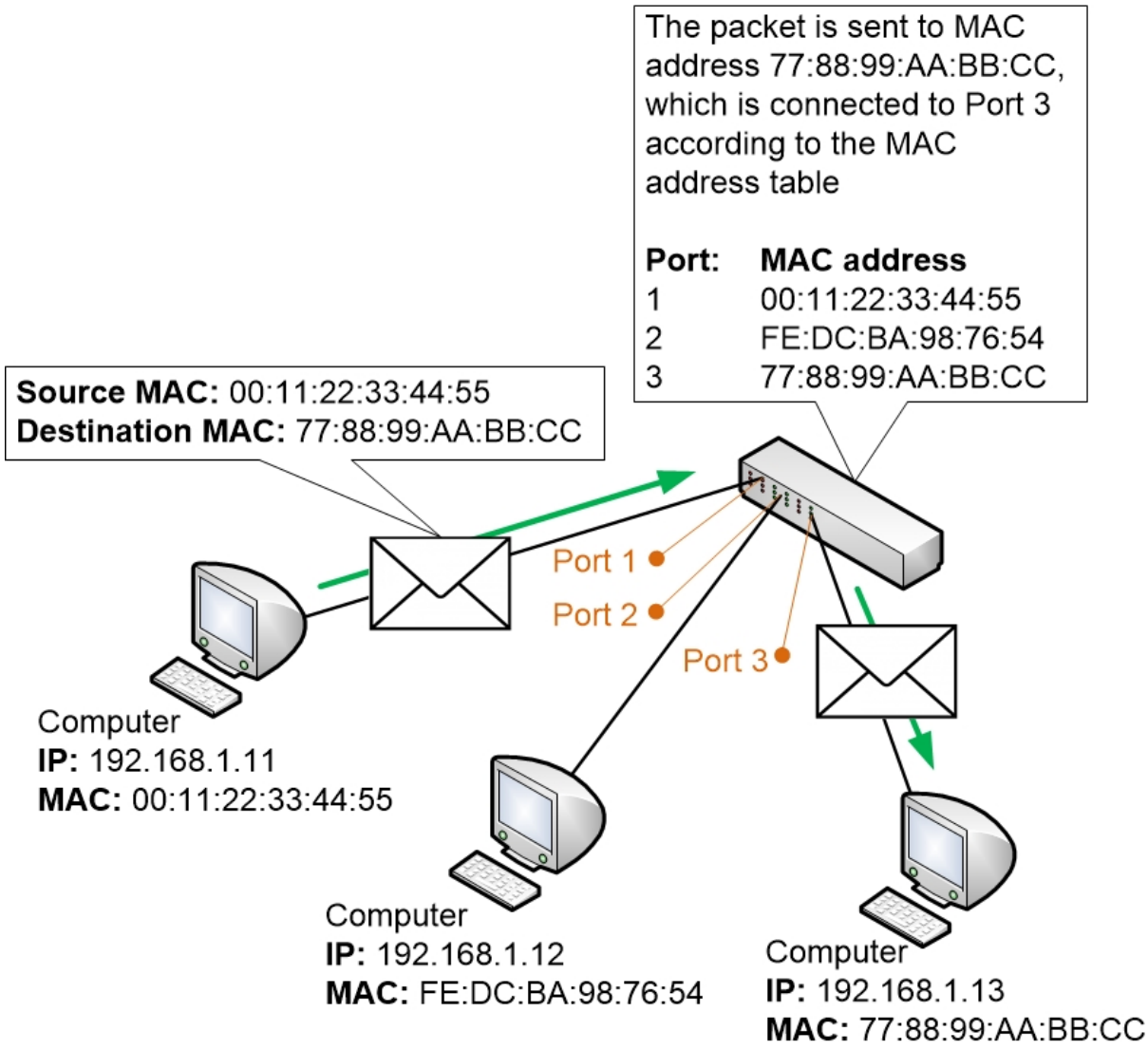


Switch

Ein Switch ist ein Kopplungselement, das mehrere Stationen in einem Netzwerk miteinander verbindet. In einem Ethernet-Netzwerk, das auf der Stern-Topologie basiert dient ein Switch als Verteiler für die Datenübertragung.

Die Funktion ist ähnlich einem Hub, mit dem Unterschied, das ein Switch direkte Verbindungen zwischen den angeschlossenen Geräten schalten kann, sofern ihm die Ports der Datenpaket-Empfänger bekannt sind. Somit kommunizieren wirklich nur jene miteinander, die auch miteinander reduzieren wollen. Wenn nicht, dann broadcastet der Switch die Datenpakete an alle Ports. Wenn die Antwortpakete von den Empfängern zurück kommen, dann merkt sich der Switch die MAC-Adressen der Datenpakete und den dazugehörigen Port und sendet die Datenpakete dann nur noch dorthin. Er baut also eine sogenannte MAC-Adressen-Tabelle auf:

Beispiel:



Während ein Hub die Bandbreite des Netzwerks limitiert, steht der Verbindung zwischen zwei Hosts, die volle Bandbreite der Ende-zu-Ende-Netzwerk-Verbindung zur Verfügung.

Ein Switch arbeitet auf der Sicherungsschicht (Schicht 2) de s OSI-Modells und arbeitet ähnlich wie eine Bridge. Daher haben sich bei den Herstellern auch solche Begriffe durchgesetzt, wie z. B. Bridging Switch oder Switching Bridge. Die verwendet man heute allerdings nicht mehr.

Switches unterscheidet man hinsichtlich ihrer Leistungsfähigkeit mit folgenden Eigenschaften:

- Anzahl der speicherbaren MAC-Adressen für die Quell- und Zielports
- Verfahren, wann ein empfangenes Datenpaket weitervermittelt wird (Switching-Verfahren)
- Latenz (Verzögerungszeit) der vermittelten Datenpakete

Ein Switch ist im Prinzip nichts anderes als ein intelligenter Hub, der sich merkt, über welchen Port welcher Host erreichbar ist. Teure Switches können zusätzlich auf der Schicht 3, der Vermittlungsschicht, des OSI-Schichtenmodells arbeiten (Layer-3-Switch oder Schicht-3-Switch). Sie sind in der Lage, die Datenpakete anhand der IP-Adresse an die Ziel-Ports weiterzuleiten. Im Gegensatz zu normalen Switche lassen sich auch ohne Router logische Abgrenzungen erreichen.

Switching-Verfahren

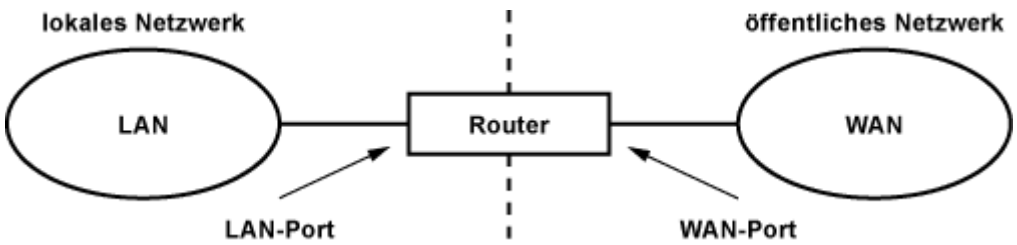
Switching-Verfahren	Beschreibung	Vorteile	Nachteile
Cut-Through	Der Switch leitet das Datenpaket sofort weiter, wenn er die Adresse des Ziels erhalten hat.	Die Latenz, die Verzögerungszeit, zwischen Empfangen und Weiterleiten ist äußerst gering.	Fehlerhafte Datenpakete werden nicht erkannt und trotzdem an den Empfänger weitergeleitet.
Store-and-Forward	Der Switch nimmt das gesamte Datenpaket in Empfang und speichert es in einem Puffer. Dort wird dann das Paket mit verschiedenen Filtern geprüft und bearbeitet. Erst danach wird das Paket an den Ziel-Port weitergeleitet.	Fehlerhafte Datenpakete können so im voraus aussortiert werden.	Die Speicherung und Prüfung der Datenpakete verursacht eine Verzögerung, abhängig von der Größe des Datenpaketes.
Fragment-Free	Der Switch empfängt die ersten 64 Byte des Daten-Paketes. Ist dieser Teil fehlerlos werden die Daten weitergeleitet. Die meisten Fehler und Kollisionen treten während den ersten 64 Byte auf. Dieses Verfahren wird trotz seiner effektiven Arbeitsweise selten genutzt.	sehr effizient	selten genutzt

Router

Ein Router verbindet mehrere Netzwerke mit unterschiedlichen Protokollen und Architekturen. Ein Router befindet sich häufig an den Außengrenzen eines Netzwerks, um es mit dem Internet oder einem anderen, größeren Netzwerk zu verbinden. Über die Routing-Tabelle entscheidet ein Router, welchen Weg ein Datenpaket nimmt. Es handelt sich dabei um ein dynamisches Verfahren, das Ausfälle und Engpässe ohne den Eingriff eines Administrators berücksichtigen kann. Ein Router hat mindestens zwei Netzwerkanschlüssen. Er arbeitet auf der Vermittlungsschicht (Schicht 3) des OSI-Schichtenmodells.

Die Aufgabe eines Routers ist ein komplexer Vorgang, der sich in 4 Schritte einteilen lässt:

- Ermittlung der verfügbaren Routen
- Auswahl der geeignetsten Route unter Berücksichtigung verschiedener Kriterien
- Herstellen einer physikalischen Verbindung zu anderen Netzwerken
- Anpassen der Datenpakete an die Übertragungstechnik (Fragmentierung)



Ein Router hat in der Regel zwei Anschlüsse. Einen für die LAN-Seite und einen für die WAN-Seite.

Häufig sind die Ports mit der Bezeichnung LAN und WAN gekennzeichnet. Manchmal gibt es Port-Beschriftungen, bei denen nicht immer eindeutig ist, um was es sich handelt. Mit LAN ist immer das lokale Netzwerk mit privaten IP-Adressen gemeint, während die WAN-Seite das öffentliche Netzwerk kennzeichnet.

Zusammenfassung

Hub, Switch & Router

Netzwerkklassen

Es wurden fünf verschiedene Netzwerkklassen festgelegt: Class A, B, C, D, E.
In der Praxis sind nur A, B, C von Bedeutung. (D und E werden nur für Testzwecke genutzt.)

Class-A-Netze:

Adresse beginnt mit einer binären 0, 7 Bit für Netzwerk-Adresse, 24 Bit für Host-Adresse. Damit gibt es weltweit 127 derartige Netzwerke, ein Class-A-Netz kann bis zu 16 Mio. Teilnehmer haben. Alle derartigen Netzadressen sind bereits belegt.

```
IP-Adressen von Class-A-Netzen
0.0.0.0 bis 127.255.255.255
```

Class-B-Netze:

Adresse beginnt mit der binären Ziffernkombination 10, 14 bit für Netzwerk-Adresse, 16 Bit für Host-Adresse. Damit gibt es weltweit 16384 derartige Netzwerke, ein Class-B-Netz kann bis zu 65536 Teilnehmer haben. Alle derartigen Netzadressen sind bereits belegt.

```
IP-Adressen von Class-B-Netzen
128.0.0.0 bis 191.255.255.255
```

Class-C-Netze:

Adresse beginnt mit der binären Ziffernkombination 110, 21 Bit für Netzwerk-Adresse, 8 Bit für Host-Adresse. Damit gibt es weltweit 2 Mio. derartige Netzwerke, ein Class-C-Netz kann bis zu 256 Teilnehmer haben. Neu zugeteilte Netzadressen sind heute immer vom Typ C. Es ist abzusehen, dass bereits in Kürze alle derartigen Adressen vergeben sein werden.

```
IP-Adressen von Class-C-Netzen
192.0.0.0 bis 223.255.255.255
```

Netzwerkmasken der verschiedenen Netze

Netzwerkmasken unterscheiden sich in der Länge des Netzwerk- (alle Bitstellen auf 1) und Hostanteils (alle Bitstellen auf 0) abhängig von der Netzwerkklasse

```
          1.Byte 2.Byte 3.Byte 4.Byte
Class A     255.     0.     0.     0
Class B     255.   255.     0.     0
Class C     255.   255.   255.     0
```

Netzwerkmasken stellen einen Filter dar, an dem Rechner entscheiden können, ob sie sich im selben (logischen) Netz befinden

Broadcast-Adresse

Die Broadcast-Adresse ergibt sich aus der IP-Adresse, bei der alle Bitstellen des Hostanteils auf 1 gesetzt sind. Sie bietet die Möglichkeit, Datenpakete an alle Rechner eines logischen Netzwerkes zu senden. Sie wird ermittelt, indem die Netzwerkadresse mit der invertierten Netzwerkmaske bitweise ODER-verknüpft wird.

```
           192.168.100.000 11000000 10101000 01100100 00000000
           000.000.000.255 00000000 00000000 00000000 11111111
Broadcast  192.168.100.255 11000000 10101000 01100100 11111111
```

Loopback-Adresse

Die Class-A-Netzwerkadresse 127 ist weltweit reserviert für das sogenannte local loopback; sie dient zu Testzwecken der Netzwerkschnittstelle des eigenen Rechners. Die IP-Adresse 127.0.0.1 ist standardmäßig dem Loopback-Interface jedes Rechners zugeordnet.

Alle an diese Adresse geschickten Datenpakete werden nicht nach außen ins Netzwerk gesendet, sondern an der Netzwerkschnittstelle reflektiert. Die Datenpakete erscheinen, als kämen sie aus einem angeschlossenem Netzwerk.

Vergabe der IP-Adressen

Private und öffentliche IP-Adressen

Da der IP-Adressbereich begrenzt ist, wurden sog. private IP-Adressen festgelegt, die im globalen Internet nicht bekannt werden:

```
  10.   0. 0. 0 –  10.255.255.255
  172. 16. 0. 0 – 172. 31.255.255
  192.168. 0. 0 – 192.168.255.255
  127.  0. 0. 1  (loopback-Adresse)
```

Es muss ein Network Address Translator (NAT) verwendet werden, um auf das globale Internet zugreifen zu können.

Vorteil: einfach, einen ISP (Internet Service Provider) zu wechseln, da nur die IP-Adresse nach außen verändert werden muss.

- am Client händisch eintragen (incl. Subnetmask und Gateway)
- mittels DHCP (dynamic host configuration protocol)
- ein DHCP-Server vergibt einem Client, der sich im Netzwerk befindet, automatisch eine IP-

Adresse für eine bestimmte Zeit (lease-time)

Domain Name Service (DNS)

DNS ist ein Protokoll, das die Zuordnung von Computernamen zu IP-Adressen regelt.
Systematischer Aufbau: **hostname.[subdomain].domain.topleveldomain** z.B.
hyper.mat.univie.ac.at

Zuordnung erfolgt über eigene Rechner im Internet (DNS-Server)

Verwaltung der Domains: **InterNIC** (.com, .net, .org, .int), **NIC.AT** (.at)

- Viele Tools rund um IP-Adressen

IP-Adressen

Damit die Wegewahl in Netzwerken und im Internet zur Übermittlung von Datenpaketen vom Senden zum Empfänger funktioniert, wird jedem Rechner im **Internet eine weltweit eindeutige (und einmalige) Adresse (IP-Adresse)** zugeordnet.

Für Rechner die ohne Router direkt mit dem Internet verbunden sind, heißt das, das deren IP-Adressen weltweit eindeutig sind.

Standardmäßig haben diese IP-Adressen eine Länge von **32 Bit (4 x 8Bit)**.

Diese werden aus Gründen der **Übersichtlichkeit in 4 Zahlen zu je 1 Byte** aufgeteilt. Meist werden die einzelnen Bytes durch einen Punkt getrennt und dezimal dargestellt.

Öffentliche IP-Adressen

Wenn man von öffentlichen Adressen spricht, meint man die IP-Adressen, die im Internet erreichbar sind. Die Zuweisung einer öffentlichen IP-Adresse erfolgt in der Regel durch einen Provider (z.B. A1, Telekom, ...) welche diese wiederum in Europa durch die Organisation RIPE-NCC bekommt.

Private IP-Adressen

Private IP-Adressen (abgekürzt Private IP) sind IP-Adressen, die von der **IANA** nicht im Internet vergeben sind. Sie wurden für die **private Nutzung aus dem öffentlichen Adressraum ausgespart**, damit sie ohne administrativen Mehraufwand (Registrierung der IP-Adressen) in lokalen Netzwerken genutzt werden können. Als die **IP-Adressen des Internet Protokolls v4 knapp** wurden und dadurch eine **bewusste Einsparung öffentlicher IP-Adressen** notwendig wurde, war es umso wichtiger, private IP-Adressen in lokalen Netzwerken zur Verfügung zu haben, die **beliebig oft bzw. in beliebigen Netzwerken genutzt** werden können.

Netzklasse: Anzahl Netze (ohne Subnetting)	**Netzadressbereich**	**Subnetzmaske**	**CIDR-Notation**	**Anzahl Adressen**
Klasse A: 1 privates Netz mit 16.777.216 Adressen	10.0.0.0 bis 10.255.255.255	255.0.0.0	10.0.0.0/8	2^{24} = 16.777.216
Klasse B: 16 private Netze mit jeweils 65.536 Adressen	172.16.0.0 bis 172.31.255.255	255.240.0.0	172.16.0.0/12	2^{20} = 1.048.576
Klasse C: 256 private Netze mit jeweils 256 Adressen	192.168.0.0 bis 192.168.255.255	255.255.0.0	192.168.0.0/16	2^{16} = 65.536

Loopback Adresse

Die Class-A-Netzwerkadresse 127 ist weltweit reserviert für das sogenannte local loopback; sie dient zu Testzwecken der Netzwerkschnittstelle des eigenen Rechners. Die IP-Adresse **127.0.0.1** ist standardmäßig dem Loopback-Interface jedes Rechners zugeordnet.

Alle an diese Adresse geschickten Datenpakete werden nicht nach außen ins Netzwerk gesendet, sondern an der Netzwerkschnittstelle reflektiert. Die Datenpakete erscheinen, als kämen sie aus einem angeschlossenem Netzwerk.

Vergleich IP-Adresse vs. Telefonnummer

Ähnlich wie bei einer Telefonnummer setzt sich eine IP-Adresse aus mehreren Segmenten zusammen. Eine Telefonnummer besteht aus einer Vorwahl und einer Teilnehmernummer. Führen Sie ein Ortsgespräch, so muss die Vorwahl nicht angegeben werden. Ähnlich ist es bei IP-Adressen.

Diese bestehen auch aus zwei Teilen, wobei die ID für Identifikation steht:

- **Netzwerk-ID** im vorderen linken Teil entspricht der Vorwahl und gibt das entsprechende IP-Subnetz an.
- **Host-ID** im hinteren rechten Teil kennzeichnet eine einzelne Netzwerkkarte und entspricht dem Teilnehmernummer im Ortsnetz.

Entsprechend können Rechner im selben Subnetz direkt miteinander kommunizieren. Dagegen erfordert Kommunikation zischen Subnetzen eine Vermittlungsstelle, einen Router (Standardgateway), wo alle nicht im selben Netz adressierten Pakete hingeschickt werden.

Um zu erkennen, wo die Netzwerk-ID endet und die Host-ID beginnt, muss zusätzlich zur IP-Adresse zwingend eine sogenannte Subnetzmaske mit angegeben werden.

Subnetzmaske

Eine Subnetzmaske ist ein Bitmuster, das (von links nach rechts) Teile der IP-Adresse „maskiert“, um den Übergang zwischen Netz-ID und Host-ID zu kennzeichnen. Binär betrachtet besteht eine Subnetzmaske aus einer Folge von Einsen, die ab einer bestimmten Stelle umschlägt in eine Folge von Nullen. Dieser Umschlagpunkt gibt an, wie viele Bits zur Netzwerk-ID (Einsen) und zur Host-ID (Nullen) gehören.

Ein Auszug aus den möglichen Subnetzmasken für ein Klasse C-Netz

Maske	Maske (kurze Schreibweise)	Anzahl Hosts pro Netz	Netze	Beispiel Klasse C-Netz (192.168.1.0)
255.255.255.0	/24	256	1	192.168.1.0 - 192.168.1.255
255.255.255.128	/25	128	2	192.168.1.0 - 192.168.1.127 192.168.1.128 - 192.168.1.255

Maske	Maske (kurze Schreibweise)	Anzahl Hosts pro Netz	Netze	Beispiel Klasse C-Netz (192.168.1.0)
255.255.255.192	/26	64	4	192.168.1.0 - 192.168.1.63 192.168.1.64 - 192.168.1.127 192.168.1.128 - 192.168.1.191 192.168.1.192 - 192.168.1.255
255.255.255.224	/27	32	8	192.168.1.0 - 192.168.1.31 192.168.1.32 - 192. 168.1.63 ... 192.168.192 - 192.168.1.223 192.168.1.224 - 192.168.1.255
255.255.255.240	/28	16	16	192.168.1.0 - 192.168.1.15 192.168.1.16 - 192.168.1.31 ... 192.168.1.224 - 192.168.1.239 192.168.1.240 - 192.168.1.255
255.255.255.248	/29	8	32	192.168.1.0 - 192.168.1.7 192.168.1.8 - 192.168.1.15 ... 192.168.1.240 - 192.168.1.247 192.168.1.248 - 192.168.1.255
255.255.255.252	/30	4	64	192.168.1.0 - 192.168.1.3 192.168.1.4 - 192.168.1.7 ... 192.168.1.248 - 192.168.1.251 192.168.1.252 - 192.168.1.255
255.255.255.254	/31	2	128	192.168.1.0 - 192.168.1.1 192.168.1.2 - 192.168.1.3 ... 192.168.1.252 - 192.168.1.253 192.168.1.254 - 192.168.1.255
255.255.255.255	/32	1	256	192.168.1.0 192.168.1.1 ... 192.168.1.254 192.168.1.255

Erklärung

Subnetting

Die Aufteilung eines zusammenhängenden Adressraums von IP-Adressen in mehrere kleinere Adressräume nennt man Subnetting. Ein Subnet, Subnetz bzw. Teilnetz ist ein physikalisches Segment eines Netzwerks, in dem IP-Adressen mit der gleichen Netzwerkadresse benutzt werden. Diese Teilnetze können über Routern miteinander verbunden werden und bilden dann ein großes zusammenhängendes Netzwerk.

Beispiel 1:

IP-Adresse: 192.168.128.17

```
Dezimale Darstellung:           192.        168.        128.          17
Bit-Darstellung:          1100 0000   1010 1000    1000 0000   0001 0001
Hex-Darstellung:                C0          A8          80          11
```

Subnetzmaske: 255.255.255.0

```
Dezimale Darstellung:           255.        255.        255.          0
Bit-Darstellung:           1111 1111   1111 1111   1111 1111  0000 0000
Hex-Darstellung:                FF          FF          FF          00
```

Verknüpft man nun die binäre IP-Adresse mit der Subnetzmaske mit einem Logischen AND, so bekommt man die Netz-ID (=Netzadresse).

```
    <----------- NETZ-ID ------------><- HOST-ID ->
    1100 0000   1010 1000   1000 0000   0001 0001    (IP-Adresse)
AND 1111 1111   1111 1111   1111 1111   0000 0000    (Subnetzmaske)
    ---------------------------------------------------------------
    1100 0000   1010 1000   1000 0000   0000 0000    (Netz-ID)
Netz-ID in Dezimaldarstellung:
 192.168.128
```

Verknüpft man nun die binäre IP-Adresse mit der negierten Subnetzmaske mit einem Logischen AND, so bekommt man die Host-ID.

```
    <----------- NETZ-ID ------------><- HOST-ID ->
    1100 0000   1010 1000   1000 0000   0001 0001    (IP-Adresse)
AND 0000 0000   0000 0000   0000 0000   1111 1111    (negierte Subnetzmaske)
    ------------------------------------------------------------------------
-
    0000 0000   0000 0000   0000 0000   0001 0001    (Host-ID)
Host-ID in Dezimaldarstellung:
 17
```

Das heißt im Netzwerk 192.168.128.0 stehen theoretisch 256 (0-255) Adressen zur Adressierung von Netzwerkgeräten zur Verfügung. Praktisch sind es aber nur 254 Adressen, da zwei Adressen

- die **Netzadresse** (= 1. Adresse im Netz –> 192.168.128.0)
- die **Broadcastadresse** (= letzte Adresse im Netz –> 192.168.128.255)

reserviert sind.

Beispiel 2:

```
IP-Adresse 130.94.122.195/27

              Dezimal          Binär                                          
Berechnung

IP Adresse    130.094.122.195  10000010 01011110 01111010 11000011            
ip-adresse
Netzmaske     255.255.255.224  11111111 11111111 11111111 11100000      AND   
netzmaske
------------------------------------------------------------------------------
----------
Netzwerkteil  130.094.122.192  10000010 01011110 01111010 11000000   =       
netzwerkanteil

IP Adresse    130.094.122.195  10000010 01011110 01111010 11000011            
ip-adresse
Netzmaske     255.255.255.224  00000000 00000000 00000000 00011111  AND(NOT   
netzmaske)
------------------------------------------------------------------------------
----------
Geräteteil                  3  00000000 00000000 00000000 00000011       =   
geräteteil
```

Bei einer Netzmaske mit 27 gesetzten Bits ergibt sich ein Netzwerkteil von 130.94.122.192. Es verbleiben 5 Bits und damit `2^5=32` Adressen für den Geräteteil. Hiervon werden noch je 1 Adresse für das Netzwerk selbst und für den Broadcast benötigt, sodass 30 Adressen für Geräte zur Verfügung stehen.

Beispiel 3:

```
IP-Adresse 130.94.122.117/28

              Dezimal          Binär                                          
Berechnung

IP Adresse   130.094.122.117   10000010 01011110 01111010 01110101            
ip-adresse
Netzmaske    255.255.255.240   11111111 11111111 11111111 11110000      AND   
netzmaske
------------------------------------------------------------------------------
----------
Netzwerkteil 130.094.122.112   10000010 01011110 01111010 01110000   =       
netzwerkanteil
```

```
IP Adresse   130.094.122.117   10000010 01011110 01111010 01110101
ip-adresse
Netzmaske    255.255.255.240   00000000 00000000 00000000 00001111  AND(NOT
netzmaske)
-------------------------------------------------------------------------
----------
Geräteteil     0.  0.  0.  5   00000000 00000000 00000000 00000101       =
geräteteil
```

Bei einer Netzmaske mit 28 gesetzten Bits ergibt sich ein Netzwerkteil von 130.94.122.112. Es verbleiben 4 Bits und damit `2^4=16` Adressen für den Geräteteil. Hiervon werden noch je 1 Adresse für das Netzwerk selbst und für den Broadcast benötigt, sodass 14 Adressen für Geräte zur Verfügung stehen.

- Zusammenfassung zu Subnetzmasken

Übungen IP-Adressierung

Bsp 1

Befinden sich 192.168.0.93/27 und 192.168.0.97/27 im gleichen Netzwerk-Segment?

Lösung

```
1) NW-ID: 192.168.0.64
2) NW-ID: 192.168.0.96
=> Nein
```

Bsp 2

Wie lauten die Neztwerkadressen des ersten/letzten Hosts des Netzwerkes 192.168.0.96/27 und die Broadcastadresse?

Lösung

```
1. Host: 192.168.0.97
letzter Host: 192.168.0.126
BC: 192.168.0.127
```

Bsp 3

Das Netz 195.1.31.0 soll in 30 Subnetze aufgeteilt werden. Bestimme die Subnetzmaske, und die Anzahl Adressen pro Subnet!

Lösung

```
Subnetzmaske: 255.255.255.248
Adressen: 6
```

Bsp 4

Das Netz 10.0.0.0 soll in 200 Subnetze aufgeteilt werden. Bestimme die Subnetzmaske, und die Anzahl Adressen pro Subnet!

Lösung

```
Subnetzmaske: 255.255.0.0
```

```
Adressen: 65534
```

Bsp 5

Das Netz 192.168.1.0 soll in Subnetze mit je 18 Host-Adressen aufgeteilt werden. Bestimme die Subnetzmaske, und die Anzahl Subnetze!

Lösung

```
Subnetzmaske: 255.255.255.224
Subnetze: 8
```

Bsp 6

Das Netz 192.168.100.0 soll in Subnetze mit je 5 Host-Adressen aufgeteilt werden. Bestimmen Sie die Subnetzmaske, und die Anzahl Subnetze!

Lösung

```
Subnetzmaske: 255.255.255.248
Subnetze: 32
```

Bsp 7

Bestimme Netz- und Broadcastadresse des Subnets, in dem die Adresse 195.1.31.135 mit Netzmaske 255.255.255.128 liegt!

Lösung

```
NW-ID: 195.1.31.128
BC: 195.1.31.255
```

Bsp 8

Bestimme Netz- und Broadcastadresse des Subnets in dem die Adresse 195.1.31.135 mit Netzmaske 255.255.255.192 liegt!

Lösung

```
NW-ID: 195.1.31.128/26
BC: 195.1.31.191
```

Bsp 9

Bestimme Netz- und Broadcastadresse des Subnets in dem die Adresse 15.3.128.222 mit Netzmaske 255.255.255.240 liegt!

Lösung

```
NW-ID: 15.3.128.208
Broadcast: 15.3.128.223
```

Bsp 10

Gib den Bereich der Rechneradressen an, den das Teilnetz Nummer drei (132.45.96.0/19) hat.

Lösung

```
132.45.96.1
bis
132.45.127.254
```

Bsp 11

Berechne Netz-, Broadcast und Hostadressen des 0.,1.,15. und 31 Subnetz des Netzes 192.168.1.0/29

Lösung

```
0. Netz:
* NW-ID: 192.168.1.0
* BC: 192.168.1.7
* Hosts: 192.168.1.1-6
```

```
1. Netz:
* NW-ID: 192.168.1.8
* BC: 192.168.1.15
* Hosts: 192.168.1.9-14
```

```
15. Netz:
* NW-ID: 192.168.1.120
* BC: 192.168.1.127
* Hosts: 192.168.1.121-126
```

```
31. Netz:
* NW-ID: 192.168.1.248
* BC: 192.168.1.255
* Hosts: 192.168.1.249-254
```

Bsp 12

Vervollständige folgende Tabelle:

IP-Adresse	10.1.1.1	Netzwerkadresse des Subnet	. . .
Netmask	255.255.0.0	Broadcastadresse des Subnet	. . .
Anzahl Host pro Subnet		Anzahl Subnets in diesem Netz	

Lösung

IP-Adresse	10.1.1.1	Netzwerkadresse des Subnet	10.1.0.0
Netmask	255.255.0.0	Broadcastadresse des Subnet	10.1.255.255
Anzahl Host pro Subnet	65536	Anzahl Subnets in diesem Netz	256

Bsp 13

Vervollständige folgende Tabelle:

IP-Adresse	10.1.1.1/24	Netzwerkadresse des Subnet	. . .
Netmask	255.255.255.0	Broadcastadresse des Subnet	. . .
Anzahl Host pro Subnet		Anzahl Subnets in diesem Netz	

Lösung

IP-Adresse	10.1.1.1/24	Netzwerkadresse des Subnet	10.1.1.0
Netmask	255.255.255.0	Broadcastadresse des Subnet	10.1.1.255
Anzahl Host pro Subnet	254	Anzahl Subnets in diesem Netz	65536

IP-Routing

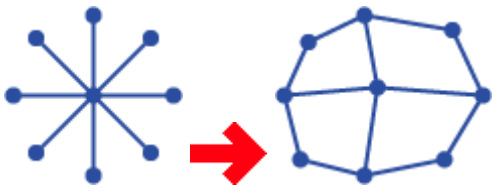
Das **Internet Protocol (IP)** ist ein **routingfähiges Protokoll** und sorgt dafür, dass **Datenpakete über Netzgrenzen** hinweg einen Weg zu anderen Hosts finden. Es kann die Daten über jede Art von physikalischer Verbindung oder Übertragungssystem vermitteln. Der hohen Flexibilität steht ein hohes Maß an Komplexität bei der Wegfindung vom Sender zum Empfänger gegenüber. Der **Vorgang der Wegfindung wird Routing** genannt

Wozu Routing?

Das grundlegende Verbindungselement in einem Ethernet-Netzwerk ist der Hub oder Switch. Daran sind alle Netzwerk-Teilnehmer angeschlossen. Wenn ein Host Datenpakete verschickt, dann werden die Pakete im Hub an alle Stationen verschickt und von diesen angenommen. Jedoch verarbeitet nur der adressierte Host die Pakete weiter. Das bedeutet aber auch, dass sich alle Hosts die Gesamtbandbreite dieses Hubs teilen müssen.

Um die Nachteile von Ethernet in Verbindung mit CSMA/CD auszuschließen, wählt man als Kopplungselement einen Switch und nutzt Fast Ethernet (kein CSMA/CD mehr). Der Switch merkt sich die Hardware-Adressen (MAC-Adressen) der Stationen und leitet die Ethernet-Pakete nur an den Port, hinter dem sich die Station befindet. Ist einem Switch die Hardware-Adresse nicht bekannt, leitet er das Datenpaket an alle seine Ports weiter (Broadcast) und funktioniert in diesem Augenblick wie ein Hub. Neben der begrenzten Speichergröße des Switches machen sich viele unbekannte Hardware-Adressen negativ auf die Performance eines Netzwerks bemerkbar.

Daher eignet sich zum **Verbinden großer Netzwerke** weder ein Hub noch ein Switch. Aus diesem Grund wird ein Netzwerk **durch Router** und IP-Adressen in **logische Segmente bzw. Subnetze** unterteilt. Die Adressierung durch das Internet Protocol ist so konzipiert, dass der Netzwerkverkehr innerhalb der Subnetze bleibt und erst dann das Netzwerk verlässt, wenn das Ziel in einem anderen Netzwerk liegt.



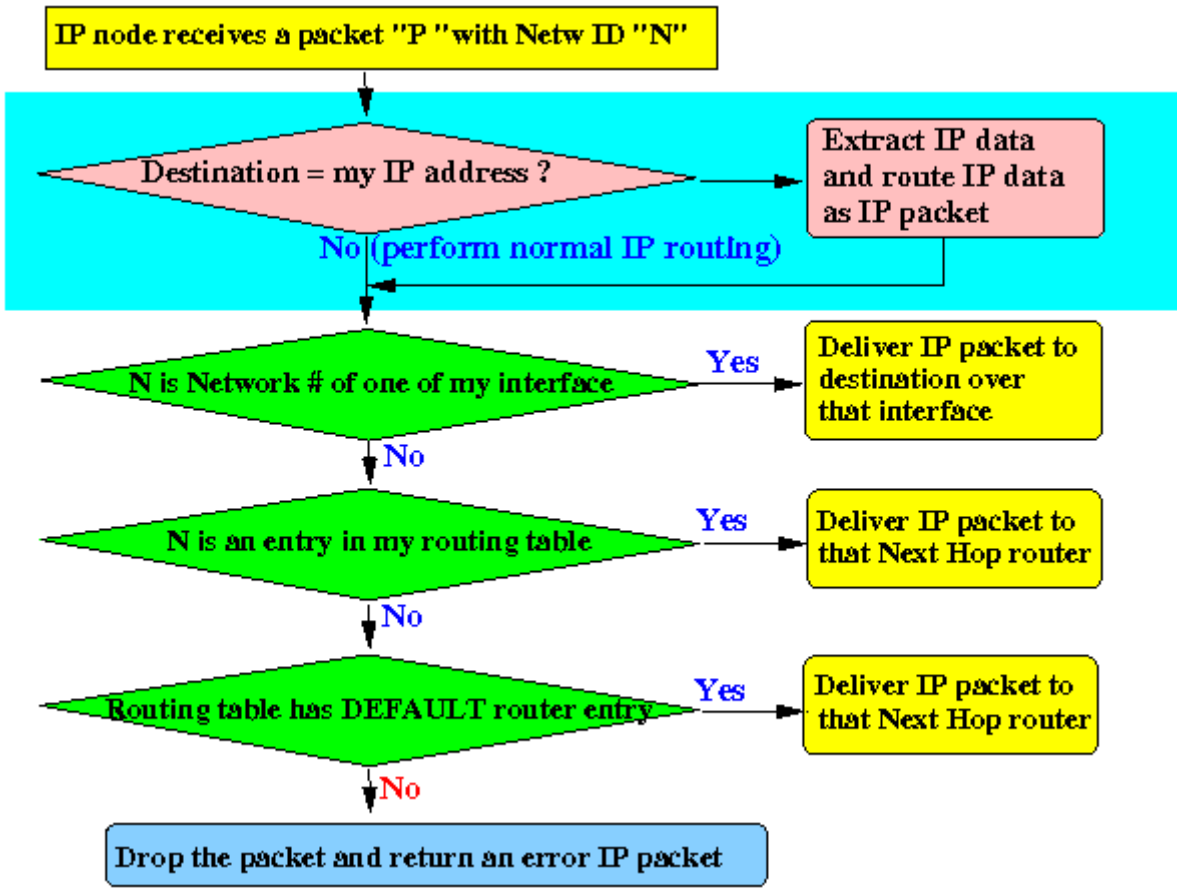
Insbesondere folgende **Probleme** in einem Ethernet-Netzwerk machen **IP-Routing notwendig**:

- **Vermeidung von Kollisionen und Broadcasts durch Begrenzung der Kollisions- und Broadcastdomäne**
- **Routing über unterschiedliche Netzarchitekturen und Übertragungssysteme**
- **Paket-Filter und Firewall**
- **Routing über Backup-Verbindungen bei Netzausfall**

IP-Routing-Algorithmus

Der IP-Routing-Algorithmus gilt nicht nur für IP-Router, sondern für alle Host, die IP-Datenpakete

empfangen können. Die empfangenen Datenpakete durchlaufen diesen Algorithmus bis das Datenpaket zugeordnet oder weitergeleitet werden kann.



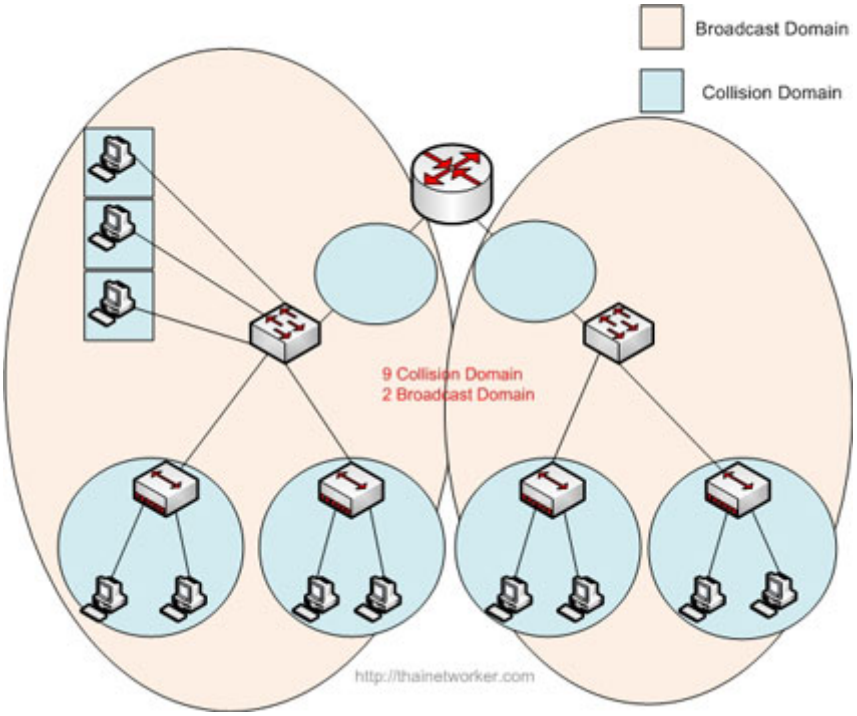
- Prüfe, ob das Datenpaket mir gehört?
  - Wenn ja, dann hat das Datenpaket sein Ziel erreicht und kann verarbeitet werden.
  - Wenn nein, prüfe ob das Datenpaket in mein Subnetz gehört?
    - Wenn ja, schick es in das eigene Subnetz weiter!
    - Wenn nein, prüfe ob die Route zum Empfänger bekannt ist?
      - Wenn ja, schicke es über die bekannte Route zum nächsten Router!
      - Wenn nein, prüfe ob es ein Standard-Gateway gibt?
        - Wenn ja, schick das Paket zum Standard-Gateway!
        - Wenn nein, schreibe eine Fehlermeldung und verwirf das Datenpaket!

Broadcastdomäne

Eine Broadcast-Domäne ist ein logischer Verbund von Netzwerkgeräten in einem lokalen Netzwerk, der sich dadurch auszeichnet, dass ein **Broadcast alle Domänenteilnehmer erreicht**.

Ein lokales Netzwerk auf der 2. Schicht des OSI-Modells (Sicherungsschicht) besteht durch seine Hubs, Switches und/oder Bridges aus einer Broadcast-Domäne. Erst durch die Unterteilung in VLANs oder durch den Einsatz von Routern, die auf Schicht 3 arbeiten, wird die Broadcast-Domäne aufgeteilt.

Eine Broadcast-Domäne besteht aus einer oder mehreren Kollisionsdomänen.



Netzwerkbefehle

Windows-Kommandos

Wichtige Netzwerkbefehle unter Windows XP

- ipconfig
  - /all ... detaillierte Informationen über Netzwerk-Konfiguration
  - /renew ... erneuert IP-Adressen
  - /release ... gibt IP-Adressen frei
- ping (sendet Datenpakete zu Rechner)
  - ping IP-Adresse
  - ping hostname
  - -n Anzahl ... Anzahl der Pakete
- tracert (Route zu Rechner)
- nslookup
  - nslookup (DNS-Abfragen)
  - nslookup IP-Adresse
  - nslookup Domain-Name

Linux-Kommandos

- ifconfig

Zeigt Informationen zu Netzwerk-Interfaces

- ping

```
ping -c 4 www.example.com
```

Pingt 4 mal www.example.com

- tracepath

```
tracepath www.example.com
```

Zeigt Hops zum Host an (benötigt eventuell SU-Rechte)

```
sudo tracepath www.example.com
```

- nslookup

```
nslookup www.example.com
```

Überprüft DNS-Eintrag

weitere hilfreiche Befehle:

- nmap (scannt („mappt“) das Netzwerk, führt Portscans aus und findet die Software eines fremden PCs heraus)
- mtr (kombiniert tracert (ohne su-Rechte) und ping, anschauliche Darstellung)

Fragen - Aufgaben - Arbeitsaufträge

Arbeitsaufträge:

1. Gib deiner/m Nachbarn/in die IP-Adresse deines Computers und notiere dir die IP-Adresse seines/ihres Computers.
2. Versuche deinen Nachbarcomputer anzupingen. Wie lange brauchte das Datenpaket zu diesem Rechner?
3. Gib deine IP-Adresse frei und versuche den Nachbarn anzupingen bzw. dich anpingen zu lassen. Erneuere anschließend deine IP-Adresse und überprüfe, wie sie nun lautet.
4. Versuche deinen Rechner (mit deiner IP-Adresse und der Loopback-Adresse) anzupingen.
5. Finde heraus, welche IP-Adresse der Domain-Name mail.bgamstetten.ac.at hat.
6. Finde einen Rechner im Internet, bis zu welchem ein Datenpaket sehr lange dauert und schicke zu diesem Rechner 20 Datenpakete hintereinander.
7. Finde heraus, welcher Domain-Name zu folgender IP-Adresse gehört 131.130.250.250
8. Lasse dir die Route zu www.yahoo.com anzeigen. Wie viele Rechner liegen zwischen dir und dem Webserver von www.yahoo.com ?
9. Versuche mit dem Internetexplorer die Website von www.google.at aufzurufen, indem du die IP-Adresse von Google in der Browserzeile eingibst.
10. Suche im Internet die Homepage von der australischen Regierung. Welche IP-Adresse hat der Rechner, auf dem die Homepage liegt? Wie viele Rechner liegen zwischen dir und diesem Rechner?
11. Finde einen Rechner im Internet, bis zu dem möglichst viele Hops dazwischen sind (Wer findet die meisten?).

Protokolle

Protokolle sind eine **Sammlung von Regeln zur Kommunikation** auf einer bestimmten Schicht des OSI-Schichtenmodells. Die Protokolle einer Schicht sind zu den Protokollen der über- und untergeordneten Schichten weitestgehend transparent, so dass die Verhaltensweise eines Protokolls sich wie bei einer direkten Kommunikation mit dem Gegenstück auf der Gegenseite darstellt.

Die **Übergänge zwischen den Schichten sind Schnittstellen**, die von den Protokollen verstanden werden müssen. Weil manche Protokolle für ganz bestimmte Anwendungen entwickelt wurden, kommt es auch vor, dass sich **Protokolle über mehrere Schichten** erstrecken und mehrere Aufgaben abdecken. Dabei kommt es vor, dass in manchen Verbindungen einzelne Aufgaben in mehreren Schichten und somit mehrfach ausgeführt werden.

Protokoll-Stack

Da sich ein einzelnes Protokoll immer nur um eine Teilaufgabe im Rahmen der Kommunikation kümmert, werden mehrere Protokolle zu Protokollsammlungen oder Protokollfamilien, den sogenannten Protokoll-Stacks, zusammengefasst. Die wichigsten Einzel-Protokolle werden dann oft stellvertertretend als Bezeichnung des gesamten Protokoll-Stapels genutzt.

Kommunikation zwischen Netzwerkkomponenten funktioniert nur dann, wenn sie denselben Protokoll-Stack benutzen oder wenn Geräte eingesetzt werden, die zwischen verschiedenen Stacks vermitteln können.

Portnummern (ab Layer 5)

Um die einzelnen Dienste (Protokolle), die bei einem Rechner über dieselbe IP-Adresse ausgeführt werden, voneinander zu differenzieren, wurden Portnummern eingeführt, um bei einer Anfrage deutlich zu machen, welcher Dienst gemeint ist.
Diese Portnummern, die im TCP- oder UDP Header angegeben werden, sind weltweit eindeutig festgelegt und können z.B. auf der Homepage der IANA eingesehen werden.
Allgemein lässt sich also sagen, dass die **IP-Adresse** den Rechner, **und** Die **Port-Nummer** den Dienst auf dem jeweiligen Rechner angibt. Diese beiden Informationen zusammen werden als **Socket** bezeichnet.

Wichtige Protokolle

OSI-LAYER	PROTOKOLL (Port)
5-7	DHCP (67+68/UDP)
	DNS (53/UDP)
	HTTP (80/TCP)
	HTTPS (443/TCP)
	FTP (20+21/TCP)
	SSH (22/TCP)
	SMTP (465 und 587 oder 25/TCP)
	POP3 (995 oder 110/TCP)
	IMAP (993 oder 143/TCP)
4	TCP
	UDP
3	IPv4
	IPv6
	NAT
	ICMP
2	Ethernet 802.3 , Wireless 802.x, MAC, ISDN,...
1	

Ethernet

Erklärung

Ethernet ist eine Technik, die **Software (Protokolle usw.) und Hardware (Kabel, Verteiler, Netzwerkkarten usw.) für kabelgebundene Datennetze spezifiziert**, welche ursprünglich für lokale Datennetze (LANs) gedacht war und daher auch als LAN-Technik bezeichnet wird. Sie ermöglicht den Datenaustausch in Form von Datenframes zwischen den in einem lokalen Netz (LAN) angeschlossenen Geräten (Computer, Drucker und dergleichen). Derzeit sind Übertragungsraten von 1, 10, 100 Megabit/s (Fast Ethernet), 1000 Megabit/s (Gigabit-Ethernet), 2,5, 5, 10, 40, 50, 100, 200 und 400 Gigabit/s spezifiziert. In seiner ursprünglichen Form erstreckt sich das LAN dabei nur über ein Gebäude; Ethernet-Varianten über Glasfaser haben eine Reichweite von bis zu 70 km.

Die Ethernet-Protokolle umfassen Festlegungen für Kabeltypen und Stecker sowie für Übertragungsformen (Signale auf der Bitübertragungsschicht, Paketformate). Im OSI-Modell ist mit Ethernet sowohl die **physische Schicht (OSI Layer 1)** als auch die **Data-Link-Schicht (OSI Layer 2)** festgelegt.

Ethernet basiert auf der Idee, dass die Teilnehmer eines LANs Nachrichten durch Hochfrequenz übertragen, allerdings **nur innerhalb eines gemeinsamen Leitungsnetzes**. Jede Netzwerkschnittstelle hat **einen global eindeutigen 48-Bit-Schlüssel**, der als **MAC-Adresse** (=Media-Access-Control-Adress) bezeichnet wird. Das stellt sicher, dass alle Systeme in einem Ethernet unterschiedliche Adressen haben.

Ethernet Frame

CSMA/CD

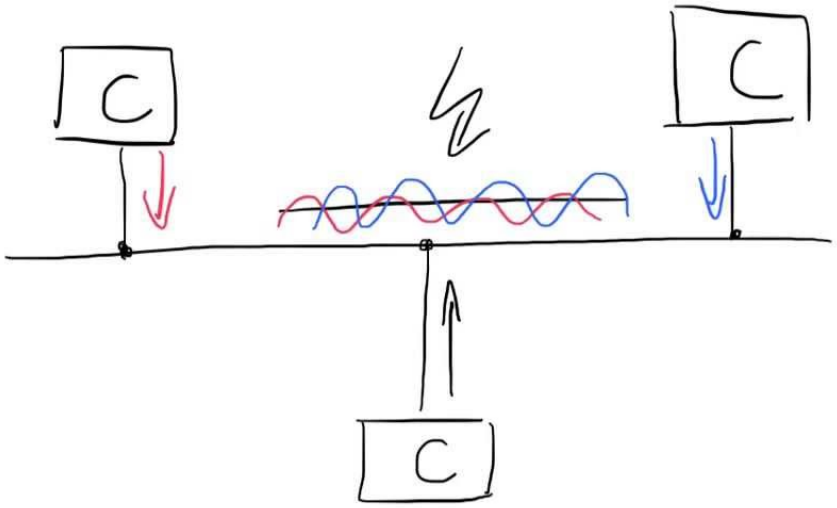
CSMA/CD (Carrier Sense Multiple Access with Collision Detection) ist das Zugriffsverfahren des Ethernets. Die Grundidee dabei ist, dass jede Station zu senden beginnen kann, wann sie will. Die einzelnen Stationen haben jederzeit und konkurrierend Zugang (Multiple Access) zum gemeinsamen Übertragungsmedium. Das grundlegende Motto könnte damit lauten:

Jeder darf, wann er will

Eingesetzt wird dieses Verfahren bei logischen Bus-Topologien. Dabei ist egal, ob physikalisch eine Bus- oder eine Stern-Topologie vorliegt.

Vorgehen

Durch Regelungen wird versucht, das Risiko zu minimieren, dass zwei Stationen ungünstigerweise gleichzeitig zu senden beginnen und somit die Signale auf dem Übertragungsweg zerstört/gestört werden **(=Kollision)**.



1) Leitung prüfen (Carrier Sense)

Der erste Teil der Abkürzung steht für Kollisionsverhinderung. Dabei wird vor einer geplanten Sendung das Übertragungsmedium abgehört, ob dieses frei ist. Wenn das Medium frei ist, wird gesendet. **–> LISTEN BEFORE TALKING**

2) Erkennen von Kollisionen (Collision Detection) Kommt es trotzdem zu einer Kollision, weil zwei Stationen gleichzeitig zu senden beginnen (**Multiple Access**). dann muss diese Kollision erkannt (**Collision Detection**) und reagiert werden. Dabei müssen alle Stationen immer am Medium horchen, ob eine Kollision auftritt. Ist dies der Fall, so sendet die erste Station, die eine Kollision erkennt, ein sogenanntes JAM-Signal aus. Jede Station, die das JAM-Signal registriert, stoppt unmittelbar das Senden von Daten. Nach einer zufälligen Zeitspanne, wird das Medium wieder überprüft und anschließend wieder begonnen zu senden.

Video

Frage: Wie kann bei einer physikalischen Stern-Topologie eine Kollision auftreten?

