Table of Contents

$ arp

Intermediate

Manipulate the ARP cache

arp displays and modifies the ARP (Address Resolution Protocol) cache. ARP maps IP addresses to MAC (hardware) addresses on the local network.

The ARP cache stores recently resolved IP-to-MAC mappings to avoid repeated ARP requests. Viewing the ARP table helps identify devices on the network and...

Options & Flags

`-a`	Show all entries
`-n`	Numeric output (no DNS resolution)
`-d`	Delete an entry
`-s`	Set a static entry

Practical Examples

Example: Show ARP table

```
$ arp -an
? (192.168.1.1) at aa:bb:cc:dd:ee:ff [ether] on eth0\n? (192.168.1.50) at 11:22:33:44:55:66 [ether] on eth0
```

Shows all known IP-to-MAC mappings.

Example: Modern equivalent

```
$ ip neigh show
192.168.1.1 dev eth0 lladdr aa:bb:cc:dd:ee:ff REACHABLE
```

Shows neighbor table using the modern ip command.

Example: Clear entry

```
$ sudo arp -d 192.168.1.50
```

Removes a specific ARP cache entry, forcing re-resolution.

Example: Set static entry

```
$ sudo arp -s 192.168.1.100 00:11:22:33:44:55
```

Creates a permanent ARP mapping.

Tips & Best Practices

Note: Use ip neigh instead: ip neigh show is the modern replacement: ip neigh show, ip neigh add, ip neigh del.

Pro Tip: Identify network devices: arp -an shows MAC addresses of devices you have communicated with. Useful for identifying devices on your network.

Warning: ARP spoofing: Static ARP entries (arp -s) can protect against ARP spoofing attacks on critical gateway addresses.

$ bridge

Advanced

Linux network bridge management

The bridge command manages Ethernet bridge devices in the Linux kernel. A network bridge connects two or more network segments at the data link layer (Layer 2), making them behave as a single network. It is essential for virtualization (KVM/QEMU), container networking, and creating transparent ne...

Options & Flags

`link show`	Show bridge port configuration
`fdb show`	Show forwarding database (MAC table)
`fdb add`	Add a static FDB entry
`vlan show`	Show VLAN filtering configuration
`vlan add`	Add VLAN to a bridge port
`mdb show`	Show multicast group database
`-j`	JSON output for scripting
`monitor`	Monitor bridge events in real-time

Practical Examples

Example: Create a bridge for VMs

```
$ sudo ip link add br0 type bridge && sudo ip link set eth0 master br0 && sudo ip link set br0 up && sudo dhclient br0
```

Create bridge br0, add physical interface eth0, bring it up, and get an IP via DHCP. VMs can now attach to br0.

Example: Show bridge ports

```
$ bridge link show
```

Display all bridge ports with their state (forwarding, blocking, learning), cost, and priority.

Example: View MAC address table

```
$ bridge fdb show br0
```

Show the forwarding database - learned MAC addresses and which port they were seen on. Essential for troubleshooting.

Example: Configure VLAN filtering

```
$ sudo ip link set br0 type bridge vlan_filtering 1 && sudo bridge vlan add vid 100 dev eth0 && sudo bridge vlan add vid 100 dev v
```

Enable VLAN filtering on the bridge and assign VLAN 100 to specific ports.

Example: Add static MAC entry

```
$ sudo bridge fdb add 00:11:22:33:44:55 dev eth0 master static
```

Add a static FDB entry. Traffic for this MAC always goes to eth0 regardless of learning.

Tips & Best Practices

Pro Tip: Use nmcli for persistent bridges: For bridges that survive reboots, create them with nmcli: nmcli connection add type bridge con-name br0 ifname br0. The bridge command manages runtime state only.

Warning: Adding your uplink to a bridge: Adding your primary network interface to a bridge will temporarily disconnect you. Always configure the bridge IP first, or do it from a console/IPMI session.

Note: STP is enabled by default: Linux bridges enable STP (Spanning Tree Protocol) by default. For simple bridges with no loops, disable it: ip link set br0 type bridge stp_state 0

Pro Tip: VLAN-aware vs traditional bridges: VLAN-aware bridges (vlan_filtering 1) handle VLANs more efficiently than creating separate bridges per VLAN. Recommended for complex setups.

$ curl

Beginner

Transfer data from or to a server using various protocols

curl is a command-line tool for transferring data using various protocols including HTTP, HTTPS, FTP, SCP, and more. It is the most versatile data transfer tool in Linux, used for API testing, file downloading, web scraping, and automation.

curl supports HTTP methods (GET, POST, PUT, DELETE), cu...

Options & Flags

`-o`	Write output to file
`-O`	Save with remote filename
`-X`	Specify HTTP method
`-H`	Add custom header
`-d`	Send POST data
`-I`	Show response headers only (HEAD request)
`-L`	Follow redirects
`-s`	Silent mode (no progress bar)
`-v`	Verbose - show request/response details
`-k`	Skip SSL certificate verification
`-u`	Provide authentication credentials

Practical Examples

Example: GET request

```
$ curl https://api.example.com/users
```

Makes a simple GET request and displays the response body.

Example: POST JSON data

```
$ curl -X POST -H 'Content-Type: application/json' -d '{"name":"John","email":"john@example.com"}' https://api.example.com/users
```

Sends a JSON payload via POST request.

Example: Download a file

```
$ curl -O -L https://github.com/user/repo/archive/main.tar.gz
```

Downloads a file following redirects, saving with the remote filename.

Example: Check HTTP status

```
$ curl -s -o /dev/null -w "%{http_code}" https://example.com
200
```

Returns only the HTTP status code - perfect for monitoring scripts.

Example: Send form data

```
$ curl -X POST -F 'file=@report.pdf' -F 'name=Report' https://upload.example.com
```

Uploads a file as multipart form data.

Tips & Best Practices

Pro Tip: Pretty-print JSON: Pipe JSON responses through jq for readable output: curl -s https://api.example.com/data | jq .

Note: curl vs wget: curl is better for API testing, custom headers, and protocols. wget is better for recursive downloads and mirroring. curl outputs to stdout by default; wget saves to files.

Warning: Security with -k: Never use -k (skip SSL verification) in production scripts. It disables certificate checking and makes the connection vulnerable to man-in-the-middle attacks.

$ dig

Intermediate

DNS lookup utility for querying name servers

dig (Domain Information Groper) is a DNS lookup utility for querying DNS servers. It is the most comprehensive and widely used DNS diagnostic tool, providing detailed information about DNS records, response times, and server behavior.

dig can query any type of DNS record (A, AAAA, MX, NS, TXT, C...

Options & Flags

`+short`	Show only the answer (concise output)
`-t`	Specify record type (A, MX, NS, TXT, etc.)
`@server`	Query a specific DNS server
`+noall +answer`	Show only the answer section
`-x`	Reverse DNS lookup (IP to hostname)
`+trace`	Trace DNS delegation from root servers
`+nssearch`	Find authoritative nameservers

Practical Examples

Example: Basic DNS lookup

```
$ dig example.com
```

Shows the A record (IP address) with full DNS response details.

Example: Quick IP lookup

```
$ dig +short example.com
93.184.216.34
```

Returns just the IP address without any extra information.

Example: Look up MX records

```
$ dig -t MX gmail.com +short
5 gmail-smtp-in.l.google.com.\n10 alt1.gmail-smtp-in.l.google.com.
```

Shows mail exchange records - essential for email troubleshooting.

Example: Query specific DNS server

```
$ dig @8.8.8.8 example.com
```

Queries Google's public DNS instead of your default resolver.

Example: Check TXT records (SPF/DKIM)

```
$ dig -t TXT example.com +short
```

Shows TXT records including SPF, DKIM, and domain verification entries.