Vorteile

- Jeder kann zu jeder Zeit senden

Nachteile

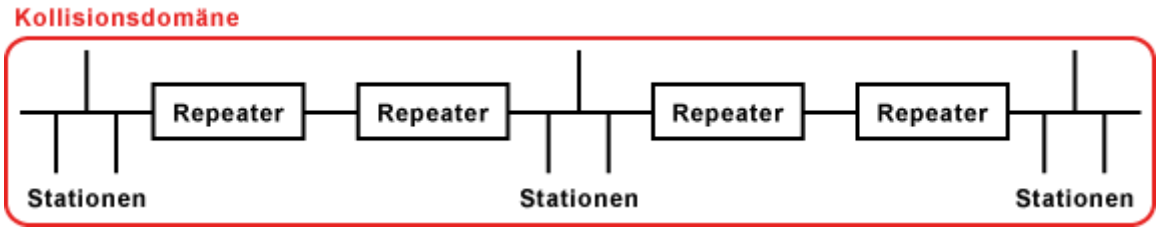
- Je mehr Stationen in einer Kollisionsdomäne, desto häufiger treten Kollisionen auf
- Der Zeitpunkt einer Sendung ist zufällig
- Das Verfahren ist ungeeignet für zeitkritischen Anwendungen

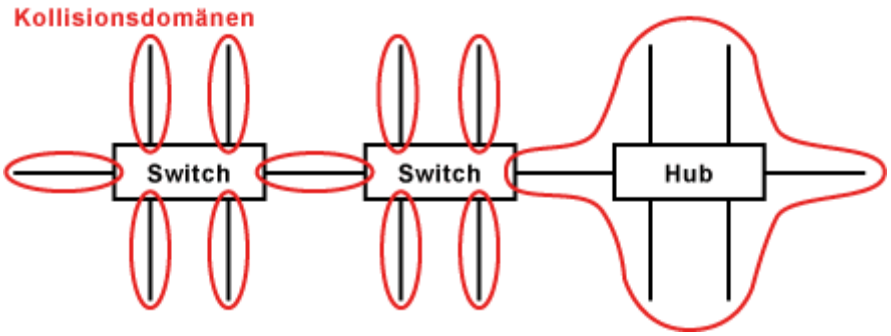
Kollisionsdomäne

Mit dem Begriff Kollisionsdomäne wird in einem Computernetz ein Teilbereich aus Teilnehmerstationen in derselben OSI-Modell-Schicht 1 bezeichnet. Eine Kollisionsdomäne umfasst alle Netzwerkgeräte, die um den Zugriff auf ein gemeinsames Übertragungsmedium konkurrieren. Das Übertragungsmedium ist daher eine zwischen allen Netzstationen geteilte Ressource. Grundlegende Vorstellung dabei ist, dass alle Netzwerkteilnehmer die Chance zur gleichberechtigten Nutzung des Netzwerkes besitzen.

Bei einem gemeinsamen Medium kann zu einer bestimmten Zeit nur jeweils eine Station Informationen übertragen, die an alle anderen Stationen übertragen bzw. von diesen empfangen wird. Fangen in einem derartigen gemeinsamen Schicht-1-Segment zwei Stationen gleichzeitig an zu senden, kommt es zu Kollisionen.

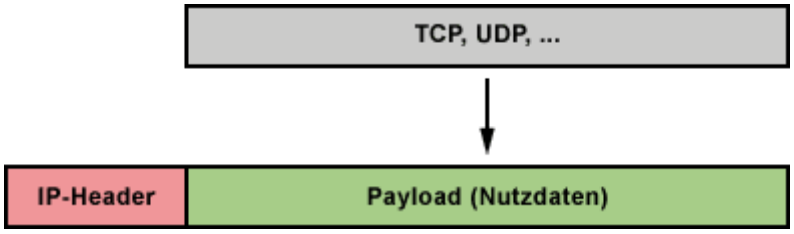
Beispiele für Kollisionsdomänen:





IPv4 - Internet Protocol Version 4 (Layer 3)

Das Internet Protocol, kurz IP, wird im Rahmen der Protokollfamilie TCP/IP zur Vermittlung von Datenpaketen verwendet. Es arbeitet auf der **Schicht 3 des OSI-Schichtenmodells** und hat maßgeblich die Aufgabe, **Datenpakete zu adressieren** und in einem dezentralen, **verbindungslosen und paketorientierten Netzwerk zu übertragen**. Dazu haben alle Netzwerk-Teilnehmer eine eigene IP-Adresse im Netzwerk. Sie dient nicht nur zur Identifikation eines Hosts, sondern auch des Netzes, in dem sich der jeweilige Host befindet.

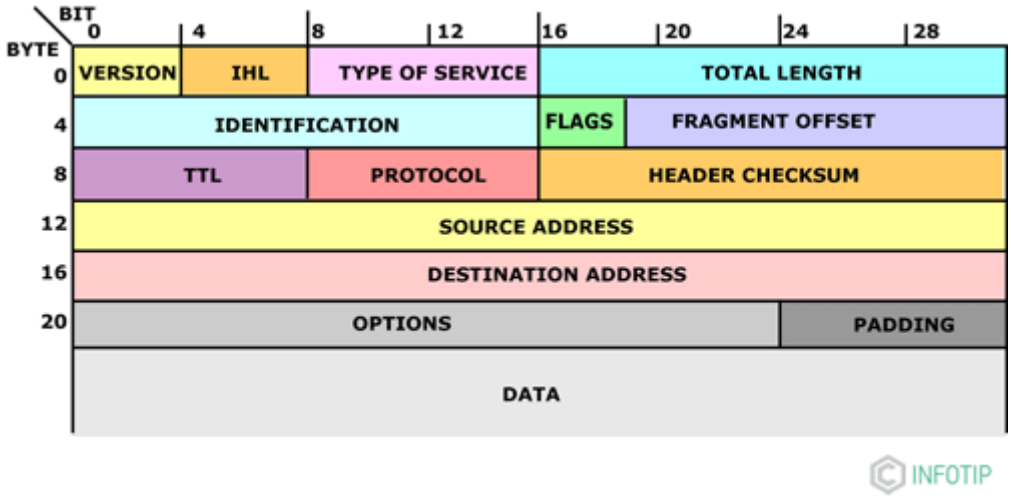


Aufgaben und Funktionen von IPv4

- Logische Adressierung (IPv4-Adresse)
- IPv4-Konfiguration
- IPv4-Header
- IP-Routing
- Namensauflösung (DNS-Dienst)

IPv4 Header

Jedes IPv4-Datenpaket besteht aus einem Header (Kopf) und dem Payload, in dem sich die Nutzdaten befinden. Der Header ist den Nutzdaten vorangestellt. Im IP-Header sind Informationen enthalten, die für die Verarbeitung durch das Internet Protocol notwendig sind.



Der Header ist in jeweils 32-Bit-Blöcke unterteilt. Dort sind Angaben zu Servicetypen, Paketlänge, Sender- und Empfängeradresse abgelegt. Ein IP-Paket muss mindestens 20 Byte Header und 8 Byte Nutzdaten bzw. Nutz- und Fülldaten enthalten. Die Gesamtlänge eines IP-Pakets darf 65.535 Byte nicht überschreiten.

IPv6 - Internet Protocol Version 6 (Layer 3)

IPv6 ist als Internet Protocol (Version 6) für die Vermittlung von Datenpaketen durch ein paketvermittelndes Netz, die Adressierung von Netzknoten und -stationen, sowie die Weiterleitung von Datenpaketen zwischen Teilnetzen zuständig. Mit diesen Aufgaben ist IPv6 der Schicht 3 des OSI-Schichtenmodells zugeordnet. Die Aufgabe des Internet-Protokolls besteht im Wesentlichen darin, Datenpakete von einem System über verschiedene Netzwerke hinweg zu einem anderen System zu vermitteln (Routing).

IPv6 ist der direkte Nachfolger von IPv4 und Teil der Protokollfamilie TCP/IP. Seit Dezember 1998 steht IPv6 bereit und wurde hauptsächlich wegen der Adressknappheit und verschiedener Unzulänglichkeiten von IPv4 entwickelt spezifiziert. Da weltweit immer mehr Menschen, Maschinen und Geräte an das Internet mit einer eindeutigen Adresse angeschlossen werden sollen, reichen die 4 Milliarden IPv4-Adressen nicht mehr aus.

Warum IPv6?

IPv6 gilt als Wunderwaffe gegen so manche Probleme mit Netzwerkprotokollen und gleichzeitig wird es als Teufelszeug verdammt, das wieder neue unbekannte Probleme hervorruft. Eine Tatsache ist, dass Administratoren, Programmierer und Hersteller IPv6 neu lernen müssen. Viele Rezepte aus der IPv4-Welt taugen unter IPv6 nicht mehr. Erschwerend kommt hinzu, dass es bei IPv6 allen Beteiligten an Erfahrung fehlt. IPv6-Gurus, die man bei einem großen Problem befragen kann, gibt es nicht so viele.

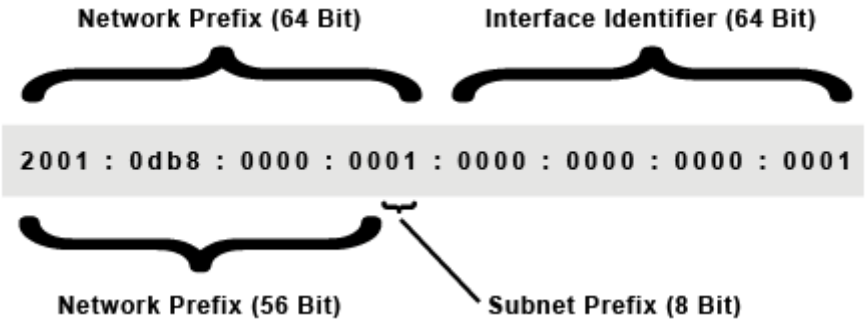
Bei IPv6 ist das Ende-zu-Ende-Prinzip konsequent weiter gedacht. Ein Interface kann mehrere IPv6-Adressen haben und es gibt spezielle IPv6-Adressen, denen mehrere Interfaces zugeordnet sind. IPv6 löst also nicht nur die Adressknappheit, sondern bietet auch Erleichterungen bei der Konfiguration und im Betrieb. Die zustandslose IPv6-Konfiguration und verbindungslokalen Adressen, die bereits nach dem Computerstart verfügbar sind, vereinfachen die Einrichtung und den Betrieb eines lokalen Netzwerks. Damit das gelingt sind Planer und Errichter von IP-Netzen gefordert sich eine neue Denkweise anzueignen.

IPv6 Adressen

Eine IPv6-Adresse ist eine Netzwerk-Adresse, die einen Host eindeutig innerhalb eines IPv6-Netzwerks logisch adressiert. Die Adresse wird auf IP- bzw. Vermittlungsebene (des OSI-Schichtenmodells) benötigt, um Datenpakete verschicken und zustellen zu können. Im Gegensatz zu anderen Adressen hat ein IPv6-Host mehrere IPv6-Adressen, die unterschiedliche Gültigkeitsbereiche haben. Konkret bedeutet das, dass wenn von IPv6-Adressen die Rede ist, dass nicht immer klar ist, welchen Gültigkeitsbereich diese IPv6-Adressen aufweisen. Grob unterscheidet man zwischen verbindungslokalen und globalen IPv6-Adressen. Die verbindungslokale IPv6-Adresse ist nur im lokalen Netzwerk gültig und wird nicht geroutet. Die globale IPv6-Adresse ist über das lokale Netzwerk hinaus im Internet gültig.

Eine IPv6-Adresse hat eine Länge von 128 Bit. Diese Adresslänge erlaubt eine unvorstellbare Menge von 2128 oder 3,4 x 1038 IPv6-Adressen. Das sind 340.282.366.900.000.000.000.000.000.000.000.000.000 IPv6-Adressen, also rund 340 Sextillionen Adressen. Bei IPv4 spricht man von rund 4,3 Milliarden Adressen. Der Adressraum von IPv6 reicht aus,

um umgerechnet jeden Quadratmillimeter der Erdoberfläche inklusive der Ozeane mit rund 600 Billiarden Adressen zu pflastern.



Eine IPv6-Adresse besteht aus 128 Bit. Wegen der unhandlichen Länge werden die 128 Bit in 8 mal 16 Bit unterteilt. Je 4 Bit werden als eine hexadezimale Zahl dargestellt. Je 4 Hexzahlen werden gruppiert und durch einen Doppelpunkt („:“) getrennt. Um die Schreibweise zu vereinfachen lässt man führende Nullen in den Blöcken weg. Eine Folge von 8 Nullen kann man durch zwei Doppelpunkte („::“) ersetzen.

Eine IPv6-Adresse besteht aus zwei Teilen. Dem Network Prefix (Präfix oder Netz-ID) und dem Interface Identifier (Suffix, IID oder EUI). Der Network Prefix kennzeichnet das Netz, Subnetz bzw. Adressbereich. Der Interface Identifier kennzeichnet einen Host in diesem Netz. Er wird aus der 48-Bit-MAC-Adresse des Interfaces gebildet und dabei in eine 64-Bit-Adresse umgewandelt. Es handelt sich dabei um das Modified-EUI-64-Format. Auf diese Weise ist das Interface unabhängig vom Network Prefix eindeutig identifizierbar.

NAT - Network Address Translation (Layer 3)

NAT (Network Address Translation) ist ein Verfahren, dass in **IP-Routern eingesetzt** wird, die lokale Netzwerke mit dem Internet verbinden. Weil Internet-Zugänge in der Regel nur über **eine einzige öffentliche** und damit routbare IPv4-Adresse verfügen, müssen sich alle anderen Hosts im lokalen Netzwerk mit privaten IPv4-Adressen begnügen. **Private IP-Adressen** dürfen zwar **mehrfach verwendet** werden, aber besitzen **in öffentlichen Netzen keine Gültigkeit**. Hosts mit einer privaten IPv4-Adresse können somit nicht mit Hosts außerhalb des lokalen Netzwerks kommunizieren.

NAT ist allerdings nur eine mittlerweile sehr lang andauernde Notlösung, um die Adressknappheit von IPv4 zu umgehen. Um die damit einhergehenden Probleme zu lösen muss langfristig auf ein Internet-Protokoll mit einem größeren Adressraum umgestellt werden. IPv6 ist ein solches Protokoll.

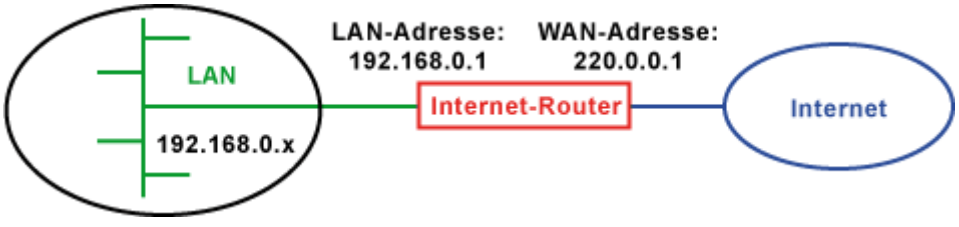
Warum NAT?

Die **ersten IPv4-Netze** waren **anfangs** eigenständige Netz **ohne Verbindung nach außen**. Hier begnügte man sich mit IPv4-Adressen aus den privaten Adressbereichen. Parallel dazu kam es bereits **Ende der 1990er** Jahre zu **Engpässen bei öffentlichen IPv4-Adressen**. Die steigende Anzahl der Einwahlzugänge über das Telefonnetz mussten mit IPv4-Adressen versorgt werden. Bis heute bekommt **ein Internet-Anschluss** nur eine **IPv4-Adresse für ein Gerät**. Damals war es undenkbar, dass an einem Internet-Anschluss ein ganzes Heimnetzwerk betrieben wird. Wenn ein Haushalt einen PC per Modem an das Telefonnetz angeschlossen und sich ins Internet eingewählt hat, dann war das schon etwas besonderes.

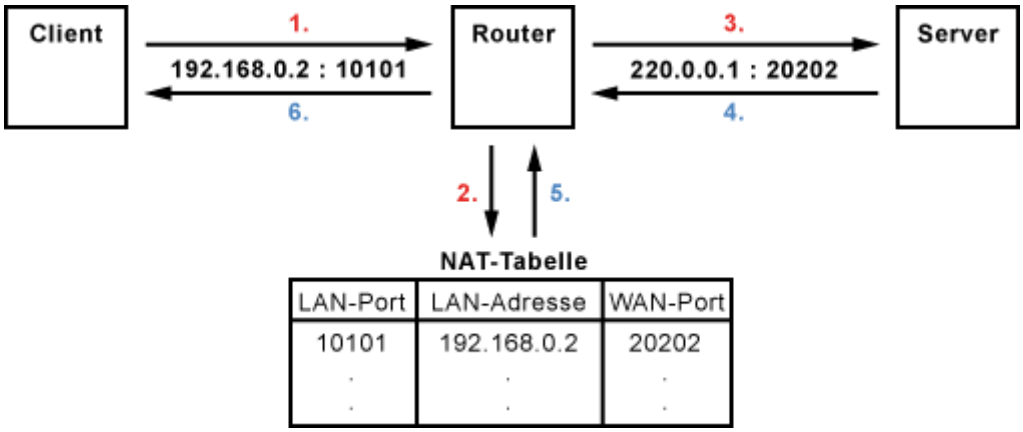
Internet in Österreich

NAT

Der Betrieb eines NAT-Routers ist üblicherweise an einem gewöhnlichen Internet-Anschluss. Zum Beispiel über DSL oder Kabelmodem. Der eingesetzte Router dient als Zugang zum Internet und als Standard-Gateway für das lokale Netzwerk. In der Regel wollen über den Router mehr Geräte ins Internet, als öffentliche IP-Adressen zur Verfügung stehen. In der Regel nur eine einzige. Beispielsweise bekommt der Router des lokalen Netzwerks die öffentliche IP-Adresse 222.0.0.1 für seinen WAN-Port vom Internet Service Provider (ISP) zugewiesen. Weil nur eine öffentliche IP-Adresse vom Internet-Provider zugeteilt wurde, bekommen die Stationen im LAN private IP-Adressen aus speziell dafür reservierten Adressbereichen zugewiesen. Diese Adressen sind nur innerhalb des privaten Netzwerks gültig. Private IP-Adressen werden in öffentlichen Netzen nicht geroutet. Das bedeutet, dass Stationen mit privaten IP-Adressen keine Verbindung ins Internet bekommen können. Damit das trotzdem funktioniert, wurde NAT entwickelt.



Ablauf von NAT



1. Der Client schickt seine Datenpakete mit der IP-Adresse 192.168.0.2 und dem TCP-Port 10101 an sein Standard-Gateway, bei dem es sich um einen NAT-Router handelt.
2. Der NAT-Router tauscht IP-Adresse (LAN-Adresse) und TCP-Port (LAN-Port) aus und speichert beides mit der getauschten Port-Nummer (WAN-Port) in der NAT-Tabelle.
3. Der Router leitet das Datenpaket mit der WAN-Adresse 220.0.0.1 und der neuen TCP-Port 20202 ins Internet weiter.
4. Der Empfänger (Server) verarbeitet das Datenpaket und schickt seine Antwort zurück.
5. Der NAT-Router stellt nun anhand der Port-Nummer 20202 (WAN-Port) fest, für welche IP-Adresse (LAN-Adresse) das Paket im lokalen Netz gedacht ist.
6. Er tauscht die IP-Adresse und die Port-Nummer wieder aus und leitet das Datenpaket ins lokale Netz weiter, wo es der Client entgegen nimmt.

Probleme

- Durch NAT können nur noch die, die über öffentliche IPv4-Adressen und in der Regel auch über das notwendige Kleingeld verfügen, Dienste im Internet anbieten.
- Die Einträge in der NAT-Tabelle des Routers sind nur für eine kurze Zeit gültig. Für eine Anwendung, die nur sehr unregelmäßig Daten austauscht, bedeutet das, dass ständig die Verbindung abgebrochen wird und dadurch die Erreichbarkeit eingeschränkt ist.

Vorteile

- Mehr Sicherheit - PCs im Netzwerk nicht direkt ansprechbar = mehr Privatsphäre
- Firewall - keine von außen eingehenden Verbindungen erlaubt, so fern keine Verbindung von intern aufgebaut wurde

ICMP - Internet Control Message Protocol (Layer 3)

Das Internet Control Message Protocol (ICMP) ist Bestandteil des Internet Protocols (IP). Es wird aber als eigenständiges Protokoll behandelt, das zur Übermittlung von Meldungen über IP dient. Hauptaufgabe von ICMP ist die **Übertragung von Statusinformationen und Fehlermeldungen der Protokolle IP, TCP und UDP**. Die ICMP-Meldungen werden zwischen Rechnern und aktiven Netzknoten, z. B. Routern, benutzt, um sich gegenseitig Probleme mit Datenpaketen mitzuteilen. Ziel ist, die Übertragungsqualität zu verbessert.

Jedes Betriebssystem mit TCP/IP hat Tools, die ICMP nutzen. Zwei bekannte Tools sind Ping und Trace Route. Beides sind sehr einfache Programme, die zur Analyse von Netzwerk-Problemen gedacht sind und damit wesentlich zur Problemlösung beitragen können.

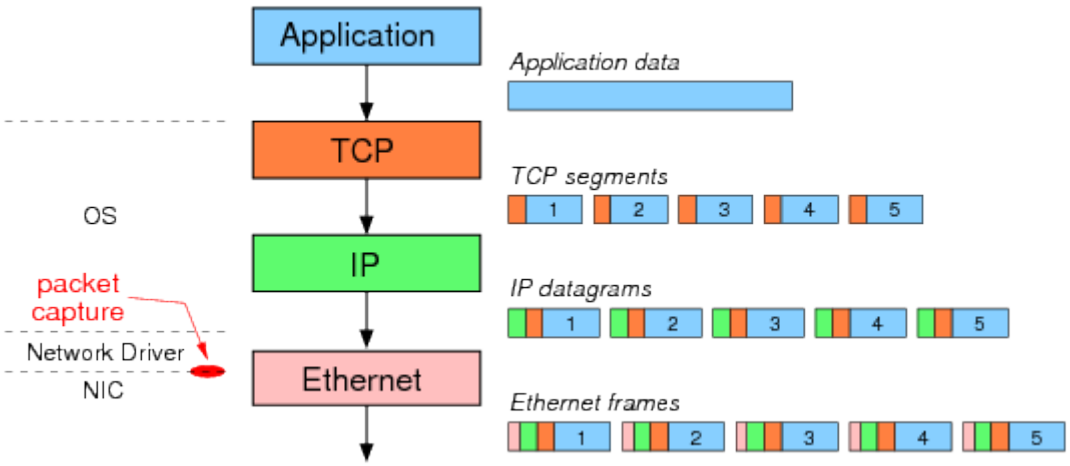
TCP - Transmission Control Protocol (Layer 4)

Das **Transmission Control Protocol** (=Übertragungssteuerungsprotokoll) ist ein Netzwerkprotokoll, das definiert, auf welche Art und Weise Daten zwischen Netzwerkkomponenten ausgetauscht werden sollen. Nahezu sämtliche aktuellen Betriebssysteme moderner Computer beherrschen TCP und nutzen es für den Datenaustausch mit anderen Rechnern. Das Protokoll ist ein zuverlässiges, verbindungsorientiertes, paketvermitteltes Transportprotokoll in Computernetzwerken. Es ist Teil der Internetprotokollfamilie, der Grundlage des Internets.

Aufgaben des TCP-Protokolls

Segmentierung (Data Segmenting)

Eine Funktion von TCP besteht darin, den von den Anwendungen kommenden **Datenstrom in Datenpakete bzw. Segmente aufzuteilen (Segmentierung)** und beim Empfang wieder zusammenzusetzen. Die Segmente werden mit einem **Header** versehen, in dem Steuer- und Kontroll-Informationen enthalten sind. Danach werden die **Segmente an das Internet Protocol (IP) übergeben**. Da beim IP-Routing die **Datenpakete unterschiedliche Wege** gehen können, entstehen unter Umständen **zeitliche Verzögerungen**, die dazu führen, dass die Datenpakete beim Empfänger in einer anderen Reihenfolge eingehen, als sie ursprünglich hatten. Deshalb werden die Segmente beim Empfänger auch wieder in die **richtige Reihenfolge** gebracht und erst dann an die adressierte Anwendung übergeben. Dazu werden die Segmente mit einer **fortlaufenden Sequenznummer** versehen (Sequenzierung).



Verbindungsmanagement (Connection Establishment and Termination)

Als **verbindungsorientiertes Protokoll** ist TCP für den **Verbindungsaufbau und Verbindungsabbau** zwischen zwei Stationen einer **Ende-zu-Ende-Kommunikation** zuständig. Durch TCP stehen Sender und Empfänger ständig in Kontakt zueinander (siehe TCP Kommunikation).

Fehlerbehandlung (Error Detection)

Obwohl es sich eher um eine virtuelle Verbindung handelt, werden während der Datenübertragung

ständig **Kontrollmeldungen** ausgetauscht. Der Empfänger bestätigt dem Sender jedes empfangene Datenpaket. Trifft keine Bestätigung beim Absender ein, wird das Paket noch mal verschickt. Da es bei Übertragungsproblemen zu doppelten Datenpaketen und Quittierungen kommen kann, werden alle TCP-Pakete und TCP-Meldungen mit einer **fortlaufenden Sequenznummer** gekennzeichnet. So sind Sender und Empfänger in der Lage, die Reihenfolge und Zuordnung der Datenpakete und Meldungen zu erkennen.

Flusssteuerung (Flow Control)

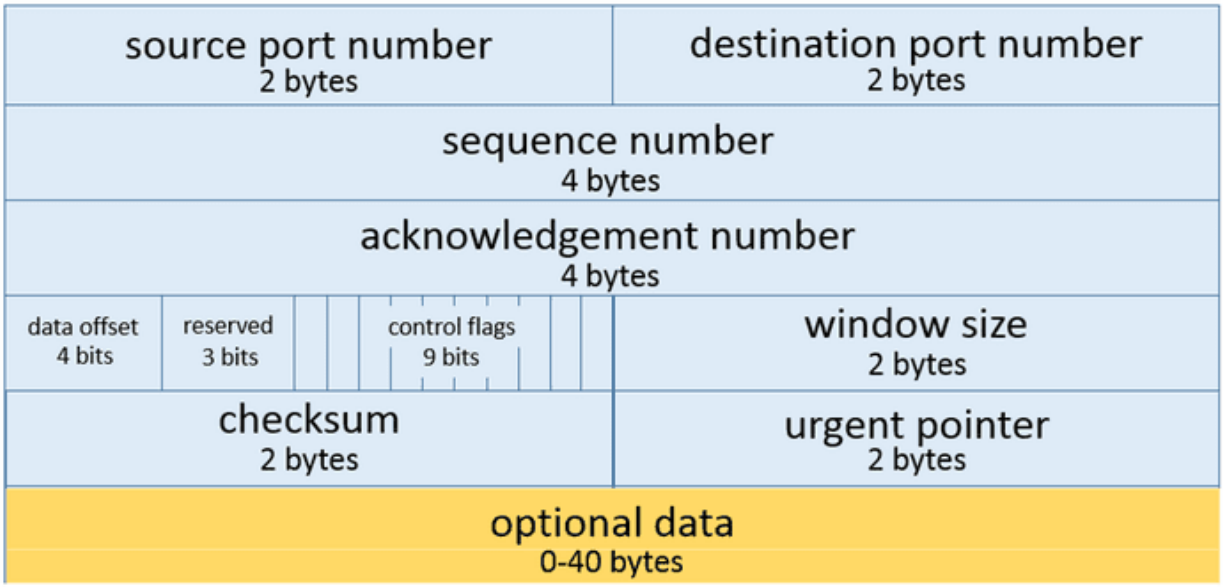
Bei einer paketorientierten Übertragung ohne feste zeitliche Zuordnung und ohne Kenntnis des Übertragungswegs erhält das Transport-Protokoll vom Übertragungssystem **keine Information über die verfügbare Bandbreite**. Mit der **Flusssteuerung** werden beliebig langsame oder schnelle **Übertragungsstrecken dynamisch auszulasten** und auch auf unerwartete Engpässe und Verzögerungen reagiert.

Anwendungsunterstützung (Application Support)

TCP- und UDP-Ports sind eine Software-Abstraktion, um Kommunikationsverbindungen voneinander unterscheiden zu können. Ähnlich wie IP-Adressen Rechner in Netzwerken adressieren, **adressieren Ports spezifische Anwendungen** oder Verbindungen, die auf einem Rechner laufen.

Aufbau eines TCP Headers

Aufbau des TCP-Headers TCP-Pakete setzen sich aus dem Header-Bereich und dem Daten-Bereich zusammen. Im Header sind alle Informationen enthalten, die für eine gesicherte TCP-Verbindung wichtig sind. Der TCP-Header ist in mehrere 32-Bit-Blöcke aufgeteilt. Mindestens enthält der Header 5 solcher Blöcke. Somit hat ein TCP-Header eine Länge von **mindestens 20 Byte**.

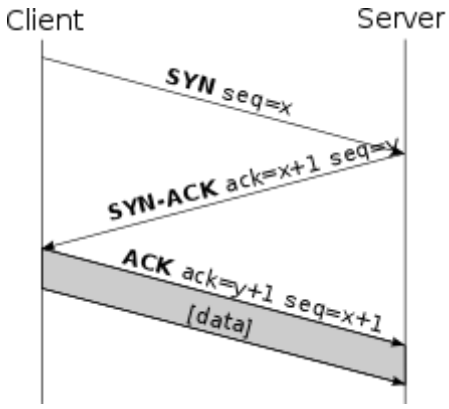


TCP Kommunikation

Verbindungsaufbau - 3-Way-Handshake

Der Client, der eine Verbindung aufbauen will, sendet dem Server ein **SYN-Paket** (von englisch synchronize) mit einer Sequenznummer x. Die Sequenznummern sind dabei für die Sicherstellung einer vollständigen Übertragung in der richtigen Reihenfolge und ohne Duplikate wichtig. Es handelt sich also um ein Paket, dessen SYN-Bit im Paketkopf gesetzt ist (siehe TCP-Header). Die Start-Sequenznummer ist eine beliebige Zahl, deren Generierung von der jeweiligen TCP-Implementierung abhängig ist. Sie sollte jedoch möglichst zufällig sein, um Sicherheitsrisiken zu vermeiden.

Der Server (siehe Abbildung) empfängt das Paket. Ist der Port geschlossen, antwortet er mit einem TCP-RST, um zu signalisieren, dass keine Verbindung aufgebaut werden kann. Ist der Port geöffnet, bestätigt er den Erhalt des ersten SYN-Pakets und stimmt dem Verbindungsaufbau zu, indem er ein **SYN/ACK-Paket** zurückschickt (ACK von engl. acknowledgement ‚Bestätigung'). Das gesetzte ACK-Flag im TCP-Header kennzeichnet diese Pakete, welche die Sequenznummer x+1 des SYN-Pakets im Header enthalten. Zusätzlich sendet er im Gegenzug seine Start-Sequenznummer y, die ebenfalls beliebig und unabhängig von der Start-Sequenznummer des Clients ist.



Der Client bestätigt zuletzt den Erhalt des SYN/ACK-Pakets durch das Senden eines eigenen **ACK-Paket**s mit der Sequenznummer x+1. Dieser Vorgang wird auch als **Forward Acknowledgement** bezeichnet. Aus Sicherheitsgründen sendet der Client den Wert y+1 (die Sequenznummer des Servers + 1) im ACK-Segment zurück. Die Verbindung ist damit aufgebaut. Im folgenden Beispiel wird der Vorgang abstrakt dargestellt:

```
1.  SYN-SENT            →    <SEQ=100><CTL=SYN>              →    SYN-RECEIVED
2.  SYN/ACK-RECEIVED    ←    <SEQ=300><ACK=101><CTL=SYN,ACK>    ←
SYN/ACK-SENT
3.  ACK-SENT            →    <SEQ=101><ACK=301><CTL=ACK>    →    ESTABLISHED
```

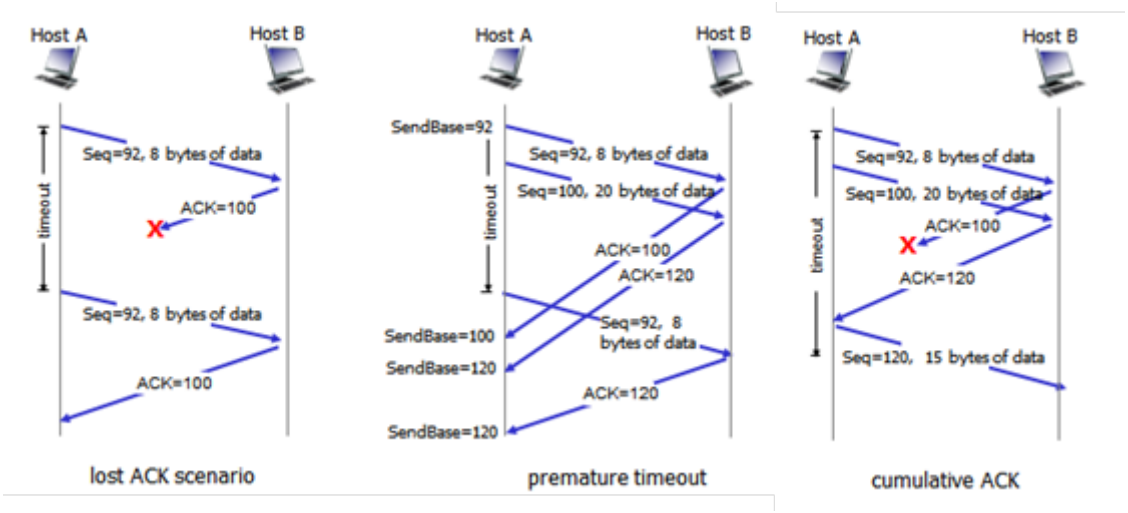
Einmal aufgebaut, ist die Verbindung für beide Kommunikationspartner gleichberechtigt, man kann einer bestehenden Verbindung auf TCP-Ebene nicht ansehen, wer der Server und wer der Client ist. Daher hat eine Unterscheidung dieser beiden Rollen in der weiteren Betrachtung keine Bedeutung mehr.

Aufzeichnung mit Wireshark

Datenaustausch

Der Sender beginnt mit dem Senden des ersten Datenpakets (Send Paket 1). Der Empfänger nimmt

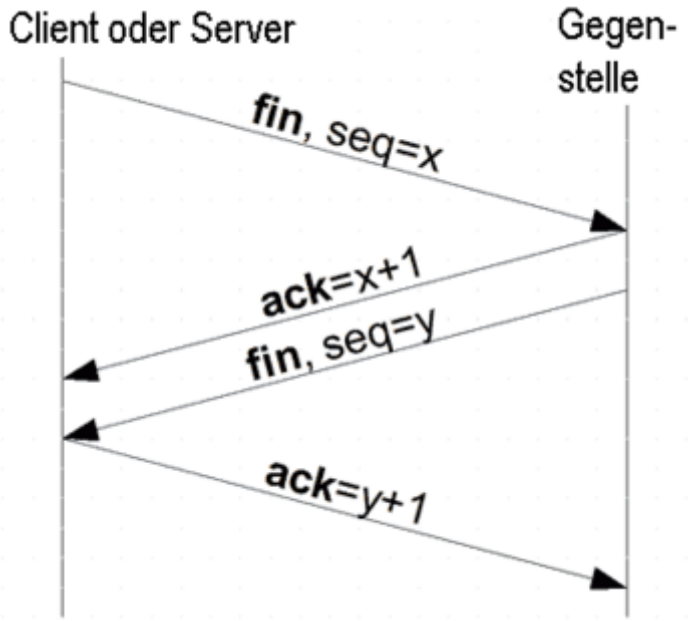
das Paket entgegen (Receive Paket 1) und bestätigt den Empfang (Send ACK Paket 1). Der Sender nimmt die Bestätigung entgegen (Receive ACK Paket 1) und sendet das zweite Datenpaket (Send Paket 2). Der Empfänger nimmt das zweite Paket entgegen (Receive Paket 2) und bestätigt den Empfang (Send ACK Paket 2). Der Sender nimmt die zweite Bestätigung entgegen (Receive ACK Paket 2). Und so läuft der Datenaustausch weiter, bis alle Pakete übertragen wurden.



Aufzeichnung mit Wireshark

Verbindungsabbau

Der Verbindungsabbau kann sowohl vom Client als auch vom Server vorgenommen werden. Zuerst schickt einer der beiden der Gegenstelle einen Verbindungsabbauwunsch (**FIN**). Die Gegenstelle bestätigt den Erhalt der Nachricht (ACK) und schickt gleich darauf ebenfalls einen Verbindungsabbauwunsch (**FIN**). Danach bekommt die Gegenstelle noch mitgeteilt, dass die Verbindung abgebaut ist (**ACK**).



Aufzeichnung mit Wireshark

UDP - User Datagram Protocol (Layer 4)

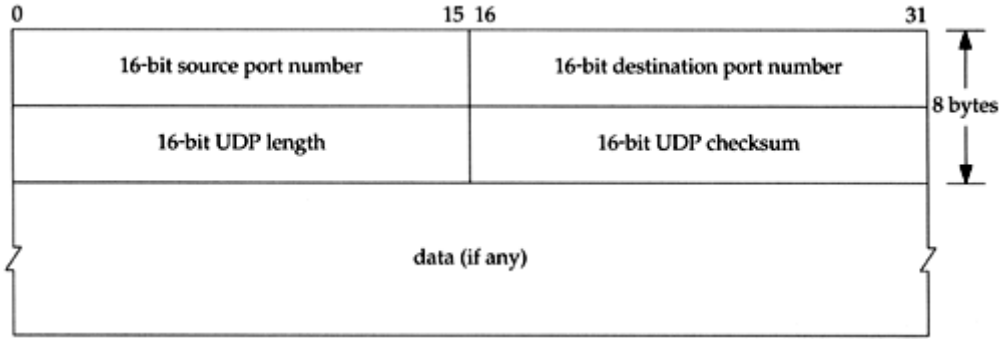
UDP ist ein **verbindungsloses Transport-Protokoll** und arbeitet auf der Schicht 4, der **Transportschicht**, des OSI-Schichtenmodells. Es hat damit eine vergleichbare Aufgabe, wie das verbindungsorientierte TCP. Allerdings arbeitet es **verbindungslos und damit unsicher**. Das bedeutet, der **Absender weiß nicht**, ob seine verschickten **Datenpakete angekommen sind**. Während TCP Bestätigungen beim Datenempfang sendet, verzichtet UDP darauf. Das hat den Vorteil, dass der Paket-Header viel kleiner ist und die Übertragungsstrecke keine Bestätigungen übertragen muss. Typischerweise wird UDP bei DNS-Anfragen, VPN-Verbindungen, Audio- und Video-Streaming verwendet.

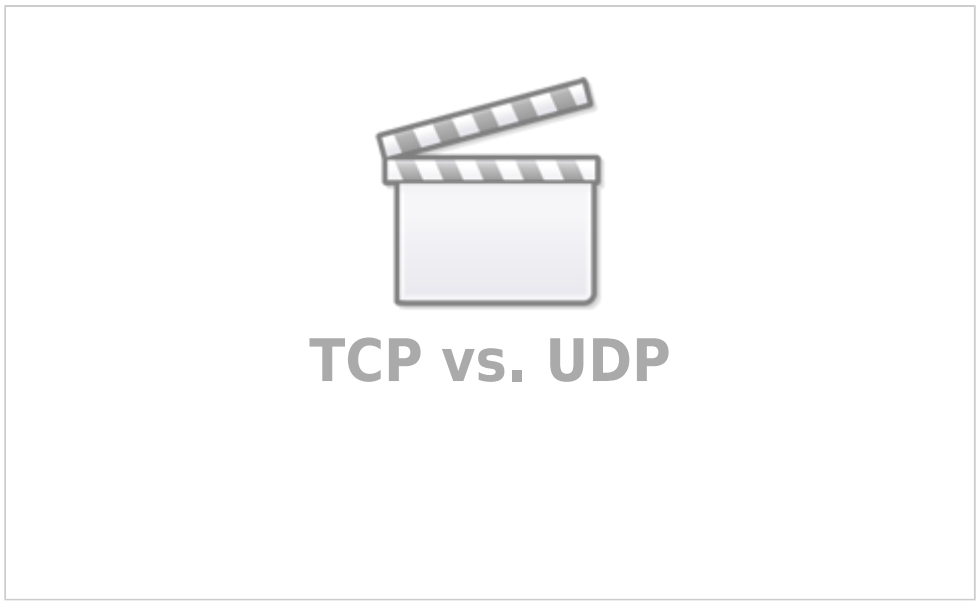
Funktionsweise

UDP hat die selbe Aufgabe wie TCP, nur das nahezu alle Kontrollfunktionen fehlen, dadurch schlanker und einfacher zu verarbeiten ist. So besitzt UDP keinerlei Methoden die sicherstellen, dass ein Datenpaket beim Empfänger ankommt. Ebenso entfällt die Nummerierung der Datenpakete. UDP ist nicht in der Lage die Datenpakete in der richtigen Reihenfolge zusammenzusetzen. Statt dessen werden die UDP-Pakete direkt an die Anwendung weitergeleitet. Für eine sichere Datenübertragung ist deshalb die Anwendung zuständig. In der Regel wird UDP für Anwendungen und Dienste verwendet, die mit Paketverlusten umgehen können oder sich selber um das Verbindungsmanagement kümmern. UDP eignet sich auch für Anwendungen, die nur einzelne, nicht zusammenhängende Datenpakete transportieren müssen.

UDP Header

UDP-Pakete setzen sich aus dem Header-Bereich und dem Daten-Bereich zusammen. Im Header sind alle Informationen enthalten, die eine einigermaßen geordnete Datenübertragung zulässt und die ein UDP-Paket als ein solches erkennen lassen. Der UDP-Header ist in 32-Bit-Blöcke unterteilt. Er besteht aus zwei solcher Blöcke, die den Quell- und Ziel-Port, die Länge des gesamten UDP-Pakets und die Check-Summe enthalten. Der UDP-Header ist mit insgesamt 8 Byte sehr schlank und lässt sich mit wenig Rechenleistung verarbeiten.





DHCP - Dynamic Host Configuration Protocol (UDP, Ports 67 + 68, Layer 7)

DHCP ist ein Protokoll, um **IP-Adressen** in einem TCP/IP-Netzwerk **zu verwalten** und an die anfragenden Hosts zu verteilen. Mit DHCP ist jeder **Netzwerk-Teilnehmer** in der Lage sich selber **automatisch zu konfigurieren**.

Warum DHCP?

Um ein Netzwerk per TCP/IP aufzubauen ist es notwendig an jedem Host eine IP-Konfiguration vorzunehmen. Für ein TCP/IP-Netzwerk müssen folgende Einstellungen an jedem Host vorgenommen werden:

- Vergabe einer eindeutigen IP-Adresse
- Zuweisen einer Subnetzmaske (Subnetmask)
- Zuweisen des zuständigen Default- bzw. Standard-Gateways
- Zuweisen des zuständigen DNS-Servers

In den ersten IP-Netzen wurden IP-Adressen noch von Hand **aufwendig vergeben** und **fest** in die Systeme **eingetragen**. Die dazu **erforderliche Dokumentation** war jedoch nicht immer fehlerfrei und schon gar nicht aktuell und vollständig. Der Ruf nach einer einfachen und automatischen Adressverwaltung wurde deshalb besonders bei Betreibern großer Netze laut. Hier war durch die manuelle Verwaltung und Konfiguration sehr **viel Planungs- und Arbeitszeit** notwendig. Um für die Betreiber der immer größer werdenden Netze eine Erleichterung zu verschaffen wurde DHCP entwickelt. Mit DHCP kann jede IP-Host die IP-Adresskonfiguration von einem DHCP-Server anfordern und sich selber automatisch konfigurieren. So müssen IP-Adressen nicht mehr manuell verwaltet und zugewiesen werden.

Funktionsweise

1. Der DHCP-Client **schickt an alle Rechner** im Netzwerk (=Broadcast) eine **DHCP-Server-Suchanfrage (DHCP-Discover)**. Bis auf den DHCP-Server verwerfen alle Rechner das Datenpaket.
2. Nur der **DHCP-Server** empfängt den DHCP-Discover und bietet dem Client mittels Broadcast eine **freie IP-Adresse an (DHCP-Offer)**
3. Nur der **DHCP-Client** empfängt den DHCP-Offer und antwortet mit einer eine **DHCP-Anfrage (DHCP-Request)**. Alle anderen Clients verwerfen wiederum das DHCP-Angebot.
4. Der **DHCP-Server** empfängt den DHCP-Request und **bestätigt** dem DHCP-Client die angefragte IP-Adresse **mittels DHCP-ACK**.

DNS - Domain Name System (UDP, Port 53 , Layer 7)

Das Domain Name System, kurz DNS, wird auch als **Telefonbuch des Internets** bezeichnet. Ähnlich wie man in einem Telefonverzeichnis nach einem Namen sucht, um die Telefonnummer heraus zu bekommen, schaut man im DNS nach einem Computernamen, um die dazugehörige IP-Adresse zu bekommen. Die IP-Adresse wird benötigt, um eine Verbindung zu einem Server aufbauen zu können, über den nur der Computername bekannt ist.

Das Domain Name System ist ein System zur Auflösung von Computernamen in IP-Adressen und umgekehrt. DNS kennt keine zentrale Datenbank. Die Informationen sind auf vielen tausend Nameservern (DNS-Server) verteilt. Möchte man zum Beispiel die Webseite www.orf.at besuchen, dann fragt der Browser einen DNS-Server, der in der IP-Konfiguration hinterlegt ist. Das ist in der Regel der Router des Internet-Zugangs. Je nach dem, ob die DNS-Anfrage beantwortet werden kann oder nicht, wird eine Kette weiterer DNS-Server befragt, bis die Anfrage positiv beantwortet und eine IP-Adresse an den Browser zurück geliefert werden kann.

Wenn ein Computername oder Domain-Name nicht aufgelöst werden kann, dann kann auch keine Verbindung zu dem betreffenden Host aufgebaut werden. Es sei denn, der Nutzer verfügt über das Wissen der IP-Adresse. Das bedeutet, ohne DNS ist die Kommunikation im Netzwerk und im Internet praktisch nicht möglich. Deshalb existieren viele tausend DNS-Server auf der ganzen Welt, die zusätzlich hierarchisch angeordnet sind und sich gegenseitig über Änderungen informieren.

Top-Level-Domain & Second-Level-Domain

Domain-Namen sind hierarchisch von rechts nach links gegliedert. Der ganz rechte Abschnitt nach dem letzten Punkt heißt Top-Level-Domain (TLD), der davor Second-Level-Domain (SLD) oder einfach „Domain“. Alle weiteren Namensteile links davon sind jeweils Sub- bzw. Third-Level-Domains (Fourth Level, Fifth Level, Sixth Level usw.). Ein Beispiel verdeutlicht die Begrifflichkeiten: Der Name „www.example.com“ besteht aus drei Ebenen:

- „.com“: die erste Ebene, auch Top-Level-Domain oder Domain-Endung genannt
- „example“: die zweite Ebene, auch als Second-Level-Domain oder Domain bezeichnet
- „www“: die dritte Ebene, auch Sub- oder Third-Level-Domain genannt

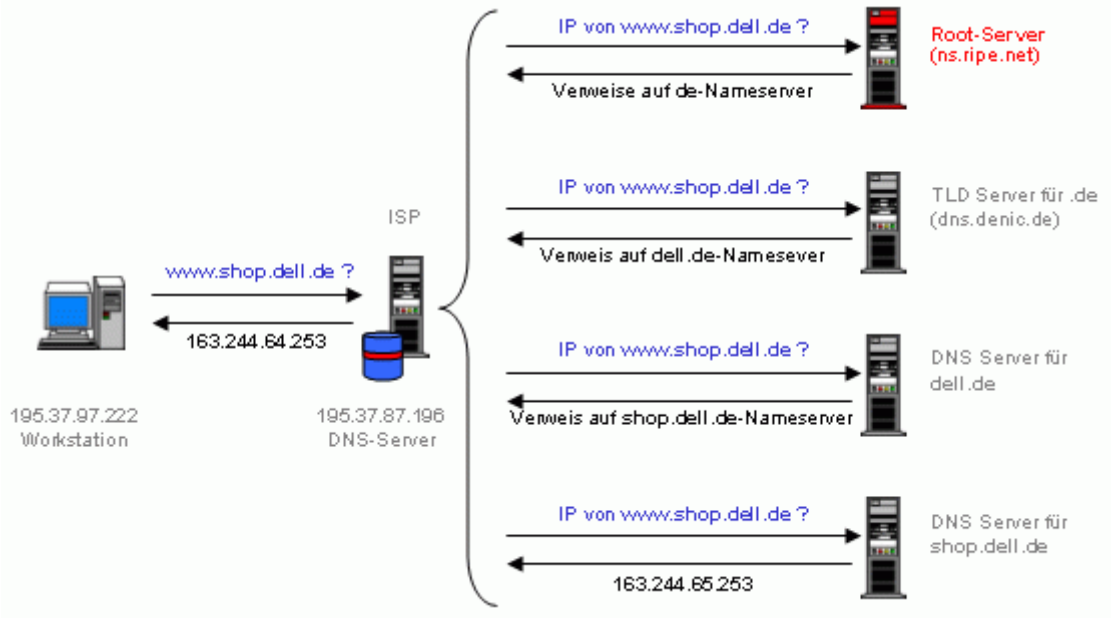
FQDN (Fully Qualified Domain Name)

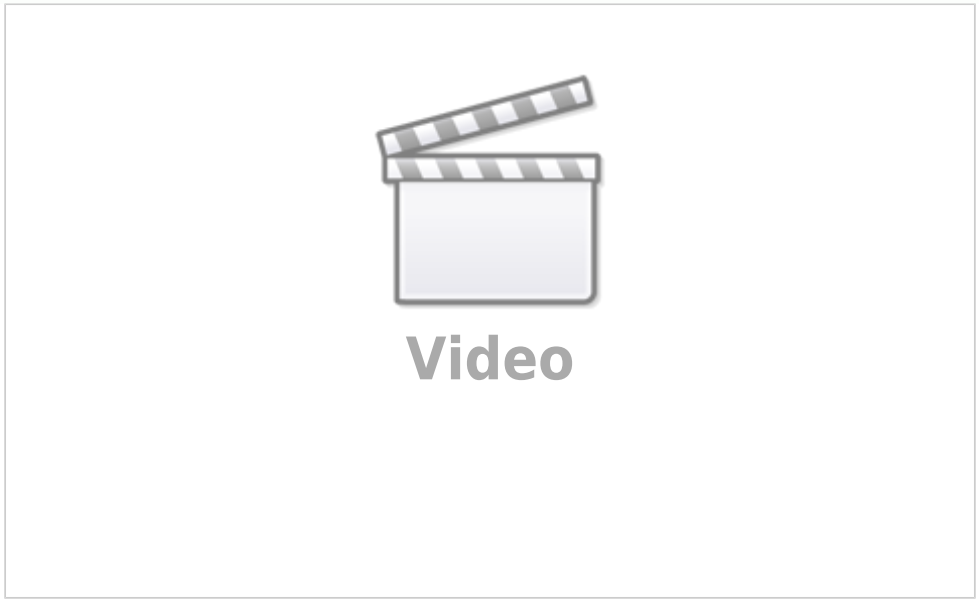
Der vollständige Name einer Domain wird als ihr Fully Qualified Domain Name (FQDN) bezeichnet. Der Domain-Name ist in diesem Fall eine absolute Adresse.

Der FQDN www.example.com. ergibt sich durch:

```
3rd-level-label. 2nd-level-label. Top-Level-Domain. root-label
```

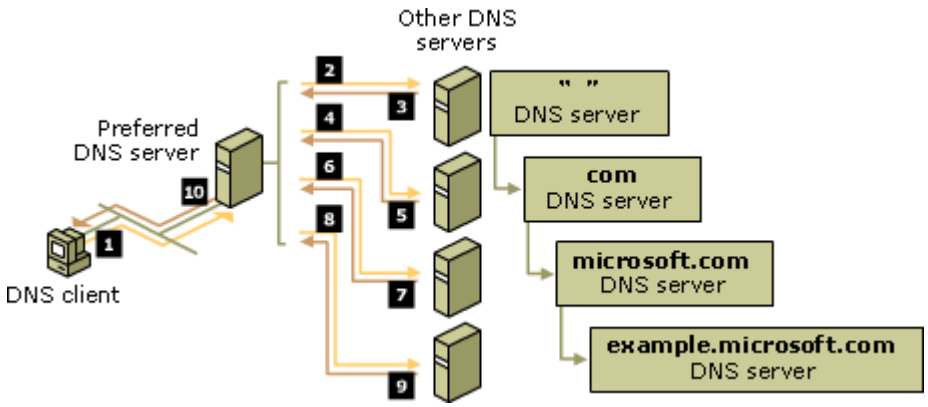
Da das Root-Label immer leer ist (es besteht aus einer leeren Zeichenkette), wird bei den meisten Benutzer-Anwendungen (zum Beispiel Browsern) in der Regel auf die Eingabe des Punktes zwischen dem Label der Top Level Domain und dem root-label verzichtet. Streng genommen handelt es sich bei dieser Schreibweise nicht mehr um eine absolute, sondern um eine relative Adresse und damit nicht mehr um einen FQDN.



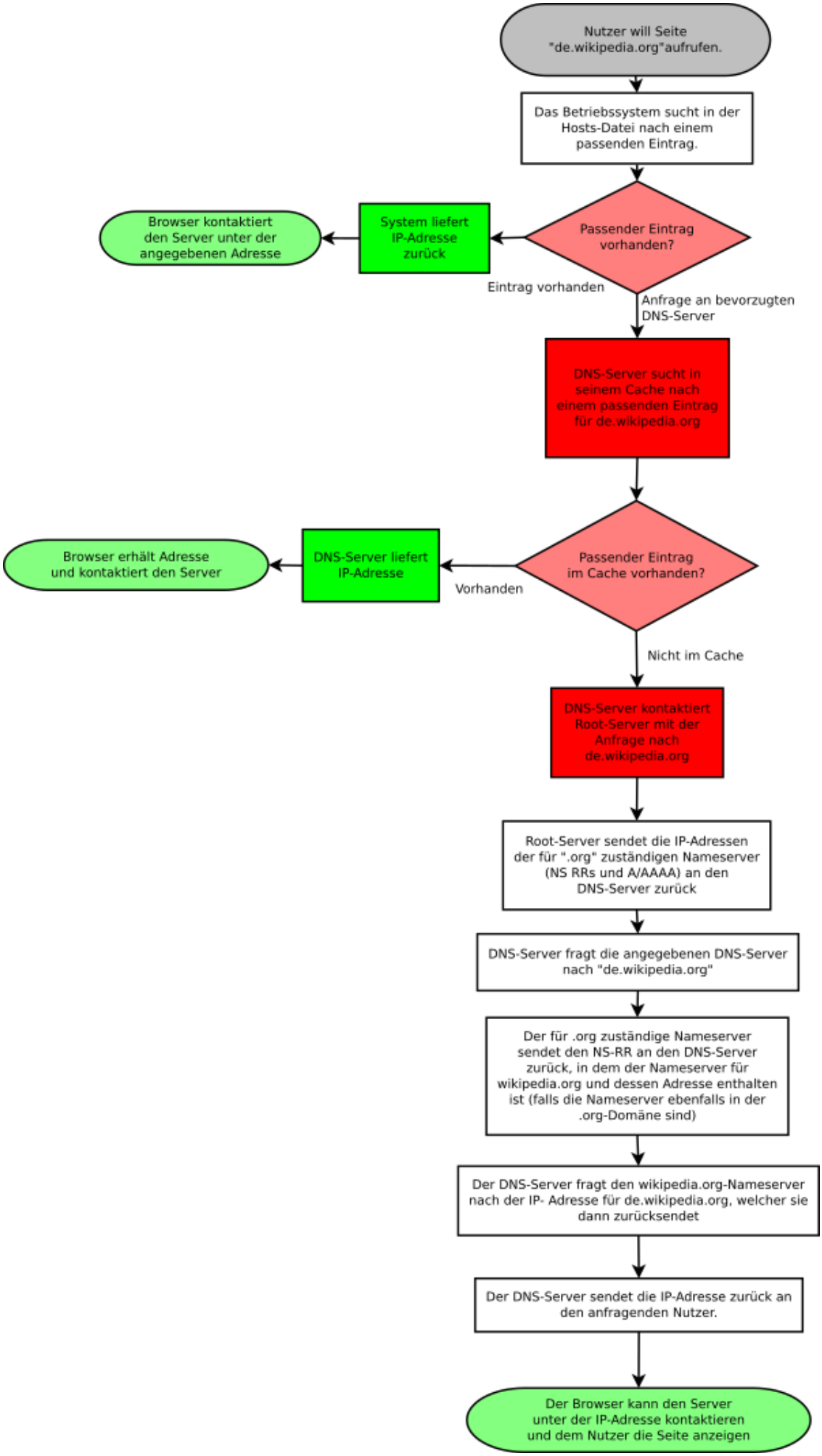


Rekursive Namensauflösung - Rekursive DNS-Serverstruktur

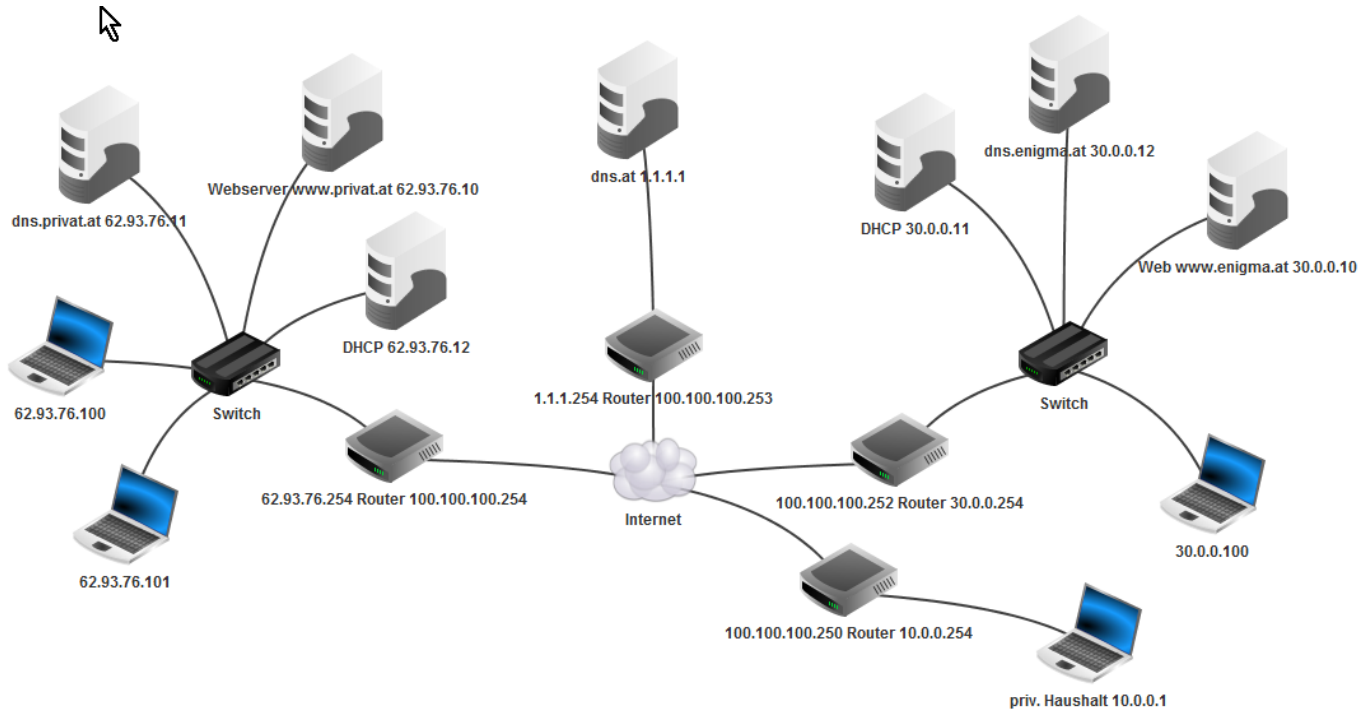
- Einzelne DNS-Server haben nicht jeden Domainnamen gespeichert. Informationen werden hierarchisch von übergeordneten DNS-Servern abgefragt. Die folgenden Grafiken sollen den Prozess veranschaulichen:



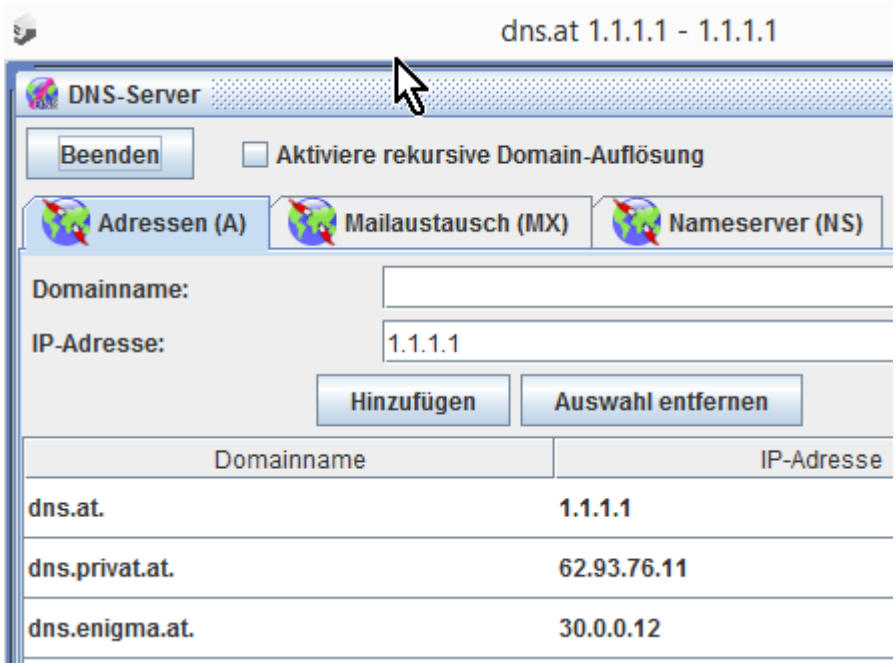
- Hier eine genauere Abfolge der Anfragen:

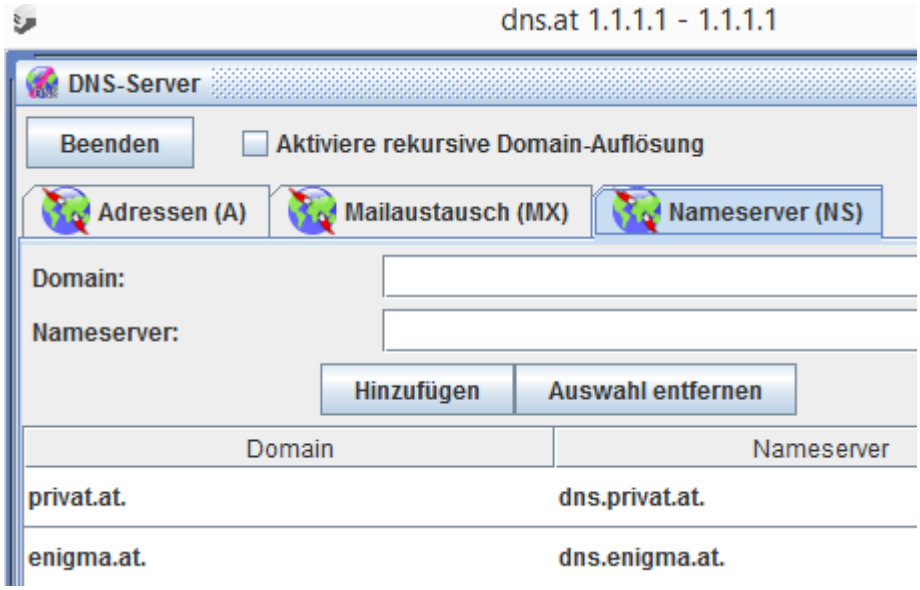

Nutzer will Seite "de.wikipedia.org"aufrufen.
Das Betriebssystem sucht in der Hosts-Datei nach einem passenden Eintrag.
Passender Eintrag vorhanden?
System liefert IP-Adresse zurück
Browser kontaktiert den Server unter der angegebenen Adresse
Eintrag vorhanden
Anfrage an bevorzugten DNS-Server
DNS-Server sucht in seinem Cache nach einem passenden Eintrag für de.wikipedia.org
Passender Eintrag im Cache vorhanden?
DNS-Server liefert IP-Adresse
Browser erhält Adresse und kontaktiert den Server
Vorhanden
Nicht im Cache
DNS-Server kontaktiert Root-Server mit der Anfrage nach de.wikipedia.org
Root-Server sendet die IP-Adressen der für ".org" zuständigen Nameserver (NS RRs und A/AAAA) an den DNS-Server zurück
DNS-Server fragt die angegebenen DNS-Server nach "de.wikipedia.org"
Der für .org zuständige Nameserver sendet den NS-RR an den DNS-Server zurück, in dem der Nameserver für wikipedia.org und dessen Adresse enthalten ist (falls die Nameserver ebenfalls in der .org-Domäne sind)
Der DNS-Server fragt den wikipedia.org-Nameserver nach der IP- Adresse für de.wikipedia.org, welcher sie dann zurücksendet
Der DNS-Server sendet die IP-Adresse zurück an den anfragenden Nutzer.
Der Browser kann den Server unter der IP-Adresse kontaktieren und dem Nutzer die Seite anzeigen

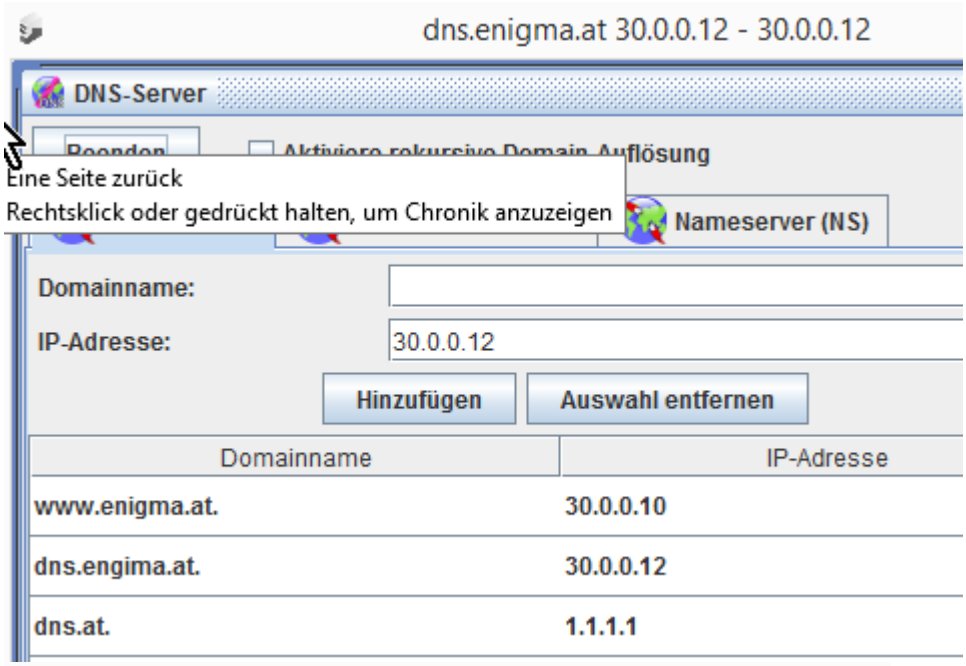
Mittels Filius kann eine hierarchische DNS-Serverstruktur aufgebaut werden:

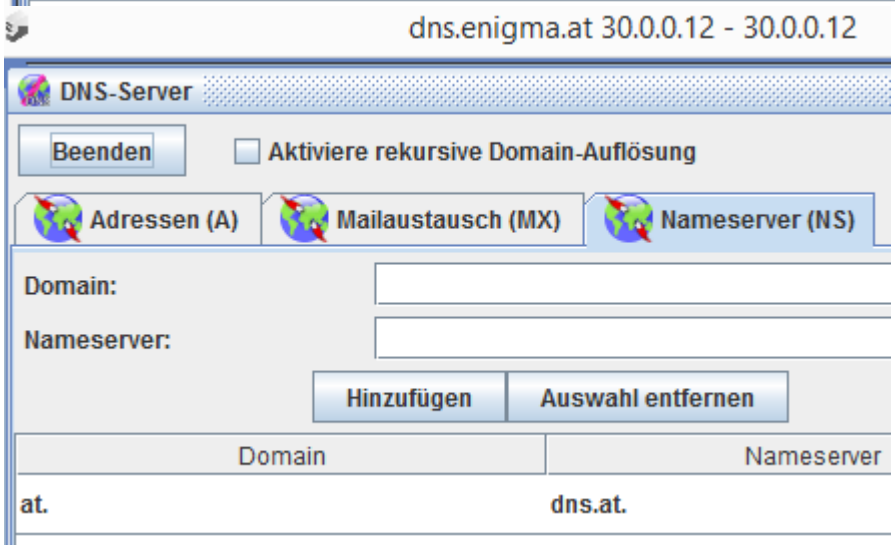


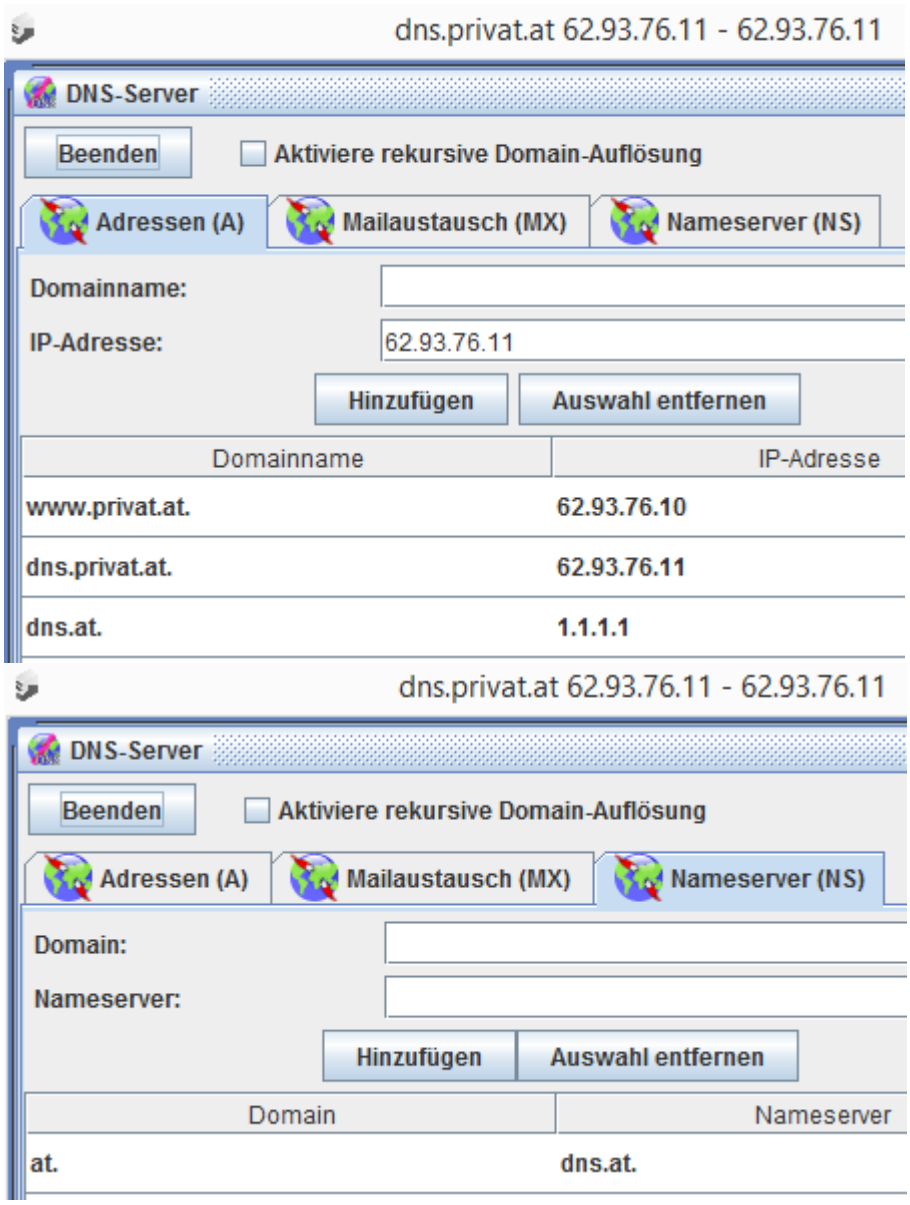
Hier die Screenshots über die entsprechenden Eintragungen in den Nameservern.




dns.at 1.1.1.1 - 1.1.1.1
DNS-Server
Beenden
Aktiviere rekursive Domain-Auflösung
Adressen (A)
Mailaustausch (MX)
Nameserver (NS)
Domain:
Nameserver:
Hinzufügen
Auswahl entfernen
Domain
Nameserver
privat.at.
dns.privat.at.
enigma.at.
dns.enigma.at.


dns.enigma.at 30.0.0.12 - 30.0.0.12
DNS-Server
Eine Seite zurück
Rechtsklick oder gedrückt halten, um Chronik anzuzeigen
Nameserver (NS)
Domainname:
IP-Adresse:
30.0.0.12
Hinzufügen
Auswahl entfernen
Domainname
IP-Adresse
www.enigma.at.
30.0.0.10
dns.engima.at.
30.0.0.12
dns.at.
1.1.1.1

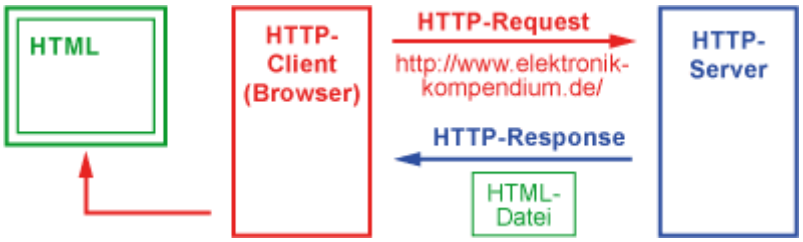

dns.enigma.at 30.0.0.12 - 30.0.0.12
DNS-Server
Beenden
Aktiviere rekursive Domain-Auflösung
Adressen (A)
Mailaustausch (MX)
Nameserver (NS)
Domain:
Nameserver:
Hinzufügen
Auswahl entfernen
Domain
Nameserver
at.
dns.at.



Hypertext Transfer Protocol (HTTP) (Port 80 - Layer 7)

HTTP ist das Kommunikationsprotokoll im World Wide Web (WWW). Die wichtigsten Funktionen sind Dateien vom Webserver anzufordern und in den Browser zu laden. Der Browser übernimmt dann die Darstellung von Texten und Bilder und kümmert sich um das Abspielen von Audio- und Video-Daten.

Funktionsweise



Die Kommunikation findet nach dem **Client-Server-Prinzip** statt. Der **HTTP-Client (Browser)** sendet seine **Anfrage (HTTP-Request)** an den **HTTP-Server (Webserver/Web-Server)**. Dieser bearbeitet die Anfrage und schickt seine **Antwort (HTTP-Response)** zurück. Nach der Antwort durch den Server ist diese Verbindung beendet. Typischerweise finden gleichzeitig mehrere HTTP-Verbindungen statt.

HTTP Adressierung

Damit der Server weiß, was er dem HTTP-Client schicken soll, adressiert der HTTP-Client eine Datei, die sich auf dem HTTP-Server befinden muss. Dazu wird vom HTTP-Client ein **URL (Uniform Resource Locator)** im HTTP-Header an den HTTP-Server übermittelt:

```
http://Servername.Domainname.Top-Level-Domain:TCP-Port/Pfad/Datei

z. B.
http://www.elektronik-kompendium.de:80/sites/kom/0902231.html
```

HTTP Request

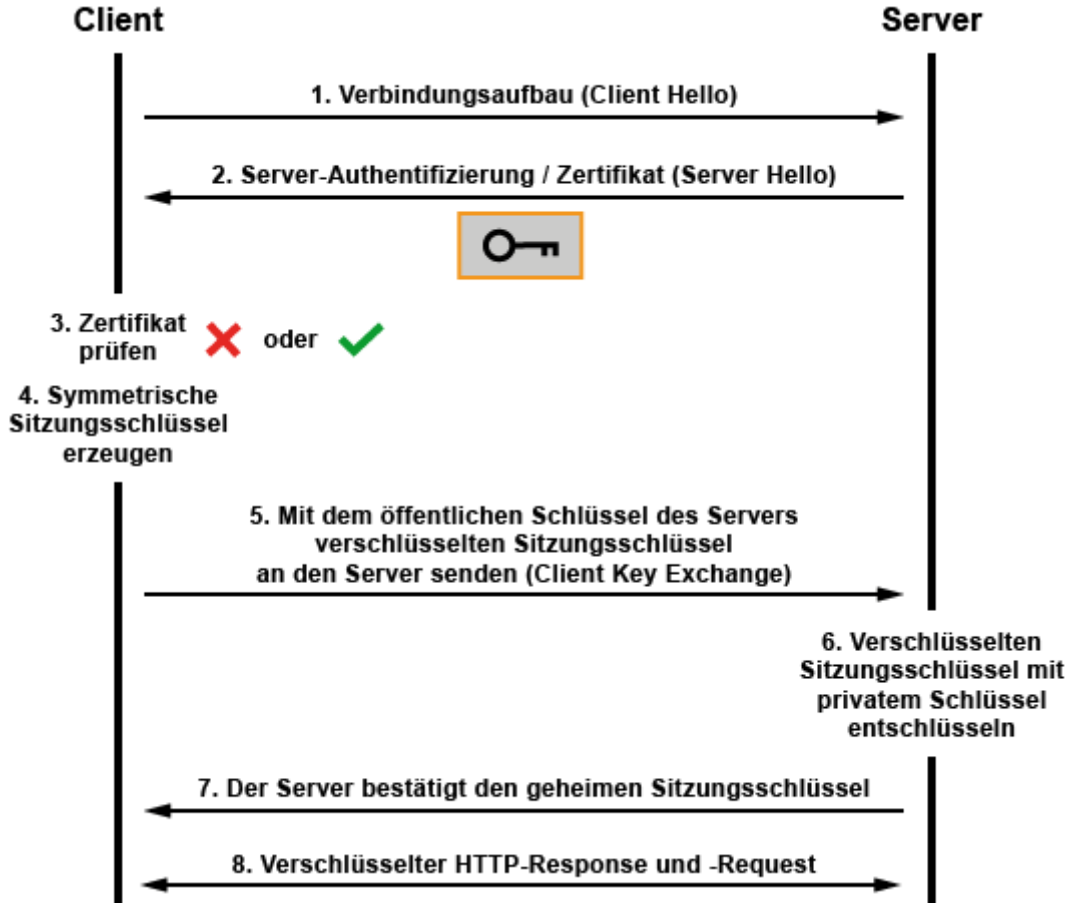
Der HTTP-Request ist die Anfrage des HTTP-Clients an den HTTP-Server. Ein HTTP-Request besteht aus den Angaben **Methode, URL und dem Request-Header**. Die häufigste Methoden sind **GET und POST**. Dahinter folgt durch ein Leerzeichen getrennt der URL und die verwendete HTTP-Version. In weiteren Zeilen folgt der Header und bei der Methode POST durch eine Leerzeile (!) getrennt die Formular-Daten.

HyperText Transfer Protocol Secure (HTTPS) (Port 443 - Layer 7)

HTTPS bzw. HTTP Secure ist die Anwendung von HTTP in **Verbindung mit Verschlüsselung und Authentifizierung**. Wobei in der Regel nur der **angefragte Webserver sich mit einem Zertifikat authentisieren** muss.

Eine **verschlüsselte Verbindung mit einem Browser** signalisiert man mit einem „https:“ *(TCP-Port 443) statt „http:*“ (TCP-Port 80). Dabei muss sich der Webserver dem Client gegenüber authentisieren, ob er tatsächlich der Webserver ist, der sich unter der eingegebenen Adresse befindet. Zusätzlich wird die **Verbindung bzw. Sitzung Ende-zu-Ende-verschlüsselt**. Das bedeutet, die **Stationen zwischen Client und Server** können die Kommunikation **nicht entschlüsseln**.

Für die Authentifizierung und Verschlüsselung ist SSL/TLS verantwortlich. Es schiebt sich zwischen HTTP und dem Transportprotokoll TCP. Damit steht SSL/TLS auch für andere Anwendungsprotokolle zur Verfügung. Beispielsweise SMTPS, IMAPS und FTPS. SSL arbeitet für den Anwender nahezu unsichtbar.



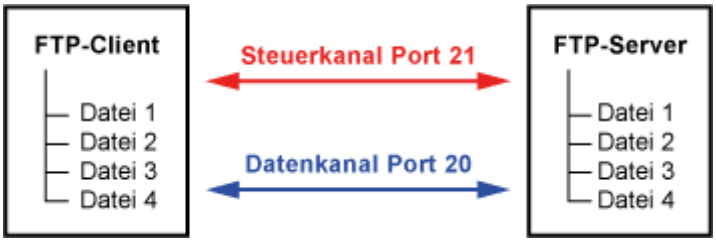
1. Client Hello: Der Client kontaktiert den Server über ein Protokoll mit Verschlüsselungsoptionen.
2. Server Hello, Certificate, Server Key Exchange, Server Hello Done: Der Server nimmt die Verbindung an und schickt sein Zertifikat mit dem öffentlichen Schlüssel seines Schlüsselpaares zur Authentifizierung an den Client.
3. Der Client überprüft das Server-Zertifikat und dessen Gültigkeit (Validierung). Erkennt der Client das Zertifikat als ungültig wird die Verbindung an dieser Stelle abgebrochen.
4. Erkennt der Client das Zertifikat als gültig erzeugt der Client den symmetrischen Sitzungsschlüssel.

5. Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message: Mit dem öffentlichen Schlüssel des Servers verschlüsselt der Client den Sitzungsschlüssel und schickt ihn an den Server.
6. Mit seinem privaten Schlüssel kann der Server den verschlüsselten Sitzungsschlüssel entschlüsseln.
7. Encrypted Handshake Message, Change Cipher Spec, Encrypted Handshake Message: Der Server bestätigt den geheimen Sitzungsschlüssel.
8. Danach werden alle HTTP-Requests und -Responses verschlüsselt, bis die Verbindung abgebaut wird.

File Transfer Protocol (FTP) (Port 20 und 21 - Layer 7)

FTP ist ein Kommunikationsprotokoll, um Dateien zwischen unterschiedlichen Computersystemen zu übertragen. Die Übertragung findet nach dem Client-Server-Prinzip statt. Ein FTP-Server stellt dem FTP-Client Dateien zur Verfügung. Der FTP-Client kann Dateien auf dem FTP-Server ablegen, löschen oder herunterladen. Mit einem komfortablen FTP-Client arbeitet man ähnlich, wie mit einem Dateimanager.

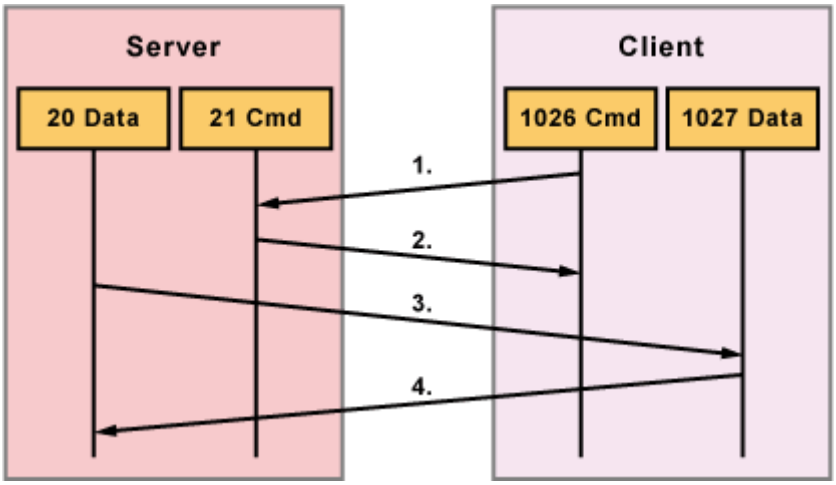
Funktionsweise

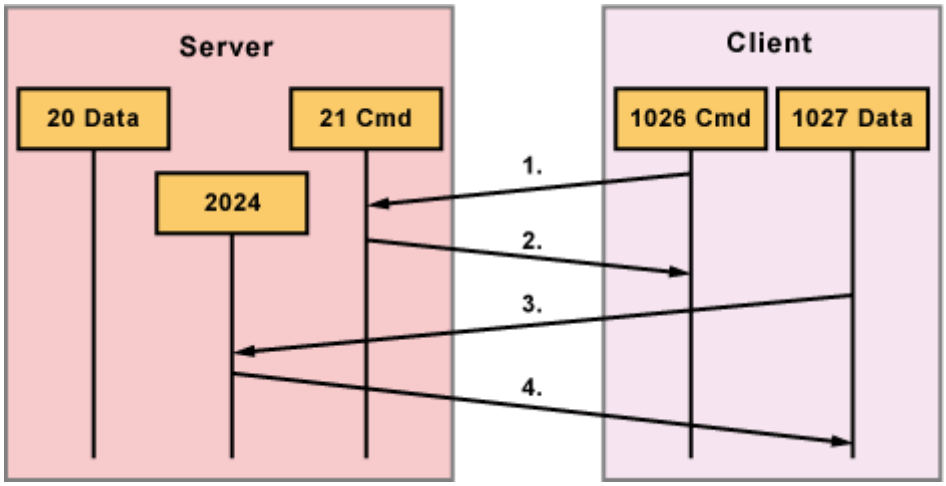


Die Kommunikation findet nach dem Client-Server-Prinzip statt. Wobei FTP zwischen Client und Server zwei logische Verbindungen herstellt. Eine Verbindung ist der Steuerkanal (command channel) über den TCP-Port 21. Dieser Kanal dient ausschließlich zur Übertragung von FTP-Kommandos und deren Antworten. Die zweite Verbindung ist der Datenkanal (data channel) über den TCP-Port 20. Dieser Kanal dient ausschließlich zur Übertragung von Daten. Über den Steuerkanal tauschen Client und Server Kommandos aus, die eine Datenübertragung über den Datenkanal einleiten und beenden.

Aktives vs. Passives FTP

Der FTP-Verbindungsaufbau sieht vor, dass der Steuerkanal vom FTP-Client zum FTP-Server aufgebaut wird. Steht der Steuerkanal wird der Datenkanal vom FTP-Server zum FTP-Client initiiert (aktives FTP). Befindet sich der FTP-Client hinter einem NAT-Router oder einer Firewall und verfügt parallel dazu nur über eine private IPv4-Adresse, dann kommt die Verbindung nicht zustande. Die Verbindungsanforderung vom Server an den Client wird von der Firewall bzw. dem Router abgeblockt, bzw. kann wegen der privaten IPv4-Adresse gar nicht geroutet werden. Für diesen Fall gibt es das passive FTP, bei dem auch der Client den Datenkanal initiiert.





Am Anfang jeder FTP-Verbindung steht die Authentifizierung des Benutzers. Danach erfolgt der Aufbau des Steuerkanals über Port 21 und des Datenkanals über Port 20. Wenn die Dateiübertragungen abgeschlossen sind, werden die Verbindungen vom Benutzer oder vom Server (Timeout) beendet.

Secure Shell (SSH) (Port 22 - Layer 7)

SSH bzw. Secure Shell ist ein **kryptografisches Protokoll** mit dem man auf einen entfernten Rechner mittels einer **verschlüsselten Verbindung über ein unsicheres Netzwerk** zugreifen kann.

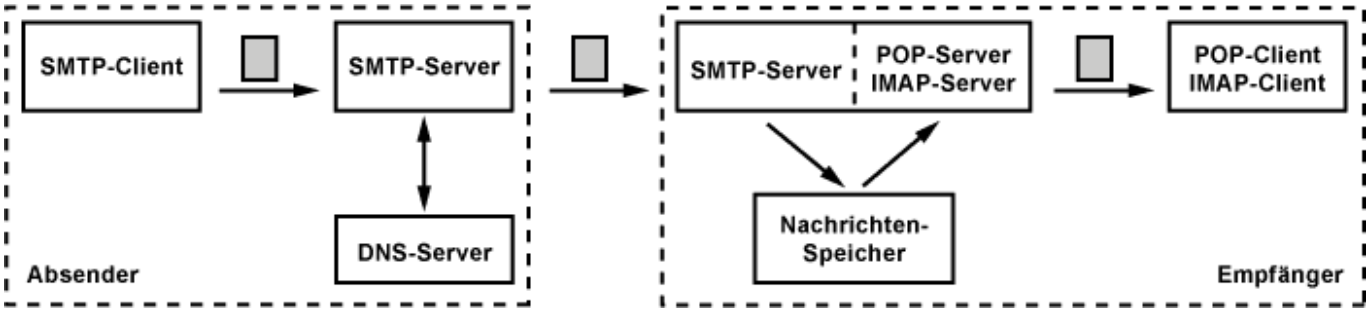
Die Shell (Kommandozeile) bietet **vollen Zugriff auf das Dateisystem und alle Funktionen** des Rechners.

Die Funktionen der Secure Shell beinhalten den Login auf entfernte Rechner, die interaktive und nicht interaktive Ausführung von Kommandos und das Kopieren von Dateien zwischen verschiedenen Rechnern eines Netzwerks. SSH bietet dazu eine kryptografisch gesicherte Kommunikation über das unsichere Netzwerk, eine zuverlässige gegenseitige Authentisierung, Verschlüsselung des gesamten Datenverkehrs auf Basis eines Passworts oder Public/Private-Key-Login-Methoden

Simple Mail Transfer Protocol (SMTP) (Port 25 - Layer 7)

SMTP ist ein **Kommunikationsprotokoll für die Übertragung von E-Mails**. Die **Kommunikation** erfolgt **zwischen** einem **E-Mail-Client und einem SMTP-Server (Postausgangsserver)** oder zwischen zwei SMTP-Server. Für den **Austausch** der E-Mails sind die **Mail Transfer Agents (MTAs)** zuständig. Untereinander verständigen sich die MTAs mit dem SMTP-Protokoll.

Neben SMTP gibt mit POP und IMAP noch zwei weitere Protokolle für den E-Mail-Austausch. Diese beiden Protokoll dienen jedoch nur dazu, um E-Mail abzuholen oder online zu verwalten. SMTP dagegen ist ein Kommunikationsprotokoll, das **E-Mails entgegennehmen und weiterleiten kann**.



Der Ablauf des E-Mail-Routings sieht in etwa so aus: Der SMTP-Server fragt einen DNS-Server ab und erhält eine Aufstellung von Mail-Servern, die E-Mails für den Ziel-SMTP-Server entgegennehmen. Jeder dieser Mail-Server (Mail Exchange) ist mit einer Priorität versehen. Der SMTP-Server versucht die Mail-Server in der vorgegebenen Reihenfolge zu kontaktieren, um die E-Mail zu übermitteln

Nachteile

- Für versendete E-Mails keine Versandbestätigung
- Geht eine E-Mail verloren, werden weder Sender noch Empfänger darüber informiert
- Nicht vorhandene Authentisierung des Benutzers beim Verbindungsaufbau zwischen SMTP-Client und SMTP-Server. Das führt dazu, dass eine beliebige Absenderadresse beim Versand einer E-Mail angegeben werden kann

Simple Mail Transfer Protocol Secure (SMTPS) (Port 465 und 587- Layer 7)

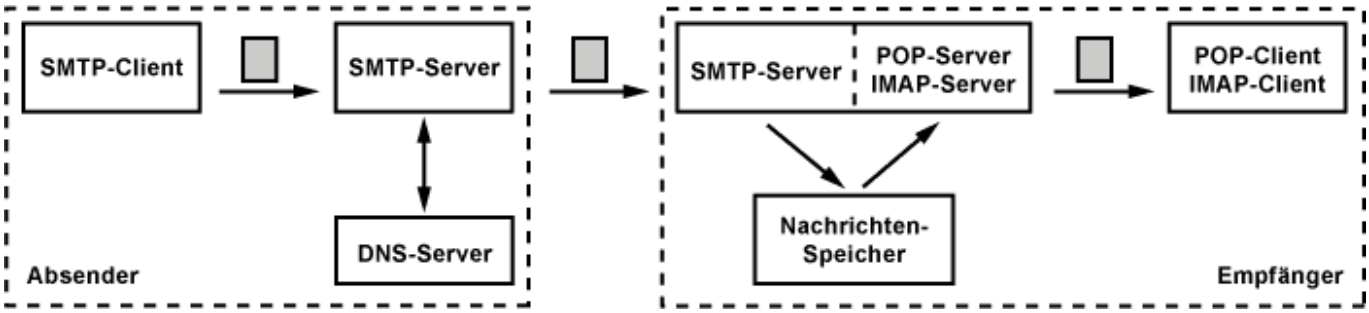
SMTPS (Simple Mail Transfer Protocol Secure) bezeichnet ein Verfahren zur **Absicherung der Kommunikation** beim E-Mail-Transport via **SMTP über SSL/TLS** und ermöglicht dadurch **Authentifizierung der Kommunikationspartner** auf Transportebene sowie **Integrität und Vertraulichkeit** der übertragenen Nachrichten.

SMTPS ist **kein eigenes Protokoll** und auch keine Erweiterung von SMTP, da es **vollkommen transparent und unabhängig** von diesem auf der Transportschicht arbeitet.

Das bedeutet, dass die Verbindung, über die SMTP abgewickelt wird, softwaremäßig mit den Verfahren **SSL oder TLS abgesichert** wird. Dies geschieht direkt beim Verbindungsaufbau, noch bevor irgendwelche Maildaten ausgetauscht werden. Da also die Verwendung der Sicherungsschicht nicht verhandelt wird, sind SMTPS-Dienste in der Regel auf einem eigenen TCP-Port erreichbar.

Post Office Protocol Version 3 (POP3) (Port 110 - Layer 7)

POP ist ein Kommunikationsprotokoll, um **E-Mails von einem Posteingangsserver (POP-Server) abzuholen**. Die **Kommunikation** erfolgt **zwischen** einem **E-Mail-Client** und einem **E-Mail-Server (Posteingangsserver)**. Das Protokoll, das diesen Zugriff regelt, nennt sich POP (aus dem Jahr 1984), das in der aktuellen Version 3 vorliegt, und deshalb manchmal auch als POP3 bezeichnet wird.



Per Fernzugriff werden die gespeicherten E-Mails abgerufen und auf dem lokalen Computer gespeichert. POP sieht das Prinzip der Offline-Verarbeitung von E-Mails vor. Online werden die E-Mails vom Posteingangsserver vom E-Mail-Client heruntergeladen. Wenn sich darunter E-Mails mit einem großen Dateianhang befinden, kann der Download schon mal etwas länger dauern. Erst nach erfolgreichem und vollständigem Zugriff werden die E-Mails auf dem Server gelöscht. Die Bearbeitung der eingegangenen E-Mails erfolgt anschließend auf dem lokalen Computer des Benutzers ohne Verbindung (offline) POP-Server.

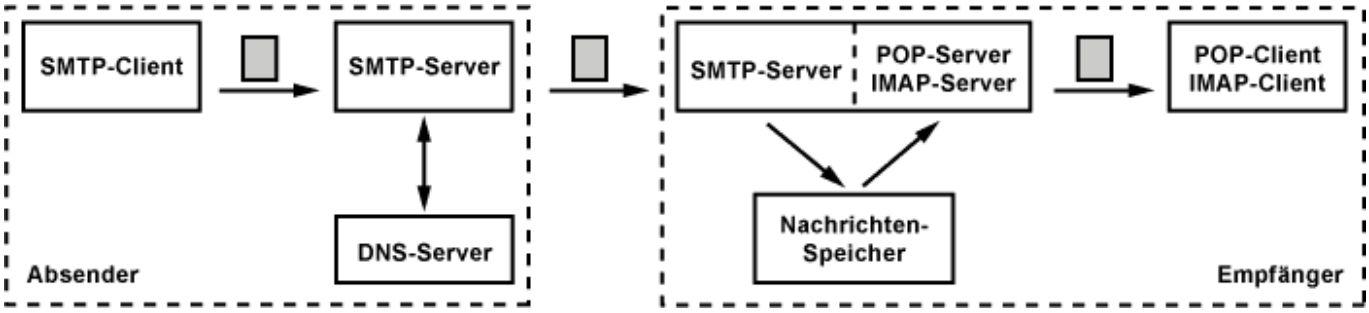
Die Verbindung zwischen POP-Server und E-Mail-Client erfolgt über TCP auf Port 110.

Post Office Protocol Version 3 Secure (POP3S) (Port 995 - Layer 7)

POP3S bezeichnet ein Netzwerkprotokoll zur Erweiterung des E-Mail-Übertragungsprotokolls POP3 um eine Verschlüsselung durch SSL/TLS. Üblicherweise wird für POP3S TCP auf Port 995 genutzt.

Internet Mail Access Protocol (IMAP) (TCP-Port 143 - Layer 7)

IMAP ist ein Kommunikationsprotokoll, um **E-Mails** auf einem **entfernten Server** ähnlich wie Dateien **zu verwalten**. Dabei **bleiben alle E-Mails auf dem IMAP-Server**. Erst wenn eine E-Mail gelesen werden soll, wird sie heruntergeladen.



IMAP erlaubt den Zugriff auf eine Mailbox, **ähnlich wie mit POP**. Der entscheidende **Unterschied** zwischen beiden Protokollen ist der **Online-Modus von IMAP**, über den der **E-Mail-Client ständig in Verbindung mit dem E-Mail-Server** steht. Während einer IMAP-Sitzung kann auf einzelne E-Mails zugegriffen werden, die so lange auf dem Server bleiben, bis sie gelöscht werden. Dadurch kann von überall auf dem Server zugegriffen werden. Auch mit einem Endgerät, das nur mit geringer Bandbreite am Netzwerk angeschlossen ist. Die **E-Mails werden nur dann heruntergeladen, wenn der Anwender diese Lesen will**. E-Mails mit einem großen Dateianhang verstopfen dann nicht mehr ungewollt den Zugang zum Netzwerk.

Internet Mail Access Protocol Secure (IMAPS) (TCP-Port 143 - Layer 7)

Bei der Verwendung von IMAPS wird die Verbindung zum Server bereits während des Verbindungsaufbaus durch SSL verschlüsselt. Damit der Server das erkennt, muss ein anderer Port verwendet werden. Dafür wurde der Port 993 reserviert.

Nach dem Aufbau der SSL-Verbindung wird IMAP verwendet. Die SSL-Schicht ist für das IMAP-Protokoll transparent, d. h., es werden keine Änderungen am IMAP-Protokoll vorgenommen.

Betriebssysteme

Einführung Betriebssysteme

Wiederholung Hardware

Wortherkunft

„Hardware“ kommt ursprünglich aus dem Englischen und bedeutet übersetzt Eisenwaren.

Hardware vs. Software

- Hardware = sind alle greifbaren/sichtbaren Elemente eines PCs
- Software = Programme und Daten, also nicht greifbare Elemente eines PCs

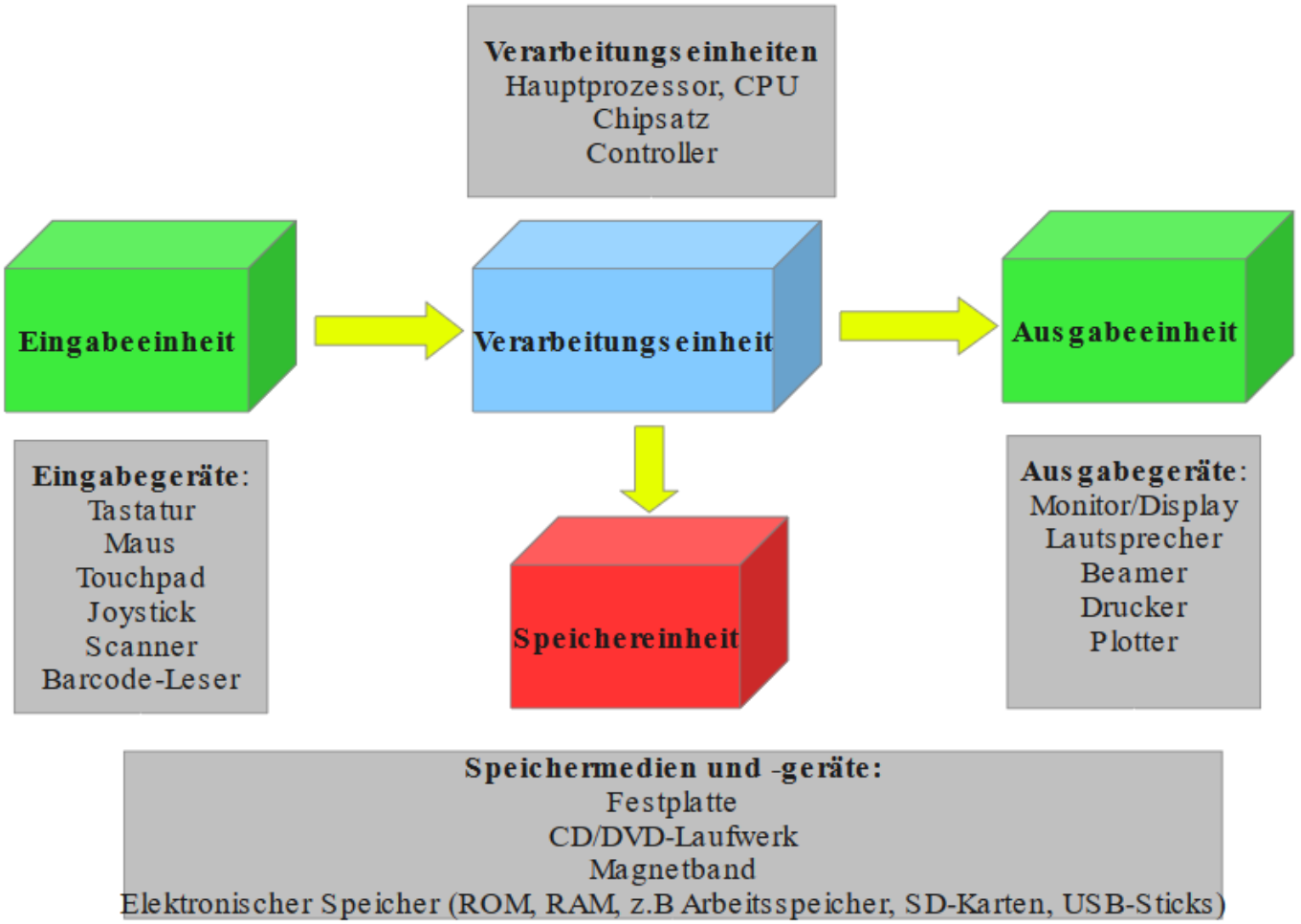
Komponenten

- **Grundbestandteile**
  - Motherboard/Mainboard - Hauptplatine
  - Central Processing Unit (CPU) - zentrale Recheneinheit (Prozessor)
  - Random Access Memory (RAM) - Arbeitsspeicher
- **Massenspeicher**
  - Festplatte (SSD, SATA)
  - Laufwerke (CD, DVD, Band,..)
- **Erweiterungskarten** (optional)
  - Grafikkarte
  - Soundkarte
  - Netzwerkkarte
- **Peripheriegeräte**
  - Eingabegeräte
    - Maus & Tastatur
    - Scanner
  - Ausgabegeräte
    - Bildschirm
    - Drucker

EVA/IPO Prinzip

Das **EVA**-Prinzip ist ein Grundprinzip für die Datenverarbeitung und beschreibt die Reihenfolge in der Daten verarbeitet werden.

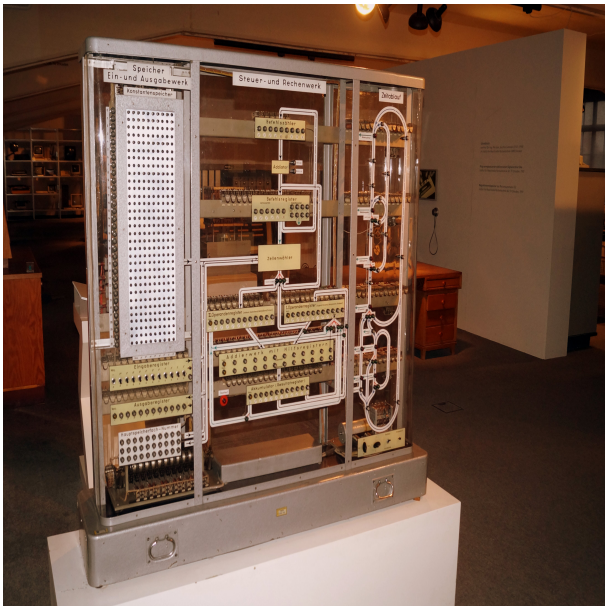
Eingabe (**I**nput) - **V**erarbeitung (**P**rocess) - **A**usgabe (**O**utput)



EVA Prinzip

Von-Neumann-Architektur (VNA)

Die **V**on-**N**eumann-**A**rchitektur ist ein Modell für Computer und bietet die Grundlage für alle heutigen Computer. Das Modell wurde von Johann von Neumann im Jahr 1945 entwickelt. Heute ist Johann von Neumann unter seinem amerikanischen Namen John von Neumann bekannt.



Er revolutionierte die bisherigen Computer, da durch seine Architektur verschiedene Programme auf derselben Hardware laufen konnten. Seine Struktur hat es erstmals ermöglicht, dass mehrere Probleme von ein und derselben Hardware gelöst werden können.