Tips & Best Practices

Pro Tip: Check DNS propagation: Query different DNS servers to check propagation: dig @8.8.8.8 domain.com vs dig @1.1.1.1 domain.com. Different results mean propagation is incomplete.

Note: dig vs nslookup: dig provides more detailed output and is preferred by professionals. nslookup is simpler but deprecated on some systems. dig is the standard DNS diagnostic tool.

Warning: Cached results: Your local resolver caches DNS responses. Query external servers (8.8.8.8, 1.1.1.1) to bypass local caching when testing changes.

$ ethtool

Advanced

Display or change ethernet device settings

ethtool displays and modifies Ethernet device settings. It shows link speed, duplex mode, auto-negotiation status, driver information, and hardware features for network interfaces.

ethtool is essential for diagnosing network performance issues, verifying link speed, checking cable connectivity (...

Options & Flags

`interface`	Show interface settings
`-i`	Show driver information
`-S`	Show NIC statistics
`-s`	Change settings
`-k`	Show offload features
`-p`	Blink LED (identify physical port)

Practical Examples

Example: Show interface info

```
$ ethtool eth0
Speed: 1000Mb/s
Duplex: Full
Link detected: yes
```

Shows link speed, duplex, auto-negotiation, and link status.

Example: Driver info

```
$ ethtool -i eth0
driver: e1000e\nversion: 3.8.7-k
```

Shows the kernel driver, version, and firmware for the NIC.

Example: NIC statistics

```
$ ethtool -S eth0 | head -20
```

Shows detailed NIC statistics including errors and drops.

Example: Blink port LED

```
$ sudo ethtool -p eth0 10
```

Blinks the port LED for 10 seconds to identify the physical NIC.

Example: Check link

```
$ ethtool eth0 | grep "Link detected"
Link detected: yes
```

Quick check if a cable is connected and link is active.

Tips & Best Practices

Pro Tip: Check link speed: ethtool eth0 | grep Speed quickly shows if you are running at 100M, 1G, or 10G. Mismatched speed = performance issue.

Note: Blink to identify: ethtool -p eth0 10 blinks the NIC LED for 10 seconds. Essential when identifying which physical port corresponds to which interface.

Warning: Requires kernel support: Not all virtual NICs support ethtool. Cloud instances may show limited information.

$ host

Beginner

DNS lookup utility (simple)

host is a simple DNS lookup utility that converts domain names to IP addresses and vice versa. It is simpler and more concise than dig or nslookup for basic DNS queries.

host supports all standard DNS record types and can query specific name servers. Its output is clean and human-readable, makin...

Options & Flags

`domain`	Look up A record
`-t`	Specify record type
`-a`	Show all DNS records
`-v`	Verbose output
`server`	Use specific DNS server

Practical Examples

Example: Basic lookup

```
$ host example.com
example.com has address 93.184.216.34\nexample.com has IPv6 address 2606:2800:220:1:248:1893:25c8:1946
```

Shows IP addresses for the domain.

Example: MX records

```
$ host -t MX example.com
example.com mail is handled by 10 mail.example.com
```

Shows mail server records.

Example: Reverse lookup

```
$ host 8.8.8.8
8.8.8.8.in-addr.arpa domain name pointer dns.google
```

Finds domain name for an IP.

Example: All records

```
$ host -a example.com
```

Shows all DNS record types for the domain.

Example: TXT records

```
$ host -t TXT example.com
```

Shows TXT records (SPF, verification, etc.).

Tips & Best Practices

Pro Tip: Simplest DNS tool: host gives the cleanest output for quick DNS checks. Use dig for detailed debugging, host for quick answers.

Note: Check DNS propagation: Query multiple DNS servers to check propagation: host domain 8.8.8.8, host domain 1.1.1.1.

Warning: May not be installed: host is part of bind-utils/dnsutils package. Install with: apt install dnsutils or yum install bind-utils.