Komponenten

Ein Von-Neumann-Rechner beruht auf folgenden Komponenten, die bis heute in Computern verwendet werden:

- ALU (Arithmetic Logic Unit) – Rechenwerk, selten auch Zentraleinheit oder Prozessor genannt, führt Rechenoperationen und logische Verknüpfungen durch. (Die Begriffe Zentraleinheit und Prozessor werden im Allgemeinen in anderer Bedeutung verwendet.)
- Control Unit – Steuerwerk oder Leitwerk, interpretiert die Anweisungen eines Programms und verschaltet dementsprechend Datenquelle, -senke und notwendige ALU-Komponenten; das Steuerwerk regelt auch die Befehlsabfolge.
- Bussystem: Dient zur Kommunikation zwischen den einzelnen Komponenten (Steuerbus, Adressbus, Datenbus)
- Memory – Speicherwerk speichert sowohl Programme als auch Daten, welche für das Rechenwerk zugänglich sind.
- I/O Unit – Eingabe-/Ausgabewerk steuert die Ein- und Ausgabe von Daten, zum Anwender (Tastatur, Bildschirm) oder zu anderen Systemen (Schnittstellen).

Register

- **Befehlszähler (Program Counter - PC)**

Enthält die Speicheradresse vom Speicherwerk des aktuellen Befehls. (Startadresse = 0x0000). Wird nach jedem Befehl um 1 erhöht.

- **Befehlsregister (Instruction Register - IR)**

Speichert den vom Speicherwerk zurückbekommenen Befehl.

- **Speicheradressregister (Memory Address Register - MAR)**

Ausschließlich für die Kommunikation zwischen Steuerwerk und Rechenwerk. Im MAR legt das Steuerwerk jeweils die Adresse ab, welche im Speicherwerk angesprochen werden soll.

- **Speicherdatenregister (Memory Data Register - MDR)**

Ausschließlich für die Kommunikation zwischen Steuerwerk und Rechenwerk. Bei einem Lesezugriff auf die Speicherzelle wird der vom Speicherwerk über den Datenbus bereitgestellte Wert im MDR abgelegt und kann von hier aus weiter verarbeitet werden. Bei einem Schreibzugriff muss sich im MDR der zu schreibende Wert befinden, so dass er über den Datenbus an das Speicherwerk übermittelt werden kann.

Prozesszyklus

1. Initialisiere das Befehlszählerregister (Program Counter- PC) mit 0 (Start)
2. Sende Adresse des aktuellen Befehls zum Speicherwerk
3. Empfange aktuellen Befehl vom Speicherwerk und speichere diesen in das Befehlsregister (Instruction Register - IR)
4. Analysiere aktuellen Befehl und treffe Vorbereitungen für die spätere Ausführung (Welcher Befehl und was ist dazu notwendig?)
5. Erhöhe den Befehlszähler (PC) um 1
6. Rufe zusätzlich benötigte Operanden ab (z.B.: Befehl ADD OP1 OP2)
7. Führe den Befehl selbst (Steuerwerk) aus oder beauftrage das Rechenwerk für die Ausführung
8. Schreibe das Ergebnis des ausgeführten Befehls an die vorgesehene Stelle

Arbeitsweise des Steuer- und Rechenwerks

Arbeitsweise des Speicherwerks

Befehlszähler und Befehlsregister im Zusammenspiel mit dem Bus-System

Arbeitsweise des Steuerwerks

Die 7 Prinzipien der Von-Neumann-Architektur

- Rechner besteht aus fünf Funktionseinheiten
- Struktur des Rechners ist unabhängig vom zu bearbeitenden Problem. Zur Lösung eines Problems muss Programm im Speicher abgelegt werden.
- Programme, Daten und Ergebnisse werden im selben Speicher abgelegt.
- Der Speicher ist in fortlaufenden nummerierten Zellen unterteilt. Über die Adresse einer Speicherzelle kann auf den Inhalt zugegriffen werden.
- Aufeinanderfolgende Befehle eines Programms werden in aufeinanderfolgende Speicherzellen abgelegt.
- Durch Sprungbefehle kann von der gespeicherten Befehlsreihenfolge abgewichen werden.
- Es gibt zumindest
    - arithmetische Befehle (Addition, Subtraktion, Multiplikation)
    - logische Befehle (EQUAL, NOR, AND, OR)
    - Transportbefehle, z.B. von Speicher zu Rechenwerk und für Ein- und Ausgabe
- Alle Daten (Befehle, Adressen, usw.) werden binär kodiert

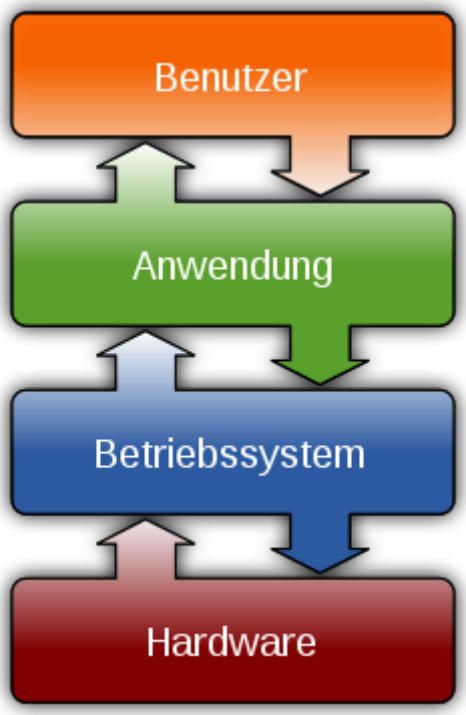
Zusammenhang Hardware - Betriebssysteme

Grundlegendes

Ein Rechner besteht zunächst aus den sichtbaren und greifbaren Komponenten der Hardware. Damit aber mit dem Rechner auch gearbeitet werden kann, benötigt er noch geeignete Programme. Nach dem Von-Neumannschen Prinzipien ist ein Rechner als Universalrechner konstruiert. Das heißt die Erledigung unterschiedlicher Aufgaben wird ausschließlich über Programme (auch Software genannt) gesteuert.

Da sich aber nicht jeder Softwareentwickler um die Details zur Veraltung der einzelnen Hardwarekomponenten (z.B.: Steuerungsinformationen für Plattenspeicher, Speicherverwaltung, Prozessverwaltung etc.) kümmern sollte, erscheint es sinnvoll diese Komponenten zentral und damit allgemein verfügbar bereitzustellen. Andernfalls würde jeder Entwickler viel Zeit verschwenden.

Somit entwickelte man eine Ebene (=Betriebssystemebene) auf der verschiedenste Verwaltungsprogramme bereits laufen und allen Anwendern bzw. Anwendungsprogrammen eine vereinfachte Sicht auf die Hardware zur Verfügung stellen.



Folgendes gilt, dass das Betriebssystem quasi der Mittler (=Übersetzer) zwischen Hardware und Benutzer ist. Dadurch kann die Komplexität der darunterliegenden Systemarchitektur verborgen werden und dem Anwender wird eine einfache und verständliche Schnittstelle (Interface) angeboten. Er kann sich somit voll und ganz auf die Implementierung seiner Funktion konzentrieren und muss sich nicht mit komplexen Maschinenbefehlen herumschlagen.

Des Weiteren verwaltet das Betriebssystem die Ressourcen des Rechners, also alle physikalischen Geräte (Prozessoren, Speicher, Grafikkarte,..). Es teilt also die Ressourcen des Gesamtsystems gerecht an die konkurrierenden Anwenderprogramme auf. Ohne einer solchen Aufteilung wäre eine

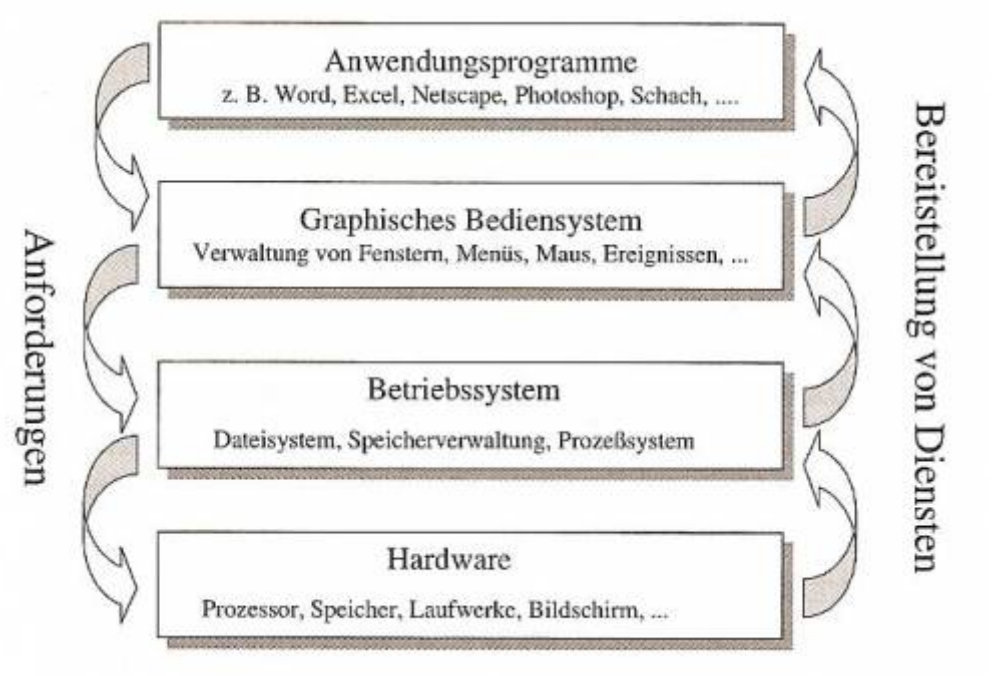
sinnvolle Benützung des PCs nicht möglich.

Als normaler Computeranwender befinden wir uns auf der **Benutzerebene**, wo der Rechner als Arbeitsgerät auch für technische Laien dienen muss.

Zwischen dem vom Anwender intuitiv zu bedienenden Rechner und den Fähigkeiten der Hardware klafft eine riesige Lücke, die vom

- Betriebssystem (Datei-, Prozess- und Speicherverwaltung) und dem
- grafischen Bediensystem (Menüs, Fenster, Maus)

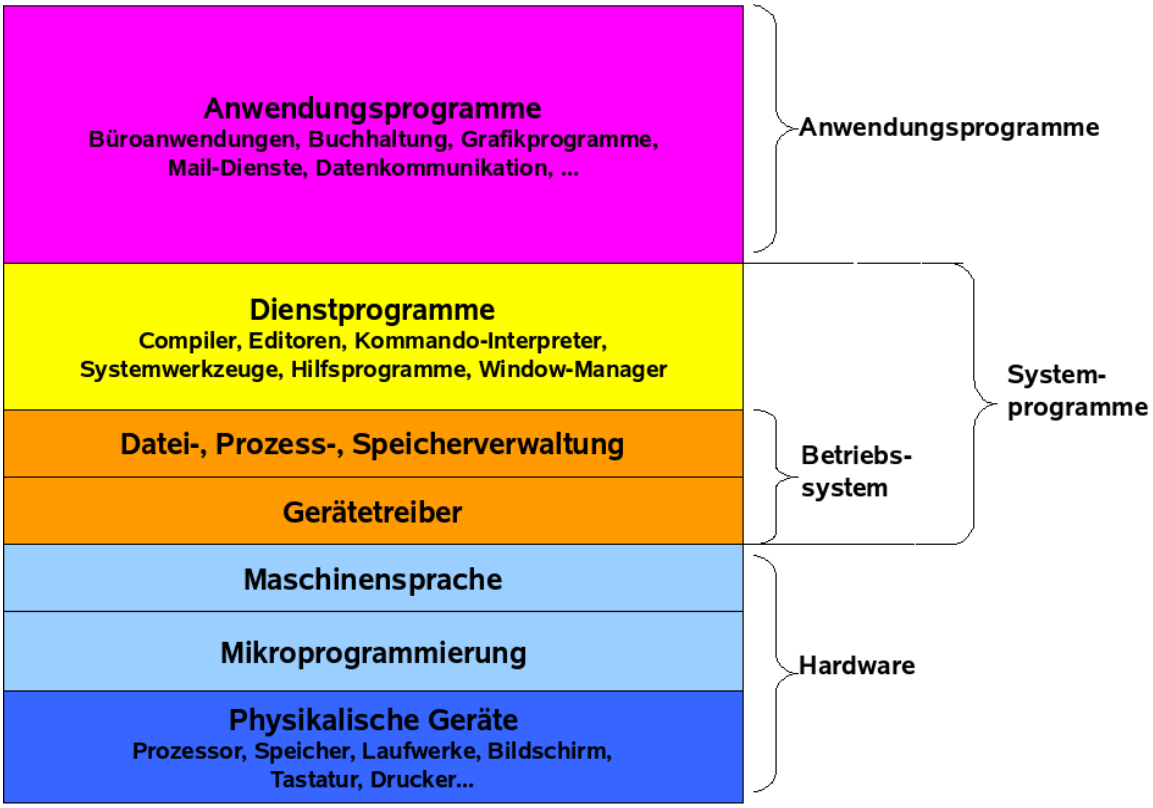
überbrückt wird.



Jede Schicht fordert von der niedrigeren Schicht Dienste an. Diese wiederum benötigt zur Erfüllung der Anforderungen selber Dienste von der nächsten Schicht. Auf diese Weise setzen sich die Anforderungen in die tiefen Schichten fort, bis die Hardware zu geeingeten Aktionen veranlasst wird.

Anders betrachtet, bietet jede Schicht der jeweils höherliegenden Schicht Dienste an.

Schichten eines Computersystems



Physikalische Geräte

Die unterste Schichte enthält die physikalischen Geräte. Sie besteht aus integrierten Schaltungen, Drähten, Stromversorgung usw.

Mikroprogrammierung/Mikrocode

Aufgabe des Mikrocodes ist die direkte Steuerung der physikalischen Geräte. In Microcode entworfene Programme werden von einem Interpreter in entsprechende Steueranweisungen übersetzt und an die Geräte übermittelt. Die Instruktionen der Maschinensprache (z.B.: ADD, MOVE, JUMP,...) werden in einzelne kleine Microcodeprogramme übersetzt.

Maschinensprache

Die Menge von Instruktionen die das Mikroprogramm ausführen kann wird als Maschinensprache bezeichnet. Maschinensprache besteht aus zahlreichen elementaren Maschinenbefehlen die im Prozessor in Microcode zu einem Programm übersetzt und ausgeführt werden. Wichtig ist, dass jeder Maschinenbefehl durch ein festgelegtes Mikroprogramm implementiert bzw. fest verdrahtet ist.

Da die Maschinensprache hardwarespezifisch ist, wird sie noch der Hardware zugeordnet.

Betriebssystem

Das Betriebssystem vermindert die Komplexität der Hardware und deren Verwaltung um dem Programmierer eine angemessene Menge an Instruktionen zur Verfügung zu stellen, mit denen er effizienter arbeiten kann.

Systemprogramme / Dienstprogramme

Das sind vom Betriebssystemhersteller mitgelieferte Programme wie z.B.: Window-Manager, Compiler für das Kompilieren von C-Programmen, Editoren, Systemwerkzeuge wie z.B.: Taskmanager, Explorer ...

Anwenderprogramme

Die oberste Schicht stellt den Benutzern des Systems Anwendersoftware zur Verfügung. Diese Programme setzen entweder auf den Dienstprogrammen oder unmittelbar auf dem Betriebssystem auf. Beispiele dafür sind:

- Browser
- Microsoft Office
- ....

Hauptaufgaben von Betriebssystemen

1) Abstraktion (Vereinfachung) der Hardware --> abstrakte Programmierschnittstelle

2) Verwaltung von Systemressourcen (CPU, Speicher, Netzwerk, Drucker, Platten..)

- **Prozessverwaltung**
- **Speicherverwaltung**
- **Dateiverwaltung**

Der Rechner mit seinen Peripheriegeräten stellt eine Fülle von Ressourcen zu Verfügung, wie CPU, Hauptspeicher, Plattenspeicher, externe Geräte. Die Verwaltung dieser Ressourcen ist Aufgabe des Betriebssystems.

Der Aufruf eines Programms führt oft zu vielen gleichzeitig und unabhängig voneinander ablaufenden Teilprogrammen, auch **Prozesse** genannt.

Ein Prozess ist also ein eigenständiges Programm mit eigenem Speicherbereich, der vor dem Zugriff durch andere Prozesse geschützt ist. Allerdings ist es erlaubt, dass verschiedene Prozesse untereinander *kommunizieren* , also Daten untereinander austauschen.

Dabei hat das Betriebssystem die Aufgabe, die *Kommunikation* zwischen diesen Prozessen zu verwalten, damit sich die gleichzeitig aktiven Prozesse nicht untereinander beeinträchtigen oder gar zerstören. Das Betriebssystem verwaltet unter anderem also gleichzeitig aktive Prozesse, so dass einerseits keiner benachteiligt wird, andererseits aber kritische Prozesse mit Priorität behandelt werden.

Selbstverständlich können Prozesse aber nicht wirklich gleichzeitig ablaufen. Das Betriebssystem erzeugt aber eine scheinbare Parallelität dadurch, dass jeder Prozess immer wieder eine kurze Zeitspanne (wenige Millisekunden) an die Reihe kommt, dann unterbrochen wird, während andere Prozesse bedient werden. Nach kurzer Zeit ist der unterbrochene Prozess wieder an der Reihe und setzt seine Arbeit fort. Diesen Vorgang nennt man **Multitasking**.

Ähnlich verhält es sich mit der Verwaltung des Hauptspeichers, in dem nicht nur der Programmcode, sondern auch die Daten der vielen Prozesse gespeichert werden. Neuen Prozessen muss freier Hauptspeicher zugeteilt werden, der Speicher beendeter Prozesse muss wiederverwendet werden.

Die dritte wichtige Aufgabe ist die **Dateiverwaltung**. Damit ein Benutzer sich nicht darum kümmern muss, in welchen Sektoren auf welchen Spuren noch Platz ist um den gerade geschriebenen Text zu speichern, stellt das Betreibssystem das Konzept „Datei“ als Behälter für Daten aller Art zur Verfügung. Die Übersetzung von Dateien und ihren Namen in bestimmte Spuren, Sektoren und Köpfe der Festplatte nimmt das *Dateisystem* als Bestandteil des Betriebssystem vor.

Eigenschaften von Betriebssystemen

- Betriebssystem muss **änderbar** sein (durch neue Hardware, Software bzw. bei Sicherheitslücken)
- **modular** und **klar strukturiert** aufgebaut
- gut **dokumentiert** –> Schnittstellenbeschreibung!!

Ziele von Betriebssystemen

- **Bequemlichkeit** - aus Sicht des Benutzers
- **Effizienz** - Nutzung der Ressourcen
- **Entwicklungsfähigkeit** - Neue Dienste

Betriebssystemarten

Man unterscheidet mehrere Arten von Betriebssystemen, die im Laufe der Zeit entstanden sind.

Mainframe-Betriebssysteme

- Betriebssysteme für Großrechner
- Einsatz: Webserver, Datenbankserver, E-Commerce, Business-to-Business (B2B)
- Viele Prozesse gleichzeitig mit hohem Bedarf an schneller Eingabe/Ausgabe
- Beispiele sind MVS, OS/390, z/OS 2.1 –> zumeist IBM-Großrechner

Server-Betriebssysteme

- Betriebssysteme die auf Servern laufen
- Bieten verschiedene Dienste im Netzwerk an (Dateidienste, Webdienste,...)
- Dienste stehen vielen Benutzern im Netz zur Verfügung z.B.: Netzlaufwerk
- Aktuelle Beispiele sind Windows Server 2016 bzw. Red Hat Enterprise Linux 7.2

PC(Desktop)-Betriebssysteme

- Betriebssystem für Personalcomputer
- Meist nur 1 oder wenige Benutzer (über Netzwerk)
- Einsatz: Programmierung, Textverarbeitung, Spiele, Internetzugriff
- Mehrere Programme pro Benutzer → quasi-parallel
- Aufteilung der Prozesse auf vorhandene Kerne
- Zuteilung der Systemressourcen
- Beispiele: Linux, Windows, Mac OS X

Echtzeit-Betriebssysteme

- Einhaltung von harten Zeitbedingungen
- Einsatz: Steuerung von maschinellen Fertigungsanlagen, Steuerung von Ampeln, Robotorsteuerung...
- Aktion in einem fest vorgegebenen Zeitintervall

Eingebettete Systeme / Embedded Systems

- = Computer die man nicht unbedingt gleich sieht
- Einsatz: Fernseher, Handy, Auto,...
- Meist Echtzeitanforderungen

- Wenig Ressourcen zur Verfügung (geringer Stromverbrauch, wenig Arbeitsspeicher)
- Beispiele: iOS, Android, Windows Phone

Historische Entwicklung von Betriebssystemen

Arbeitsauftrag

Recherchiere im Internet hinsichtlich den historischen Entwicklungen von Betriebssystemen.

Beschreibe und erkläre die folgenden Entwicklungsstadien (= 4 Generationen) und gib die ungefähre Zeitspanne an:

1. Generation - Serielle Systeme
2. Generation - Einfache Stapelvearbeitungssysteme
3. Generation - ICS, Multiprogramming, Timesharing
4. Generation - Personal Computer

Das Ergebnis soll ein Word-Dokument (gegliedert in den 4 Generationen) im Umfang von 2-4 Seiten exklusive Titelblatt (Download unter

Titelblatt

- vor der Abgabe bitte den Namen im Titelblatt und den Dateinamen auf Betriebssysteme_NACHNAME.docx ändern) sein (d.h. mindestens 3 Seiten).

Letzter Abgabetermin ist der 21.02.2019 @ 23:59.

Die bis dahin abgegebenen Arbeitsaufträge werden bewertet und zählen zur Teilnote Praktische Arbeiten (PA)!!

Nicht abgegebene Arbeitsaufträge werden negativ beurteilt.

Abgabe unter

Google Classroom

ACHTUNG: Die Musterlösung findet ihr in Google Classroom!

Formatierung von Datenträgern

Stufen der Formatierung

Wenn man von Formatierung spricht, unterscheidet man dabei mehrere Stufen:

1. **Low-Level-Formatierung** = physikalische Einteilung eines Speichermediums
2. **Partitionierung** = logische Einteilung des Speichermediums in zusammenhängende Strukturen
3. **High-Level-Formatierung** = die logische Einteilung der Partitionsstruktur mit einem Dateisystem durch eine Software (meist durch das Betriebssystem).

Soweit durch die Beschaffenheit, industrielle Standards oder durch spezielle Verwendung die physikalische Einteilung des Mediums als Norm feststeht, ist eine separate Low-Level-Formatierung nicht erforderlich. In diesen Fällen können beide Vorgänge gleichzeitig vorgenommen werden, so zum Beispiel bei Disketten, CD-ROM, CD-RW oder DVD-ROM/RW.

1) Physikalische Formatierung (Low-Level-Formatierung)

Damit eine Festplatte logisch formatiert werden kann, muss sie physikalisch formatiert sein.

Die physikalische Formatierung eines Festplattenlaufwerks (auch als Zwei-Stufen-Formatierung bezeichnet) wird in der Regel vom Hersteller durchgeführt.

Durch die physikalische Formatierung wird eine Festplatte in ihre physikalischen Grundbausteine unterteilt: **Spuren**, **Sektoren** und **Zylinder**. Durch diese Elemente wird die Art und Weise vorgegeben, mit der Daten physikalisch auf der Festplatte aufgezeichnet und von dort gelesen werden können.

Spuren

Die **Spuren** sind konzentrische Kreispfade, die auf jede Scheibenseite geschrieben werden, wie auf einer Schallplatte oder einer CD. Die Spuren werden mit einer Nummer identifiziert, die mit Spur 0 am äußeren Rand einsetzt.

Zylinder

Der Spurensatz, der auf allen Seiten der Platte im gleichen Abstand von der Mitte angelegt ist, wird als "**Zylinder**" bezeichnet. Hardware und Software arbeiten häufig mit diesen Zylindern.

Sektoren

Die Spuren sind in Bereiche, sogenannte "**Sektoren**" oder „**Blöcke**" unterteilt, in denen eine

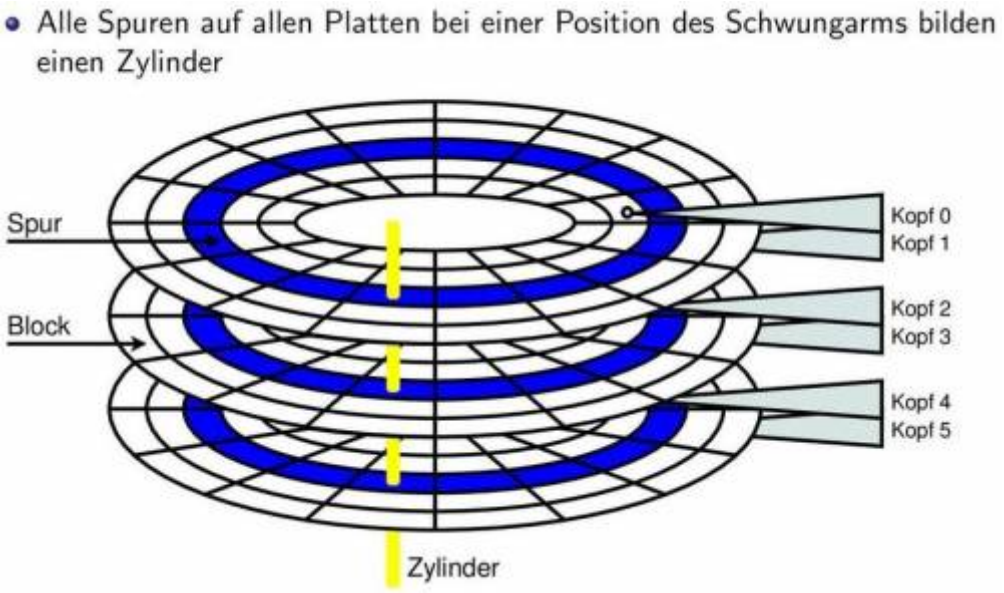
festgeschriebene Datenmenge gespeichert wird. Die Sektoren werden in der Regel für eine Speicherkapazität von 512 Byte (ein Byte besteht aus 8 Bit) formatiert.

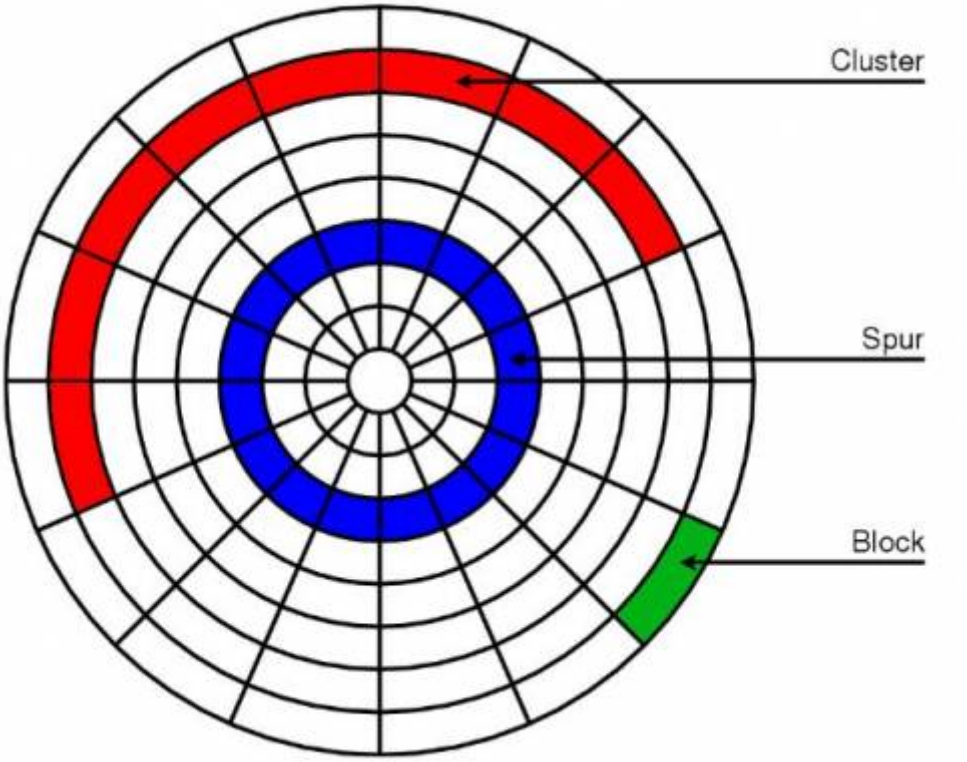
Fehlerhafte Sektoren

Nachdem ein Festplattenlaufwerk physikalisch formatiert wurde, kommt es gelegentlich vor, dass die magnetischen Eigenschaften der Eisenoxydschicht an einigen Stellen auf der Festplatte nach und nach an Wirksamkeit verlieren. Dies hat zur Folge, dass es für die Schreib-/Leseköpfe der Festplatte in zunehmenden Maße schwieriger wird, ein Bit-Muster auf der Festplatte zu speichern, das später von der Festplatte gelesen werden kann.

Wenn dieser Prozess einsetzt, bezeichnet man die Sektoren, in denen Daten nicht mehr zuverlässig gespeichert werden können, als “fehlerhafte Sektoren”. Glücklicherweise ist die Qualität moderner Festplatten so hoch, dass solche “fehlerhaften Sektoren” nur sehr selten auftreten.

Darüber hinaus sind die modernen Computer normalerweise in der Lage, einen fehlerhaften Sektor zu identifizieren und ihn als solchen zu kennzeichnen, damit er nicht mehr genutzt wird; es wird dann ein alternativer Sektor benutzt.





Adressierung der Daten auf einer (magnetischen) Festplatte

CHS - Adressierung

Bei Festplatten mit einer Kapazit< 8 GB werden die einzelnen Sektoren mit der sogenannten "**Zylinder-Kopf-Sektor-Adressierung(Cylinder-Head-Sector-Adressierung)**" durchgeführt.

Jeder Sektor lässt sich mit **CHS** klar lokalisieren und adressieren.

Einschränkungen durch das BIOS

- Zylinder = 10 Bits
- Kopf = 4 Bits –> später 8 Bits (erweitertes CHS)
- Sektor/Spur = 8 Bits

Maximaler Speicherplatz früher: 1.024 Zylinder * 16 Köpfe * 63 Sektoren/Spur * 512 Byte/Sektor = 528.482.304 Byte = 504 MB

Maximaler Speicherplatz heute: 1.024 Zylinder * 255 Köpfe * 63 Sektoren/Spur * 512 Byte/Sektor = 8.422.686.720 Byte = 7,844 GB

- Größe einer Spur berechnen?
- Größe einer Scheibe berechnen?

LBA- Adressierung

Seit der Einführung der logischen Blockadressierung (**Logical Block Addressing (LBA)**) 1995 werden alle Sektoren auf der Festplatte von 0 beginnend durchnummeriert. Durch LBA wurde das

BIOS - und CHS-Korsett umgangen und es kann damit jede derzeit gängige Festplatte voll adressiert werden. Die Zahlen zu Zylinder, Köpfen und Sektoren auf der Festplatte haben heute nichts mehr mit der tatächlichen Lage der Sektoren auf der Festplatte zu tun. Es handelt sich um eine logische Plattengeometrie, die nur aus Kompatibilitätsgründen existiert

2) Partitionierung von Festplatten

Eine Partition ist ein zusammenhängender Bereich einer Festplatte, der in der Regel ein Dateisystem enthält.
Wenn man einen PC oder ein Notebook mit vorinstalliertem Windows kaufen, enthält die Festplatte zumeist zwei Partitionen: eine winzige Partition mit Windows-Boot-Dateien und eine zweite Partition, die den Rest der Festplatte füllt und Windows enthält. Unter Umständen kann es auch weitere Partitionen geben, die beispielsweise ein Recovery-System enthalten.

Um mehrere Betriebssysteme (Windows, Linux etc.) auf einem Rechner zu installieren, benötigt man mehrere Partitionen. Für jedes Betriebssystem ist mindestens eine Partition erforderlich; für Linux sind sogar mehrere Partitionen sinnvoll.

Master-Boot-Record (MBR)

Der Master-Boot-Record ist der **erste Sektor** auf der Festplatte. Er beinhaltet zum einen den „**Bootstrap**“. Dies ist ein Programm, das von dem BIOS aufgerufen wird, um das eigentliche Betriebssystem zu laden. Zum anderen enthält dieser Sektor auch eine Beschreibung, ob / wie die Festplatte in unterschiedliche Bereiche (Partitionen) unterteilt ist. Diese Beschreibung erfolgt in der sogenannten „Partitionstabelle“. Sie enthält für jede Partition einen Eintrag. Dieser besteht aus der Lage der Partition auf der Festplatte und dem „Typ“ dieser Partition.
Die Master-Boot-Record Partitionstabelle ist eine Tabelle, welche die Unterteilung der Festplatte in Partitionen beschreibt. Sie ist seit der Verbreitung von Festplatten im Jahr 1980 fest definiert und wird von allen Betriebssystemen zwingend vorausgesetzt.

Aus historischen Gründen kann diese Partitionstabelle nur vier Einträge (primäre Partitionen oder erweiterte Partitionen) aufnehmen. Das Format dieses Master-Boot-Records (Bootstrap / Partitionstabelle) ist fest definiert und wird von allen Betriebssystemen zwingend vorausgesetzt.

Partitionsarten

Es gibt im wesentlichen zwei Partitionsarten: **Primärpartitionen** und **erweiterte Partitionen**. Darüber hinaus lassen sich erweiterte Partitionen noch weiter in **logische Partitionen** unterteilen. Sie können bis zu vier Hauptpartitionen auf dem Festplattenlaufwerk definieren, wovon eine als erweiterte Partition definiert werden kann, d.h. maximal vier Primärpartitionen oder drei Primärpartitionen und eine erweiterte Partition.

a) Primärpartitionen

In einer Primärpartition können Sie jedes beliebige Betriebssystem sowie Datendateien wie z.B. Programm- und Benutzerdateien speichern. Eine Primärpartition ist logisch formatiert, damit sich ein Dateisystem darauf definieren lässt, das mit dem auf der Partition installierten Betriebssystem kompatibel ist.

Wenn Sie mehrere Betriebssysteme auf Ihrem Festplattenlaufwerk installieren müssen, ist es in der Regel notwendig, mehrere Primärpartitionen zu erstellen, da die meisten Betriebssysteme (vor allem Windows) nur von einer Primärpartition gebootet werden können. Pro Festplatte ist es möglich 4 primäre Partitionen einzurichten, ohne den Bootsektor der Festplatte anzupassen.

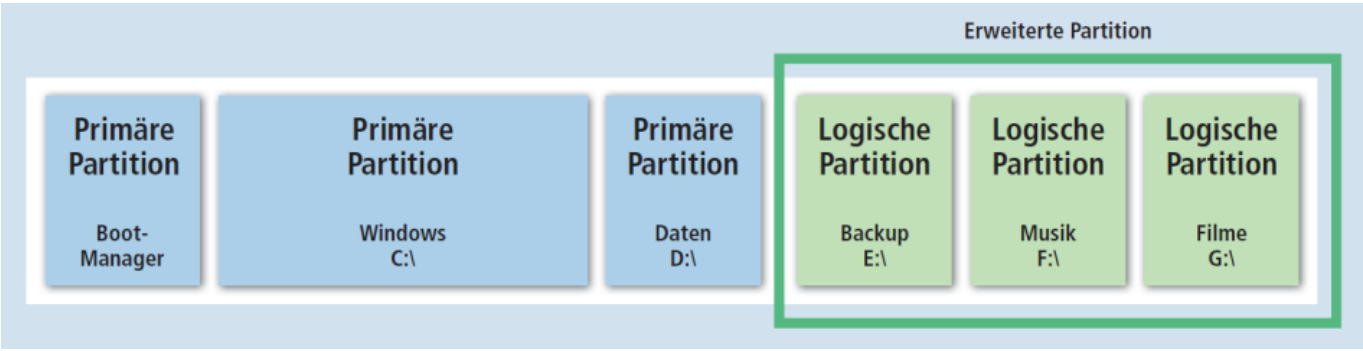
b) Erweiterte Partitionen

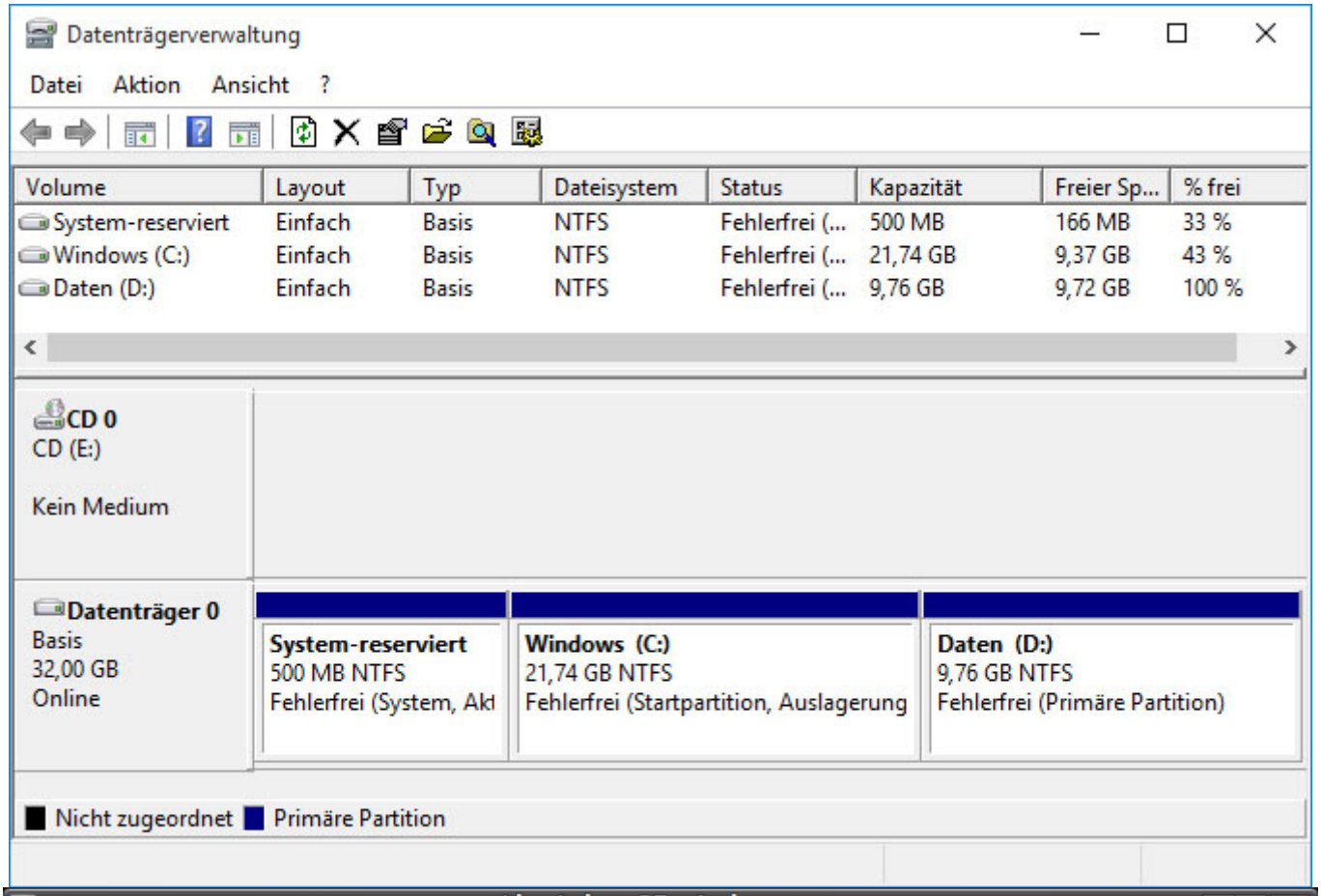
Die erweiterte Partition wurde entwickelt, um die willkürliche 4-Partitionen-Begrenzung zu umgehen. Es handelt sich im Prinzip um einen Bereich, in dem Sie den Datenträgerspeicher weiter physikalisch unterteilen können, indem Sie eine unbegrenzte Anzahl logischer Partitionen definieren (weitere physikalische Unterteilungen des Datenträgerspeichers).

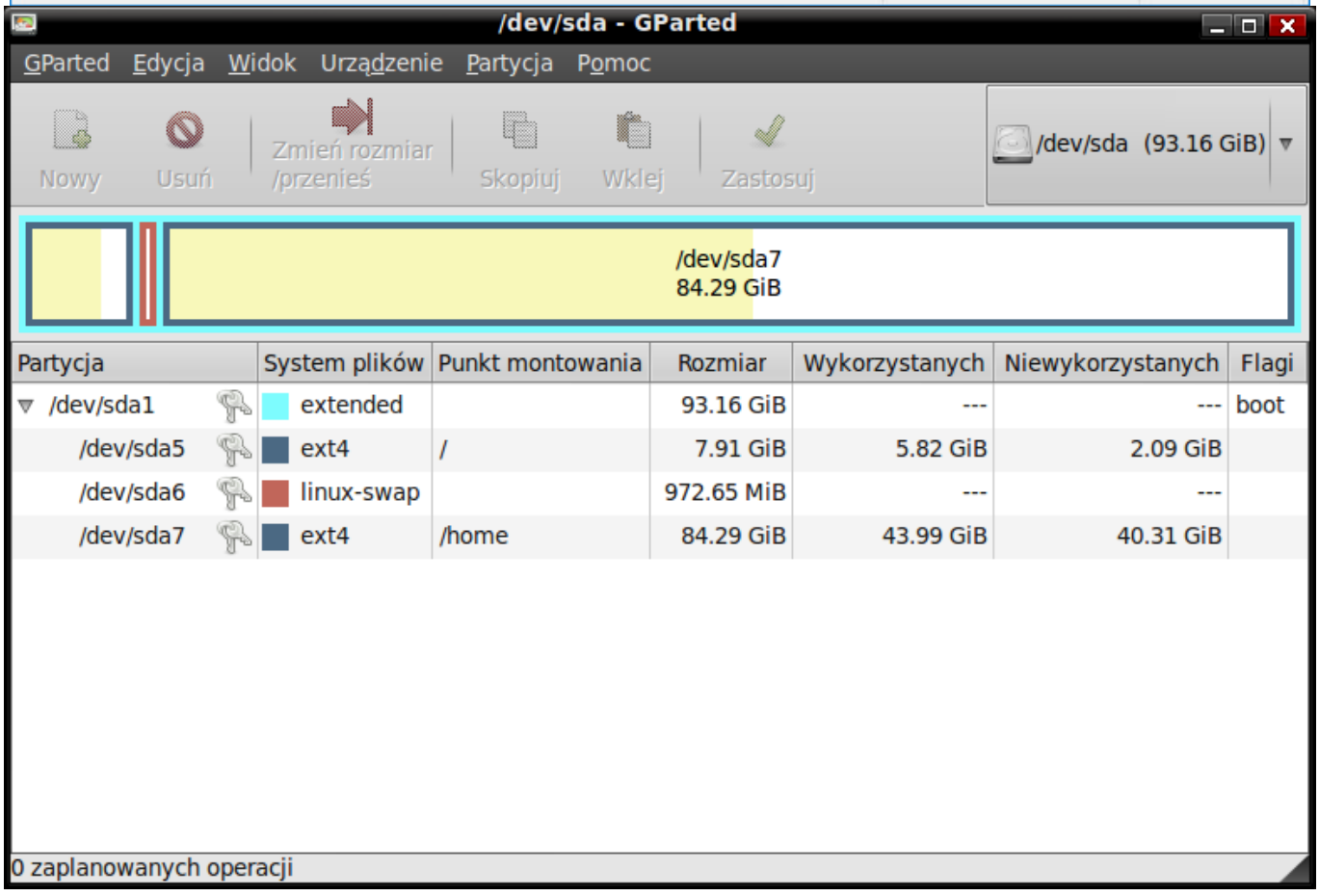
Eine erweiterte Partition dient nur indirekt der Datenspeicherung. Der Benutzer muss in der erweiterten Partition logische Partitionen definieren, in denen die Daten gespeichert werden. Logische Partitionen müssen logisch formatiert sein; auf jeder logischen Partition kann ein anderes Dateisystem definiert sein. Nach der logischen Formatierung gilt jede logische Partition als separater Datenträger.

c) Logische Partitionen

Logische Partitionen können nur innerhalb einer erweiterten Partition definiert werden und sind nur für die Speicherung von Datendateien und Betriebssystemen vorgesehen, die von einer logischen Partition gebootet werden können (z.B. Linux und Windows NT). Betriebssysteme, die von einer logischen Partition gebootet werden können, sollten gewöhnlich in einer logischen Partition installiert werden; damit können Primärpartitionen für andere Zwecke freigehalten werden.







Frage: Wie finde ich die aktuelle Partitionierung von meinem PC?

Gründe für Partitionen

Das Problem früher waren die Festplatten-Controller die nicht in der Lage waren, einen größeren Adressbereich anzusprechen. Und, der technische Fortschritt und die höheren Kapazitäten von Festplatten wurden schneller eingeführt als neue und bessere Dateisysteme. Vor allem unter Windows-Betriebssystemen war das FAT-Dateisystem (File Allocation Table) lange führend. FAT ermöglichte durch die Zusammenführung mehrerer Blöcke zu einer logischen Ansprecheinheit (Cluster), um die Adressierungsbeschränkung zu umgehen. Es hatte den Nachteil, dass es Festplatten nur bis zu einer bestimmten Kapazität verwalten und die Dateien nicht besonders platzsparend speichern konnte.

Beschränkungen der Partitionsgröße vom Dateisystem:

- FAT16 ⇒ max. Partitionsgröße = 2 GByte
- Nachfolger FAT32 ⇒ 2 TByte
- Über 32 GByte verwendet man üblicherweise das Dateisystem NTFS.

Heutige Gründe für Partitionierungen:

- Installation mehrerer Betriebssysteme
- Einrichten verschiedener Dateisysteme
- Reservierung für eine Windows- oder Linux-Auslagerungsdatei (SWAP-Partition)
- Installationsdateien
- sehr großen Speicher verwalten
- Trennung von Programmen und Daten

3) Logische Formatierung (High-Level-Formatierung)

Ein physikalisch formatiertes Festplattenlaufwerk muss noch logisch formatiert werden. Durch die logische Formatierung wird ein **Dateisystem** auf der Festplatte definiert. Erst ein Dateisystem gestattet es einem Betriebssystem wie z.B. DOS, OS/2, Windows 95/98, Windows NT/Windows 2000/WindowsXP/Windows7/Windows8 oder Linux), den verfügbaren Speicher für das Speichern und Laden von Dateien zu nutzen.

Vor der logischen Formatierung kann eine Festplatte in **Partitionen** unterteilt werden. Auf jeder Partition kann ein Dateisystem (logisches Format) definiert werden. Eine Festplattenpartition, die bereits logisch formatiert ist, wird als Datenträger (Volume) bezeichnet. Als Teil des Formatierungsvorgangs werden Sie durch das Formatierungsprogramm aufgefordert, der Partition einen Namen zuzuweisen, die sogenannte “Datenträgerbezeichnung”. Mit diesem Namen können Sie den Datenträger (die Partition) später identifizieren.

Normal- und Schnellformatierung

Es gibt zwei unterschiedliche Methoden, den Datenträger high-level zu formatieren:

Die Schnell- und die Normalformatierung.

Sie unterscheiden sich in folgenden Punkten:

Normalformatierung – Es wird eine Suche nach fehlerhaften Sektoren durchgeführt. Diese Suche nimmt den Großteil der Zeit in Anspruch. Anschließend erfolgt das Schreiben der Dateisystem-Metadaten und somit die Löschung der vorhandenen Dateien.

Schnellformatierung – Es werden zwar die Dateien aus dem Inhaltsverzeichnis des Datenträgers beziehungsweise der betreffenden Partition entfernt, die Suche nach fehlerhaften Sektoren wird jedoch ausgelassen.

Bei einer High-Level-Formatierung sind in dem neuen Dateisystem die alten Daten nicht verfügbar, da sie nicht mehr durch entsprechende Verweise im Dateisystem referenziert werden. Sie werden jedoch nicht notwendigerweise gelöscht. Meistens verbleiben sie rein physikalisch auf der Festplatte, bis sie mit neuen Daten überschrieben werden. Solange die verwendeten Datenblöcke nicht erneut beschrieben werden, ist mit entsprechender Software eine weitgehende Wiederherstellung noch möglich, gestaltet sich jedoch schwerer, als wenn die Dateien einfach nur gelöscht würden. Nicht selten können daher via Software nur die einzelnen Dateien, nicht aber die Ordnerstruktur wiederhergestellt werden.

Dateisysteme

Jeder Computer benötigt ein funktionsfähiges Betriebssystem, auch wenn es lediglich aus den DOS- oder Windows 98-Bootdateien besteht. Verschiedene Betriebssysteme verwenden unterschiedliche Dateisysteme, um auf Datenträger zuzugreifen und die Speicherorte von Daten zu ermitteln. Das Dateisystem bestimmt hierbei die Art der Verwaltung und des Zugriffs auf Dateien auf einem Datenträger.

Die Methoden und Datenstrukturen, die ein Betriebssystem verwendet, um Speicherorte von Dateien auf einer Partition zu ermitteln, werden also als das **Dateisystem** definiert. Bei den hier besprochenen Dateisystemen handelt es sich um FAT, FAT32, HPFS, NTFS, ReFS, Linux Ext2 und Linux ReiserFS.

Windows

FAT16

Das FAT16-Dateisystem ist das ursprünglich von MS-DOS für die Dateiverwaltung verwendete System. Das FAT (File Allocation Table) ist eine Datenstruktur, die MS-DOS beim Formatieren eines Datenträgers auf demselben erstellt. FAT16-Partitionen können bis zu 2 GB groß sein. Die Größe einer Partition bestimmt die Größe der Cluster, in denen Daten gespeichert werden. Bei einer Partitionsgröße von über einem GB kann dies zu einem erheblichen Speicherverlust führen. FAT16-Partitionen werden bei den meisten Versionen von DOS und Windows 95 eingesetzt und von den folgenden Betriebssystemen unterstützt: Windows 95, Windows 98, Windows NT, Windows 2000, DOS (inklusive MS-DOS, PC-DOS und DR-DOS), OS/2, Linux und viele andere Betriebssysteme.

FAT32

FAT32 ist ein Dateisystem, das von Microsoft für Windows 95, Version B (OSR2), entwickelt wurde. Es verfügt über die Fähigkeit, 32-Bit-Clusteradressen zu verwalten und unterstützt Partitionen von bis zu 8 GB bei einer Clustergröße von nur 4 KB. Partitionen von bis zu 16 GB können bei einer Clustergröße von 8 KB verwaltet werden, was den unnötigen Verbrauch von Festplattenspeicher aufgrund ineffizienter Clusternutzung deutlich reduzieren kann. FAT32-Partitionen kommen ausschließlich unter Windows 95B oder höher, Windows 98, Windows 2000 und unter neueren Versionen von Linux zum Einsatz.

NTFS

NTFS (New Technology File System) wurde von Microsoft speziell für die Verwendung mit Windows NT, Windows 2000 und Windows XP entwickelt und unterstützt die verbesserten Sicherheitsmerkmale, die über dieses Betriebssystem zur Verfügung stehen. NTFS unterstützt vollständige Zugriffskontrolle, Dateisystemwiederherstellung und besonders große Datenträger. NTFS-Partitionen werden ausschließlich unter Windows NT (Workstation und Server) sowie unter Windows 2000, Windows XP, Windows 7 und Windows 8 eingesetzt.

ReFS

ReFS (Resilient File System) ist ein Dateisystem von Microsoft, wobei engl. „resilient“ für „elastisch“, „belastbar“ oder gar „unverwüstlich“ steht. Es wurde mit den Betriebssystemen Windows 8 und Windows Server 2012 als Ergänzung zum NTFS-Dateisystem eingeführt.

Linux

Ext2

Das Ext2-Dateisystem war bis ca. 2002 das Standardsystem für Linux. Ist außerordentlich stabil und sicher, aber leider muss nach einem Systemabsturz (z.B. Stromausfall) eine sehr zeitaufwändige Überprüfung des gesamten Dateisystems durchgeführt werden.

Ext3

Das Ext3-Dateisystem ist z.Z. das Standardsystem für Linux. Weitgehend kompatibel zu Ext2, hat aber den Vorteil das es über eine **Journaling-Funktion** verfügt. D.h. nach einem Stromausfall entfällt eine langwierige Überprüfung des gesamten Dateisystems.

Ext4

Ext4 (Fourth Extended File System): weiterentwickelte Variante von Ext3, unter anderem mit

erweiterten Limits.

ReiserFS

Das ReiserFS ist eine Weiterentwicklung des Ext2-Dateisystem ist das Standardsystem für Linux (Verbesserte Dateisystemwiederherstellung nach Unterbrechungen im laufenden Betrieb). Linux ReiserFS-Partitionen werden ausschließlich unter Linux-Installationen verwendet.

MAC

HFS

- HFS

Weiterführende Informationen

Festplatten Fakten

Planung eines Systems

Man sollte sich vor der Installation der Betriebssysteme (oder eines Bootmanagers) einige Gedanken über den Aufbau des Systems machen: Hierbei ist am wichtigsten, sich zuerst überlegen, welche Betriebssysteme eingesetzt werden sollen.
Als nächstes ist abzuklären, wie viel Festplattenplatz für welches Betriebssystem benötigt wird(Größe der Festplatte). Hierzu macht der Hersteller des Betriebssystem bereits einen Vorschlag, der als Minimum eingehalten werde sollte.
Unter Umständen kann es sinnvoll sein, einen Teil der Festplatte (eine Partition) ausschließlich dazu zu verwenden, Daten unter allen, oder zumindest unter mehreren Betriebssystemen, zur Verfügung zu stellen. Am besten lässt man einen Teil der Festplatte unbenutzt (falls diese groß genug ist), damit dieser später für Erweiterungen genutzt werden kann.

Besonderheiten und Probleme der einzelnen Betriebssysteme

Probleme mit DOS / Windows 95/98/ME

Bei der Verwendung von FAT 16 durfte die Partition nicht größer als 2 GB sein.
Von der 2. Festplatte kann nur gebootet werden, wenn auf der 1. Festplatte keine primäre Partition sichtbar ist.

Probleme mit Windows NT/2000/XP

- Windows 2000/XP: Das Setup-Programm von Windows 2000/XP überschreibt während der

Installation den Master-Boot-Record.

- Unter Windows 2000/XP kann eine einmal angelegte Partition nicht mehr verkleinert werden.

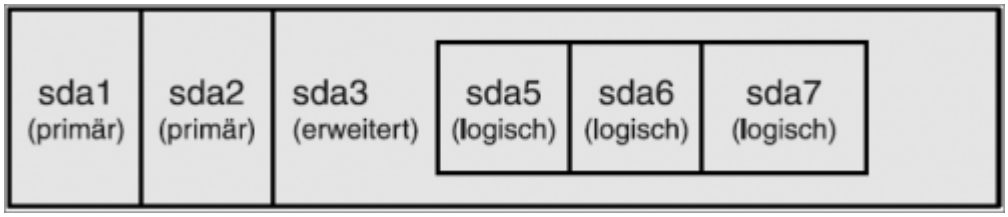
Windows 7, Windows 8

- ab Windows 7 können Partitionen auch verkleinert werden:
    - Datenträgerverwaltung - Rechtsklick auf die Partition - Volume verkleinern

Linux

Linux kann auf jeder Festplatte maximal 15 Partitionen ansprechen, davon maximal 11 logische Partitionen.
Unter Linux erfolgt der interne Zugriff auf Festplatten bzw. deren Partitionen über so genannte Device-Dateien: Die Festplatten erhalten der Reihe nach die Bezeichnungen /dev/sda, /dev/sdb, /dev/sdc etc.

Um eine einzelne Partition und nicht die ganze Festplatte anzusprechen, wird der Name um die Partitionsnummer ergänzt. Die Zahlen 1 bis 4 sind für primäre und erweiterte Partitionen reserviert. Logische Partitionen beginnen mit der Nummer 5 – auch dann, wenn es weniger als vier primäre oder erweiterte Partitionen gibt. Die folgende Abbildung veranschaulicht die Nummerierung: Auf der Festplatte gibt es zwei primäre Partitionen und eine erweiterte Partition, die drei logische Partitionen enthält.



Die maximale Partitionsgröße beträgt 2 TByte. Da es mittlerweile Festplatten mit mehr als 2 TByte Speichervolumen gibt, ist eine sinnvolle Nutzung von Festplatten mit mehr als 2 TByte nur noch mit GPT-Partitionstabellen möglich.

Anzahl und Größe der Linux-Partitionen

Leider gibt es auf die Frage, wie viele Linux-Partitionen ein System haben sollte, keine allgemein gültige Antwort.

Die **Systempartition** ist die einzige Partition, die man unbedingt benötigt. Sie nimmt das Linux-System mit all seinen Programmen auf. Diese Partition bekommt immer den Namen /. Dabei handelt es sich genau genommen um den Punkt, an dem die Partition in das Dateisystem eingebunden wird (den mount-Punkt).
Eine vernünftige Größe für die Installation und den Betrieb einer gängigen Distribution liegt bei 10 bis 20 GByte.

Mit einer **Datenpartition** trennt man den Speicherort für die Systemdateien und für die eigenen Dateien. Das hat einen wesentlichen Vorteil: Man kann später problemlos eine neue Distribution in die Systempartition installieren, ohne die davon getrennte Datenpartition mit Ihren eigenen Daten zu gefährden. Bei der Datenpartition wird **/home** als Name bzw. mount-Punkt verwendet, weswegen oft auch von einer Home-Partition die Rede ist. Die Größe hängt von den eigenen Bedürfnissen ab.

Die **Swap-Partition** ist das Gegenstück zur Auslagerungsdatei von Windows: Wenn Linux zu wenig RAM hat, lagert es Teile des gerade nicht benötigten RAM-Inhalts dorthin aus. Im Gegensatz zu den anderen Partitionen bekommt die Swap-Partition keinen Namen (keinen mount-Punkt). Die Größe sollte das ein- bis zweifache des RAMs betragen.

Mehrere Betriebssysteme verwalten

Es gibt verschiedene Möglichkeiten, mehrere Betriebssysteme zu verwalten:

- Mit Hilfe von Bootmanagementprogrammen, wie beispielsweise BootMagic von PowerQuest, Bootstar von Star-Tools, Acronis von SWSoft oder GRUB von SuSE-Linux
- Mit Hilfe einer von einem Betriebssystem verwalteten doppelten Bootkonfiguration, wie

beispielsweise der Boot Loader von Windows NT oder GRUB (bzw. mit dem älteren) LILO unter Linux.
- Manuell, indem Sie ein Betriebssystem als „aktiv“ einstellen. Nehmen Sie diese Einstellung über ein Dienstprogramm wie PQBoot vor, oder bearbeiten Sie den Masterbootdatensatz von Hand (nicht zu empfehlen!).

Windows-Betriebssysteme

- MS-DOS / DOS
- Windows 3.11
- Windows 95
- Windows 98
- Windows NT
- Windows 2000
- Windows XP
- Windows Vista
- Windows 7
- Windows 8
- Windows 10

Linux-Betriebssysteme

- OpenSuSE Linux
  - Open Suse für Profis
- SuSE Linux

Ubuntu

- Ubuntu
- www.ubuntu.com
  - Ubuntu Austria
  - Ubuntu Server
- Kubuntu
- Edubuntu
- Xubuntu
- Gobuntu
- Knoppix
- RedHat Linux
- Debian Linux
- Xubuntu von USB-Stick: Pendrive Linux

Macintosh-Betriebssysteme

- Leopard
- Snow Leopard

Betriebssystem-Virtualisierung

- Virtualisierung

Online-Betriebssysteme

- Ein Online-Desktop von eyeOS

Portable-Betriebssysteme

- PC am Datenstick
- Linux Advanced

Windows

- https://www.script-example.com/themen/direkter_Aufruf_von_Elementen_aus_Windows_bzw._der_Systemsteuerung.php
- MS-DOS
- Benutzerverwaltung
- Dateimanagement
- Sicherung & Wiederherstellung

Das Betriebssystem MSDOS

MSDOS (DOS = Disk Operating System) ist ein Betriebssystem, das von der Firma Microsoft in den USA entwickelt wurde und früher zu den am meisten verbreiteten Betriebssystemen zählte. Die Entwicklungsgeschichte reicht sehr lange zurück, die erste Version gab es 1981.

Mittlerweile ist MSDOS weitgehend duch die neuen Windows-Betriebssysteme abgelöst worden - trotzdem wird DOS für bestimmte Aufgaben auch heute noch verwendet, wie z.B. zur Erstellung von Batch-Dateien. Die Kenntnisse über DOS könnten auch insbesondere bei einem Crash nützlich sein, wenn Windows z.B. nicht mehr startet oder man ein Uralt-DOS-PC-Spiel noch mal spielen will.

Arbeiten mit DOS auf der Systemkonsole

Speicher zuvor folgenden Ordner auf deinem Verzeichnis: dos-befehle.zip

Start der Systemkonsole

- Start - Ausführen: cmd
- Start - Programme - Zubehör - Eingabeaufforderung

- Vollbildmodus: ALT + ENTER
- Beenden: exit

Einige wichtige DOS-Befehle

Allgemeine Befehle

Befehl	**Beschreibung**
help	zeigt eine Liste aller MS-DOS-Befehle und eine kurze Beschreibung
help *Befehl*	man erhält Informationen zu einem bestimmten Befehl
dir	zeigt Inhaltsverzeichnis des aktuellen Verzeichnisses an
dir i:	zeigt den Inhalt des Laufwerks i
dir *.exe	Stern ist eine „Wild Card“. Er steht für eine beliebige Zeichenkombination, es werden somit alle .EXE-Dateien ausgegeben
dir m??er.doc	Fragezeichen ist ebenfalls eine „Wild Card“, jedoch für ein einzelnes Zeichen
whoami	Gibt den Usernamen des ausführenden Benutzers aus
hostname	Gibt den Rechnernamen aus

Befehle für Verzeichnisse

Die folgende Befehle sind zum Anlegen, Wechseln und Löschen neuer Verzeichnisse. Das Verzeichnis, welches im aktuellen Eingabebereitschaftszeichen aufscheint, wird meist als **aktuelles Verzeichnis** oder **Arbeitsverzeichnis** bezeichnet.

Befehl	**Beschreibung**
cd	change directory, Bedeutung von Syntax abhängig
cd \..	geht zum root (oberste Verzeichnisebene)
cd \programme	wechselt in das Verzeichnis Programme ausgehend vom root
cd programme	wechselt - ausgehend vom derzeitigen Arbeitsverzeichnis - in das Unterverzeichnis `programme`
cd .	Zeiger auf aktuelles Verzeichnis
cd ..	Zeiger auf übergeordnetes Verzeichnis
md	make directory - **erzeugt Unterverzeichnis**
rd	remove directory - **löscht Unterverzeichnis**
tree	zeigt die Verzeichnisstruktur auf dem aktuellen Laufwerk an

Befehle für Dateien

Wichtiger Operationen mit Dateien sind das Kopieren, das Löschen und das Umbenennen.

Befehl	**Beschreibung**
copy	benötigt zwei Parameter: 1. Quelldatei, die kopiert werden soll, 2. Zieldatei
copy x.dat x.old	kopiert die Datei x.dat auf x.old (dabei wird x.dat nicht gelöscht!). Existiert bereits eine Datei x.old, so wird diese gelöscht und mit den Daten von x.dat gefüllt.
copy *.* a:	kopiert alle Dateien im Arbeitsverzeichnis nach A:
copy adam.dat a:	kopiert adam.dat **nach** a:adam.dat
copy *.* c:\xy\z	kopiert alle Dateien des aktuellen Ordners auf die Festplatte c: ins Unterverzeichnis Z des Verzeichnisses xy
del x.dat	löscht die Datei `x.dat` im aktuellen Verzeichnis
del *.*	löscht alle Dateien im aktuellen Verzeichnis
ren x.dat y.dat	rename, benennt x.dat in y.dat um
move muster.doc windows	verschiebt `muster.doc` ind das Verzeichnis `windows`
type *Dateiname*	zeigt den Inhalt der angegebenen Datei am Bildschirm an
edit *Dateiname*	ruft einen Editor auf, um den Inhalt einer Datei verändern zu können
print *Dateiname*	gibt den Inhalt einer Datei auf dem Drucker aus
xcopy /E	Kopiert ganze Verzeichnisse
rmdir /S	Löscht Verzeichnisse trotz Inhalt

DOS-Übung

Speichere den Ordner „DOS“ dos.rar in dein Verzeichnis, bevor du mit der Übung beginnst!

1. Erzeuge ein Verzeichnis `uebungen` in deinem homedirectory.
2. Wechsle in dieses Verzeichnis.
3. Erstelle eine Datei `test1.txt` und `test2.txt` mit beliebigen Inhalt.
4. Kopiere die Datei `test1.txt` in die Datei `test21.txt`.
5. Erstelle ein Verzeichnis `aufgabe`.
6. Benenne `test2.txt` um in `test12.txt`.
7. Kopiere die Dateien test1?.txt in das Verzeichnis `aufgabe`. Welche Dateien wurden kopiert?
8. Kopiere dir alle Dateien aus dem Ordner`DOS` in das Verzeichnis `aufgabe`. Wie viele Dateien

befinden sich nun in diesem Verzeichnsi?
9. Lasse dir alle Dateien mit der Endung `*.txt` ausgeben.
10. Lösche alle Dateien mit der Endung `*.dat`.
11. Benenne alle Dateien mit der Endung `*.bak` um in `*.txt`.
12. Lasse dir alle Dateien sortiert nach der Dateigröße anzeigen. Finde den Befehl selber heraus.
13. Gib den Inhalt der Datei `hallo.txt` aus.
14. Lasse dir alle Dateien anzeigen, bei denen im Dateinamen die Zahl 1 vorkommt.
15. Lösche diese Dateien.
16. Verschiebe alle Dateien aus dem Verzeichnis Aufgabe in das Verzeichnis uebungen.
17. Lösche das Verzeichnis Aufgabe.
18. Gib den Befehl `attrib +h *.*` ein.
19. Lasse dir alle Dateien anzeigen. Was ist passiert? Schreibe deine Antwort in die Datei antwort.txt.
20. Mache dir über den Befehl `attrib` schlau und gib einen Befehl ein, sodass alle Dateien wieder angezeigt werden.

BATCH (.bat) Files

Befehl	Beschreibung
echo	Gibt Text aus
@echo off	Schaltet die Autobefehlsanzeige aus
title	Definiert den Titel des BATCH-Files
set VAR=Text	Speichert Text in die Variable VAR
set /P VAR=	Fordert den User auf eine Zeichenfolge einzugeben
set/A C=%A%+%B%	/A gibt an, dass die Zeichenfolge rechts vom = ein numerischer Ausdruck ist, der ausgewertet wird. in C wird das Ergebnis von A+B gespeichert
echo %VAR%	Ausgabe einer Variable → Text
echo %USERNAME%	Gibt den Usernamen des ausführenden Benutzers aus
REM	Definiert einen Kommentar

Batch-Programmieren

Benutzerverwaltung

Microsoft Windows ist ein Multi-User Betriebssystem. Was ist ein Benutzer? Was ist eine Gruppe? Welche Rolle spielt die Sicherheitsrichtlinie?

BENUTZER

SID

Jeder Benutzer wird durch eine Nummer, einer SID (Security Identifier) eindeutig gekennzeichnet und erhält ein eigenes Profil unter c:\users\%username%.

Mithilfe von SIDs ist es u.a. möglich Benutzer umzubenennen, ohne Veränderung seiner Zugriffsrechte. Selbst wenn das Administratorkonto umbenannt wird, so hat dieser immer die SID mit der Endung 500. Welcher User welche SID verwendet kann in der CMD oder mit PowerShell eingesehen werden:

Befehl für die PowerShell:

```
PS C:\> get-wmiobject win32_useraccount


AccountType : 512
Caption     : SERVER01\Administrator
Domain      : SERVER01
SID         : S-1-5-21-140281148-68646805-4244902722-500
FullName    :
Name        : Administrator

AccountType : 512
Caption     : SERVER01\Gast
Domain      : SERVER01
SID         : S-1-5-21-140281148-68646805-4244902722-501
FullName    :
Name        : Gast
```

Befehl für die CMD:

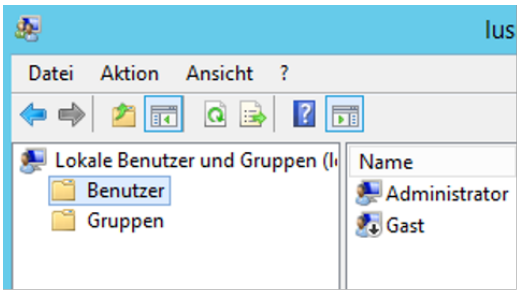
```
wmic useraccount get name,sid
```

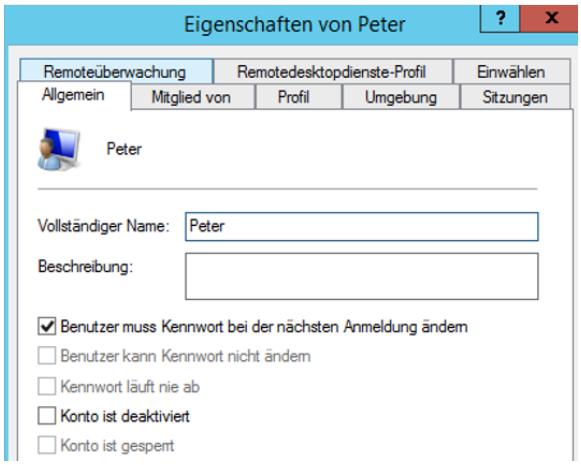
Kennwörter/Passwörter

Kennwörter werden in der SAM (Security Account Manager) Datenbank gespeichert. Diese Datenbank kann im laufenden Betrieb (mit Bordmitteln) nicht eingesehen werden. Die Kennwörter werden als Hashwert gespeichert, d.h. nicht im Klartext.

Anlegen von Lokalen Benutzern

Dies ist grafisch mit lusrmgr.msc oder mithilfe des Befehls net user möglich.

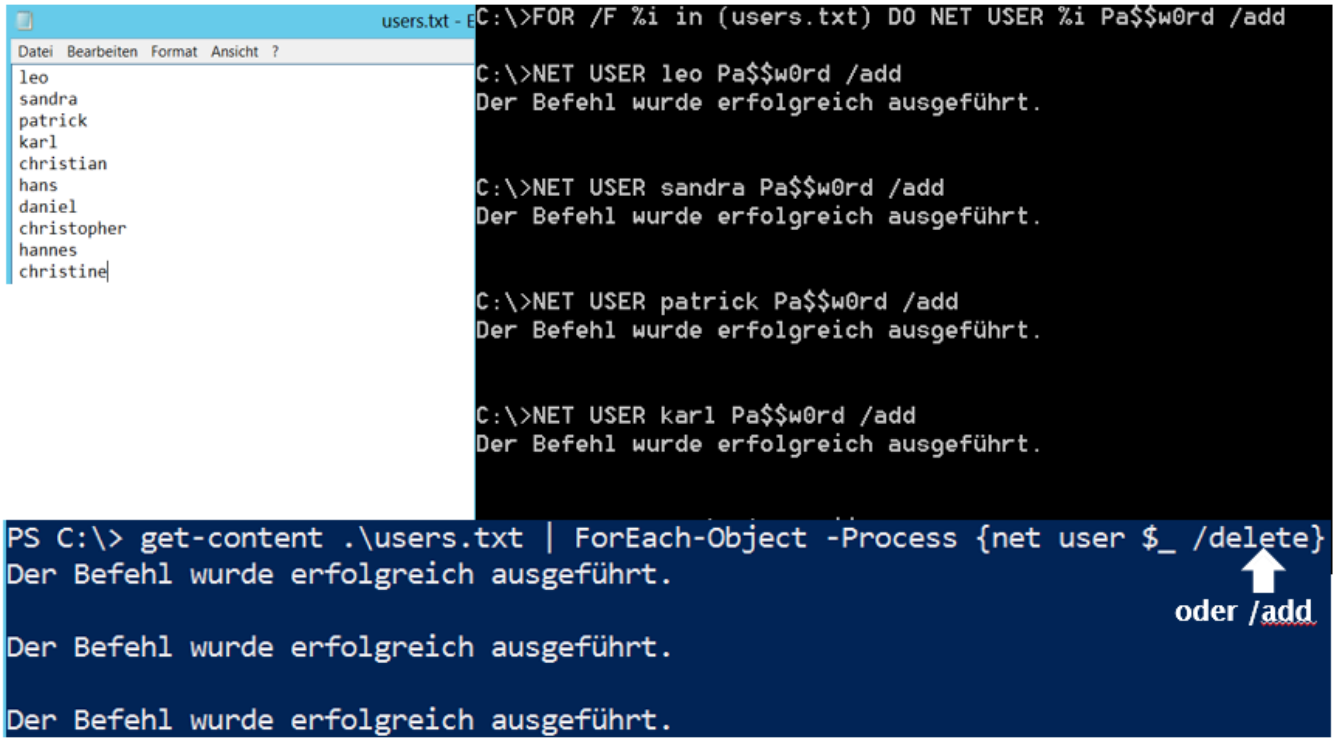





```
net user Max 'Pa$$w0rd' /add
```

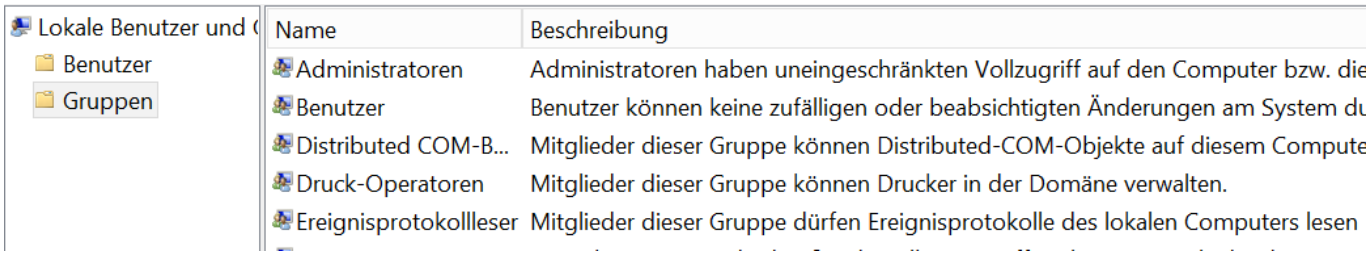
Massenimport von Usern

Hier am Beispiel einer Textdatei users.txt mit einer Auflistung von Vornamen:



GRUPPEN

Bei einer Gruppe handelt es sich um eine Sammlung von Benutzer- und Computerkonten, Kontakten sowie weiteren Gruppen, die als einzelne Einheit verwaltet werden können. Gruppen findet man ebenfalls in lusrmgr.msc bzw. auch mit net localgroup.

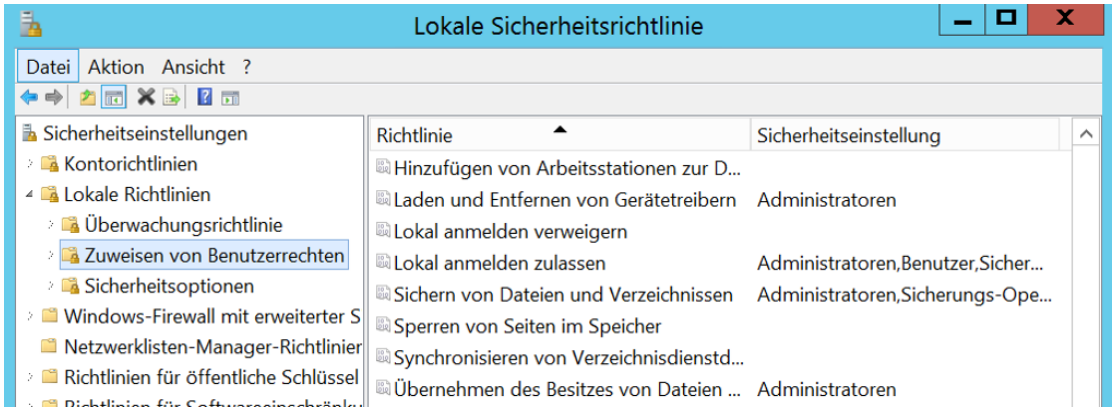


Hier wird der Benutzer admin01 der Gruppe Administratoren hinzugefügt:

```
net localgroup administratoren admin01 /add
```

SICHERHEITSRICHTLINIEN

Wozu das Ganze? Gruppen, Benutzer? Nun, es ist wichtig zu unterscheiden, wer, was, wann, wo am System tun darf oder nicht. Und dies regelt die Sicherheitsrichtlinie, welche unter secpol.msc zu finden ist.



Fügen Sie einen Benutzer in die Gruppe der Administratoren hinzu, so darf dieser laut der Liste Gerätetreiber installieren oder auch Dateien sichern. Würde es keine Gruppen(-richtlinien) geben, so müsste man bei jedem einzelnen Benutzer jedes einzelne Recht hinzufügen oder entfernen. Mithilfe von Gruppen braucht man dies nur einmal für die Gruppe und kann später alle gewünschten Benutzer in diese Gruppe hinzufügen. Somit spart man sehr viel Zeit als IT-Administrator.

Dateimanagement

Eine Hauptaufgabe des Betriebssystems ist die Verwaltung der Daten. Dabei werden Daten in sogenannten **Dateien** zusammengefasst, die auf Festplatten und anderen Speichermedien abgespeichert werden können. Eine Datei lässt sich mit dem Inhalt einer Karteikarte vergleichen.

Mehrere Dateien („Karteikarten“) werden in einem Verzeichnis (directory) abgelegt. Ein Verzeichnis ist vergleichbar mit einer Schachtel, in der verschiedene Karteikarten untergebracht werden können. Selbstverständlich ist es auch möglich, in einer Schachtel mehrere kleinere Schachteln unterzubringen (Unterverzeichnisse - subdirectory) usw.

Klarerweise entsteht dadurch ein baumartiges System, weshalb man dieses auch als „Tree“ bezeichnet. Die größte Schachtel, die alle anderen enthält bezeichnet man als **Hauptverzeichnis** oder **Root** (Wurzel).

Dateiformate

Die Art und Weise, wie Informationen innerhalb einer Datei gespeichert werden, und wie diese zu interpretieren sind, wird als Dateiformat bezeichnet. Um den Inhalt einer Datei wieder benutzen oder weiterverwenden zu können, muss bekannt sein, auf welche Weise und in welcher Reihenfolge die Informationen in der Datei abgelegt wurden.

Für unterschiedliche Inhalte und Einsatzzwecke gibt es zahlreiche Methoden zur Speicherung in Dateien, die unter dem Oberbegriff **Dateiformate** zusammengefasst werden. Um also z.B. den Inhalt eines einfachen Briefes in einer Datei abzulegen, wird ein auf Textspeicherung ausgerichtetes Format verwendet, während bei der Speicherung eines Bildes ein entsprechendes Grafikformat verwendet wird.

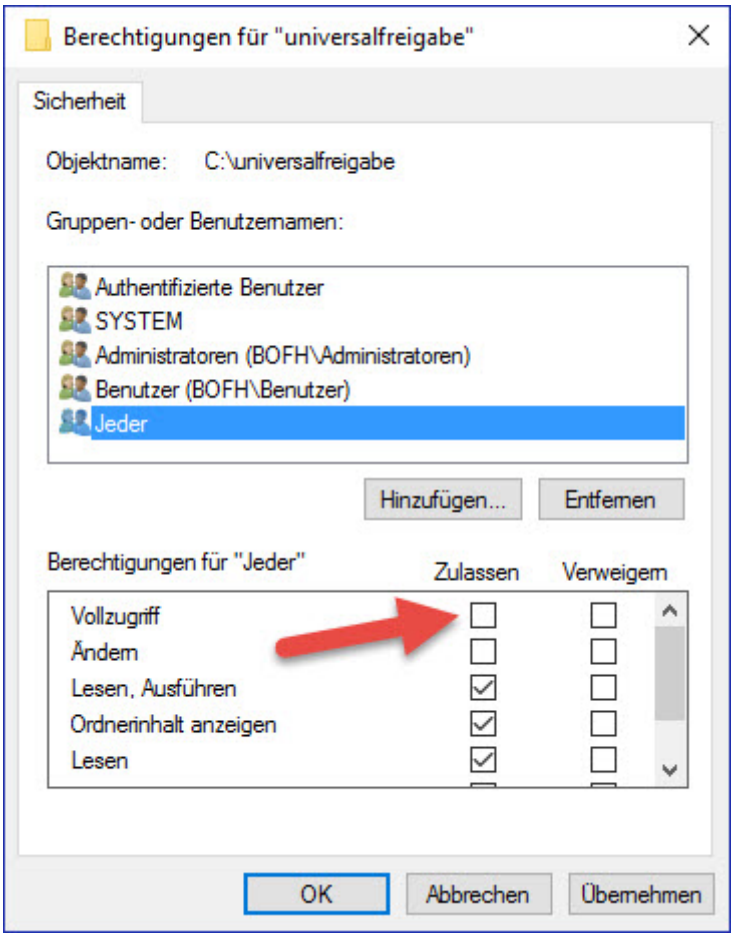
Vor allem bei Microsoft-Betriebssystemen hat es sich eingebürgert, einen Hinweis auf das verwendete Dateiformat durch einen Zusatz am jeweiligen Namen der Datei anzubringen. Diesen Hinweis nennt man **Dateiendung** und wird durch einen **Punkt** vom Dateinamen getrennt.

Übersicht über einige Standarderweiterungen

für ausführbare Dateien	
EXE	ausführbare Datei
BAT	Batch-File
programmspezifische Erweiterungen	
DOC	formatiertes Dokument
TXT	Textdatei ohne Sonderzeichen
SYS	System(Programm)datei
DBF	Datenbankfile
...	...

Dateirechte

Da nicht jeder Benutzer immer alle Rechte auf einem Rechner haben darf/soll/muss, gibt es im Dateisystem (z.B. NTFS) die Rechte für die jeweiligen Benutzer bzw. Gruppen zu definieren. Diese Dateirechte sind die Sicherheitseinstellungen, die man auf einer NTFS-Partition einem Dateiobjekt vergibt. Diese werden üblicherweise im Kontextmenü des Objektes unter Eigenschaften > Sicherheitseinstellungen vergeben:



Es gibt die folgenden Rechte in der Registerkarte Sicherheit bei einem Dateiobjekt

Berechtigung	**Rechte**
Vollzugriff / Full Control	Alle
Ändern / Modify	Alle, außer Berechtigungen ändern und Besitzerrechte übernehmen
Lesen & Ausführen / Read & Execute	Datei öffnen und lesen, ausführbare Dateien und Batchdateien starten
Ordnerinhalt auflisten / List Folder Contents	Nur bei Ordner: Lesen und Lesen & Ausführen
Lesen / Read	Datei öffnen und lesen
Schreiben / Read	Datei ändern oder neu erzeugen
Spezielle Berechtigungen	Siehe unten

Kopieren / Verschieben von Dateien

Wenn eine Datei kopiert wird, wird sie am Zielort neu erstellt und erbt daher die Berechtigungen des Ordners, in den sie kopiert wurde. Beim Verschieben innerhalb einer Partition behält die Datei Ihre Ursprungsrechte.

Will man Dateien verschieben und Berechtigungen des Zielordners annehmen lassen, kopiert man also am Besten und löscht dann die Ursprungsdaten.

Sichern und Wiederherstellen in Windows

Ein Windows 10 Backup könnt ihr mit integrierten Backup-Tools im Handumdrehen erstellen, eure persönlichen Daten durch eine Sicherung schützen und ein komplettes Windows-Systemabbild mit Bordmitteln erstellen.

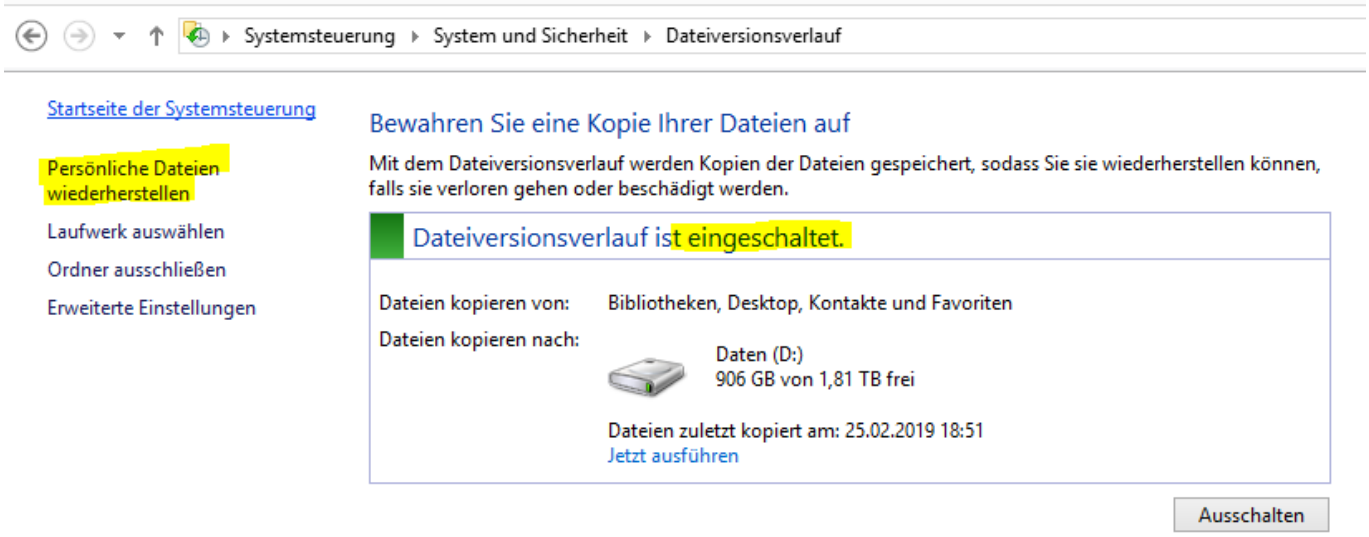
Mit den Windows 10-Bordmitteln könnt ihr bei einem Defekt der Festplatte nicht nur einzelne Daten, sondern euer komplettes Windows-System wiederherstellen. Außerdem könnt ihr mit den Windows-Systemprogrammen eure Daten automatisch sichern.

Hier sind die einzelnen Varianten beschrieben:

1) Dateiversionsverlauf

Mit dem Dateiversionsverlauf werden Kopien der Dateien gespeichert, sodass sie wiederhergestellt werden können, falls sie verloren gehen oder beschädigt werden.

- Systemsteuerung - System und Sicherheit - Dateiversionsverlauf
- Wenn der Dateiversionsverlauf eingeschaltet ist, kann auf ältere Dateibestände zurückgegriffen werden.

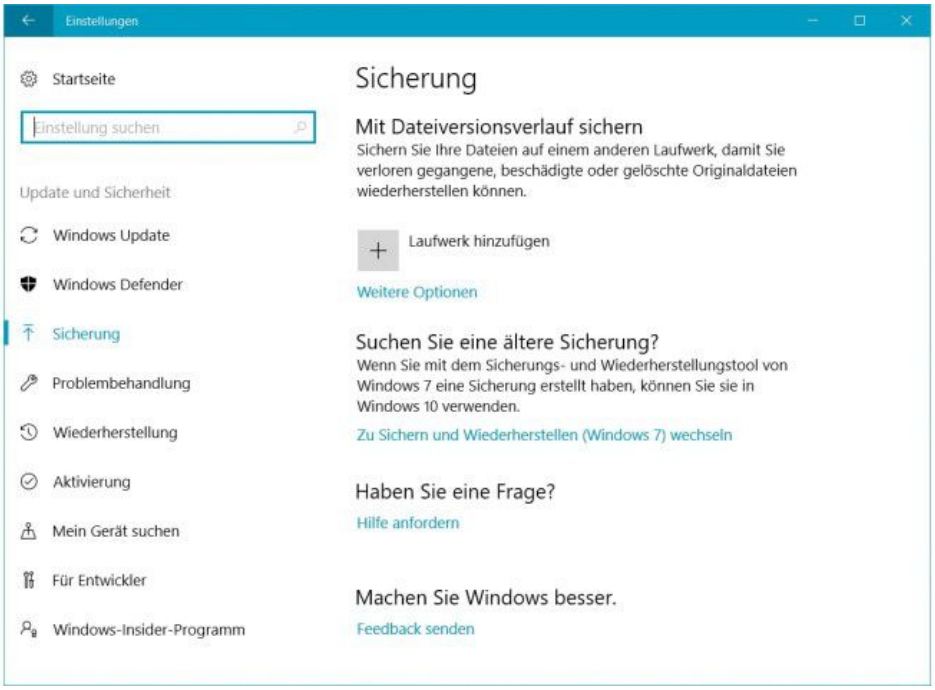


2) Backup erstellen und einzelne Dateien sichern

Mit den folgenden Schritten könnt ihr einzelne Ordner und Dateien, etwa Bilder, Videos und Dokumente, automatisch sichern und einzelne Dateien im Fehlerfall wiederherstellen:

- Öffnet unten links das vierkachelige Windows-Startmenü und klickt auf den Reiter „Einstellungen“.
- Wählt im neuen Fenster das Feld „Update und Sicherheit“ und klickt anschließend auf die Kategorie „Sicherung“.
- Klickt auf „Laufwerk hinzufügen“ und Windows listet euch alle verfügbaren Datenträger aus.

Wählt die Festplatte oder den USB-Stick aus, auf dem eure Daten gesichert werden sollen.
- Klickt danach auf „Weitere Optionen“, wo ihr über „Ordner hinzufügen“ weitere Ordner und Daten dem Backup hinzufügen könnt. Über das Drop-Down-Menü bei „Meine Daten sichern“ könnt ihr außerdem auswählen, ob und wie oft eure Daten automatisch gesichert werden sollen.
- Schießt die Datensicherung mit einem Klick auf „Jetzt sichern“ ab.

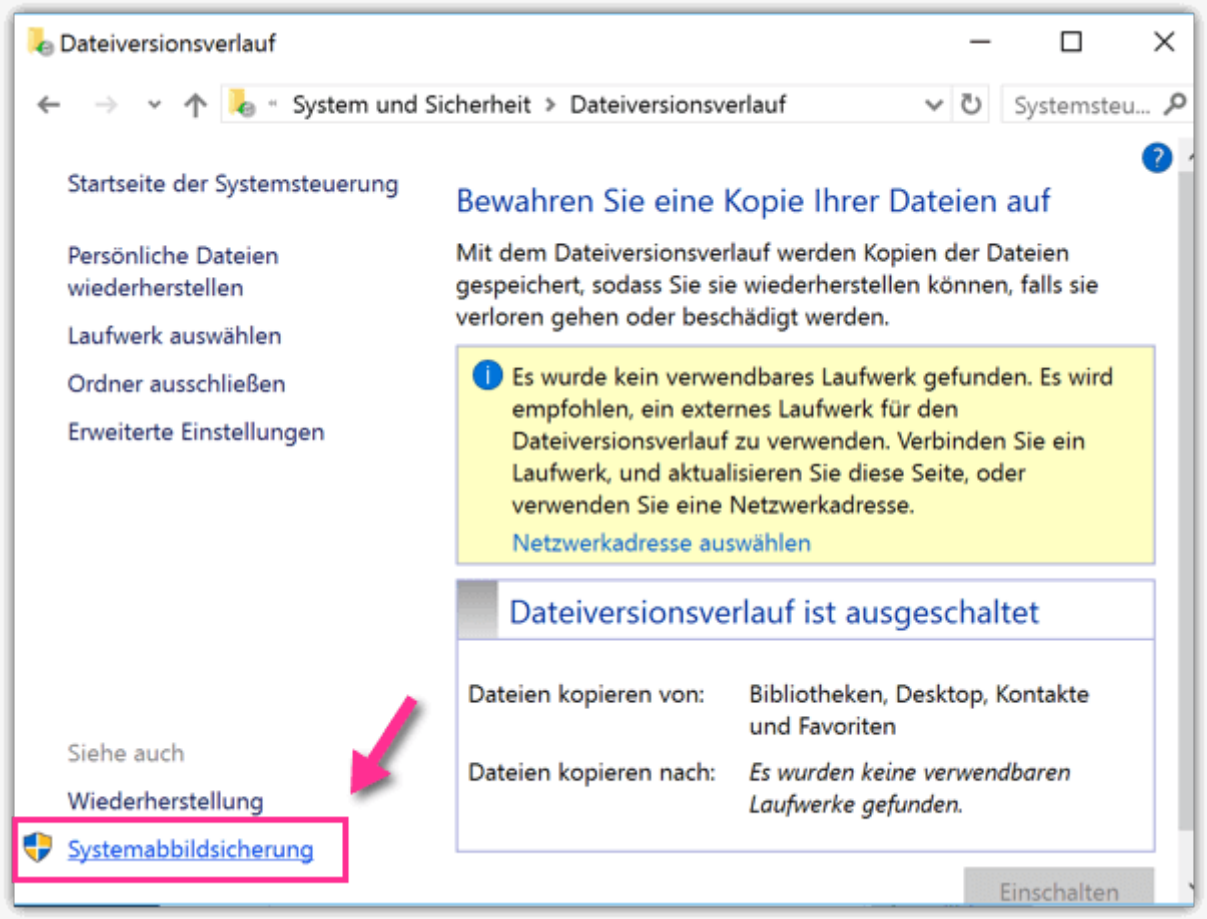


3) Komplettes Systemabbild mit Bordmitteln erstellen

Wollt ihr aber euer komplettes System absichern und ein Windows 10-Systemabbild erstellen, so sind folgende Schritte notwendig. Damit kann man nicht nur einzelne Dateien sondern auch das komplette Betriebssystem im Fehlerfall wiederherstellen.

Achtung!!: Speichert das Backup nie auf das Systemlaufwerk, denn im Falle eines Festplattendefekts hilft das beste Backup nichts!! Als Speicherort eignet sich am besten eine externe Festplatte oder ein Serverlaufwerk mit genügende Speicherplatz.

- Navigiert, wie oben beschrieben, erneut in das „Weitere Optionen“-Menü.
- Scrollt in den Sicherungsoptionen bis an das Ende der Seite und klickt auf „Siehe erweiterte Einstellungen“.
- Wählt im neu geöffneten Fenster unten links „Systemabbildsicherung“ und anschließend „Systemabbild erstellen“ aus.
- Wählt die Festplatte aus, auf dem das Systemabbild erstellt werden soll und bestätigt mit „Weiter“.
- Windows zeigt euch an, welche Laufwerke mit dem Backup gesichert werden. Schließt den Vorgang mit „Sicherung starten“ ab



4) Alternative Backup-Methoden

Neben den Windows-Bordmitteln könnt ihr für eure Datensicherung auch auf verschiedene Software-Lösungen zurückgreifen.

- Acronis
- Duplicati

LINUX

Linux ist ein freies Multiplattform-Mehrbenutzer-Betriebssystem, das den Linux-Kernel enthält. Im praktischen Einsatz werden meist sogenannte Linux-Distributionen genutzt, in denen der Linux-Kernel und verschiedene Software zu einem fertigen Paket zusammengestellt sind.

Das wichtigste gleich vorweg, in Linux ist alles eine Datei –> Everything is a file.

Everything is a file beschreibt eine der definierenden Eigenschaften von Unix und seinen Abkömmlingen, demnach Ein-/Ausgabe-Ressourcen wie Dateien, Verzeichnisse, Geräte (z. B. Festplatten, Tastaturen, Drucker) und sogar Interprozess- und Netzwerk-Verbindungen als einfache Byteströme via Dateisystem verfügbar sind

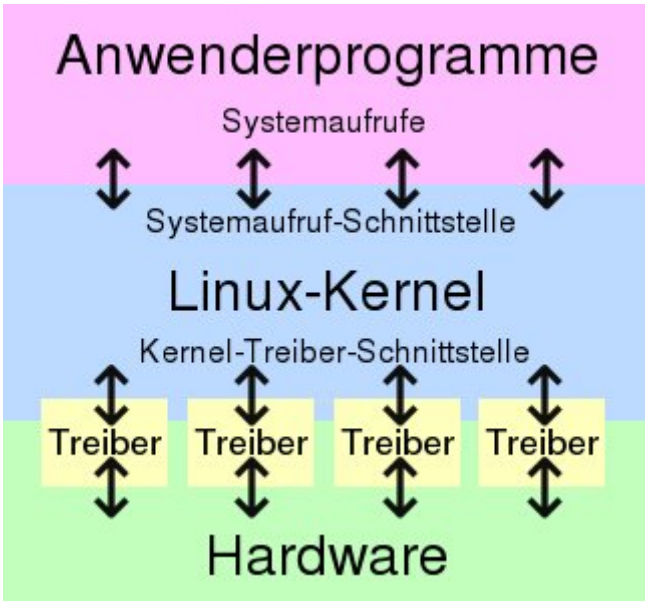
- Aufbau des Betriebssystems
- Benutzer
- Dateimanagement
- Dateirechte
- Inodes
- Distributionen und Desktops
- Konsole/Bash/Terminal
- Shell Scripts

Aufbau des Betriebssystems

Das zentrale Kernstück des Betriebssystems, der Linux-Kernel (meist nur Kernel genannt) bildet eine Trennschicht zwischen Hardware und Anwenderprogrammen. Das heißt, wenn ein Programm auf ein Stück Hardware zugreifen will, so kann es niemals direkt darauf zugreifen, sondern nur über das Betriebssystem.

Dazu bedient sich das Programm der **Systemaufrufe**. Über den Systemaufruf teilt das Anwenderprogramm dem Betriebssystem mit, dass es etwas zu tun gibt. Will etwa ein Programm eine Zeile Text auf dem Bildschirm ausgeben, so wird ein Systemaufruf gestartet, dem der Text übergeben wird. Das Betriebssystem erst schreibt ihn auf den Bildschirm.

Auf der anderen Seite muss das Betriebssystem die Möglichkeit haben, mit den einzelnen Hardware-Komponenten zu sprechen. Mittels seiner **Treiberschnittstelle** spricht es spezielle Geräte-Treiber an. Erst die Treiber kommunizieren dann direkt mit den Geräten.



Zu den **Anwenderprogrammen** zählen alle von uns gestarteten Programme (Videoplayer, Webbrowser ...), wie auch die grafische Oberfäche des Betriebssystems, das Desktop-Environment. Letzteres ist nicht ein Programm, sondern eine Sammlung von Programmen, die zusammen die gewohnten Funktionalitäten beisteuern.

Ein ganz spezielles Anwenderprogramm ist die Shell - die „Benutzeroberfläche“. Es existieren viele verschiedene Shells - wir werden hier mit der Bash (Bourne again shell) arbeiten. Diese ist die Standardshell auf Linuxsystemen. Alle Shells stellen dem Benutzer eine Kommandozeile zur Verfügung, mit der Befehle eingegeben werden können, die direkt als Systemaufrufe an das Betriebssystem weitergeleitet werden.

Linux ist ein **Multitasking-Betriebssystem**: das heißt, es können mehrere Prozesse - so nennt man Programme, sobald sie in den Speicher geladen sind und laufen - gleichzeitig laufen. Das bedingt, dass das System die verfügbare Rechenzeit des Prozessors in kleine Zeitscheiben aufteilt (im Millisekundenbereich), die dann den jeweiligen Prozessen zur Verfügung stehen. Diese Aufgabe übernimmt eine übergeordnete Instanz - der Scheduler. Dieser verwaltet die Zuteilung der Zeitscheiben an die verschiedenen Prozesse.

Daher kann kein Prozess die ganze Rechenleistung für sich beanspruchen und auch ein „hängender“ Prozess kann nicht das ganze System lahmlegen.

Benutzer

Linux ist auch ein **Multiuser**-System, das heißt, es können mehrere Benutzer an verschieden Terminals auf dem selben Rechner arbeiten. Dazu ist es natürlich notwendig, dass jeder Benutzer eindeutig identifiziert ist. Die User (engl., Benutzer) werden zwar mit ihren Namen verwaltet, intern arbeitet ein Unix-System aber mit Usernummern. Jeder Benutzer hat also eine Nummer welche UserID oder kurz UID genannt wird. Jeder Benutzer ist auch Mitglied mindestens einer Gruppe. Es kann beliebig viele Gruppen in einem System geben und auch sie haben intern Nummern (GroupID oder GID). Im Prinzip sind Gruppen nur eine Möglichkeit, noch detailiertere Einstellungsmöglichkeiten zu haben, wer was darf.

Eine spezielle Rolle hat der Benutzer mit der UserID 0 - er ist Root (engl., Wurzel). Root steht außerhalb aller Sicherheitseinrichtungen des Systems - kurz - er darf alles. Er kann mit einem Befehl das ganze System zerstören, er kann die Arbeit von Wochen und Monaten löschen usw. Aus diesem Grund meldet sich auch der Systemverwalter im Normalfall als normaler Benutzer an - zum Root-Benutzer wird er nur dann, wenn er Systemverwaltungsarbeiten abwickelt, die diese Identität benötigen.

Benutzertypen

root

Der Benutzer root ist mit allen Rechten ausgestattet, die ihm die Administration (bei Unachtsamkeit natürlich auch die Beschädigung!) des Systems erlauben. Diesem auch als Superuser bezeichneten Benutzer ist immer die UID 0 zugeordnet.

Systembenutzer

Je nach System kann eine Vielzahl von Prozessen und Diensten erwünscht sein, die bereits beim Hochfahren des Systems verfügbar sein sollen. Nicht jeder dieser Prozesse benötigt jedoch die volle Rechteausstattung des Superusers. Man möchte natürlich so wenige Prozesse wie nur möglich unter einer root Kennung starten, da die weitreichenden Rechte solcher Prozesse unnötige Möglichkeiten für Missbrauch und Beschädigung des Systems liefern.

Ein Systembenutzerkonto ist in diesem Sinne ein Benutzerkonto, das jedoch (nahezu) ausschließlich zur Ausführung von Programmen unter einer speziellen Benutzerkennung verwendet wird. Kein menschlicher Benutzer meldet sich normalerweise unter einem solchen Konto an.

Standardbenutzerkonto

Dies ist das normale Benutzerkonto, unter welchem jeder üblicherweise arbeiten sollte.

Die zentralen Benutzerdateien

Die Dateien zur Benutzerverwaltung finden Sie unter Linux im Verzeichnis /etc. Es handelt sich dabei um die Dateien **/etc/passwd**, **/etc/shadow** und **/etc/group**.

/etc/passwd

Die Datei /etc/passwd ist die zentrale Benutzerdatenbank.

Mit cat /etc/passwd können Sie einen Blick in diese zentrale Benutzerdatei werfen. Hier werden alle Benutzer des Systems aufgelistet. Zu beachten ist, dass alle Benutzertypen eingetragen sind, also sowohl der Superuser root als auch die Standard- und Systembenutzer.

Ein Benutzerkonto in der Datei /etc/passwd hat generell folgende Syntax:

```
Benutzername : Passwort : UID : GID : Info : Heimatverzeichnis : Shell
```

Spalte	**Erklärung**
Benutzername	Dies ist der Benutzername in druckbare Zeichen, meistens in Kleinbuchstaben.
Passwort	Hier steht verschlüsselt das Passwort des Benutzers (bei alten Systemen). Meist finden Sie dort ein x. Dies bedeutet, dass das Passwort verschlüsselt in der Datei /etc/shadow steht. Es ist auch möglich, den Eintrag leer zu lassen. Dann erfolgt die Anmeldung ohne Passwortabfrage (in der Datei /etc/shadow muss dann an Stelle des verschlüsselten Passwortes ein * stehen).
UID	Die Benutzer-ID des Benutzers. Die Zahl hier sollte größer als 100 sein, weil die Zahlen unter 100 für Systembenutzer vorgesehen sind. Weiterhin muss die Zahl aus technischen Gründen kleiner als 64000 sein.
GID	Die Gruppen-ID des Benutzers. Auch hier muss die Zahl wie bei der UID kleiner als 64000 sein.
Info	Hier kann weitere Information vermerkt werden, wie z.B. der vollständige Name des Benutzers und persönliche Angaben (Telefonnummer, Abteilung, Gruppenzugehörigkeit u.ä.).
Heimatverzeichnis	Das Heimatverzeichnis des Benutzers bzw. das Startverzeichnis nach dem Login.
Shell	Die Shell, die nach der Anmeldung gestartet werden soll. Bleibt dieses Feld frei, dann wird die Standardshell /bin/sh gestartet.

Hier ein Beispiel für einen Systembenutzer:

```
uucp:x:10:14:Unix-to-Unix CoPy system:/etc/uucp:/bin/bash
```

Der Benutzer heißt uucp, das Passwort ist in der Datei /etc/shadow gespeichert (x), die UID ist 10, die GID 14, als erläuternde Bezeichnung trägt der Benutzer den Namen „Unix-to-Unix CoPy system“, das Startverzeichnis nach der Anmeldung ist /etc/uucp, und die vorgeschlagene Shell ist die bash.

An dieser Stelle sei nochmals darauf hingewiesen, dass die meisten Linux-Distributionen komfortable Werkzeuge zur Benutzerverwaltung mitliefern und es auch eine Reihe von Befehlen gibt, die für die Benutzerverwaltung verwendet werden können

/etc/shadow

Bei früheren Versionen von Linux speicherte man die die Passwörter direkt in die passwd-Datei. Allerdings war dies durch einen sogenannten Wörterbuchangriff und der beispielsweise mit Hilfe des Programmes crypt möglich, diese Passwörter in vielen Fällen zu entschlüsseln und auszulesen. Deshalb hat man die Datei /etc/shadow eingeführt, in der die Angaben über die Passwörter durch ein spezielles System besser geschützt werden.