$ ifconfig

Beginner

Configure or display network interface parameters (legacy)

ifconfig displays or configures network interfaces. It shows IP addresses, MAC addresses, MTU, and traffic statistics. While ifconfig has been the standard for decades, the ip command is its modern replacement.

ifconfig is part of the net-tools package and is gradually being replaced by the ip c...

Options & Flags

`(no args)`	Show all active interfaces
`-a`	Show all interfaces (including down)
`interface up/down`	Enable/disable interface
`interface IP`	Assign IP address
`interface netmask`	Set subnet mask

Practical Examples

Example: Show interfaces

```
$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>
    inet 192.168.1.100  netmask 255.255.255.0
    ether aa:bb:cc:dd:ee:ff
```

Displays all active network interfaces.

Example: Specific interface

```
$ ifconfig eth0
```

Shows details for eth0 only.

Example: Get IP address

```
$ ifconfig eth0 | grep 'inet ' | awk '{print $2}'
192.168.1.100
```

Extracts just the IP address.

Example: Bring interface down

```
$ sudo ifconfig eth0 down
```

Disables the network interface.

Example: Set IP address

```
$ sudo ifconfig eth0 192.168.1.200 netmask 255.255.255.0 up
```

Assigns IP address and brings interface up.

Tips & Best Practices

Note: ip command is the replacement: ifconfig ? ip addr show. ifconfig eth0 down ? ip link set eth0 down. ip is more powerful and actively maintained.

Pro Tip: Quick IP check: hostname -I is faster than ifconfig for just getting IP addresses.

Warning: Changes are temporary: ifconfig changes are lost on reboot. Use /etc/network/interfaces or NetworkManager for permanent configuration.

$ ip

Intermediate

Show and manipulate network interfaces, routing, and tunnels

The ip command is the modern replacement for ifconfig, route, and arp. It is the primary tool for managing network interfaces, IP addresses, routing tables, and ARP entries in modern Linux systems.

ip is part of the iproute2 package and provides a unified interface for all network configuration ...

Options & Flags

`addr`	Show or manipulate IP addresses
`link`	Show or manipulate network interfaces
`route`	Show or manipulate routing table
`neigh`	Show or manipulate ARP/neighbor entries
`-c`	Colorized output
`-br`	Brief output format
`-4/-6`	Show only IPv4 or IPv6
`-s`	Show statistics

Practical Examples

Example: Show all IP addresses

```
$ ip addr show
```

Shows all network interfaces with their IP addresses.

Example: Brief address listing

```
$ ip -br -c addr show
lo       UNKNOWN  127.0.0.1/8\neth0     UP       192.168.1.100/24
```

Compact, colorized view of all interfaces and their IPs.

Example: Show routing table

```
$ ip route show
default via 192.168.1.1 dev eth0\n192.168.1.0/24 dev eth0 proto kernel
```

Displays the current routing table with default gateway.

Example: Add an IP address

```
$ sudo ip addr add 192.168.1.200/24 dev eth0
```

Adds a secondary IP address to eth0.

Example: Bring interface up/down

```
$ sudo ip link set eth0 up
```

Enables the eth0 network interface.

Tips & Best Practices

Pro Tip: Use -br for clean output: ip -br addr show gives a much cleaner output than the default. Combined with -c for colors, it is the quickest way to check network status.

Warning: Changes are not persistent: ip command changes are temporary and lost on reboot. For persistent configuration, edit /etc/network/interfaces, /etc/netplan/, or use nmcli.

Note: ip replaces ifconfig: ifconfig is deprecated. ip addr replaces ifconfig, ip route replaces route, ip neigh replaces arp. Learn ip as the modern standard.

$ ip route

Intermediate

Show and manipulate the routing table

ip route displays and manages the kernel routing table. It shows how the system decides where to send network packets - which interface and gateway to use for each destination network.

The routing table is fundamental to networking. Every packet sent by the system is matched against routing rule...