Der Eintrag in diese Datei erfolgt nach einem ähnlichen Schema, wie in der Datei /etc/passwd:

```
Benutzername : Passwort : DOC : MinD : MaxD : Warn : Exp : Dis : Res
```

Benutzername	**Dies ist der Benutzername in druckbaren Zeichen, meistens in Kleinbuchstaben.**
Passwort	Hier steht verschlüsselt das Passwort des Benutzers. Wenn hier ein * oder ! steht, dann bedeutet dies, dass kein Passwort vorhanden bzw. eingetragen ist.
DOC	Day of last change: der Tag, an dem das Passwort zuletzt geändert wurde. Besonderheit hier: Der Tag wird als Integer-Zahl in Tagen seit dem 1.1.1970 angegeben.
MinD	Minimale Anzahl der Tage, die das Passwort gültig ist.
MaxD	Maximale Anzahl der Tage, die das Passwort gültig ist.
Warn	Die Anzahl der Tage vor Ablauf der Lebensdauer, ab der vor dem Verfall des Passwortes zu warnen ist.
Exp	Hier wird festgelegt, wieviele Tage das Passwort trotz Ablauf der MaxD noch gültig ist.
Dis	Bis zu diesem Tag (auch hier wird ab dem 1.1.1970 gezählt) ist das Benutzerkonto gesperrt
Res	Reserve, dieses Feld hat momentan keine Bedeutung.

Es folgt wieder ein Beispiel:

```
selflinux:/heSIGnYDr6MI:11995:1:99999:14:::
```

Der Benutzer heißt selflinux, das Passwort lautet verschlüsselt „/heSIGnYDr6MI“. Es wurde zuletzt geändert, als 11995 Tage seit dem 1.1.1970 vergangen waren. Das Passwort ist minimal einen Tag gültig, maximal 99999 Tage (was man als immer deuten kann - 99999 Tage sind ca. 274 Jahre). Es soll ab 14 Tage vor Ablauf des Passwortes gewarnt werden. Die anderen Werte sind vom Administrator nicht definiert und bleiben daher leer.

/etc/group

In dieser Datei finden Sie die Benutzergruppen und ihre Mitglieder. In der Datei /etc/passwd wird mit der GID eigentlich schon eine Standardgruppe für den Benutzer festgelegt. In der /etc/group können Sie weitere Gruppenzugehörigkeiten definieren. Das hat in der Praxis vor allem in Netzwerken eine große Bedeutung, weil Sie so in der Lage sind, z.B. Gruppen für Projekte oder Verwaltungseinheiten zu bilden. Für diese Gruppen kann man dann entsprechend die Zugriffsrechte einstellen. Dies hat dann wiederum den Vorteil, dass man die Daten gegen eine unbefugte Benutzung absichern kann.

Der Eintrag einer Gruppe in die Datei sieht so aus:

```
Gruppenname : Passwort : GID : Benutzernamen (Mitgliederliste)
```

Gruppenname	**Der Name der Gruppe in druckbare Zeichen, auch hier meistens Kleinbuchstaben.**
Passwort	Die Besonderheit hier ist folgende: Wenn das Passwort eingerichtet ist, können auch Nichtmitglieder der Gruppe Zugang zu den Daten der Gruppe erhalten, wenn ihnen das Passwort bekannt ist. Ein x sagt hier aus, das das Passwort in /etc/gshadow abgelegt ist. Der Eintrag kann auch entfallen, dann ist die Gruppe nicht durch ein Passwort geschützt. In diesem Fall kann jedoch auch kein Benutzer in die Gruppe wechseln, der nicht in diese Gruppe eingetragen ist.
GID	Gruppen-ID der Gruppe
Benutzernamen	hier werden die Mitglieder der Gruppe eingetragen. Diese sind durch ein einfaches Komma getrennt.

Für einen korrekten Eintrag in die /etc/group reicht eigentlich der Gruppenname und die GID aus. Damit ist die Gruppe dem System bekannt gemacht. Die Felder für das Passwort und die Benutzernamen können frei bleiben.

Soll der Benutzer nur in seiner Standardgruppe bleiben, ist kein Eintrag in die /etc/group notwendig. Hier reicht der Eintrag in die /etc/passwd völlig aus, weil dort die Standardgruppe schon mit angegeben wird. Nur wenn der Benutzer in weiteren bzw. mehreren Gruppen Mitglied sein soll, muss dies in die /etc/group-Datei eingetragen werden. Für Passwörter gilt das oben in der Tabelle Gesagte.

Hier sehen Sie ein Beispiel für einen Eintrag:

```
dialout:x:16:root,tatiana,steuer,selflinux
```

Sie sehen eine Gruppe mit der GID „16“ und den Namen dialout. (Zur Information: dialout erlaubt es normalen Benutzern einen ppp-Verbindungsaufbau zu starten, normalerweise hat nur root dieses Recht). Das x bedeutet hier, dass das Passwort in /etc/shadow abgelegt ist. Da in /etc/gshadow hier bei Passwort ein * steht, ist also kein Passwort für die Gruppe vorhanden (Das bedeutet wiederum, das nur die eingetragenen Mitglieder Zugang zu dieser Gruppe haben). Mitglieder der Gruppe sind: root, tatiana, steuer, selflinux.

Benutzerklassen: user, group und others

Aus der Sicht des Systems existieren drei Benutzerklassen, wenn entschieden werden soll, ob die Berechtigung für einen Dateizugriff existiert oder nicht. Soll beispielsweise eine Datei gelöscht werden, so muss das System ermitteln, ob der Benutzer, welcher die Datei löschen möchte, das erforderliche Recht besitzt:

user@linux ~$ rm testdatei rm: Entfernen (unlink) von „testdatei“ nicht möglich: Keine Berechtigung

In diesem Fall wurde dem rm Kommando der beabsichtigte löschende Zugriff auf die Datei verwehrt - der ausführende Benutzer hatte nicht das Recht, die Datei zu löschen. Um diese Entscheidung zu treffen, verwendet das System das Konzept der Benutzerklassen. Drei Benutzerklassen werden unterschieden: user, group und others. Jede dieser Benutzerklassen ist wiederum in ein Lese-, Schreib- und Ausführrecht unterteilt. Diese werden im Folgenden als Berechtigungsklassen bezeichnet. Somit ergibt sich folgende Körnung für die einfachen Zugriffsrechte einer Datei:

Recht	Beschreibung
user-read	Leserecht für Dateieigentümer
user-write	Schreibrecht für Dateieigentümer
user-execute	Ausführrecht für Dateieigentümer
group-read	Leserecht für Gruppe des Dateieigentümers
group-write	Schreibrecht für Gruppe des Dateieigentümers
group-execute	Ausführrecht für Gruppe des Dateieigentümers
other-read	Leserecht für alle anderen Benutzer
other-write	Schreibrecht für alle anderen Benutzer
other-execute	Ausführrecht für alle anderen Benutzer

Benutzerklassen sind also eng mit der Eigentümerschaft von Dateien verbunden. Jede Datei und jedes Verzeichnis ist sowohl einem Benutzer (einer UID) als auch einer Gruppe (einer GID) zugeordnet. UID und GID gehören zur elementaren Verwaltungsinformation von Dateien und Verzeichnissen und werden in der sogenannten Inode gespeichert.

Beim Zugriff auf eine Datei werden nun UID und GID des zugreifenden Prozesses mit UID und GID der Datei verglichen. Ist other-read gesetzt, darf jeder Benutzer lesend zugreifen und ein weiterer Vergleich erübrigt sich. Ist lediglich group-read gesetzt, muss der Zugreifende mindestens der Gruppe des Dateieigentümers angehören, d.h. eine identische GID aufweisen. Ist ausschließlich user-read gesetzt, so darf nur der Eigentümer selbst die Datei lesen. root ist von dieser Einschränkung freilich ausgenommen. („Ich bin root, ich darf das!“). Von sehr speziellen Ausnahmen abgesehen, die sich außerhalb der hier besprochenen Rechteklassen bewegen, ist root in seinen Aktionen in keinerlei Weise eingeschränkt.

Dateimanagement

Nachdem unter Linux das Prinzip **Everything is a file** gilt, werden hier die Besonderheiten von Dateien beschrieben.

Dateien

Datei- und Verzeichnisnamen können bis zu 255 Zeichen lang sein. Dabei wird in jedem Fall zwischen Groß- und Kleinschreibung unterschieden. Die Dateinamen

- DATEI
- datei
- Datei

bezeichnen drei unterschiedliche Dateien. Ein Dateiname darf beliebig viele Punkte enthalten, also zum Beispiel auch Datei.Teil.1.txt. Ein Punkt gilt als normales Zeichen in einem Dateinamen. Dateien, die mit Punkt beginnen, gelten als versteckt und werden normalerweise nicht angezeigt - zum Beispiel .datei. Das Zeichen zum Trennen von Verzeichnis- und Dateinamen ist der Slash („/“) statt dem Backslash („\“) bei Windows.

Es gibt verschiedene Dateiarten: (in Klammer die offizielle Darstellung, wie sie symbolisiert werden)

- Normale Dateien (-)
- Verzeichnisse (d)
- Symbolische Links (l)
- Blockorientierte Geräte (b)
- Zeichenorientierte Geräte (c)
- Named Pipes (p)

Wir sehen hier schon, dass auch Verszeichnisse bloß eine bestimmte Dateiart sind. Eine spezielle nämlich, in der andere Dateien aufgelistet sind. Mit einem Dateibrowser (von Windows kennen wir „Explorer“, bei Apple den „Finder“) sehen wir uns immer nur genau diese Verzeichnisse an, sofern wir nicht mittels verschiedener Plugins die Dateien selbst auswerten und Textdokumente, Bilder anzeigen oder Videos und Musik wiedergeben.

In einem Unix-Dateisystem hat jede einzelne Datei jeweils einen Eigentümer und eine Gruppenzugehörigkeit. Neben diesen beiden Angaben besitzt jede Datei noch einen Satz Attribute, die bestimmen, wer die Datei wie benutzen darf. Diese Attribute werden dargestellt als „rwx“. Dabei steht r für lesen (read), w für schreiben (write) und x für ausführen (execute).

Das Dateisystem

Das Dateisystem ist die Ablageorganisation auf einem Datenträger eines Computers. Um die Funktionsweise zu verstehen, betrachten wir einen Datenträger, die Festplatte, näher:

Die Festplatte besteht aus mehreren Scheiben mit einer magnetisierbaren Oberfläche, auf die die

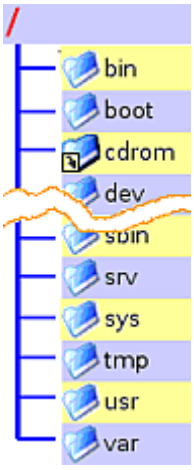
Schreibköpfe unsere Daten als Einsen (ein) und Nullen (aus) abspeichern. Um diese aber vernünftig adressieren zu können, benutzen wir Dateisysteme. Ein solches teilt die Festplatte (eigentlich die „Partition“, denn die Festplatte wird häufig in mehrere Partitionen aufgeteilt, die dann unabhängig formatiert werden können) in kleine Einheiten, die „Blöcke“, welche aus Performancegründen häufig noch zu „Clustern“ zusammengefasst werden.

Der Block (oder Cluster) ist dann die kleinste Einheit, in die eine Datei geschrieben wird, jede Datei benötigt dadurch immer diesen Speicherplatz (oder ein vielfaches) auf der Festplatte.

Von Windows kennen wir NTFS und FAT32, bzw. Apple-Benutzer werden schon von HFS+ gehört haben. Unter Linux werden meist ext2, ext3 oder ext4 (Second, Third bzw. Fourth Extented File System) verwendet. „ext3“ unterscheidet sich von „ext2“ nur dadurch, dass zusätzlich ein „Journal“ geschrieben wird, welches bei Systemabstürzen eine zuverlässige Wiederherstellung möglich macht. „ext4“ ist eine performantere Weiterentwicklung von „ext3“ und heute Standard. Daneben gibt es gelegentlich noch ReiserFS, XFS oder JFS, aber die Wahl des Dateisystems bestimmt tatsächlich immer das Abwägen zwischen höherer Sicherheit und schnellerer Schreibgeschwindigkeit - mit oder ohne Journal.

Verzeichnisstruktur

Das Dateisystem beginnt mit einem Wurzelverzeichnis (auch Rootverzeichnis genannt - /). Es enthält im Regelfall keine Dateien, sondern nur die folgenden Verzeichnisse (Ubuntu):



/bin

Von: binaries (Programme); muss bei Systemstart vorhanden sein; enthält für Linux unverzichtbare Programme; diese Programme können im Gegensatz zu /sbin von allen Benutzern ausgeführt werden; /bin darf keine Unterverzeichnisse enthalten.

/boot

Muss bei Systemstart vorhanden sein; Enthält zum Booten benötigte Dateien.

/dev

Von devices (Geräte); muss bei Systemstart vorhanden sein; enthält alle Gerätedateien, über die die Hardware im Betrieb angesprochen wird

/etc

Von: et cetera („alles übrige“), später auch: editable text configuration (änderbare Text Konfiguration); muss bei Systemstart vorhanden sein; enthält Konfigurations- und Informationsdateien des Basissystems.

- /etc/init.d: dort liegen alle Start- und Stopskripte
- /etc/opt: Verzeichnisse und Konfigurationsdateien für Programme in /opt
- /etc/network: Verzeichnisse und Konfigurationsdateien des Netzwerkes
- ….

/home

Von: home-directory (Heimatverzeichnis); enthält pro Benutzer ein Unterverzeichnis; jedes Verzeichnis wird nach dem Anmeldenamen benannt

/lib

Von: libraries (Bibliotheken); muss bei Systemstart vorhanden sein; enthält unverzichtbare Bibliotheken fürs Booten und die dynamisch gelinkten Programme des Basissystems;

/lost+found

(verloren und gefunden); Dateien und Dateifragmente, die beim Versuch, ein defektes Dateisystem zu reparieren, übrig geblieben sind.

/media

Für (Speicher-)Medien. Enthält Unterverzeichnisse, welche als mount- oder Einhängepunkte für transportable Medien wie z.B. externe Festplatten, USB-Sticks, CD-ROMs, DVDs und andere Datenträger dienen. Ubuntu legt hier auch die Einhängepunkte für Partitionen an. Unterverzeichnisse sind u.a.:

- /media/floppy: Einhängepunkt für Disketten
- /media/cdrom0: Einhängepunkt für CD-ROMs

/mnt

Von: mount (eingehängt); normalerweise leer; kann für temporär eingehängte Partitionen verwendet werden. Für Datenträger, die hier eingehängt werden, wird im Gegensatz zu /media kein Link auf dem Desktop angelegt

/opt

Von: optional; ist für die manuelle Installation von Programmen gedacht, die ihre eigenen Bibliotheken mitbringen und nicht zur Distribution gehören;

/proc

Von: processes (laufende Programme); muss bei Systemstart vorhanden sein; enthält Schnittstellen zum aktuell geladenen Kernel und seinen Prozeduren; Dateien lassen sich mittels cat auslesen; Beispiele: version (Kernelversion), swaps (Swapspeicherinformationen), cpuinfo, interrupts, usw.;

/root

Ist das Homeverzeichnis des Superusers (root). Der Grund, wieso sich das /root-Verzeichnis im Wurzelverzeichnis und nicht im Verzeichnis /home befindet, ist, dass das Homeverzeichnis von Root immer erreichbar sein muss, selbst wenn die Home-Partition aus irgendeinem Grund (Rettungs-Modus, Wartungsarbeiten) mal nicht eingehängt ist.

/sbin

Von: system binaries (Systemprogramme); muss bei Systemstart vorhanden sein; enthält alle Programme für essentielle Aufgaben der Systemverwaltung; Programme können nur vom Systemadministrator (root) oder mit Superuserrechten ausgeführt werden

/srv

Von: services (Dienste); Verzeichnisstruktur noch nicht genau spezifiziert; soll Daten der Dienste enthalten; unter Ubuntu in der Regel leer

/sys

Von: system; im FHS noch nicht spezifiziert; erst ab Kernel 2.6. im Verzeichnisbaum enthalten; besteht ebenso wie /proc hauptsächlich aus Kernelschnittstellen

/tmp

Von: temporary (temporär); enthält temporäre Dateien von Programmen; Verzeichnis soll laut FHS beim Booten geleert werden.

/usr

Von: user (siehe: Herkunft); enthält die meisten Systemtools, Bibliotheken und installierten Programme; der Name ist historisch bedingt - früher, als es /home noch nicht gab, befanden sich hier auch die Benutzerverzeichnisse;

- /usr/bin : Anwenderprogramme; Hier liegen die Desktopumgebungen und die dazu gehörigen Programme, aber auch im Nachhinein über die Paketverwaltung installierte Programme, wie Audacity

/var

Von variable (variabel); enthält nur Verzeichnisse; Dateien in den Verzeichnissen werden von den Programmen je nach Bedarf geändert (im Gegensatz zu /etc); Beispiele: Log-Dateien, Spielstände, Druckerwarteschlange

- /var/log : Alle Log-Dateien der Systemprogramme; Beispiele: Xorg.0.log (Log-Datei des XServer), kern.log (Logdatei des Kernels), dmesg (letzte Kernelmeldungen), messages (Systemmeldungen); Siehe auch Logdateien

DATEIRECHTE

UNIX-Systeme wie Linux verwalten ihre Dateien in einem virtuellen Dateisystem (VFS, Virtual File System). Dieses ordnet jeder Datei über eindeutig identifizierbare Inodes unter anderem folgende Eigenschaften zu:

- Dateityp (einfache Datei, Verzeichnis, Link, …)
- Zugriffsrechte (Eigentümer-, Gruppen- und sonstige Rechte)
- Größe
- Zeitstempel
- Verweis auf Dateiinhalt

Jedes unter Linux gängige UNIX-Dateisystem (z.B. ext2/3/4, ReiserFS, xfs usw.) unterstützt diese Rechte. Gar nicht umgesetzt werden die Rechte hingegen auf VFAT-Dateisystemen; dort können Dateirechte lediglich beim Einhängen simuliert werden. Partitionen mit dem Windows-Dateisystem NTFS werden zwar in Linux standardmäßig ähnlich wie VFAT-Partitionen behandelt; mit den Mount-Optionen permissions und acl lässt sich aber auch auf NTFS-Partitionen eine echte Rechteverwaltung wie bei UNIX-Dateisystemen einrichten. Siehe hierzu Windows-Partitionen einbinden sowie NTFS-3G.

Rechte in symbolischer Darstellung

Im Terminal lassen sich die Rechte mit dem Befehl ls -l anzeigen. Im Folgenden sind als Beispiel die Dateirechte des Verzeichnisses /var/mail/ dargestellt

```
 ls -l /var/mail/
drwxrwsr-x 2 root mail 4096 Apr 23  2012 /var/mail/
```

Für die Darstellung der Rechte sind die markierten Teile der Ausgabe relevant:

- Der erste Buchstabe (d) kennzeichnet den Dateityp.
- Danach folgen die Zugriffsrechte (rwxrwsr-x).
- Eigentümer der Datei
- Gruppe

Wie auch in anderen Betriebssystemen kann man verschiedene Rechte für Eigentümer (Owner) und Gruppe (Group) vergeben. Neben Eigentümer und Gruppe gibt es noch eine weitere, allgemeine Gruppe. Diese Gruppe nennt sich andere (Others).

Darstellungsarten

Neben der symbolischen Darstellung (z.B. rwxrwxr-x) gibt es auch noch eine oktale Darstellung (nach dem Oktalsystem). Die Grundrechte (Lesen, Schreiben, Ausführen) und Kombinationen daraus werden hierbei durch eine einzelne Ziffer repräsentiert und dem Eigentümer, der Gruppe und allen anderen zugeordnet. Je nach Anwendung wird dabei von unterschiedlichen Grundwerten ausgegangen und entweder Rechte gegeben oder entzogen. Bei chmod wird beispielsweise von der Grundeinstellung „keine Rechte“ (000) ausgegangen und Rechte gegeben, wohingegen bei umask von „alle Rechte

vorhanden“ (777) ausgegangen und Rechte entzogen werden. Entsprechend sind die Werte je nach Anwendung anders.

Mögliche Werte für:

Recht(e)	chmod (octal)	umask (octal)	Symbolisch	Binäre Entsprechung
Lesen, schreiben und ausführen	7	0	rwx	111
Lesen und Schreiben	6	1	rw-	110
Lesen und Ausführen	5	2	r-x	101
Nur lesen	4	3	r–	100
Schreiben und Ausführen	3	4	-wx	011
Nur Schreiben	2	5	-w-	010
Nur Ausführen	1	6	–x	001
Keine Rechte	0	7	—	000

Hier ein paar Beispiele:

- rwxrwxrwx entspricht 0777 (chmod) oder 0000 (umask): Jeder darf lesen, schreiben und ausführen.
- rwxr-xr-x entspricht 0755 (chmod) oder 0022 (umask): Jeder darf lesen und ausführen, aber nur der Dateibesitzer darf diese Datei (oder das Verzeichnis) auch verändern.

Die nachfolgenden Erklärungen beziehen sich vor allem auf Dateien vom Typ File (ohne Kennbuchstaben) und „Ordner“ (Directory, Kennbuchstabe d).

Nach dem Dateityp kommen drei Zeichengruppen zu je drei Zeichen. Diese kennzeichnen die Zugriffsrechte für die Datei bzw. das Verzeichnis. Hat der Benutzer/Gruppe/andere ein Recht, so wird der Buchstabe dafür angezeigt; ansonsten wird ein - dafür angezeigt.

In obigen Beispiel erscheint nach dem Dateityp dann die Zeichenfolge rwxrwsr-x. Wenn man diese in drei Dreiergruppen aufteilt, erhält man diese Gruppen:

- rwx: Rechte des Eigentümers
- rws: Rechte der Gruppe
- r-x: Recht von allen anderen (others)

Die folgende Tabelle erklärt die Bedeutung der einzelnen Buchstaben, Diese stehen immer in der gleichen Reihenfolge:

Symbole für Zugriffsrechte		
Zeichen	**Bedeutung**	**Beschreibung**
r	Lesen (read) Erlaubt lesenden Zugriff auf die Datei. Bei einem Verzeichnis können damit die Namen der enthaltenen Dateien und Ordner abgerufen werden (nicht jedoch deren weitere Daten wie z.B. Berechtigungen, Besitzer, Änderungszeitpunkt, Dateiinhalt etc.).	
w	Schreiben (write) Erlaubt schreibenden Zugriff auf eine Datei. Für ein Verzeichnis gesetzt, können Dateien oder Unterverzeichnisse angelegt oder gelöscht werden, sowie die Eigenschaften der enthaltenen Dateien/Verzeichnisse verändert werden.	

Symbole für Zugriffsrechte		
Zeichen	**Bedeutung**	**Beschreibung**
x	Ausführen (execute) Erlaubt das Ausführen einer Datei, wie das Starten eines Programms. Bei einem Verzeichnis ermöglicht dieses Recht, in diesen Ordner zu wechseln und weitere Attribute zu den enthaltenen Dateien abzurufen (sofern man die Dateinamen kennt ist dies unabhängig vom Leserecht auf diesen Ordner). Statt x kann auch ein Sonderrecht angeführt sein.	

Sonderrechte

Die oben gezeigten Dateirechte kann man als Basisrechte bezeichnen. Für besondere Anwendungen gibt es zusätzlich noch besondere Dateirechte. Der Einsatz dieser ist nur dann ratsam, wenn man genau weiß, was man tut, da dies unter Umständen zu Sicherheitsproblemen führen kann.

Sonderrechte		
Zeichen	**Bedeutung**	**Beschreibung**
s	Set-UID-Recht (SUID-Bit)	Das Set-UID-Recht („Set User ID“ bzw. „Setze Benutzerkennung“) sorgt bei einer Datei mit Ausführungsrechten dafür, dass dieses Programm immer mit den Rechten des Dateibesitzers läuft. Bei Ordnern ist dieses Bit ohne Bedeutung.
s (S)	Set-GID-Recht (SGID-Bit)	Das Set-GID-Recht („Set Group ID“ bzw. „Setze Gruppenkennung“) sorgt bei einer Datei mit Ausführungsrechten dafür, dass dieses Programm immer mit den Rechten der Dateigruppe läuft. Bei einem Ordner sorgt es dafür, dass die Gruppe an Unterordner und Dateien vererbt wird, die in diesem Ordner neu erstellt werden.
t (T)	Sticky-Bit	Das Sticky-Bit („Klebrig“) hat auf modernen Systemen nur noch eine einzige Funktion: Wird es auf einen Ordner angewandt, so können darin erstellte Dateien oder Verzeichnisse nur vom Dateibesitzer gelöscht oder umbenannt werden. Verwendet wird dies z.B. für /tmp.

Die Symbole für die Sonderrechte erscheinen an der dritten Stelle der Zugriffsrechte, die normalerweise dem Zeichen x (für executable) vorbehalten ist, und ersetzen ggf. dieses. Die Set-UID/GID-Rechte werden anstelle des x für den Besitzer bzw. die Gruppe angezeigt, das Sticky-Bit anstelle des x für andere. Wenn das entsprechende Ausführrecht gesetzt ist, wird ein Kleinbuchstabe verwendet, ansonsten ein Großbuchstabe.

INODES

Ein Inode (englisch index node, gesprochen „eye-node“) ist die grundlegende Datenstruktur zur Verwaltung von Dateisystemen mit unixartigen Betriebssystemen. Jeder Inode wird innerhalb einer Partition eindeutig durch seine Inode-Nummer identifiziert. Jeder Namenseintrag in einem Verzeichnis verweist auf genau einen Inode. Dieser enthält die Metadaten der Datei und verweist auf die Daten der Datei beziehungsweise die Dateiliste des Verzeichnisses.

Die Anwendungssoftware unterscheidet beim Lesen oder Schreiben von Daten nicht mehr zwischen Gerätetreibern und regulären Dateien. Durch das Inode-Konzept gilt bei den Unixvarianten alles als Datei („On UNIX systems it is reasonably safe to say that everything is a file: …“). Dadurch unterscheiden sich solche Betriebssysteme in der Verwaltung ihres Datenspeichers von anderen Systemen wie Microsoft Windows mit NTFS, aber auch von VMS oder MVS.

Grundsätzliches

Speichert man eine Datei auf einem Computer ab, so muss nicht nur der Dateiinhalt (Nutzdaten) gespeichert werden, sondern auch Metadaten, wie zum Beispiel der Zeitpunkt der Dateierstellung oder der Besitzer der Datei. Gleichzeitig muss der Computer einem Dateinamen – inklusive Dateipfad – die entsprechenden Nutzdaten und Metadaten effizient zuordnen können. Die Spezifikation, wie diese Daten organisiert und auf einem Datenträger gespeichert werden, nennt man Dateisystem. Dabei gibt es abhängig von Einsatzbereich und Betriebssystem unterschiedliche Dateisysteme. Umgesetzt wird die Dateisystemspezifikation von einem Treiber, der wahlweise als ein Kernel-Modul des Betriebssystemkern (Kernel) oder seltener als gewöhnliches Programm im Userspace umgesetzt sein kann.

Dateisysteme unixoider Betriebssysteme – wie Linux und macOS – verwenden sogenannte Inodes. Diese enthalten die Metadaten sowie Verweise darauf, wo Nutzdaten gespeichert sind. An einem speziellen Ort des Dateisystems, dem Superblock, wird wiederum die Größe, Anzahl und Lage der Inodes gespeichert. Die Inodes sind durchnummeriert und an einem Stück auf dem Datenträger gespeichert. Das Wurzelverzeichnis eines Dateisystems besitzt eine feste Inodenummer. Unterordner sind „gewöhnliche“ Dateien, welche eine Liste der darin enthaltenen Dateien mit der Zuordnung der dazugehörenden Inodenummern als Nutzdaten enthalten.

Soll also beispielsweise die Datei /bin/ls geöffnet werden, so läuft dies, vereinfacht, wie folgt ab:

- Der Dateisystemtreiber liest den Superblock aus, dadurch erfährt er die Startposition der Inodes und deren Länge, somit kann nun jeder beliebige Inode gefunden und gelesen werden.
- Nun wird der Inode des Wurzelverzeichnisses geöffnet. Da dies ein Ordner ist, befindet sich darin ein Verweis auf die Speicherstelle der Liste aller darin enthaltenen Dateien mitsamt ihren Inodenummern. Darin wird das Verzeichnis bin gesucht.
- Nun kann der Inode des bin-Verzeichnisses gelesen werden und analog zum letzten Schritt der

Inode der Datei ls gefunden werden.
- Da es sich bei der Datei ls nicht um ein Verzeichnis, sondern um eine reguläre Datei handelt, enthält ihr Inode nun einen Verweis auf die Speicherstelle der gewünschten Daten.

Aufbau

Jedem einzelnen von einem Schrägstrich / (slash) begrenzten Namen ist genau ein Inode zugeordnet. Dieser speichert folgende Metainformationen zur Datei, aber nicht den eigentlichen Namen:

- Die Art der Datei (reguläre Datei, Verzeichnis, Symbolischer Link, …), siehe unten;
- die nummerische Kennung des Eigentümers (UID, user id) und der Gruppe (GID, group id);
- die Zugriffsrechte für den Eigentümer (user), die Gruppe (group) und alle anderen (others);
- Die klassische Benutzer- und Rechteverwaltung geschieht mit den Programmen chown (change owner), chgrp (change group) und chmod (change mode). Durch Access Control Lists (ACL) wird eine feinere Rechtevergabe ermöglicht.
- verschiedene Zeitpunkte der Datei: Erstellung, Zugriff (access time, atime) und letzte Änderung (modification time, mtime);
- die Zeit der letzten Status-Änderung des Inodes (status, ctime);
- die Größe der Datei;
- den Linkzähler (siehe unten);
- einen oder mehrere Verweise auf die Blöcke, in denen die eigentlichen Daten gespeichert sind.

Reguläre Dateien

Reguläre Dateien (engl. regular files) sind sowohl Anwenderdaten als auch ausführbare Programme. Letztere sind durch das executable-Recht gekennzeichnet und werden beim Aufruf durch das System in einem eigenen Prozess gestartet. Als „ausführbar“ gelten nicht nur kompilierte Programme, sondern auch Skripte, bei denen der Shebang den zu verwendenden Interpreter angibt. Bei „dünnbesetzten Dateien“, sogenannten sparse files, unterscheidet sich die logische Größe vom durch die Datenblöcke tatsächlich belegten Festplattenplatz.

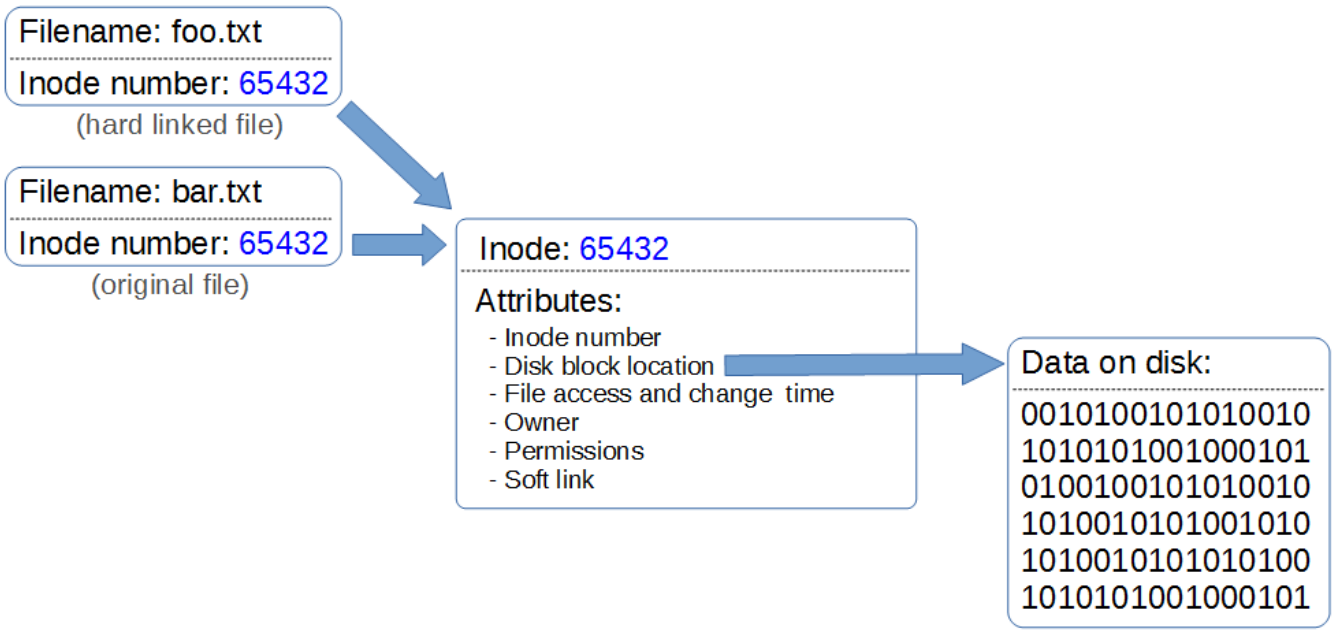
Verzeichnisse

Verzeichnisse sind Dateien, deren „Dateiinhalt“ aus einer tabellarischen Liste der darin enthaltenen Dateien besteht. Die Tabelle enthält dabei eine Spalte mit den Dateinamen und eine Spalte mit den zugehörigen Inodenummern. Bei manchen Dateisystemen umfasst die Tabelle noch weitere Informationen, so speichert ext4 darin auch den Dateityp aller enthaltenen Dateien ab, so dass dieser beim Auflisten eines Verzeichnisinhalts nicht aus den Inodes aller Dateien ausgelesen werden muss. Für jedes Verzeichnis existieren immer die Einträge . und .. als Verweise auf das aktuelle bzw. übergeordnete Verzeichnis.

Harte Links (hard links)

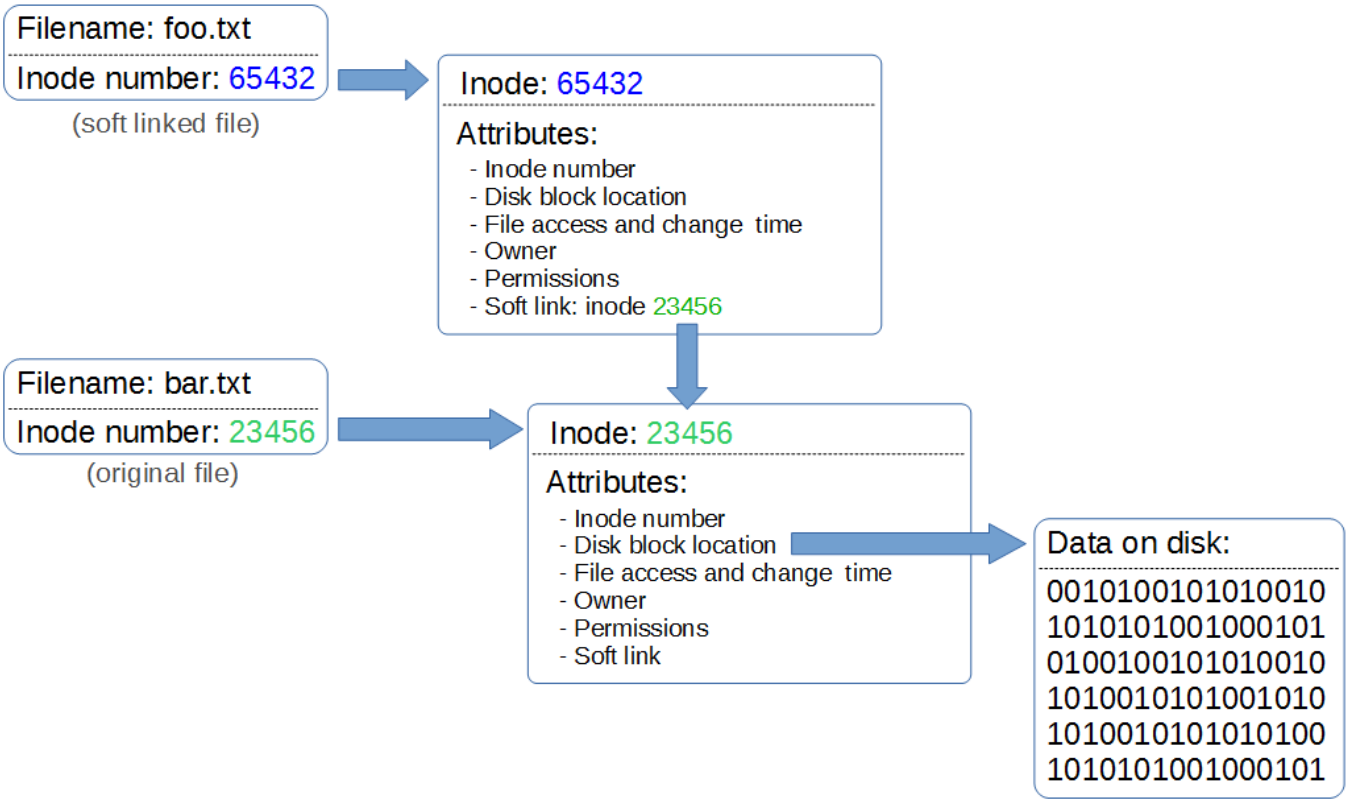
Bei Harten Links hingegen handelt es sich nicht um spezielle Dateien. Von einem Harten Link spricht man, wenn auf einen Inode mehrfach von verschiedenen Ordnern oder verschiedenen Dateinamen verwiesen wird. Alle Verweise auf den Inode sind gleichwertig, es gibt also kein Original. Im Inode gibt

der Linkzähler an, wie viele Dateiennamen auf diesen verweisen, er steht nach dem Anlegen einer Datei also bei 1 und wird erhöht, sobald weitere Hardlinks für diese Datei erstellt werden. Bei einem Verzeichnis beträgt er zwei mehr, als Unterordner darin enthalten sind, da neben dem Eintrag im Ordner darüber und dem Eintrag '.' im Ordner noch die Einträge '..' in allen Unterordnern darauf verweisen. Wird eine Datei gelöscht, so wird ihr Eintrag aus dem übergeordneten Verzeichnis entfernt und der Linkzähler um eins reduziert. Beträgt der Linkzähler dann 0, wird gegebenenfalls abgewartet, bis die Datei von keinem Programm mehr geöffnet ist, und erst anschließend der Speicherplatz freigegeben.



Symbolische Links (symbolic links, soft links, symlinks)

Bei symbolischen Links handelt es sich um spezielle Dateien, die anstelle von Daten einen Dateipfad enthalten, auf den der Link verweist. Je nach Dateisystem und Länge des Dateipfads wird der Link entweder direkt im Inode gespeichert oder in einem Datenblock, auf welchen der Inode verweist.



Praxis

Die Inodenummer einer Datei lässt sich mittels des Befehls ls -li Dateiname anzeigen

LINUX-DISTRIBUTIONEN

Wie schon besprochen besteht ein Linux-Betriebssystem aus dem Linux-Kernel und einer großen Anzahl verschiedener Anwenderprogramme. Tatsächlich gibt es ein Projekt, das eine Anleitung bietet, wie man aus den Kernelquellen und selbst selektierten Programmen ein komplettes, maßgeschneidertes Betriebssystem bauen kann. Das Projekt nennt sich „Linux From Scratch“ oder „LFS“ (www.linuxfromscratch.org) und ich kann jedem, der etwas Zeit übrig hat, nur empfehlen, dies selbst einmal zu probieren. Man erhält eine ultrakompaktes, ultraschnelles Betriebssystem und kann sagen, „das ist mein eigenes reinrassiges Linux-System“. Aber was kommt dann?

Man braucht schon einige Zeit bis alle Programme, die so benötigt werden, kompiliert und konfiguriert sind und dann müssen diese auch noch laufend aktualisiert werden. Sicherheitsupdates müssen selbst organisiert und kompiliert werden. Mit all der Administrationsarbeit kommt man zu sonst nichts mehr.

Um das zu vermeiden, gibt es Linux-Distributionen. Diese bieten nicht nur einen fertigen Satz von notwendigen Programmen, sondern auch die regelmäßige Versorgung mit Updates an. Die aus meiner Sicht wichtigsten Distributionen sollen hier aufgeführt werden:

Debian

„Debian“ (www.debian.org) ist eine nicht-kommerzielle Distribution und das „Debian-Projekt“ ist nach der „Debian-Verfassung“ geregelt, die eine demokratische Organisationsstruktur vorsieht. Darüberhinaus ist das Projekt über den „Debian Social Contract“ zu völlig freier Software verpflichtet. Mit einigen 1000 Mitarbeitern ist Debian der Gigant unter den Linux-Systemen.

Da bei Debian eine „stabile Version“ immer eine wirklich stabile Version ist, sind die Entwicklungszeiten relativ lang und böse Zungen behaupten auch, dass die stabile Version schon bei Erscheinen veraltet ist.

Allerdings bietet Debian auch immer schon die zukünftigen Versionen an und so gibt es mehrere Zweige, aus denen man sich bedienen kann:

stable - wirklich stabile Version, die auch für den kommerziellen Serverbetrieb geeignet ist!

testing - die zukünftige stable-Version. Ab einem gewissen Entwicklungsstand wird die Distribution „eingefroren“ (engl. „frozen“) - d.h. es werden keine neueren Versionen von Programmen mehr aufgenommen, sondern nur noch an der Fehlerbeseitigung bei den vorhandenen gearbeitet. Dies entspricht ungefähr dem Zustand, bei dem andere Distributoren ihre „stabilen“ Versionen veröffentlichen. Da ich schon mehrmals eine testing-Version ab dem Anfangsstadium benutzt habe, glaube ich sogar sagen zu können, dass testing nie so instabil ist, wie manche andere Distribution im „ausgereiften“ Zustand. Für den Desktopbetrieb kann ich ein eingefrorenes testing jedenfalls empfehlen.

unstable - ist der erste Anlaufpunkt für neue Versionen von Paketen und Programmen, bevor sie in testing integriert werden. Man installiert sich mit unstable das neueste vom neuen, muss aber wissen,

dass das nicht immer stabil ist.

experimental - ist kein vollständiger Zweig, denn es dient nur dazu, Programme und deren Funktionen zu testen, die sonst das ganze System gefährden würden. Es enthält immer nur die gerade getesteten, bzw. die von diesen benötigten Programmpakete.

Diese Zweige haben auch immer Codenamen und sind, einer Vorliebe der frühen Entwickler folgend, immer nach Figuren aus dem Film „Toy Story“ benannt. So heißt im Moment die stable-Version „Jessie“ und „Stretch“ ist testing. unstable ist immer „Sid“, der Junge von nebenan, der die Spielsachen zerstört, aber es lässt sich auch als Abkürzung für „still in development“ (noch in Entwicklung) deuten.

Und wer einen dieser Zweige installiert hat, kann auf eine unüberschaubare Vielzahl an Programmen zurückgreifen, auf Wunsch (und auf eigene Gefahr) auch aus den anderen Zweigen. Für Anfänger ist es wohl nur bedingt zu empfehlen, obwohl sich in den letzten Jahren sehr viel in Sachen Benutzerführung getan hat. Auch steht ein deutschsprachiges Forum (debianforum.de) zur Verfügung, wo man Hilfe bekommt und wo auch dumme Fragen gestellt werden dürfen. Für ambitionierte Linux-EinsteigerInnen, die sich auch mit den Möglichkeiten ihres Betriebssystems auseinandersetzen wollen, könnte es sogar die beste Distribution sein.

Fedora

Das nicht-kommerzielle „Fedora“ (fedoraproject.org) ist der Nachfolger des traditionsreichen, kommerziellen „Red Hat Linux“, welches nicht mehr selbständig weiterentwickelt wird. Statt dessen verkauft die Firma Red Hat, das auf Fedora basierende „Red Hat Enterprise Linux“.

Fedora ist eine sehr innovative Distribution und vor allem in den USA sehr beliebt. Es werden nur völlig unter freier Lizenz stehende Inhalte akzeptiert, weshalb nach der Installation zum Beispiel keine MP3-unterstützenden Programme zu finden sind. Für Anfänger gibt es bessere Distributionen.

Gentoo Linux

Das nicht-kommerzielle „Gentoo Linux“ (www.gentoo.de) ist eine quellbasierte Linux-Metadistribution - das heißt, alle Programme, inklusive des Kernels, werden selbst kompiliert. Das klingt sehr anstrengend, ist es aber gar nicht so, da die Distribution geeignete

Werkzeuge zur Verfügung stellt, mit denen dies einfachst möglich gelingt. Auch sorgt eine große, sehr aktive „Community“ bei jedem Problem für Rat und Hilfe. Dennoch ist sie für Linux-Neulinge wohl nicht empfehlenswert.

OpenSUSE

„openSUSE“ (www.opensuse.org) ist die zweitbeliebteste Distribution am Heim-PC. Die nicht-kommerzielle Variante der von Novell aufgekauften kommerziellen SUSE-Distribution (heute „SUSE Linux Enterprise“) glänzt mit einem universellen Konfigurationswerkzeug. Sie gilt als anfängerfreundlich. Die neueste Versionsnummer 42.1 bezieht sich übrigens auf die Antwort auf die Frage „nach dem Leben, dem Universum und dem ganzen Rest“ aus Douglas Adam's „Per Anhalter durch die Galaxis“. Schon 1996 hatte die Version 4.2 diesen Bezug.

Slackware

„Slackware“ (www.slackware.com) ist die älteste noch heute existierende Distribution. Sie verzichtet aus Prinzip auf grafische Einrichtungswerkzeuge und ist daher eher nur für fortgeschrittene BenutzerInnen geeignet.

Bisher nicht vorgekommen sind Distributionen, die nur Abwandlungen anderer Distributionen sind und häufig auch deren Quellen benutzen. Vor allem von Debian gibt es unzählige davon. Sie werden als „Derivate“, oder oft auch, etwas abfällig, als „Klone“ bezeichnet. Ein solcher „Debian-Klon“ hat allerdings Geschichte geschrieben:

Ubuntu

Das von der Firma des Gründers gesponserte kostenlose Betriebssystem soll nach dem Willen der Entwickler ein einfach zu installierendes und leicht zu bedienendes Betriebssystem mit aufeinander abgestimmter Software sein. Es bedient sich dazu aus den Quellen von Debian unstable und hat das Ziel, nach der enormen Popularität zu schließen, eindeutig geschafft.

Tatsächlich kann Ubuntu von der Live-CD mit wenigen Mausklicks problemlos auf die Festplatte installiert werden und dann erwartet den Benutzer ein weitgehend komplettes Betriebssystem mit vielen Multimedia-Programmen. Releases erscheinen mit schöner halbjährlicher Regelmäßigkeit und der Upgrade auf diese lässt sich ebenfalls auf Mausklick bewerkstelligen. Alle 2 Jahre gibt es eine „Versionen mit verlägerter Unterstützung“ (Long Term Support oder kurz LTS), die dann deutlich stabiler als die kürzer unterstützten ist. Für EinsteigerInnen ist Ubuntu (www.ubuntu.com) bestens geeignet.

Linux Mint

Diese Distribution war ursprünglich ein Ubuntu-Klon, aber neuerdings ist Linux Mint auch als LMDE - Linux Mint Debian Edition - erhältlich. Den Entwicklern ist es wichtig , die bestmögliche Integration von Programmen zu bieten, die bei Benutzern beliebt, aber eben nicht quelloffene freie Software sind. Die anderen Distribution, inklusive Ubuntu, bieten zwar auch die Installation von „non-free“-Paketen an, aber in einem eigenen Zweig und erst nach der Basisinstallation.

Für absolute Stabilität setzt Linux Mint immer auf LTS (Ubuntu) oder stable (Debian) Versionen auf, aber die integrierten Programme erhalten auch zwischenzeitig Versionsupgrades. Als Vorbild wird die Benutzerfreundlichkeit und Stabilität von Apple's OS X genannt. Auch Linux Mint (www.linuxmint.com) ist für EinsteigerInnen bestens geeignet.

GRAFISCHE OBERFLÄCHEN (GUIs, Desktops)

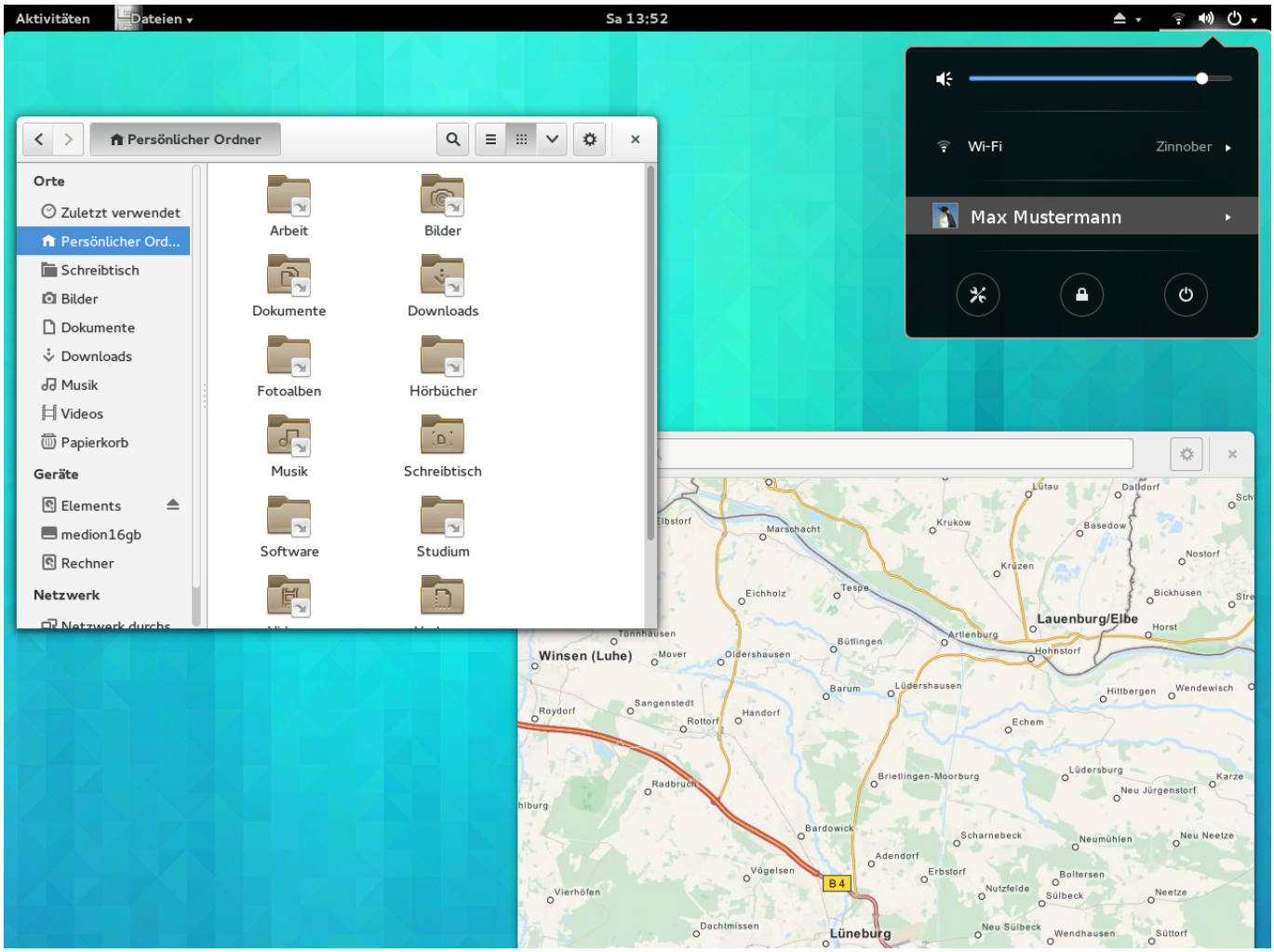
Anders als bei Microsoft und Apple gibt es bei Linux-Distributionen keinen festgelegten Desktop. Alle

Distributionen bieten die Möglichkeit den Desktop zu ändern und zunehmend kann schon bei der Installation der gewünschte Desktop festgelegt werden. Wenngleich für Anfänger im allgemeinen der Standard-Desktop der jeweiligen Distribution sicher eine gute Wahl ist, möchte ich abschließend auch noch kurz einige Desktops vorstellen. Gnome und KDE sind die Platzhirsche auf Linux-Bildschirmen.

Gnome

Gnome (Eigenschreibweise GNOME) [gnoʊm][3] ist eine Desktop-Umgebung für Unix- und Unix-ähnliche Systeme mit einer grafischen Benutzeroberfläche und einer Sammlung von Programmen für den täglichen Gebrauch. Gnome wird unter den freien Lizenzen GPL und LGPL veröffentlicht und ist Teil des GNU-Projekts.

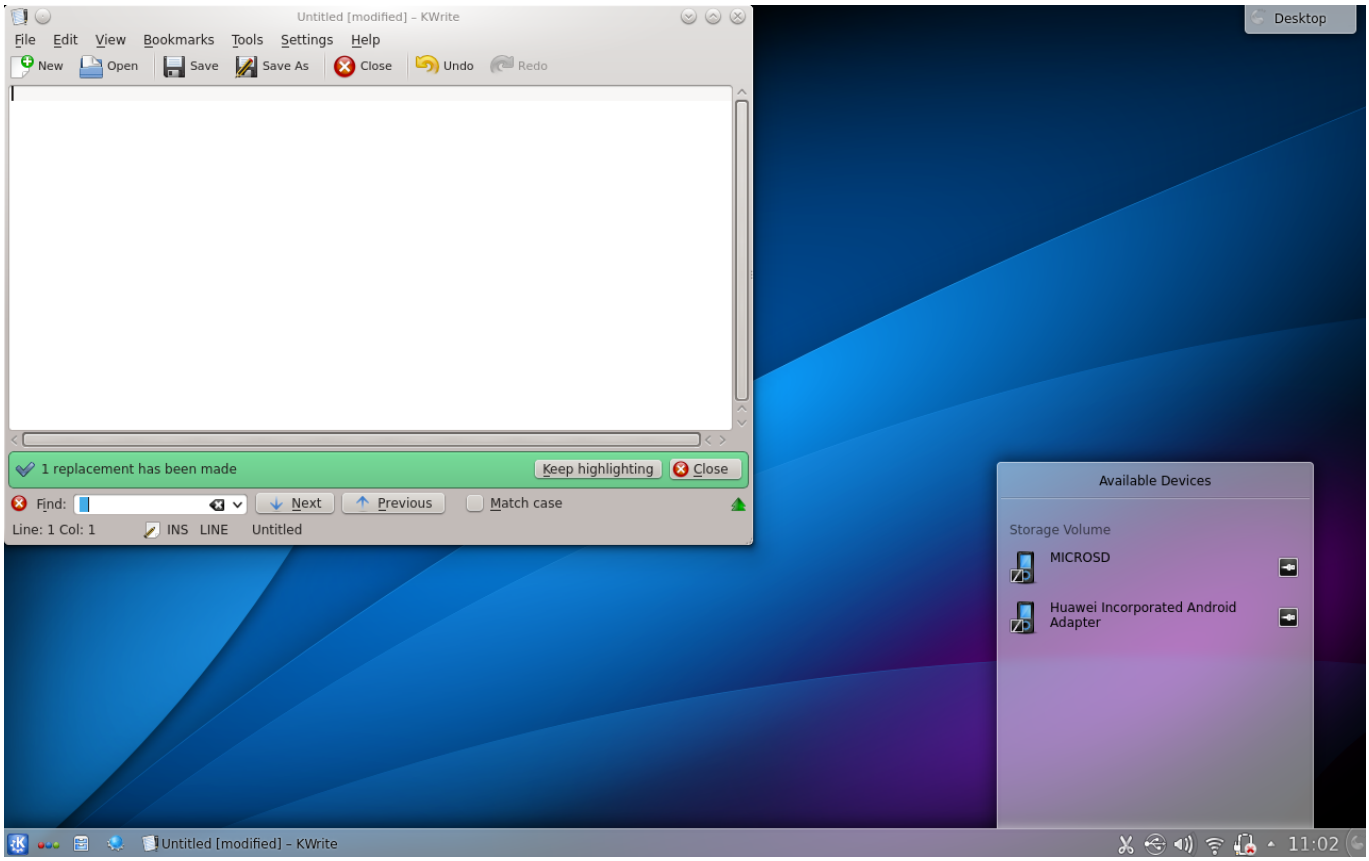
Gnome ist unter anderem der Standard-Desktop von Fedora und Ubuntu. Einige Komponenten von Gnome wurden nach Windows und MacOS portiert, etwa Evolution oder GStreamer, werden jedoch teilweise, wie im Falle von Evolution, nicht mehr länger gepflegt.



KDE

KDE ist eine Community, die sich der Entwicklung freier Software verschrieben hat. Eines der bekanntesten Projekte ist die Desktop-Umgebung KDE Plasma 5 (früher K Desktop Environment,

abgekürzt KDE).

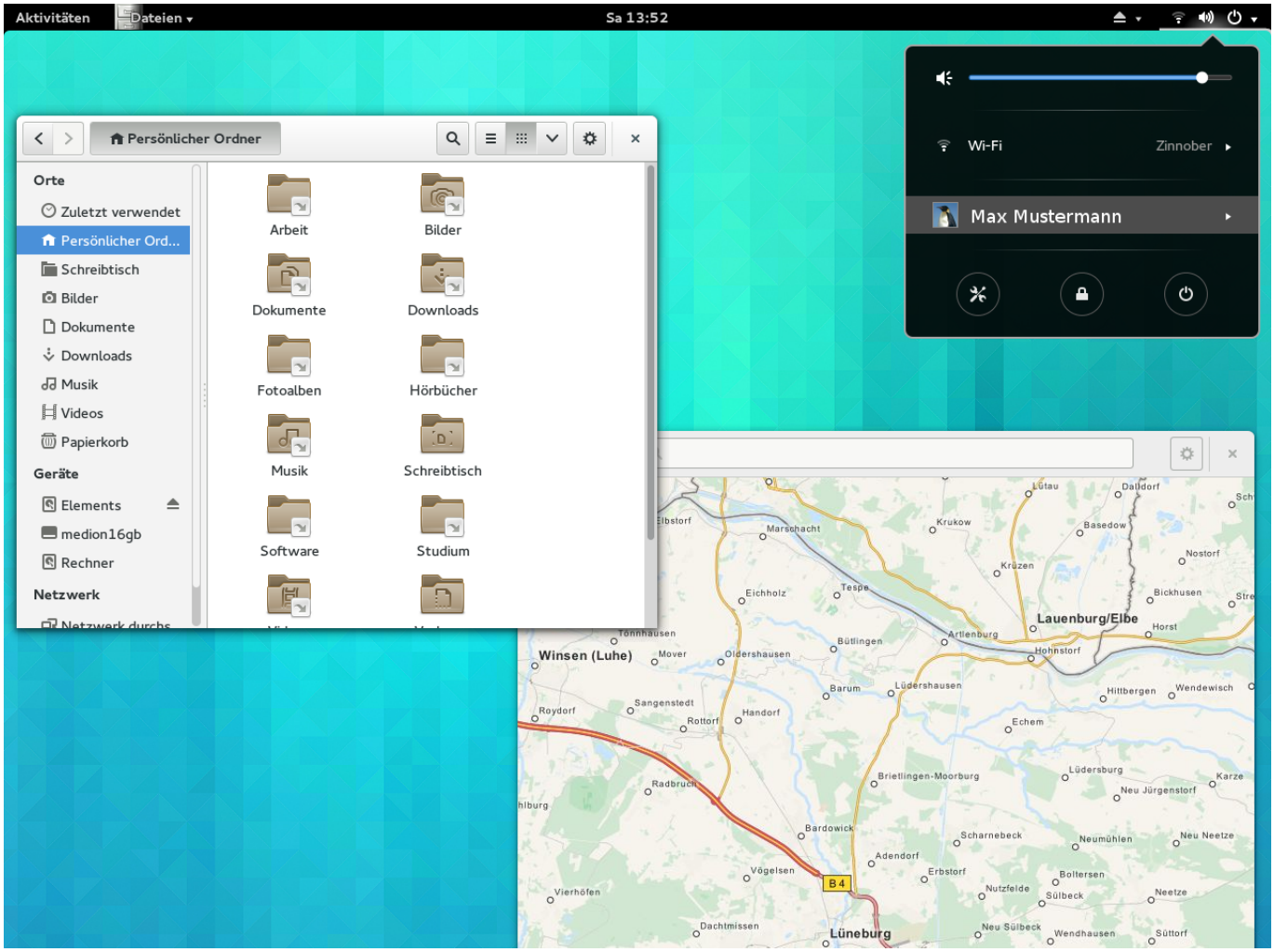


Unity

Unity ist eine durch das Unternehmen Canonical entwickelte Desktop-Umgebung für Linux-Betriebssysteme, die besonders sparsam mit Bildschirmplatz umgehen soll.

XFCE

Während erstgenannte Desktops mit Features protzen gehen die EntwicklerInnen bei XFCE einen anderen Weg. XFCE möchte einen schlanken, performanten Desktop bieten, der auf „unnötige Spielereien“ verzichtet und auch auf leistungsschwachen und zum Teil für andere Aufgaben (Multimedia) bestimmten Systemen resourcenschonend läuft. Viele heutige Distributionen bieten XFCE als vorkonfigurierte Alternative zum Standarddesktop an.



MATE und Cinnamon

Der umstrittene Upgrade von Gnome2 auf Gnome3 führte zur Geburt von MATE. Dieser Desktop ist eine direkte Fortentwicklung von Gnome2. Cinnamon dagegen ist aus den Quellen von Gnome3 entstanden, aber deutlich performanter als dieser. Was beide gemeinsam haben - sie sind die Standarddesktops von „Linux Mint“ und nur als Ubuntu-Editionen von Linux Mint gibt es auch KDE und XFCE vorkonfiguriert.

Es können auch mehrere Desktops gleichzeitig installiert werden. Bei der Anmeldung kann dann zwischen den Desktops gewechselt werden. So kann jeder seinen Lieblingsdesktop herausfinden.

Arbeiten auf der Konsole

Gnome Terminal findet man unter *Weiter Anwendungen* .

prompt *user@linux-aosk:~→* hat folgende Bedeutung:

- *user* : Benutzername
- *linux-aosk* : Rechnername
- ~ : Homeverzeichnis

Kommandos zur Bearbeitung von Dateien

Obwohl unter KDE und Gnome moderne Dateimanager zur Verfügung stehen, verwenden erfahrene Linux-Anwender oft noch immer diverse, text-orientierte Kommandos.

Kommando	**Beschreibung**	**Kommando in DOS**
Hilfe		
man *Befehl*	Hilfe zum Kommando	
Befehl - - help	Hilfe zum Kommando	
Als Root		
su	wechselt als Root (Passwort eingeben)	
sudo	einen Befehl als Root ausführen	
Verzeichnisbaum		
cd	wechselt das aktuelle Verzeichnis	
cd /	wechselt ins root-Verzeichnis	

Kommando	Beschreibung	Kommando in DOS
Hilfe		
ls	zeigt alle Dateien des aktuellen Verzeichnisse an	dir
ls -l	zeigt eine detaillierte Liste	
ls -a	zeigt versteckte Dateien an	
mkdir	erzeugt ein neues Verzeichnis	md
rmdir	löscht Verzeichnisse	rd
pwd	zeigt aktuellen Pfad an	
Joker		
*	steht für eine beliebige Anzahl von beliebigen Zeichen	
?	steht für ein beliebiges Zeichen	
Dateien		
mv quelle ziel	verschiebt Dateien bzw. ändert ihren Namen	move
cp quelle ziel	kopiert Dateien	copy
cp ordner ziel -r	kopiert gesamten ordner inkl. aller unterordner an ziel	
cat	zeigt Dateiinhalt an	type
less	öffnet Anzeigeprogramm	
more	zeigt Dateiinhalt seitenweise an	
touch Dateiname	erstellt leere Datei	
mcedit Dateiname	öffnet Datei in einem Editor zur Bearbeitung	edit
vim Dateiname	öffnet Datei mit dem Editor VIM zur Bearbeitung	
vimtutor	Tutorial zum Erlernen vom Editor VIM	
rm	löscht Dateien	del
rm unterordner -r	löscht gesamten Unterordner inkl. aller Dateien	
find -name dateinamen	sucht Dateien nach Namen	
Packen und Komprimieren von Verzeichnissen und Dateien		
tar	vereint mehrere Dateien (und Verzeichnisse) in einer Datei	
tar -t	Inhalt eines Archivs anzeigen	
tar -x	Dateien aus Archiv holen	
tar -c	neues Archiv erzugen	
tar -f	um Namen des Archiv anzugeben	
tar -xvjf	entzippen	
Dateirechte chown und chmod		
chown student meinedatei.txt	Ändert den Besitzer der Datei „meinedatei.txt“ auf den User „student“	
chown student:www-data meinedatei.txt	Ändert den Besitzer der Datei „meinedatei.txt“ auf den User „student“ und die Gruppe der Datei auf „www-data“	
chown -R www-data:www-data /var/www	Ändert den Ordner, Inhalt und alle Unterordner von „/var/www“ auf den Besitzer und Gruppe“www-data“	
chmod 777 meinedatei.txt	Ändert die Rechte der Datei auf Lesen, Schreiben und Ausführen für Besitzer, Gruppe und Andere im im Hexadezimalmodus	

Kommando	Beschreibung	Kommando in DOS
Hilfe		
chmod a+rwx meinedatei.txt	Ändert die Rechte der Datei auf Lesen, Schreiben und Ausführen für Besitzer, Gruppe und Andere im symbolischen Modus (a...all)	
chmod -R 700 /var/www/html/	Setz die Dateirechte rekursiv auf 700 im Ordner /var/www/html, also auf alle Dateien und Ordner die sich in /var/www/html befinden.	
chmod u=rw,g=rw,o=r meinedatei.txt	Setzt explizit die rechte für Besitzer und Gruppe auf lesen und schreiben und andere dürfen nur lesen (u...user, g...group, a...others	

Weitere Befehle

- http://www.admintalk.de/konsolenbefehle.php
- http://www.shellbefehle.de/befehle/

Übung 1

1. Erstelle in deinem Home-Directory einen Ordner uebungen.
2. Speichere das File uebung1.tar in diesen Ordner!
3. Entpacke das Archiv mit Hilfe von `tar -xvjf uebung1.tar`. Welche Verzeicnisse und Dateien befinden sich nun in deinem Home-Directory?
4. Gib den Befehl `./hallo` ein.
5. Finde die Datei `ichbinhier`.
6. Wechsle in das Verzeichnis, in dem sich die Datei befindet.
7. Erstelle ein Verzeichnis mit dem Namen `backup` in deinem Homedirectory.
8. Kopiere das gesamte Verzeichnis `uebung1` in das Verzeichnis `backup`.
9. Lösche das Verzeichnis `uebung1`.
10. Erstelle ein Verzeichnis mit dem Namen `aufgabe` und wechsle hinein.
11. Erstelle drei leere Dateien `datei1` bis `datei3`.
12. Öffne mit einem Editor `datei1` und gib drei Zeilen Text ein. Speicher ab!
13. Lasse dir die Datei mit einem entsprechendem Kommando ausgeben!
14. Gib den Befehl `tac datei1` ein. Was passiert?
15. Wechsle in die grafische Oberfläche!
16. Orientiere dich an der Oberfläche!
17. Versuche den Bildschirmhintergrund umzustellen.
18. Öffne ein Konsolenfenster. Lösche darin den gesamten Ordner uebungen inklusive Unterverzeichnis.

Übung 2

Grundkurs 1-3 & 7, Für Experten 4-6

Shell Scripts sh (Bourne Shell)

Shell-Scripts (Kommandoprozeduren) sind unter Unix das Analogon zu Batch-Dateien (Stapeldateien) von MS-DOS, sie sind jedoch wesentlich leistungsfähiger. Ihre Syntax hängt allerdings von der verwendeten „Shell“ ab. In diesem Artikel werden Bourne Shell (sh) Scripts betrachtet. Sie gelten im allgemeinen als zuverlässiger als csh-Scripts.

Ein Shell-Script ist eine Textdatei, in der Kommandos gespeichert sind. Es stellt selbst ein Kommando dar und kann wie ein Systemkommando auch mit Parametern aufgerufen werden. Shell-Scripts dienen der Arbeitserleichterung und (nach ausreichenden Tests) der Erhöhung der Zuverlässigkeit, da sie gestatten, häufig gebrauchte Sequenzen von Kommandos zusammenzufassen.

Aufruf

Bourne Shell Scripts lassen sich generell wie folgt aufrufen:

```
sh myscript
```

Im allgemeinen ist es aber praktischer, die Datei als ausführbar anzumelden (execute permission), mittels

```
chmod +x myscript
```

Das Shell-Script läßt sich dann wie ein „normales“ (binäres) Kommando aufrufen:

```
myscript
```

Vorausgesetzt ist hierbei allerdings, daß das Betriebssystem das Script mit der richtigen Shell abarbeitet. Moderne Unix-Systeme prüfen zu diesem Zweck die erste Zeile. Für die sh sollte sie folgenden Inhalt haben:

```
#!/bin/sh
```

Obwohl das Doppelkreuz normalerweise einen Kommentar einleitet, der vom System nicht weiter beachtet wird, erkennt es hier, daß die „sh“ (mit absoluter Pfadangabe: /bin/sh) eingesetzt werden soll.

Ein Beispiel

```
#!/bin/sh
# Einfaches Beispiel
echo Hallo, Welt!
echo Datum, Uhrzeit und Arbeitsverzeichnis:
date
pwd
```

```
echo Uebergabe-Parameter: $*
```

Das vorstehende einfache Script enthält im wesentlichen normale Unix-Kommandos. Abgesehen von der ersten Zeile liegt die einzige Besonderheit im Platzhalter „$*“, der für alle Kommandozeilen-Parameter steht.

Testen eines Shell-Scripts

```
sh -n myscript
```

Syntax-Test (die Kommandos werden gelesen und geprüft, aber nicht ausgeführt)

```
sh -v myscript
```

Ausgabe der Shell-Kommandos in der gelesenen Form

```
sh -x myscript
```

Ausgabe der Shell-Kommandos nach Durchführung aller Ersetzungen, also in der Form, wie sie ausgeführt werden

Kommandozeilen-Parameter

```
$0
```

Name der Kommandoprozedur, die gerade ausgeführt wird

```
$#
```

Anzahl der Parameter

```
$1
```

erster Parameter

```
$2
```

zweiter Parameter

```
$3
```

dritter ... Parameter

```
$*
```

steht für alle Kommandozeilen-Parameter ($1 $2 $3 ...)

```
$@
```

wie $* ($1 $2 $3 ...)

```
$$
```

Prozeßnummer der Shell (nützlich, um eindeutige Namen für temporäre Dateien zu vergeben)

```
$-
```

steht für die aktuellen Shell-Optionen

```
$?
```

gibt den Return-Code des zuletzt ausgeführten Kommandos an (0 bei erfolgreicher Ausführung)

```
$!
```

Prozessnummer des zuletzt ausgeführten Hintergrund-Prozesses

Beispiel

```
#!/bin/sh
# Variablen
echo Uebergabeparameter: $*
echo user ist: $USER
echo shell ist: $SHELL
echo Parameter 1 ist: $1
echo Prozedurname ist: $0
echo Prozessnummer ist: $$
echo Anzahl der Parameter ist: $#
a=17.89        # ohne Luecken am = Zeichen
echo a ist $a
```

Prozesssteuerung

Bedingte Ausführung: if

```
if [ bedingung ]
  then kommandos1
  else kommandos2
fi
```

Anmerkungen: „fi“ ist ein rückwärts geschriebenes „if“, es bedeutet „end if“ (diese Schreibweise ist eine besondere Eigenheit der Bourne Shell). Die „bedingung“ entspricht der Syntax von test, siehe auch weiter unten. Im if-Konstrukt kann der „else“-Zweig entfallen, andererseits ist eine Erweiterung

durch einen oder mehrere „else if“-Zweige möglich, die hier „elif“ heißen:

```
if [ bedingung1 ]
    then kommandos1
  elif [ bedingung2 ]
    then kommandos2
  else kommandos3
fi
```

Die Formulierung

```
if [ bedingung ]
```

ist äquivalent zu

```
if test bedingung
```

Alternativ ist es möglich, den Erfolg eines Kommandos zu prüfen:

```
if kommando
```

(beispielsweise liefert das Kommando „true“ stets „wahr“, das Kommando „false“ hingegen „unwahr“)

Wichtige Vergleichsoperationen (test)

Hinweis: Es ist unbedingt notwendig, daß alle Operatoren von Leerzeichen umgeben sind, sonst werden sie von der Shell nicht erkannt! (Das gilt auch für die Klammern.)

Zeichenketten

```
"s1" = "s2"
```

wahr, wenn die Zeichenketten gleich sind

```
"s1" != "s2"
```

wahr, wenn die Zeichenketten ungleich sind

```
-z "s1"
```

wahr, wenn die Zeichenkette leer ist (Länge gleich Null)

```
-n "s1"
```

wahr, wenn die Zeichenkette nicht leer ist (Länge größer als Null)

(Ganze) Zahlen

```
n1 -eq n2
```

wahr, wenn die Zahlen gleich sind

```
n1 -ne n2
```

wahr, wenn die Zahlen ungleich sind

```
n1 -gt n2
```

wahr, wenn die Zahl n1 größer ist als n2

```
n1 -ge n2
```

wahr, wenn die Zahl n1 größer oder gleich n2 ist

```
n1 -lt n2
```

wahr, wenn die Zahl n1 kleiner ist als n2

```
n1 -le n2
```

wahr, wenn die Zahl n1 kleiner oder gleich n2 ist

Sonstiges

```
!
```

Negation

```
-a
```

logisches „und“

```
-o
```

logisches „oder“ (nichtexklusiv; -a hat eine höhere Priorität)

```
\( ... \)
```

Runde Klammern dienen zur Gruppierung. Man beachte, daß sie durch einen vorangestellten Backslash, \, geschützt werden müssen.

```
-f filename
```

wahr, wenn die Datei existiert. (Weitere Optionen findet man in der man page zu test)

Beispiel:

```
#!/bin/sh
# Interaktive Eingabe, if-Abfrage
echo Hallo, user, alles in Ordnung?
echo Ihre Antwort, n/j:
read answer
echo Ihre Antwort war: $answer
# if [ "$answer" = "j" ]
if [ "$answer" != "n" ]
  then echo ja
  else echo nein
fi
```

Mehrfachentscheidung: case

```
case var in
  muster1) kommandos1 ;;
  muster2) kommandos2 ;;
  *) default-kommandos ;;
esac
```

Anmerkungen: „esac“ ist ein rückwärts geschriebenes „case“, es bedeutet „end case“. Die einzelnen Fälle werden durch die Angabe eines Musters festgelegt, „muster)“, und durch ein doppeltes Semikolon abgeschlossen. (Ein einfaches Semikolon dient als Trennzeichen für Kommandos, die auf derselben Zeile stehen.) Das Muster „*)“ wirkt als „default“, es deckt alle verbleibenden Fälle ab; die Verwendung des default-Zweiges ist optional.

Beispiel:

```
#!/bin/sh
# Interaktive Eingabe, Mehrfachentscheidung (case)
echo Alles in Ordnung?
echo Ihre Antwort:
read answer
echo Ihre Antwort war: $answer
case $answer in
  j*|J*|y*|Y*) echo jawohl ;;
  n*|N*) echo nein, ueberhaupt nicht! ;;
  *) echo das war wohl nichts ;;
esac
```

Schleife: for

```
for i in par1 par2 par3 ...
  do kommandos
done
```

Anmerkungen: Die for-Schleife in der Bourne Shell unterscheidet sich von der for-Schleife in üblichen Programmiersprachen dadurch, daß nicht automatisch eine Laufzahl erzeugt wird. Der Schleifenvariablen werden sukzessive die Parameter zugewiesen, die hinter „in“ stehen. (Die Angabe „for i in $*“ kann durch „for i“ abgekürzt werden.)

Beispiel:

```
#!/bin/sh
# Schleifen: for
echo Uebergabeparameter: $*
# for i
for i in $*
  do echo Hier steht: $i
done
```

Schleife: while und until

```
while [ bedingung ]
  do kommandos
done
until [ bedingung ]
  do kommandos
done
```

Anmerkung: Bei „while“ erfolgt die Prüfung der Bedingung vor der Abarbeitung der Schleife, bei „until“ erst danach. (Anstelle von „[bedingung]“ oder „test bedingung“ kann allgemein ein Kommando stehen, dessen Return-Code geprüft wird; vgl. die Ausführungen zu „if“.)

Beispiel:

```
#!/bin/sh
# Schleifen: while
# mit Erzeugung einer Laufzahl
i=1
while [ $i -le 5 ]
do
  echo $i
  i=`expr $i + 1`
done
```

C++

DATEIBEHANDLUNG

Programme speichern ihre Informationen in Variablen. Leider bleiben diese immer nur bis zum nächsten Stromausfall oder Betriebssystemabsturz erhalten. Und wenn das Programm verlassen wird, ist ihr Inhalt ebenfalls Geschichte. Damit Sie auf Ihre Daten auch morgen noch kraftvoll zugreifen können, empfiehlt es sich, diese in einer Datei abzulegen. Dazu können Sie die Daten als Ausgabestrom in die Datei schreiben. Das Vorgehen entspricht dem bei der Bildschirmausgabe per cout. Diese Form wird sequenziell genannt, weil die Daten nacheinander in der Reihenfolge, wie sie geschrieben wurden, in der Datei landen. Sie können aber auch einen Datenblock an eine beliebige Stelle der Datei schreiben. Später können Sie diesen Datenblock wieder zurückholen, indem Sie den internen Dateizeiger an diese Stelle positionieren und den Datenblock wieder lesen. Diese Vorgehensweise ist typisch für Klassen, insbesondere, wenn sie in irgendeiner Form sortiert abgelegt werden sollen.

fstream

Über die Headerdatei fstream wird die Funktionalität zum Lesen und Schreiben von Dateien zur Verfügung gestellt. Soll nur geschrieben werden, dann kann auch die Headerdatei ofstream genutzt werden. Ebenso kann, wenn nur gelesen werden soll, als Header ifstream zum Einsatz kommen. Der übliche Ablauf zum Lesen oder Schreiben von Dateien ist folgender:

- Objekt zum Lesen oder Schreiben im Programm anlegen
- Objekt mit dem Dateinamen zum öffnen verknüpfen
- Text über das Objekt schreiben oder lesen
- Datei schliessen

Schreiben einer Datei

In Zeile 5 wird ein Objekt angelegt, dass ähnlich wie cout oder cin die Ausgabe in die Datei übernimmt. Der Name des Objekts kann beliebig vergeben werden, so lange dieser noch nicht verwendet wird. Die Datei beispiel.txt wird in Zeile 6 geöffnet und im aktuellen Verzeichnis angelegt, falls diese noch nicht existiert. Nun da die Datei geöffnet wurde, kann in Zeile 7 etwas in die Datei geschrieben werden. Zum Schluss muss noch die Datei geschlossen werden, damit alle gepufferten Schreibvorgänge abgeschlossen werden (Zeile 8).

```
1  #include <fstream>
2  using namespace std;
3
4  int main () {
5      ofstream fileout;
6      fileout.open("beispiel.txt");
7      fileout << "Das steht jetzt in der Datei.";
8      fileout.close();
9      return 0;
10 }
```

Lesen einer Datei

Nun wollen wir den gerade geschriebenen Inhalt der Datei wieder auslesen und auf dem Bildschirm anzeigen.

```
1  #include <iostream>
2  #include <fstream>
3  #include <string>
4  using namespace std ;
5
6  int main() {
7     string dateizeile;
8     ifstream fin;
9     fin.open("beispiel.txt");
10    getline(fin,dateizeile);
11    fin.close () ;
12    cout << "Zeile in der Datei: " << dateizeile << endl ;
13    return 0;
14 }
```

Wir benötigen einen String dateizeile, in dem wir die gelesene Zeile aus der Datei aufbewahren können. Eine komplette Zeile lässt sich mittels der Funktion getline einlesen. Es wird das mit der Datei verknüpfte Objekt und der String, in den diese Zeile gespeichert werden, übergeben (Zeile 10). Nach dem Schliessen der Datei wird der gelesene String in Zeile 12 ausgegeben.

Daten aus einer strukturierten Datei einlesen

Oftmals speichert man Daten in kommagetrennten Dateien (.CSV-Dateien) strukturiert ab. Um diese abgespeicherten Werte in einem C++-Programm einzulesen und in temporäre Variablen, Arrays oder Datenstrukturen zu speichern ist es notwendig diese Zeile für Zeile einzulesen und anschließend zu parsen (zergliedern). Dafür zerschneidet man die Zeile wie folgt:

- Man schneidet immer von der ganzen Zeile die Zeichenkette von Beginn bis zum ersten Komma aus und speichert diesen Teilstring in eine Variable ab. Anschließend speichert man den restlichen Teilstring neu ab, so dass der zweite kommagetrennte Wert nun am Anfang steht.

```
#include <iostream>
#include <fstream>

using namespace std;

int main(int argc, char** argv) {

        //Einlesen der CSV-Datei
        ifstream filein;
        filein.open("rangliste.csv");

    //Zeilen auslesen
    string zeile;
```

```
    while(getline(filein,zeile)){
        string wert = zeile.substr(0,zeile.find(';'));
        zeile=zeile.substr(zeile.find(';')+1,zeile.length());
        }

        filein.close();

        return 0;

}
```

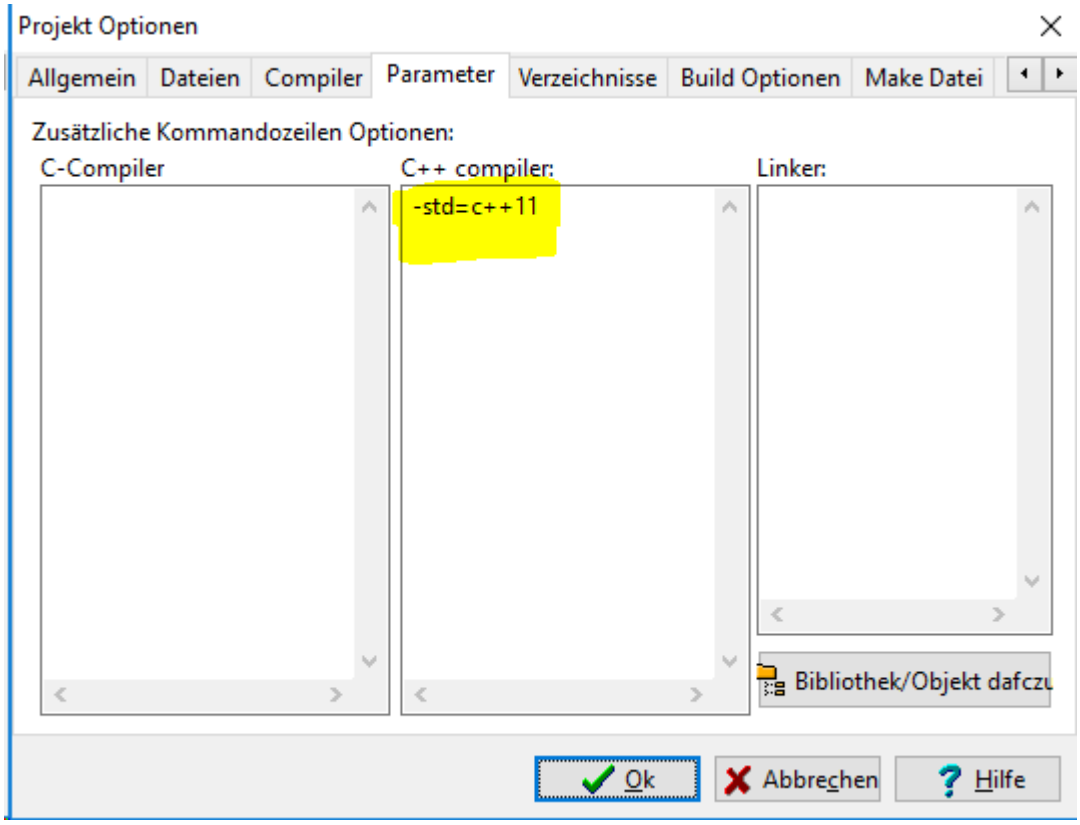
Oftmals ist es auch notwendig die eingelesene Zeichenkette zu konvertieren. Hier gibt es verschiedene Konvertierungsfunktionen:

- int stoi(string str) - String to Int, retourniert einen Integer-Value und benötigt einen String als Parameter
- float stof (string str); - String to Float, retourniert einen Float-Value und benötigt einen String als Parameter
- ...

```
#include <iostream>
#include <string>

using namespace std;

int main(int argc, char** argv) {

    string str="10";

    int zahl=0;

    zahl=stoi(str);

    cout << zahl;


    return 0;
}
```

Funktioniert dies im Programm DEV-C++ nicht, so ist der Compiler veraltet und man muss dem Compiler einen zusätzlichen Parameter (-std=c++11) mit angeben.

Dies funktioniert im Menü unter Projekt → Projekt Optionen (ALT+P).



Zeiger (Pointer)

Zeiger (engl. pointers) sind Variablen, die als Wert die Speicheradresse einer anderen Variable enthalten.

Jede Variable wird in CPP an einer bestimmten Position im Hauptspeicher abgelegt. Diese Position nennt man Speicheradresse (engl. memory address). CPP bietet die Möglichkeit, die Adresse jeder Variable zu ermitteln. Solange eine Variable gültig ist, bleibt sie an ein und derselben Stelle im Speicher.

Am einfachsten vergegenwärtigt man sich dieses Konzept anhand der globalen Variablen. Diese werden außerhalb aller Funktionen und Klassen deklariert und sind überall gültig. Auf sie kann man von jeder Klasse und jeder Funktion aus zugreifen. Über globale Variablen ist bereits zur Kompilierzeit bekannt, wo sie sich innerhalb des Speichers befinden (also kennt das Programm ihre Adresse).

Zeiger sind nichts anderes als normale Variablen. Sie werden deklariert (und definiert), besitzen einen Gültigkeitsbereich, eine Adresse und einen Wert. Dieser Wert, der Inhalt der Zeigervariable, ist aber nicht wie in unseren bisherigen Beispielen eine Zahl, sondern die Adresse einer anderen Variable oder eines Speicherbereichs. Bei der Deklaration einer Zeigervariable wird der Typ der Variable festgelegt, auf den sie verweisen soll.

```
#include <iostream>

int main() {
     int   Wert;      // eine int-Variable
     int *pWert;      // eine Zeigervariable, zeigt auf einen int
     int *pZahl;      // ein weiterer "Zeiger auf int"

     Wert = 10;       // Zuweisung eines Wertes an eine int-Variable

     pWert = &Wert;   // Adressoperator '&' liefert die Adresse einer
Variable
     pZahl = pWert;   // pZahl und pWert zeigen jetzt auf dieselbe Variable
```

Beispielhafte Speicherbelegung des Programms im Hauptspeicher:

Datentyp	Variable	Adresse	Wert
int	Wert	0x0001	10
int *	pWert	0x0005	0x0001
int *	pZahl	0x0009	0x0001
...	...	.	.
...	...	.	.

Der Adressoperator & kann auf jede Variable angewandt werden und liefert deren Adresse, die man einer (dem Variablentyp entsprechenden) Zeigervariablen zuweisen kann. Wie im Beispiel gezeigt, können Zeiger gleichen Typs einander zugewiesen werden. Zeiger verschiedenen Typs bedürfen einer Typumwandlung. Die Zeigervariablen pWert und pZahl sind an verschiedenen Stellen im Speicher abgelegt, nur die Inhalte sind gleich.

Wollen Sie auf den Wert zugreifen, der sich hinter der im Zeiger gespeicherten Adresse verbirgt, so verwenden Sie den Dereferenzierungsoperator *.

```
  *pWert += 5;
  *pZahl += 8;

   std::cout << "Wert = " << Wert << std::endl;
```

Beispielhafte Speicherbelegung des Programms im Hauptspeicher:

Datentyp	Variable	Adresse	Wert
int	Wert	0x0001	10 –> 15 –> 23
int *	pWert	0x0005	0x0001
int *	pZahl	0x0009	0x0001
...	...	.	.
...	...	.	.

Ausgabe:

```
Wert = 23
```

Man nennt das den Zeiger dereferenzieren. Im Beispiel erhalten Sie die Ausgabe Wert = 23, denn pWert und pZahl verweisen ja beide auf die Variable Wert.

Um es noch einmal hervorzuheben: Zeiger auf Integer (int) sind selbst keine Integer. Den Versuch, einer Zeigervariablen eine Zahl zuzuweisen, beantwortet der Compiler mit einer Fehlermeldung oder mindestens einer Warnung. Hier gibt es nur eine Ausnahme: die Zahl 0 darf jedem beliebigen Zeiger zugewiesen werden. Ein solcher Nullzeiger zeigt nirgendwohin. Der Versuch, ihn zu dereferenzieren, führt zu einem Laufzeitfehler.

Bespiel Zeigerübung

```
int main()
{
        int zahl=10;
        int *z=NULL;
        int **zz=NULL;


        cout<< zahl<<endl;            //Wert wird ausgegeben
        cout<<&zahl<<endl;            //adresse wird ausgegeben
        cout<<&zahl+1<<endl;        //adresse cpn zahl+1
        cout<< zahl +1<<endl;        //11


        z=&zahl;
```

```
        cout<< *z<< endl;         //10
        cout<< z<< endl;         //Adresse von zahl= Wert von z
        cout<< &z<< endl;         //Adresse von Zeiger z

        zz=&z;                     //Adresse von Zeiger z wird in Zeiger zz

        cout<< *zz<<endl;         //Wert von z= Adresse von zahl    //&zahl
//z
        cout<< **zz<<endl;         //Wert von zahl                    //zahl
        cout<< zz<<endl;         //Adresse von z =Wert von zz    //&z
//zz
        cout<< &zz<<endl;         //Adresse von Zeiger zz            //&zz

  getch();
  return 0;
}
```

Verschiedene Konventionen bei der Definition von Zeigern

Bei der Definition einer Zeigervariablen muss das Zeichen * nicht unmittelbar auf den Datentyp folgen. Die folgenden vier Definitionen sind gleichwertig:

```
int* i;  // Whitespace (z.B. ein Leerzeichen) nach *
int *i;  // Whitespace vor *
int * i; // Whitespace vor * und nach *
int*i;   // Kein whitespace vor * und nach *
```

Versuchen wir nun, diese vier Definitionen nach demselben Schema wie eine Definition von „gewöhnlichen“ Variablen zu interpretieren. Bei einer solchen Definition bedeutet

```
T v;
```

dass eine Variable v definiert wird, die den Datentyp T hat. Für die vier gleichwertigen Definitionen ergeben sich verschiedene Interpretationen, die als Kommentar angegeben sind:

```
int* i;   // Definition der Variablen i des Datentyps int*
int *i;   // Irreführend: Es wird kein "*i" definiert,
          // obwohl die dereferenzierte Variable *i heißt.
int * i;  // Datentyp int oder int* oder was?
int*i;    // Datentyp int oder int* oder was?
```

Offensichtlich passen nur die ersten beiden in dieses Schema. Die erste führt dabei zu einer richtigen und die zweite zu einer falschen Interpretation.

Allerdings passt die erste Schreibweise nur bei der Definitionen einer einzelnen Zeigervariablen in dieses Schema, da sich der * bei einer Definition nur auf die Variable unmittelbar rechts vom *

bezieht:

```
int* i,j,k; // definiert int* i, int j, int k
            // und nicht: int* j, int* k
```

Zur Vermeidung solcher Missverständnisse sind zwei Schreibweisen verbreitet:

- C-Programmierer verwenden oft die zweite Schreibweise von oben, obwohl sie nicht in das Schema der Definition von „gewöhnlichen“ Variablen passt. Diese Schreibweise wird auch in Turbo CPP verwendet.

```
int *i,*j,*k; // definiert int* i, int* j, int* k
```

- Bjarne Stroustrup (einer der C bzw. CPP-Entwickler) empfiehlt, auf Mehrfachdefinitionen zu verzichten. Er schreibt den * wie in „int* pi;“ immer unmittelbar nach dem Datentyp.

Dynamisch erzeugte Variablen: new und delete

Mit dem Operator **new** kann man Variablen während der Laufzeit des Programms erzeugen. Dieser Operator versucht, in einem eigens dafür vorgesehenen Speicherbereich (der oft auch als **Heap, dynamischer Speicher** oder **freier Speicher** bezeichnet wird) so viele Bytes zu reservieren, wie eine Variable des angegebenen Datentyps benötigt.

Falls der angeforderte Speicher zur Verfügung gestellt werden konnte, liefert new seine Adresse zurück. Andernfalls wird eine **Exception** des Typs `std::bad_alloc` ausgelöst, die einen Programmabbruch zur Folge hat, wenn sie nicht mit einer `try`- Anweisung abgefangen wird. Da unter 32-bit-Systemen wie Windows unabhängig vom physisch vorhandenen Hauptspeicher 2 GB virtueller Speicher zur Verfügung stehen, kann man meist davon ausgehen, dass der angeforderte Speicher verfügbar ist.

Variablen, die zur Laufzeit erzeugt werden, bezeichnet man auch als **dynamisch erzeugte Variablen**. Wenn der Unterschied zu vom Compiler erzeugten Variablen (wie „`int i;`“) betont werden soll, werden diese als „gewöhnliche“ Variablen bezeichnet. Die folgenden Beispiele zeigen, wie man new verwenden kann:

1. Für einen Datentyp T reserviert „new T“ so viele Bytes auf dem Heap, wie für eine Variable des Datentyps T notwendig sind. Der Ausdruck „new T“ hat den Datentyp „Zeiger auf T“. Sein Wert ist die Adresse des reservierten Speicherbereichs und kann einer Variablen des Typs „Zeiger auf T“ zugewiesen werden:

```
int* pi;
pi=new int; //reserviert sizeof(int) Bytes und weist pi die Adresse dieses
Speicherbereichs zu
*pi=17;     // initialisiert den reservierten Speicherbereich
```

Am besten initialisiert man eine Zeigervariable immer gleich bei ihrer Definition:

```
int* pi=new int; // initialisiert pi mit der Adresse
*pi=17;
```

Die explizit geklammerte Version von *new* kann zur Vermeidung von Mehrdeutigkeiten verwendet werden. Für einfache Datentypen sind beide Versionen gleichwertig:

```
pi=new(int); // gleichwertig zu pi=new int;
```

2. In einem *new-expression* kann man nach dem Datentyp einen *new-initializer* angeben: Er bewirkt die Initialisierung der mit new erzeugten Variablen. Die zulässigen Ausdrücke und ihre Bedeutung hängen vom Datentyp der dynamisch erzeugten Variablen ab.

Für einen fundamentalen Datentyp (wie *int, double* usw.) gibt es drei Formen:

a) Der in Klammern angegebene Wert wird zur Initialisierung verwendet:

```
double* pd=new double(1.5); //Initialisierung *pd=1.5
```

b) Gibt man keinen Wert zwischen den Klammen an, wird die Variable mit 0 (NULL) initialisiert:

```
double* pd=new double(); //Initialisierung *pd=0
```

c) Ohne einen Initialisierer ist ihr Wert unbestimmt:

```
double* pd=new double;// *pd wird nicht initialisiert
```

Für einen Klassentyp müssen die Ausdrücke Argumente für einen Konstruktor sein.

3. Gibt man nach einem Datentyp T in eckigen Klammern einen ganzzahligen Ausdruck >= 0 an, wird ein **Array dynamisch erzeugt** reserviert.

Die Anzahl der Arrayelemente ist durch den ganzzahligen Ausdruck gegeben. Im Unterschied zu einem gewöhnlichen Array muss diese Zahl keine Konstante sein:

```
typedef double T; // T irgendein Datentyp, hier double
int n=100; // nicht notwendig eine Konstante
T* p=new T[n]; // reserviert n*sizeof(T) Bytes
```

Wenn durch einen *new*-Ausdruck ein Array dynamisch erzeugt wird, ist der Wert des Ausdrucks die Adresse des ersten Arrayelements. Die einzelnen Elemente des Arrays können folgendermaßen angesprochen werden:

```
p[0], ..., p[n–1] // n Elemente des Datentyps T
```

Die Elemente eines dynamisch erzeugten Arrays können nicht bei ihrer Definition initialisiert werden.

4. Mit dem nur selten eingesetzten *new-placement* kann man eine Variable an eine bestimmte Adresse platzieren. Damit wird keine Variable wie bei einem gewöhnlichen *new*-Ausdruck auf dem Heap angelegt. Die Adresse wird als Zeiger nach new angegeben und ist z.B. die Adresse eines zuvor reservierten Speicherbereichs:

```
int* pi=new int;
double* pd=new(pi) double;// erfordert #include<new.h>
```

Durch die letzte Anweisung kann man den Speicherbereich ab der Adresse von `i` als *double* ansprechen wie nach

```
double* pd=(double*)pi; // explizite Typkonversion
```

Dieses Beispiel zeigt, dass die meisten Programme keine Anwendungen für ein *new placement* haben. Nützlichere Anwendungen gibt es bei Betriebssystemen, die (anders als Windows) Geräte über physikalische Adressen ansprechen.

5. Variable, deren Datentyp eine Klasse der VCL ist, müssen mit new angelegt werden. Dabei muss dem Konstruktor der Eigentümer (z.B. *Form1*) übergeben werden.

```
TEdit* pe = new TEdit(Form1);
pe->Parent = Form1;
pe ->SetBounds(10, 20, 100, 30);
pe ->Text = "blablabla";
```

Es ist nicht möglich, VCL-Komponenten vom Compiler erzeugen zu lassen:

```
TEdit e; bzw. TEdit e(Form1); // Fehler
```

Eine gewöhnliche Variable existiert von ihrer Definition bis zum Ende des Bereichs (bei einer lokalen Variablen ist das der Block), in dem sie definiert wurde. Im Unterschied dazu existiert eine dynamisch erzeugte Variable bis der für sie reservierte Speicher mit dem Operator delete wieder freigegeben oder das Programm beendet wird. Man sagt auch, dass eine dynamisch erzeugte Variable durch den Aufruf von *delete* **zerstört** wird.

Die erste dieser beiden Alternativen ist für Variable, die keine Arrays sind, und die zweite für Arrays. Dabei muss *cast-expression* ein Zeiger sein, dessen Wert das Ergebnis eines *new*-Ausdrucks ist. Nach *delete p* ist der Wert von *p* unbestimmt und der Zugriff auf **p* unzulässig. Falls *p* den Wert 0 hat, ist *delete p* wirkungslos.

Damit man mit dem verfügbaren Speicher sparsam umgeht, sollte man den Operator delete immer dann aufrufen, wenn eine mit new erzeugte Variable nicht mehr benötigt wird. Unnötig reservierter Speicher wird auch als **Speicherleck (memory leak)** bezeichnet. Ganz generell sollte man diese **Regel** beachten: Jede mit ***new*** erzeugte Variable sollte mit ***delete*** auch wieder freigegeben werden.

Die folgenden Beispiele zeigen, wie die in den letzten Beispielen reservierten Speicherbereiche wieder freigegeben werden.

1. Der Speicher für die unter 1. und 2. erzeugten Variablen wird folgendermaßen wieder freigegeben:

```
delete pi;
delete pd;
```

2. Der Speicher für das unter 3. erzeugte Array wird freigegeben durch

```
delete[] p;
```

3. Es ist möglich, die falsche Form von *delete* zu verwenden, ohne dass der Compiler eine Warnung oder Fehlermeldung ausgibt:

```
delete[] pi; // Arrayform für Nicht-Array
delete p; // Nicht-Arrayform für Array
```

Im C++-Standard ist explizit festgelegt, dass das Verhalten nach einem solchen falschen Aufruf undefiniert ist. Oben wurde empfohlen, einen Zeiger, der nicht auf reservierten Speicher zeigt, immer auf 0 (Null) zu setzen. Deshalb sollte man einen Zeiger nach *delete* immer auf 0 setzen, z.B. nach Beispiel 1:

```
pi=0;
pd=0;
```

Stroustrup empfiehlt dafür eine Funktion wie ***destroy***:

```
void destroy(int*& p)
{//http://www.research.att.com/~bs/bs_faq2.html
delete p; // delete[] für Zeiger auf dynamische Arrays
p = 0;
}
```

In dieser Function wird der Zeiger-Parameter als Referenz übergeben, da sich die Zuweisung von 0 auf das Argument auswirken soll. Die wichtigsten Unterschiede zwischen dynamisch erzeugten und „gewöhnlichen“ (vom Compiler erzeugten) Variablen sind:

1. Eine dynamisch erzeugte Variable hat im Unterschied zu einer gewöhnlichen Variablen **keinen Namen** und kann nur **indirekt** über einen Zeiger angesprochen werden.

Nach einem erfolgreichen Aufruf von p=new type enthält der Zeiger p die Adresse der Variablen. Falls p überschrieben und nicht anderweitig gespeichert wird, gibt es keine Möglichkeit mehr, sie anzusprechen, obwohl sie weiterhin existiert und Speicher belegt. Der für sie reservierte Speicher wird erst beim Ende des Programms wieder freigegeben.

Da zum Begriff “Variable” auch ihr Name gehört, ist eine “namenlose Variable” eigentlich widersprüchlich. Im C++-Standard wird „object“ als Oberbegriff für namenlose und benannte Variablen verwendet. Ein **Objekt** in diesem Sinn hat wie eine Variable einen Wert, eine Adresse und einen Datentyp, aber keinen Namen. Da der Begriff „Objekt“ aber auch oft für Variable eines Klassentyps (siehe Objekte) verwendet wird, wird zur Vermeidung von Verwechslungen auch der Begriff „namenlose Variable“ verwendet.

2. Der Name einer gewöhnlichen Variablen ist untrennbar mit reserviertem Speicher verbunden. Es ist nicht möglich, über einen solchen Namen nicht reservierten Speicher anzusprechen. Im Unterschied dazu existiert ein Zeiger p, über den eine dynamisch erzeugte Variable angesprochen wird, unabhängig von dieser Variablen. Ein Zugriff auf *p ist nur nach new und vor delete zulässig. Vor *new p* oder nach *delete p* ist der Wert von *p* unbestimmt und der Zugriff auf **p* ein Fehler, der einen Programmabbruch zur Folge haben kann:

```
int* pi = new int(17);
delete pi;
...
```

```
*pi=18; // Jetzt knallts - oder vielleicht auch nicht?
```

3. Der Speicher für eine gewöhnliche Variable wird **automatisch** freigegeben, wenn der Gültigkeitsbereich der Variablen verlassen wird. Der Speicher für eine dynamisch erzeugte Variable **muss** dagegen mit genau einem Aufruf von ***delete*** wieder freigegeben werden.

– Falls *delete* überhaupt nicht aufgerufen wird, kann das eine Verschwendung von Speicher (Speicherleck, memory leak) sein. Falls in einer Schleife immer wieder Speicher reserviert und nicht mehr freigegeben wird, können die swap files immer größer werden und die Leistungsfähigkeit des Systems nachlassen.

– Ein zweifacher Aufruf von *delete* mit demselben, von Null verschiedenen Zeiger, ist ein Fehler, der einen Programmabsturz zur Folge haben kann.

Beispiel: Anweisungen wie

```
int* pi = new int(1);
int* pj = new int(2);
```

zur Reservierung und

```
delete pi;
delete pj;
```

zur Freigabe von Speicher sehen harmlos aus. Wenn dazwischen aber eine ebenso harmlos aussehende Zuweisung stattfindet,

```
pi = pj;
```

haben die beiden *delete* Anweisungen den Wert von *pj* als Operanden. Diese Zuweisung führt also dazu, dass *pj* doppelt und *pi* überhaupt nicht freigegeben wird.

4. Oben haben wir gesehen, dass eine Zuweisung von Zeigern zu Aliasing führen kann. Bei einer Zuweisung an einen Zeiger auf eine dynamisch erzeugte Variable besteht außerdem noch die Gefahr von Speicherlecks. Das kann z.B. mit den folgenden Strategien vermieden werden:

– Man vermeidet solche Zuweisungen. Diese Strategie wird von der *smart pointer* Klasse *scoped_ptr* der Boost-Bibliothek (siehe http://boost.org/) verfolgt.

– Der Speicher für diese Variable wird vorher freigegeben.

```
int* pi = new int(17);
int* pj = new int(18);
delete pi; // um ein Speicherleck zu vermeiden
pi = pj;
```

Da anschließend zwei Zeiger pi und pj auf die mit new(18) erzeugte Variable *pj zeigen, muss darauf geachtet werden, dass der Speicher für diese Variable nicht freigegeben wird, solange über einen anderen Zeiger noch darauf zugegriffen werden kann. Diese Strategie wird von der smart pointer Klasse shared_ptr verfolgt.

– Der Speicher für eine dynamisch erzeugte Variable wird automatisch wieder freigegeben, wenn es keine Referenz mehr auf diese Variable gibt. Das wird als **garbage collection** bezeichnet. Garbage collection gehört allerdings noch nicht zum C++-Standard 2003. Es steht aber über die *smart pointer* Klasse *shared_ptr* der Boost-Bibliothek (siehe Abschnitt 3.12.5) sowie in speziellen Erweiterungen wie z.B. C++/CLI zur Verfügung. Es soll außerdem in den nächsten C++-Standard aufgenommen werden.