Options & Flags

`show`	Display routing table
`get`	Show route for specific destination
`add`	Add a route
`del`	Delete a route
`replace`	Replace or add a route

Practical Examples

Example: Show routing table

```
$ ip route show
default via 192.168.1.1 dev eth0
192.168.1.0/24 dev eth0 proto kernel scope link src 192.168.1.100
10.0.0.0/8 via 192.168.1.254 dev eth0
```

Displays all routes in the routing table.

Example: Check route for destination

```
$ ip route get 8.8.8.8
8.8.8.8 via 192.168.1.1 dev eth0 src 192.168.1.100
```

Shows exactly which route would be used to reach 8.8.8.8.

Example: Add static route

```
$ sudo ip route add 10.0.0.0/8 via 192.168.1.254
```

Routes all 10.x.x.x traffic through 192.168.1.254.

Example: Change default gateway

```
$ sudo ip route replace default via 192.168.1.1 dev eth0
```

Sets the default gateway.

Example: Delete route

```
$ sudo ip route del 10.0.0.0/8
```

Removes a static route.

Tips & Best Practices

Pro Tip: ip route get is very useful: ip route get IP shows exactly which route, gateway, and interface will be used. Great for debugging.

Warning: Not persistent: ip route changes are lost on reboot. For permanent routes, configure /etc/netplan/ or /etc/network/interfaces.

Note: Replaces route command: ip route replaces the legacy route command. route ? ip route show. route add ? ip route add.

$ iperf3

Intermediate

Network bandwidth measurement tool

The iperf3 command is the industry-standard tool for measuring network bandwidth, latency, jitter, and packet loss between two endpoints. It generates TCP or UDP traffic at maximum or controlled rates and reports detailed performance statistics.

iperf3 works in a client-server model: one host ru...

Options & Flags

`-s`	Run as server (listen for connections)
`-c SERVER`	Run as client, connect to server
`-p PORT`	Set server port (default: 5201)
`-t SECONDS`	Test duration (default: 10 seconds)
`-P N`	Number of parallel streams
`-R`	Reverse mode (server sends, client receives)
`-u`	Use UDP instead of TCP
`-b RATE`	Target bandwidth (UDP default: 1Mbps)
`-J`	Output in JSON format
`--bidir`	Bidirectional test (simultaneous send/receive)

Practical Examples

Example: Start iperf3 server

```
$ iperf3 -s
Server listening on 5201
```

Listen on port 5201 for incoming test connections. Keep running until terminated.

Example: Basic TCP bandwidth test

```
$ iperf3 -c 10.0.0.1 -t 10
```

Run a 10-second TCP throughput test. Reports bandwidth, retransmissions, and congestion window size.

Example: Test with parallel streams

```
$ iperf3 -c 10.0.0.1 -P 4 -t 30
```

Use 4 parallel TCP streams for 30 seconds. Multiple streams can saturate high-bandwidth links better.

Example: UDP test with packet loss

```
$ iperf3 -c 10.0.0.1 -u -b 100M -t 10
```

Send 100 Mbps of UDP traffic. Reports jitter, packet loss percentage, and out-of-order packets.

Example: Test download speed (reverse)

```
$ iperf3 -c 10.0.0.1 -R
```

Reverse mode: server sends to client. Tests download bandwidth instead of upload.

Tips & Best Practices

Pro Tip: Open firewall for iperf3: Allow port 5201/tcp on the server: sudo ufw allow 5201/tcp or sudo firewall-cmd --add-port=5201/tcp. Remember to close it after testing.

Warning: iperf3 uses significant bandwidth: iperf3 will saturate the link by default. Run tests during maintenance windows or use -b to limit bandwidth. Avoid running on production networks during peak hours.

Note: TCP vs UDP testing: TCP tests measure maximum achievable throughput. UDP tests measure jitter and packet loss at a specified rate. Use TCP for bandwidth measurement, UDP for VoIP/video quality assessment.

Pro Tip: Multiple test runs: Run tests multiple times and at different times of day. Single tests can be misleading due to network congestion, CPU load, or temporary conditions.