5. Der Operand von *delete* muss einen Wert haben, der das Ergebnis eines *new*-Ausdrucks ist. Wendet man *delete* auf einen anderen Ausdruck an, ist das außer bei einem Nullzeiger ein Fehler, der einen Programmabbruch zur Folge haben kann. Insbesondere ist es ein Fehler, *delete* auf einen Zeiger anzuwenden,

a) dem nie ein *new*-Ausdruck zugewiesen wurde.
b) der nach *new* verändert wurde.
c) der auf eine gewöhnliche Variable zeigt.

Beispiele: Der Aufruf von *delete* mit *p1, p2* und *p*3 ist ein Fehler:

```
int* p1; // a)
int* p2 = new int(1);
p2++; // b)
int i = 17;
int* p3 = &i; // c)
```

6. Bei gewöhnlichen Variablen prüft der Compiler bei den meisten Operationen anhand des Datentyps, ob sie zulässig sind oder nicht. Ob für einen Zeiger *delete* aufgerufen werden muss oder nicht aufgerufen werden darf, ergibt sich dagegen nur aus dem bisherigen Ablauf des Programms.

Beispiel: Nach Anweisungen wie den folgenden ist es unmöglich, zu entscheiden, ob *delete p* aufgerufen werden muss oder nicht:

```
int i,x;
int* p;
if (x>0) p=&i;
else p = new int;
```

Diese Beispiele zeigen, dass mit dynamisch erzeugten Variablen **Fehler** möglich sind, die mit „gewöhnlichen“ Variablen nicht vorkommen können. Zwar sehen die Anforderungen bei den einfachen Beispielen hier gar nicht so schwierig aus. Falls aber new und *delete* in verschiedenen Teilen des Quelltextes stehen und man nicht genau weiß, welche Anweisungen dazwischen ausgeführt werden, können sich leicht Fehler einschleichen, die nicht leicht zu finden sind.

– Deshalb sollte man **„gewöhnliche“ Variablen** möglichst immer **vorziehen**.
– Falls sich Zeiger nicht vermeiden lassen, sollte man das Programm immer so **einfach** gestalten, dass möglichst keine Unklarheiten aufkommen können.
– **Smart pointer** sind oft eine Alternative, die viele Probleme vermeidet.

Dynamisch erzeugte Variable bieten in der bisher verwendeten Form keine Vorteile gegenüber „gewöhnlichen“ Variablen. Trotzdem gibt es Situationen, in denen sie notwendig sind:

- Falls gewisse Informationen erst zur Laufzeit und nicht schon bei der Kompilation verfügbar sind. So muss zum Beispiel
- bei dynamisch erzeugten Arrays die Größe,

- bei verketteten Datenstrukturen (wie z.B. Listen und Bäume) die Verkettung, und
- bei einem Zeiger auf ein Objekt einer Basisklasse der Datentyp des Objekts, auf den er zeigt, erst während der Laufzeit festgelegtwerden. Aus diesem Grund werden die Elemente von objektorientierten Klassenhierarchien oft dynamisch erzeugt.
- Es gibt Datentypen wie z.B. die Klassen der VCL, mit denen man keine gewöhnlichen Variablen anlegen kann. Wenn man z.B. eine Variable des Typs *TEdi*t will, muss man sie dynamisch mit *new* erzeugen.
- Es gibt Betriebssysteme und Compiler, bei denen der für globale und lokale Variablen verfügbare Speicher begrenzt ist (z.B. auf 64 KB bei 16-bit-Systemen wie Windows 3.x oder MS-DOS). Für den Heap steht dagegen mehr Speicher zur Verfügung. Unter 32-bit-Betriebssystemen stehen für globale und lokale Variablen unabhängig vom physikalisch verfügbaren Hauptspeicher meist 2 GB oder mehr zur Verfügung. Deshalb besteht heutzutage meist keine Veranlassung, Variablen aus Platzgründen im Heap anzulegen. Allerdings findet man noch Relikte aus alten Zeiten, in denen Variablen nur aus diesem Grund im Heap angelegt werden.
- Sie können zur Reduzierung von Abhängigkeiten bei der Kompilation beitragen (Sutter 2000, Items 26-30).

Hinweis: In der Programmiersprache C gibt es anstelle der Operatoren new und delete die Funktionen malloc bzw. free. Sie funktionieren im Prinzip genauso wie new und delete und können auch in C++ verwendet werden. Da sie aber mit void*-Zeigern arbeiten, sind sie fehleranfälliger. Deshalb sollte man immer new und delete gegenüber malloc bzw. free bevorzugen. Man kann in einem Programm sowohl malloc, new, free und delete verwenden. Speicher, der mit new reserviert wurde, sollte aber nie mit free freigeben werden. Dasselbe gilt auch für malloc und delete.

Aufgabe 1

Gib nach jeden Programmierbefehl alle Adressen und Inhalte der einzelnen Variablen aus!

```
int a=2, b=5, *c=&a, *d=&b;

a=*c**d;
*d-=3;
b=a*b;
c=d;
b=7;
a=*c+*d;
```

Aufgabe 2

Gib nach jeden Programmierbefehl alle Adressen und Inhalte der einzelnen Variablen aus!

```
int a=2, b=5, *c=&a, *d=&b;
int **zz=NULL;

a = *c + *d;
zz=&d;
**zz=*zz-10;
*c *= 3;
b = a * *c;
c = d;
a = 7-*d;
b = *c * *d;
*c = *c + **zz;
```

Aufgabe 3: Verständnisübung mit Zeigern

Nach den Definitionen

```
  int i=5, j=7;
  int *pi, *pj;
```

sollen die folgenden Zuweisungen in einem Programm stehen:

```
  pi=i;
  pi=&i;
  *pi=i;
  *pi=&i;
  pi=j;
  pj=&j;
  pi=*pj;
```

```
  j=*pi;
  *pi=i**pj;
  pi=0;
```

Gib an, welche dieser Zuweisungen syntaktisch korrekt sind. Gib außerdem für jeden syntaktisch korrekten Ausdruck den Wert der linken Seite an, wobei die Ergebnisse der zuvor ausgeführten Anweisungen vorausgesetzt werden sollen. Falls die linke Seite ein Zeiger ist, gib den Wert der dereferenzierten Variablen an.

Aufgabe 4: Verständnisübung mit Zeigern

Beschreibe für die Funktion

```
int f(int a, bool b, bool c, bool d)
{
  int* p; int* q; int x=0;
  if (a==1) { p=0; q=0; }
  else if (a==2) { p=new int(17); q=new int(18); }
  else if (a==3) { p=&x; q=&x; }
  if (b) p=q;
  if (c) x=*p+*q;
  if (d) { delete p; delete q; }
  return x;
}
```

das Ergebnis der Aufrufe:

a) `f(0,false,true,true);`
b) `f(1,false,false,true);`
c) `f(2,false,true,false);`
d) `f(2,true,true,true);`
e) `f(3,true,true,true);`
f) `f(3,true,true,false);`

Aufgabe 5: Verständnisübungen mit Zeigern

Erstelle zwei Zeiger pAS1 und pAS2 auf zwei AnsiStrings.
a) Belege die beiden Speicherplätzen, auf die die beiden Zeiger zielen, jeweils einen kurzen Text zu und gib diesen aus.
b) Führe die Zuweisung `pAS2=pAS1;` aus und erkläre mit einer Graphik, warum nun beide Ausgaben den gleichen Text liefern.
c) Führe die Zuweisung `pAS2=pAS1;` aus, gib `*pAS2` aus, verändere den Text auf den pAS2 zeigt und gib dann `*pAS1` aus. Erkläre wieder mit einer Graphik was hier geschieht.
d) Deklariere zwei Variablen `AS1,AS2` und weise ihnen einen Text zu. Deklariere zwei weitere Pointervariablen `pAS3,pAS4` und lasse sie (mittels Adressoperator) auf jene Speicherbereiche zeigen, die durch die Variablen AS1 und AS2 besetzt werden.
e) Lasse den Zeiger pAS1 auf den Speicherinhalt von AS2 und den Zeiger pAS2 auf den

Speicherinhalt von AS1 zeigen.
f) Lasse einen der beiden Zeiger auf die definierte Leerstelle (NULL) zeigen und versuche auszugeben, wohin beide Zeiger zielen.

Aufgabe 6: Verständnisübung mit Zeigern

a) Erkläre, an welcher Stelle und warum das folgende Programm (erst während der Laufzeit) einen Fehler meldet.
b) Verändere das Programm so, dass der Inhalt beider Speicherstellen (auf die die beiden Zeiger zielen) ausgegeben wird.

```
//---------------------------------------------------------------------------
-
#include <vcl.h>
#pragma hdrstop
#include "Pointer1Unit1.h"
//---------------------------------------------------------------------------
-
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm2 *Form2;
int *pi1, *pi2;
//---------------------------------------------------------------------------
-
__fastcall TForm2::TForm2(TComponent* Owner)
    : TForm(Owner)
{
}
//---------------------------------------------------------------------------
-
void __fastcall TForm2::Button1Click(TObject *Sender)
{
 int zahl1=13,zahl2=17;
 pi1=&zahl1;
 *pi2=zahl2;
 Memo1->Lines->Add("Inhalt des Speicherpatzes auf den pi1 zeigt:
"+IntToStr(*pi1));
 Memo1->Lines->Add("Inhalt des Speicherpatzes auf den pi2 zeigt:
"+IntToStr(*pi2));
}
//---------------------------------------------------------------------------
-
```

Aufgabe 7: Verständnisübung mit Zeigern

Beschreibe das Ergebnis der folgenden Anweisungen:

```
  int* f_retp()
  {
  int x=1;
  return &x;
  }
...

  Memo1->Lines->Add(IntToStr(*f_retp()));
```

Struktur - eine zusammengesetzter Datentyp

Mit Arrays können Variablen gleichen Typs zusammengestellt werden. In der realen Welt gehören aber meist Daten unterschiedlichen Typs zusammen. So hat ein Auto einen Markennamen und eine Typbezeichnung, die als Zeichenkette unterzubringen ist. Dagegen eignet sich für Kilometerzahl und Leistung eher der Typ Integer. Für den Preis bietet sich der Typ float an. Bei bestimmten Autohändlern könnte auch double erforderlich sein. Alles zusammen beschreibt ein Auto.

Modell

Vielleicht werden Sie einwerfen, dass ein Auto noch mehr Bestandteile hat. Da gibt es Bremsscheiben, Turbolader und Scheibenwischer. Das ist in der Realität richtig. Ein Programm interessiert sich aber immer nur für bestimmte Eigenschaften, die der Programmierer mit dem Kunden zusammen festlegt. Unser Beispiel würde für einen kleinen Autohändler vielleicht schon reichen. Eine Autovermietung interessiert sich vielleicht überhaupt nicht für den Wert des Autos, aber möchte festhalten, ob es für Nichtraucher reserviert ist. Eine Werkstatt dagegen könnte sich tatsächlich für alle Teile interessieren. Ein Programm, das die Verteilung der Firmenfahrzeuge verwaltet, interessiert sich vielleicht nur für das Kennzeichen. Es entsteht also ein Modell eines Autos, das bestimmte Bestandteile enthält und andere vernachlässigt, je nachdem was das Programm benötigt. Bereits in C gab es für solche Zwecke die Struktur, die mehrere Variablen zu einer zusammenfasst. Das Schlüsselwort für die Bezeichnung solch zusammengesetzter Variablen lautet struct. Nach diesem Schlüsselwort folgt der Name des neuen Typen. In dem folgenden geschweiften Klammernblock werden die Bestandteile der neuen Struktur aufgezählt. Diese unterscheiden sich nicht von der bekannten Variablendefinition. Den Abschluss bildet ein Semikolon.

struct

Um ein Auto zu modellieren, wird ein neuer Variablentyp namens TAutoTyp geschaffen, der ein Verbund mehrerer Elemente ist.

```
struct TAutoTyp // Definiere den Typ
{
    char Marke[MaxMarke];
    char Modell[MaxModell];
    long km;
    int kW;
    float Preis;
};  // Hier vergisst man leicht das Semikolon!
```

Syntaxbeschreibung

Das Schlüsselwort struct leitet die Typdefinition ein. Es folgt der Name des neu geschaffenen Typs, hier TAutoTyp. In dem nachfolgenden geschweiften Klammerpaar werden alle Bestandteile der

Struktur nacheinander aufgeführt. Am Ende steht ein Semikolon, das man selbst als erfahrener Programmierer immer wieder einmal vergisst. Variablendefinition Damit haben wir den Datentyp TAutoTyp geschaffen. Er kann in vieler Hinsicht verwendet werden wie der Datentyp int. Sie können beispielsweise eine Variable von diesem Datentyp anlegen. Ja, Sie können sogar ein Array und einen Zeiger von diesem Datentyp definieren.

```
TAutoTyp MeinRostSammler; // Variable anlegen
TAutoTyp Fuhrpark[100];   // Array von Autos
TAutoTyp *ParkhausKarte;  // Zeiger auf ein Auto
```

Elementzugriff

Die Variable MeinRostSammler enthält nun alle Informationen, die in der Deklaration von TAutoTyp festgelegt sind. Um von der Variablen auf die Einzelteile zu kommen, wird an den Variablenname ein Punkt und daran der Name des Bestandteils gehängt.

```
// Auf die Details zugreifen
MeinRostSammler.km = 128000;
MeinRostSammler.kW = 25;
MeinRostSammler.Preis = 25000.00;
```

Zeigerzeichen

Wenn Sie über einen Zeiger auf ein Strukturelement zugreifen wollten, müssten Sie über den Stern referenzieren und dann über den Punkt auf das Element zugreifen. Da aber der Punkt vor dem Stern ausgewertet wird, müssen Sie eine Klammer um den Stern und den Zeigernamen legen.

```
TAutoTyp *ParkhausKarte = 0;      // Erst einmal keine Zuordnung
ParkhausKarte = &MeinRostSammler; // Nun zeigt sie auf ein Auto
(*ParkhausKarte).Preis = 12500;   // Preis für MeinRostSammler
```

Das mag zwar logisch sein, aber es ist weder elegant noch leicht zu merken. Zum Glück gibt es in C und C++ eine etwas hübschere Variante, über einen Zeiger auf Strukturelemente zuzugreifen. Dazu wird aus Minuszeichen und Größer-Zeichen ein Symbol zusammengesetzt, das an einen Pfeil erinnert.

```
ParkhausKarte->Preis = 12500;
L-Value
Strukturen sind L-Values. Sie können also auf der linken Seite einer
Zuweisung stehen. Andere Strukturen des gleichen Typs können ihnen
zugewiesen werden. Dabei wird die Quellvariable Bit für Bit der Zielvariable
zugewiesen.
TAutoTyp MeinNaechstesAuto, MeinTraumAuto;
MeinNaechstesAuto = MeinTraumAuto;
```

Trotzdem die beiden Strukturvariablen nach dieser Operation ganz offensichtlich gleich sind, kann man dies nicht einfach durch eine Anwendung des doppelten Gleichheitszeichen nachprüfen. Sie können bei Strukturen die Typdeklaration und die Variablendefinition zusammenfassen, indem der

Name der Variablen direkt nach der geschweiften Klammer eingetragen wird.

```
struct // hier wird kein Typ namentlich festgelegt
{
    char Marke[MaxMarke];
    char Modell[MaxModell];
    long km;
    int kW;
    float Preis;
} MeinErstesAuto, MeinTraumAuto;
```

Hier werden im Beispiel die Variablen MeinErstesAuto und MeinTraumAuto gleich mit ihrer Struktur definiert. Werden auf diese Weise gleich Variablen dieser Struktur gebildet, muss ein Name für den Typ nicht unbedingt angegeben werden. Damit ist dann natürlich keine spätere Erzeugung von Variablen dieses Typs möglich. Initialisierung Auch Strukturen lassen sich initialisieren. Dazu werden wie bei den Arrays geschweifte Klammern verwendet. Auch hier werden die Werte durch Kommata getrennt.

```
TAutoTyp JB = {"Aston Martin", "DB5", 12000, 90, 12.95};
TAutoTyp GWB = {0};
```

Beispiel Schule

```
#include <iostream>
using namespace std;

struct PC
{
    string Marke;
    double CPUGeschwindigkeit;
    int RAM;
    int Festplatte;
};

struct Raum
{
    int AnzPCs;
    string Name;
    double Groesse;
    int AnzTische;
    PC computer[20];
};

struct Schule
{
    string Name;
    string Adresse;
    int AnzSchueler;
    string Administrator;
```

```
    double Groesse;
    int AnzLehrer;
    int Gruendungsjahr;
    Raum rarr[100];    //Array von Räumen
    Raum externeraeume[5];
};

int main(int argc, char** argv)
{

    Schule s;
    s.Name="BG BRG Amstetten";
    s.Adresse="Anzengruberstraße 6";
    s.Gruendungsjahr=1938;
    s.AnzLehrer=90;
    s.AnzSchueler=726;
    s.Administrator="MMag. Matthias Haslauer";
    s.rarr[0].Name="Sekretariat";
    s.rarr[0].AnzTische=2;
    s.rarr[0].Groesse=30;
    s.rarr[0].AnzPCs=2;
    s.rarr[1].Name="TUS1";
    s.rarr[1].AnzTische=0;
    s.rarr[1].Groesse=500;
    s.rarr[1].AnzPCs=0;
    s.rarr[2].Name="TUS2";
    s.rarr[2].AnzTische=0;
    s.rarr[2].Groesse=300;
    s.rarr[2].AnzPCs=0;
    s.rarr[0].computer[0].CPUGeschwindigkeit=3.07;
    s.rarr[0].computer[0].Marke="HP";
    s.rarr[0].computer[0].RAM=4;
    s.rarr[0].computer[0].Festplatte=256;



    for (int i=0;i<3;i++)
    {
        cout << "Raum: " << s.rarr[i].Name << endl;
        cout << "Anzahl Tische: " << s.rarr[i].AnzTische << endl;
        cout << "Groesse: " << s.rarr[i].Groesse << endl;
        cout << "Anzahl PCs: " << s.rarr[i].AnzPCs << endl;
        cout << "===============";
    }
}
```

Objektorientierte Programmierung (OOP)

Am Anfang steht wohl die Frage was Objektorientierte Programmierung überhaupt ist. Dies ist nicht ganz einfach zu erklären und es gibt viele Definitionen die dies versuchen.

Grob gesagt ist **Objektorientierte Programmierung (OOP) ein Verfahren zur Strukturierung von Programmen, bei dem Programmlogik zusammen mit Zustandsinformationen (Datenstrukturen) in Einheiten, den Objekten, zusammengefasst werden**.

Es ist also möglich mehrere Objekte des gleichen Typs zu haben. Jedes dieser Objekte hat dann seinen individuellen Zustand. Dies bietet die Möglichkeit einer besseren Modularisierung der Programme, sowie einer höheren Wartbarkeit des Quellcodes.

Klassen

Die Klasse (class) ist die zentrale Datenstruktur in C + +. Sie kapselt zusammengehörige Daten und Funktionen vom Rest des Programmes ab. Sie ist das Herz der objektorientierten Programmierung (OOP).

Klassen sind ein erweitertes Konzept von Datenstrukturen denen es erlaubt ist, außer Daten, noch Methoden zu beinhalten. Ein Objekt ist eine Instanz einer Klasse, welches sich die Eigenschaften der Klasse zunutze machen kann. Es ist möglich mehrere Objekte einer Klasse zu instanziieren, wobei jedes dieser Objekte zwar die selben Eigenschaften hat, intern aber einen ganz individuellen Zustand haben kann.

Beispiel: Zwei Instanzen der Klasse „Gegner“ werden erzeugt. Nachdem ein Gegner Schaden erlitten hat sind seine Lebenspunkte auf 50 gesunken. Die des anderen Gegners haben allerdings noch den Wert 100.

Klassen werden normalerweise mit dem Schlüsselwort **class** deklariert. Außerdem haben sie immer folgendes Format:

```
class Klassenname
{
Zugriffsspezifizierung 1:
Member 1;
...
Zugriffsspezifizierung 2:
Member 2;
...
...
};
```

Der Klassenname identifiziert die Klasse, der Objektname ist eine optionale Liste von Namen von Objekten dieser Klasse. Der Körper der Klasse wird durch geschweifte Klammern gekennzeichnet und kann verschiedene Member (Methoden, Membervariablen) beinhalten. Diesen können verschiedene Zugriffsspezifierungen zugewiesen werden.

Zugriffsrechte

public: Auf Members kann von überall zugegriffen werden von wo das Objekt sichtbar ist

private: Auf Member kann nur innerhalb anderer Member zugegriffen werden oder aus befreundeten Klassen und Funktionen

protected: Members sind verfügbar für Member der selben Klasse, befreundeten Funktionen/Klassen und Abgeleiteten Klassen

Attribute

Die Festlegung, welche Daten zu einem Typ gehören, erfolgt bei der Definition der Klasse. Die Objekte enthalten jeweils unabhängig voneinander einen Satz von Datenkomponenten (Attributen). Ihre Startwerte können durch Konstruktoren festgelegt werden. C + + erlaubt auch die Zuweisung von Anfangswerten bei der Definition:

```
class Enemy {
    public:   //Zugriffsrecht public
        void setHealth(int h);
        int getHealth();
        string name;
                int id;
    private:  //Zugriffsrecht private
        int health;
};
```

Elementzugriff

Im Beispiel davor wird ein Objekt vom **Typ Enemy** angelegt. Das **Objekt** enthält die **3 Variablen health (integer), id (int) und name (String)**, wie es in der Klassendefinition zu sehen ist. Um auf eine öffentliche Elementvariable zugreifen zu können, wird an den Objektnamen ein **Punkt** und dann der in der Klassendefinition verwendete Elementname gehängt. Im Beispiel wird der Name auf Andreas gesetzt.

```
    ...
    //Zugriff auf öffentliche Variable name
    e1.name="Andreas";
    //Zugriff auf öffentliche Variable id
    e1.id=1;
    ...
    //e1.health=100; ist nicht erlaubt, da health das Schlüsselwort private
besitzt!!!
```

Klassenmethoden

Nun können Sie ein Objekt von der Klasse Enemy erzeugen. Aber was nützt Ihnen die schönste Datenstruktur in Ihrem Programm, wenn sie nicht durch Funktionen zum Leben erweckt wird? Sie werden z.B. einen Namen und Lebenspunkte eingeben und ausgeben wollen. Vielleicht wollen Sie die Lebenspunkte verringern oder erhöhen. Kurz gesagt, ein Datenverbund ist nichts wert ohne Funktionen, die auf ihn wirken. Aber die Funktionen sind auch nur im Zusammenhang mit ihrem Datenverbund sinnvoll. Aus diesem Grund werden die Funktionen ebenso in die Klasse integriert wie die Datenelemente. Eine Funktion, die zu einer Klasse gehört, nennt man Elementfunktion oder auf englisch member function. In anderen objektorientierten Programmiersprachen spricht man auch von einer Methode oder Operation. Aufruf So, wie Sie auf Datenelemente nur über ein real existierendes Objekt, also eine Variable dieser Klasse zugreifen können, kann auch eine Funktion nur über ein Objekt aufgerufen werden. Objekt und Funktionsnamen werden dabei durch einen Punkt getrennt. Die Funktion arbeitet mit den Daten des Objekts, über das sie gerufen wurde. Aus prozeduraler Sicht könnte man es so sehen, dass eine Elementfunktion immer bereits einen Parameter mit sich trägt, nämlich das Objekt, über das sie aufgerufen wurde.

Beispiel:

```
#include <iostream>
#include <conio.h>
#include <string.h>

using namespace std;

//Klasse Enemy
class Enemy {
    public:   //Zugriffsrecht public
        void setHealth(int h);
        int getHealth();
        string name;
                int id;
    private:  //Zugriffsrecht private
        int health;
};
//Methode setHealth zum Setzen der Lebenspunkte
void Enemy::setHealth(int h)
{
    health=h;
}

//Methode getHealth() zum Auslesen der Lebenspunkte
int Enemy::getHealth()
{
    return health;
}

//Hauptprogramm
int main()
{
    Enemy e1;   //Objekt e1 der Klasse Enemy wird erzeugt

    //Zugriff auf öffentliche Variable name
```

```
    e1.name="Andreas";
    //Zugriff auf öffentliche Variable id
    e1.id=1;

    //Aufruf der Methode setHealth
    e1.setHealth( 100 );
    //Aufruf der Methode getHealth und Zugriff auf die öffentliche Variable
name
    cout << "Der Gegner " << e1.name << " hat " << e1.getHealth() << "
Lebenspunkte.\n";
    //Aufruf der Methode setHealth
    e1.setHealth( 50 );
    //Aufruf der Methode getHealth und Zugriff auf die öffentliche Variable
name
    cout << "Der Gegner " << e1.name << " hat " << e1.getHealth() << "
Lebenspunkte.n";

    return 0;
}
```

Methodendefinition innerhalb einer Klasse

Alternativ können Sie die Elementfunktion auch direkt in der Klasse definieren. Das wird leicht unübersichtlich, darum sollten Sie das nur bei sehr kurzen Funktionen tun.

```
#include <iostream>
#include <conio.h>
#include <string.h>

using namespace std;

class Enemy {
    public:     //Zugriffsrecht public
                /***** INLINE - METHODENDEFINITION *****/
        //Methode setHealth zum Setzen der Lebenspunkte
        void setHealth(int h)
        {
            health=h;
        }
        //Methode getHealth() zum Auslesen der Lebenspunkte
        int getHealth()
        {
            return health;
        }

        string name;
                int id;


    private:     //Zugriffsrecht private
```

```
        int health;
};

//Hauptprogramm
int main()
{
    Enemy e1;   //Objekt e1 der Klasse Enemy wird erzeugt

    //Zugriff auf öffentliche Variable name
    e1.name="Andreas"

    //Aufruf der Methode setHealth
    e1.setHealth( 100 );
    //Aufruf der Methode getHealth und Zugriff auf die öffentliche Variable
name
    cout << "Der Gegner " << e1.name << " hat " << e1.getHealth() << "
Lebenspunkte.\n";
    //Aufruf der Methode setHealth
    e1.setHealth( 50 );
    //Aufruf der Methode getHealth und Zugriff auf die öffentliche Variable
name
    cout << "Der Gegner " << e1.name << " hat " << e1.getHealth() << "
Lebenspunkte.\n";

    return 0;
}
```

Aufruf

So, wie Sie auf Datenelemente nur über ein real existierendes Objekt, also eine Variable dieser Klasse zugreifen können, kann auch eine Funktion **nur über ein Objekt aufgerufen werden**.
Objekt und **Funktionsnamen** werden dabei durch einen **Punkt** getrennt. Die Funktion arbeitet mit den Daten des Objekts, über das sie gerufen wurde. Aus prozeduraler Sicht könnte man es so sehen, dass eine Elementfunktion immer bereits einen Parameter mit sich trägt, nämlich das Objekt, über das sie aufgerufen wurde.

```
    ....
    Enemy e1;   //Objekt e1 der Klasse Enemy wird erzeugt

    //Aufruf der Methode setHealth
    e1.setHealth( 100 );
    ....
```

Konstruktor

Die Elementfunktion, die beim Erzeugen eines Objekts aufgerufen wird, nennt man Konstruktor. In dieser Funktion können Sie dafür sorgen, dass alle Elemente des Objekts korrekt initialisiert sind. Der Konstruktor trägt immer den Namen der Klasse selbst und hat keinen Rückgabetyp, auch nicht void.

Der Standardkonstruktor hat keine Parameter.

```
class Enemy {
    public:     //Zugriffsrecht public

                /**** Konstruktor ****/
                Enemy()
                {
                       health=0;
                       name="";
                       id=0;
                }

        string name;
                int id;


    private:     //Zugriffsrecht private
        int health;
};
```

Dekonstruktor

Im Falle einer Datumsklasse wäre es sinnvoll, dass der Konstruktor alle Elemente auf 0 setzt. Daran kann jede Elementfunktion leicht erkennen, dass das Datum noch nicht festgelegt wurde. Sie könnten alternativ das aktuelle Datum ermitteln und eintragen. Im Beispiel ist auch ein Destruktor definiert worden, obwohl er im Falle eines Datums keine Aufgabe hat.

```
class Enemy {
    public:     //Zugriffsrecht public

                /**** Destruktor ****/
                ~Enemy()
                {
                     cout << "Ressourcen von " <<  name << " wurden
freigegeben!";
                }

        char name[50];
                int id;


    private:     //Zugriffsrecht private
        int health;
};
```

Überladen von Konstruktor

Konstruktoren können genauso überladen werden wie normale Funktionen auch. Es kann neben dem Standardkonstruktor auch mehrere weitere Konstruktoren mit verschiedenen Parametern geben. Der Compiler wird anhand der Aufrufparameter unterscheiden, welcher Konstruktor verwendet wird.

```
class Enemy {
    public:     //Zugriffsrecht public


                /**** Konstruktor ****/
                Enemy()
                {
                     name="Max Mustermann";
                }
                Enemy(string n)
                {
                     name=n;
                }
                Enemy(string n, int h)
                {
                     name=n;
                     health=h;
                }
                /**** Destruktor ****/
                ~Enemy()
                {
                     cout << "Ressourcen von " <<  name << " wurden
freigegeben!";
                }

        char name[50];
                int id;


    private:     //Zugriffsrecht private
        int health;
};
```

Je nachdem wie viele Parameter beim Konstruktoraufruf übergeben werden, wird der passende Konstruktor ausgewählt. Existiert gar kein Konstruktor so wird vom Compiler ein leerer Konstruktor eingefügt.

```
int main (void)
{

     Enemy e;                    //führt den Standardkonstruktor Enemy() aus
-> name="Max Mustermann"
     Enemy e("Fritz Phantom");   //führt den 2. Konstruktor mit den
passenden Argumenten aus -> name="Fritz Phantom"
     Enemy e("Hans Wurst", 100); //führt den 3. Konstruktor mit den
passenden Argumenten aus -> name="Hans Wurst", health=100
```

```
    return 0;
}
```

Vererbung

Die Informatik steckt voller schöner Analogien. So hat die objektorientierte Programmierung den Begriff der »Vererbung« eingeführt, wenn eine Klasse von einer anderen Klasse abgeleitet wird und deren Eigenschaften übernimmt.

Basisklasse

Eine Klasse kann als Basis zur Entwicklung einer neuen Klasse dienen, ohne dass ihr Code geändert werden muss. Dazu wird die neue Klasse definiert und dabei angegeben, dass sie eine abgeleitete Klasse der Basisklasse ist. Daraufhin gehören alle öffentlichen Elemente der Basisklasse auch zur neuen Klasse, ohne dass sie erneut deklariert werden müssen. Man sagt, die neue Klasse erbt die Eigenschaften der Basisklasse.

Spezialisierung

Durch die Elemente, die in der abgeleiteten Klasse definiert werden, wird die abgeleitete Klasse zu einem besonderen Fall der Basisklasse. Sie besitzt alle Eigenschaften der Basisklasse. Sie können neue Elemente hinzufügen. Am folgenden Beispiel wird deutlich, warum das Hinzufügen von Eigenschaften eine Spezialisierung ist.

Ein Beispiel

Aus Sicht eines Computerprogramms haben alle **Personen** (=Basisklasse)

- Namen
- Adressen und
- Telefonnummern.

Geschäftspartner haben darüber hinaus eine

- Bankverbindung.

Da die Geschäftspartner auch Personen sind, haben sie neben ihrer Bankverbindung auch Namen, Adressen und Telefonnummern (von der Basisklasse).

Einige Geschäftspartner können auch **Kunden** sein. **Kunden** haben zusätzlich zu den Eigenschaften eines Geschäftspartners noch eine

- Lieferanschrift.

Lieferanten sind keine Kunden, aber auch Geschäftspartner. Sie haben noch

- eine Anzahl an offenen Rechnungen.

Selbst **Mitarbeiter** sind eigentlich Geschäftspartner, denn sie haben eine Bankverbindung. Darüber

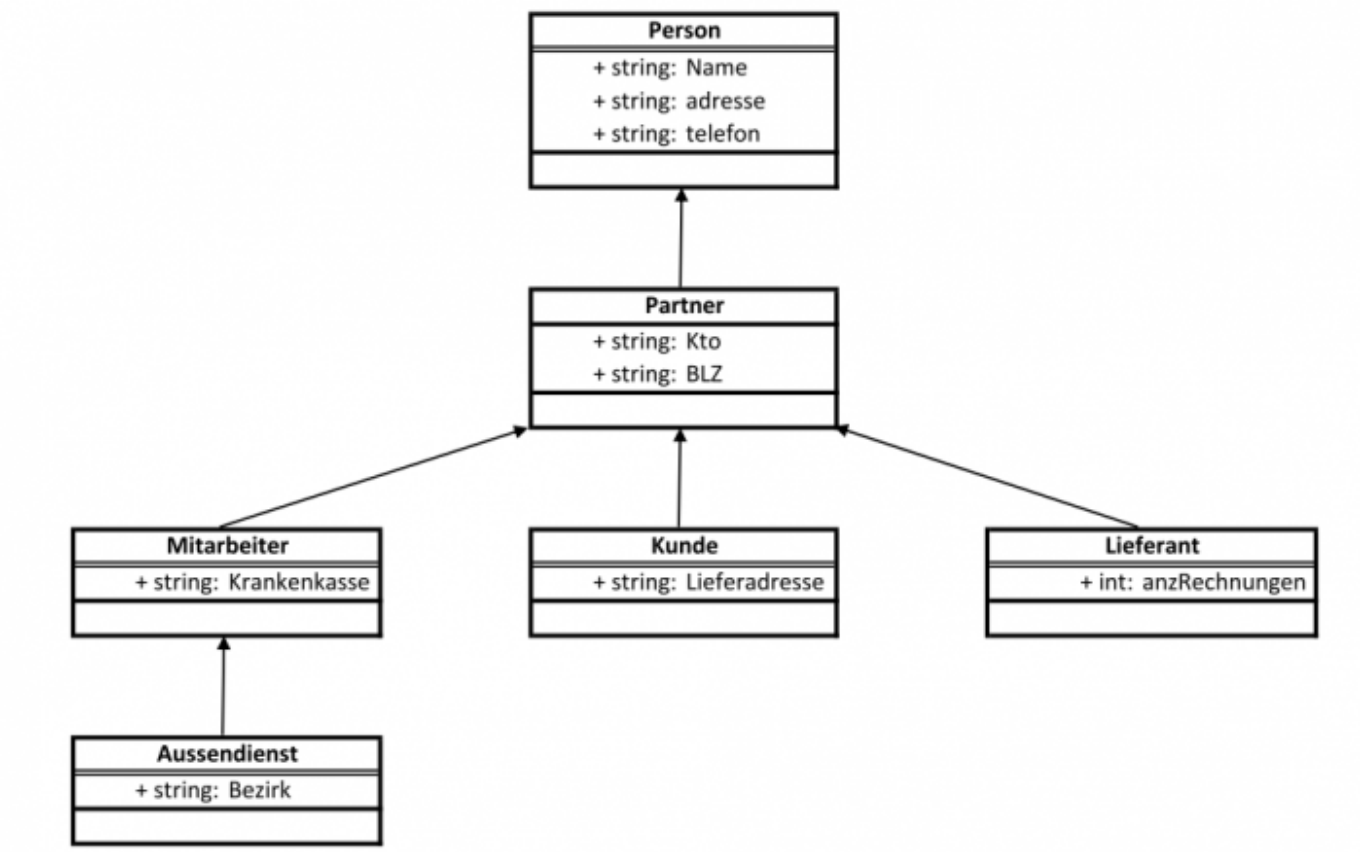
hinaus haben sie eine

- Krankenkasse.

Außendienstler haben alle Eigenschaften eines Mitarbeiters und zusätzlich ihren

- Bezirk.

Beispiel als UML-Diagramm:



Beispiel in C + +

```
class Person
{
    public:
       string Name, Adresse, Telefon;
};

class Partner : public Person
{
    public:
       string Kto, BLZ;
};

class Mitarbeiter : public Partner
{
    public:
       string Krankenkasse;
```

```
};

class Kunde : public Partner
{
    public:
       string Lieferadresse;
};

class Lieferant : public Partner
{
    public:
       int anzRechnungen;
};

class Aussendienst : public Mitarbeiter
{
    public:
       string Bezirk;
};

int main (void) {

   Aussendienst a1;

   a1.Bezirk="Amstetten";
   a1.Name="Hans Wurst";
   a1.Krankenkasse="Gebietskrankenkasse";
   a1.BLZ="14200";

   return 0;
}
```

Vorteil von Vererbungen:

Damit stellt die Ausgangsklasse Person die Verallgemeinerung dar und jede abgeleitete Klasse eine Spezialisierung. Der Vorteil dieser Technik ist, dass der bestehende Code der Basisklasse nicht noch einmal für die neu geschaffene Klasse geschrieben werden muss. Beispielsweise würde eine Prüffunktion der Adresse, die der Klasse Person hinzugefügt wird, automatisch auch allen anderen Klassen, die direkt oder indirekt von Person abgeleitet wurden, hinzugefügt, ohne dass eine Zeile Code mehr geschrieben werden müsste. Eine Änderung in der Klasse Mitarbeiter würde immer auch auf die Außendienstler durchschlagen. Es gibt aber keine Rückwirkung auf die Geschäftspartner.

Zusammenfassung

Die objektorientierte Programmierung hat einige neue Begriffe aufgebracht.

Objekt und Klasse

Der zentrale Begriff des Objekts bezeichnet einen Speicherbereich, der durch eine Klasse beschrieben wird. Prinzipiell kann sich der Anfänger ein Objekt als eine besondere Art einer Variablen vorstellen und die Klasse als die Typbeschreibung. Im Buch wird ein Objekt auch hin und wieder als Variable bezeichnet, insbesondere dann, wenn sich ein Objekt an dieser Stelle wie jede andere Variable verhält.

Attribut

Die Daten-Elemente einer Klasse werden meist Elementvariable genannt. In der objektorientierten Literatur findet sich dafür auch die Bezeichnung Attribut. Damit wird angedeutet, dass die Elementvariablen die Eigenschaften eines Objekts beschreiben.

Methode und Operation

Die Funktionen einer Klasse, die hier als Elementfunktionen bezeichnet werden, finden sich in der objektorientierten Literatur unter dem Namen Methode oder Operation wieder. Diese Bezeichnung bringt zum Ausdruck, dass die Funktion nur über das Objekt erreichbar und damit eine Aktion des Objekts ist.

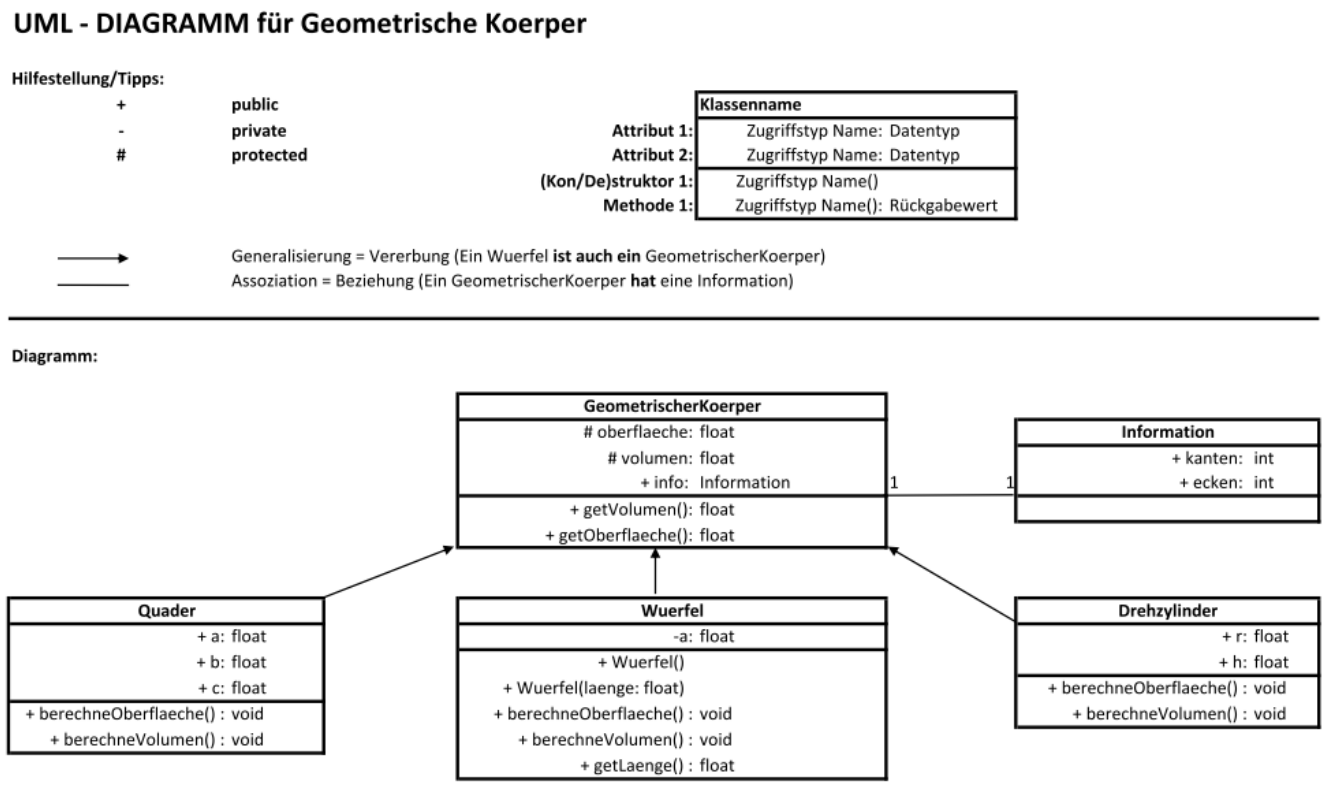
Konstruktor

Die Elementfunktion, die beim Erzeugen eines Objekts aufgerufen wird, nennt man Konstruktor. In dieser Funktion können Sie dafür sorgen, dass alle Elemente des Objekts korrekt initialisiert sind. Der Konstruktor trägt immer den Namen der Klasse selbst und hat keinen Rückgabetyp, auch nicht void. Der Standardkonstruktor hat keine Parameter.

Dekonstruktor

Im Falle einer Datumsklasse wäre es sinnvoll, dass der Konstruktor alle Elemente auf 0 setzt. Daran kann jede Elementfunktion leicht erkennen, dass das Datum noch nicht festgelegt wurde. Sie könnten alternativ das aktuelle Datum ermitteln und eintragen. Im Beispiel ist auch ein Destruktor definiert worden, obwohl er im Falle eines Datums keine Aufgabe hat

Angabe



Loesung

```
/* Beispiel: Klassen (Arrays von Objekten,Vererbung, Attribute, Methoden, Konstruktor)
   Filename: main.cpp
   Author: Lahmer
   Title: Geometrische Formen
   Description: In diesem Beispiel lernen Sie wie man Arrays von Objekten anlegt und verwendet.
   Change: 20.02.2018
*/
#include <iostream>
#include <stdlib.h>
#include <time.h>

#define PI 3.14

using namespace std;

class Information
{
    public:
        int kanten;
        int ecken;
};
```

```
class GeometrischerKoerper
{
    protected:
        float oberflaeche;
        float volumen;

    public:
        Information info;    //Assoziation, ein geometrischer Koerper hat
eine (Beziehung mit) Information
        float getVolumen()
        {
            return volumen;
        }
        float getOberflaeche()
        {
            return oberflaeche;
        }
};

class Wuerfel : public GeometrischerKoerper    //Vererbung, Klasse Wuerfel
wird von Klasse GeometrischerKoerper abgeleitet
{
    private:
        float a;
    public:
        Wuerfel()
        {
            a=0.0;
        }
        Wuerfel(float laenge)
        {
            a=laenge;
        }
        void berechneOberflaeche()
        {
            oberflaeche=a*a*6;    //6 Seiten hat ein Würfel
        }
        void berechneVolumen()
        {
            volumen=a*a*a;
        }
        float getLaenge()
        {
            return a;
        }

};

class Drehzylinder : public GeometrischerKoerper
{
```

```
    public:
        float r;
        float h;

        void berechneOberflaeche()
        {
            oberflaeche=2*r*r*PI+2*r*PI*h;    //2x Grundfläche + Mantel
        }
        void berechneVolumen()
        {
            volumen=r*r*PI*h;  //Grundfläche * H
        }

};

class Quader : public GeometrischerKoerper
{
    public:
        float a;    //a*b = Grundfläche
        float b;
        float c;    //c=Höhe
        void berechneOberflaeche()
        {
            oberflaeche=2*(a*b+a*c+b*c);    //
        }
        void berechneVolumen()
        {
            volumen=a*b*c;
        }

};




int main(int argc, char** argv) {

    Wuerfel *w=new Wuerfel(3.3);
    w->berechneOberflaeche();
    w->berechneVolumen();
    w->info.ecken=8;
    w->info.kanten=12;

    cout << "Wuerfel: \t" << endl;
    cout << "---------------------------" << endl;
    cout << "Seitenlaenge: \t" << w->getLaenge() << endl;
    cout << "Oberflaeche: \t" << w->getOberflaeche() << endl;
    cout << "Volumen: \t" << w->getVolumen() << endl;
    cout << "Anzahl Kanten: \t" << w->info.kanten << endl;
    cout << "Anzahl Ecken: \t" << w->info.ecken << endl;
```

```
    cout << endl << "##############################" << endl;

    srand(time(NULL));

    cout << endl << "Array von Wuerfeln: " << endl;
    Wuerfel *warray=new Wuerfel[3];   //Alternativ geht auch Wuerfel
warray[3];
    for(int i=0; i<3; i++)
    {

        warray[i]=Wuerfel((rand()%10+1)/2.0);
        warray[i].berechneOberflaeche();
        warray[i].berechneVolumen();
        cout << endl << "Wuerfel " << i << ":" << endl;
        cout << "----------------------------" << endl;
        cout << "Seitenlaenge: \t" << warray[i].getLaenge() << endl;
        cout << "Oberflaeche:\t" << warray[i].getOberflaeche() << endl;
        cout << "Volumen:\t" << warray[i].getVolumen() << endl;
    }

    cout << endl << "##############################" << endl;

    cout << endl << "Array von Drehzylindern: " << endl;
    Drehzylinder *zylinder=new Drehzylinder[3];
    for(int i=0; i<3; i++)
    {

        zylinder[i].h=rand()%10;
        zylinder[i].r=(rand()%20+1)/2.0;
        zylinder[i].berechneOberflaeche();
        zylinder[i].berechneVolumen();
        cout << endl << "Drehzylinder " << i << ":" << endl;
        cout << "----------------------------" << endl;
        cout << "Radius: \t" << zylinder[i].r << endl;
        cout << "Hoehe: \t\t" << zylinder[i].h << endl;
        cout << "Oberflaeche:\t" << zylinder[i].getOberflaeche() << endl;
        cout << "Volumen:\t" << zylinder[i].getVolumen() << endl;
    }



    return 0;
}
```

Übergabe der Argumente

C+ + kennt mehrere Varianten, wie einer Funktion die Argumente übergeben werden können: call-by-value, call-by-reference und call-by-pointer.

call-by-value (Wertübergabe)

Bei call-by-value (Wertübergabe) wird der Wert des Arguments in einen Speicherbereich kopiert, auf den die Funktion mittels Parametername zugreifen kann. Ein Werteparameter verhält sich wie eine lokale Variable, die „automatisch“ mit dem richtigen Wert initialisiert wird. Der Kopiervorgang kann bei Klassen (Thema eines späteren Kapitels) einen erheblichen Zeit- und Speicheraufwand bedeuten!

Beispiel

```
#include <iostream>
#include <conio.h>

using namespace std;

//Prototyp, Deklaration
int sumW(int x,int y);   //Parameterübergabe per Wert (Value)

int main(int argc, char** argv) {
    cout << "Hello World" << endl;

    int a=5,b=6,erg=0;
    cout << "Parameteruebergabe per Wert" << endl;
    erg=sumW(a, b);
    cout << "a: " << a <<endl;
    cout << "b: " << b <<endl;

    getch();

    return 0;
}

//Funktionendefinition - Parameterübergabe per Wert
int sumW(int x,int y){
    x+=1; //x=x+1; x++; ++x;
    y+=1;

    cout << "x: " << x <<endl;
    cout << "y: " << y <<endl;

    return x+y;
```

```
}
```

Beispielhafte Speicherbelegung des Programms im Hauptspeicher:

Variable	Adresse	Wert
a	0x0001	5
b	0x0005	6
...	.	.
...	.	.
x	0x00a1	5 → 6
y	0x00a5	6 → 7

Die Ausgabe des Programms ist:

```
Hello World
Parameteruebergabe per Wert
x: 6
y: 7
a: 5
b: 6
```

call-by-reference (Übergabe per Referenz)

Sollen die von einer Funktion vorgenommen Änderungen auch für das Hauptprogramm sichtbar sein, müssen in C sogenannte Zeiger verwendet werden. C+ + stellt ebenfalls Zeiger zur Verfügung. C+ + gibt Ihnen aber auch die Möglichkeit, diese Zeiger mittels Referenzen zu umgehen, was im alten C nicht möglich war. Beide sind jedoch noch Thema eines späteren Kapitels.

Im Gegensatz zu call-by-value wird bei call-by-reference die Speicheradresse des Arguments übergeben, also der Wert nicht kopiert. Änderungen der (Referenz-)Variable betreffen zwangsläufig auch die übergebene Variable selbst und bleiben nach dem Funktionsaufruf erhalten. Um call-by-reference anzuzeigen, wird der Operator & verwendet, wie Sie gleich im Beispiel sehen werden. Wird keine Änderung des Inhalts gewünscht, sollten Sie den Referenzparameter als const deklarieren, um so den Speicherbereich vor Änderungen zu schützen. Fehler, die sich aus der ungewollten Änderung des Inhaltes einer übergebenen Referenz ergeben, sind in der Regel schwer zu finden.

```
#include <iostream>
#include <conio.h>

using namespace std;

//Prototyp, Deklaration
int sumR(int &x,int &y);   //Parameterübergabe per Referenz (Reference)


int main(int argc, char** argv) {
    cout << "Hello World" << endl;
```

```
    cout << "Parameteruebergabe per Referenz" << endl;
    a=5,b=6,erg=0;
    erg=sumR(a, b);
    cout << "a: " << a <<endl;
    cout << "b: " << b <<endl;


    getch();

    return 0;
}

//Funktionendefinition - Parameterübergabe per Referenz
int sumR(int &x,int &y){
    x+=1; //x=x+1; x++; ++x;
    y+=1;

    cout << "x: " << x <<endl;
    cout << "y: " << y <<endl;

    return x+y;
}
```

Beispielhafte Speicherbelegung des Programms im Hauptspeicher:

Variable	Adresse	Wert
x,a	0x0001	5 → 6
y,b	0x0005	6 → 7
...	...	...
...	...	...
...	...	...
...	...	...

Die Ausgabe des Programms ist:

```
Hello World
Parameteruebergabe per Referenz
x: 6
y: 7
a: 6
b: 7
```

call-by-pointer (Übergabe per Zeiger)

Wenn Sie einen Zeiger als Parameter an eine Funktion übergeben, können Sie den Wert an der übergebenen Adresse ändern.

```
#include <iostream>
#include <conio.h>

using namespace std;

//Prototyp, Deklaration
int sumZ(int *x,int *y);   //Parameterübergabe per Zeiger (Pointer)

int main(int argc, char** argv) {
    cout << "Hello World" << endl;
    cout << "Parameteruebergabe per Zeiger" << endl;
    a=5,b=6,erg=0;
    erg=sumZ(&a, &b);
    cout << "a: " << a <<endl;
    cout << "b: " << b <<endl;

    getch();

    return 0;
}


//Funktionendefinition - Parameterübergabe per Zeiger
int sumZ(int *x,int *y){
    *x+=1; //x=x+1; x++; ++x;
    *y+=1;

    cout << "x: " << *x <<endl;
    cout << "y: " << *y <<endl;

    return *x+*y;
}
```

Beispielhafte Speicherbelegung des Programms im Hauptspeicher:

Variable	Adresse	Wert
a	0x0001	5 → 6
b	0x0005	6 → 7
...	...	...
...	...	...
*x	0x00a1	0x0001
*y	0x00a5	0x0005

Die Ausgabe des Programms ist:

```
Hello World
Parameteruebergabe per Zeiger
x: 6
y: 7
```

```
a: 6
b: 7
```

Themen

- Tipp 10 (Fleissner, Mohsen)
- Geogebra 3d-Brille (David)
- 3D-Druck (Lorenz)
- Robotics (Jonas)
- Spiele (code.org , Tik Tak Toe - Scratch - Unity3d) (René und Max)
- Bildbearbeitung Photoshop - optische Täuschungen
- Joomla (Alexander und Daniel)