Cheat Sheet

```
Unset
# All examples assume "eth0" as ethernet device name and "wlan0" for wifi.
# When a command says "permission denied" put "sudo" in front
#
# List, add to and remove from address tables
ip -c a
ip a add 10.42.43.128/24 dev eth0
ip a del 10.42.43.128/24 dev eth0

# List, add to and remove from routing table
ip -c r
ip r add 10.42.43.0/24 dev eth0
ip r del 10.42.43.0/24 dev eth0
ip r add default via 10.42.43.1 dev eth0  # Default gateway is 10.42.43.1

# DNS setup
resolvectl dns               # List
resolvectl dns eth0 9.9.9.9  # Set DNS server as 9.9.9.9

# Testing connections
ping 10.42.43.128  # Ping another local computer
ping 9.9.9.9       # Ping internet
curl google.com    # Check if DNS works
iperf3 --client 10.42.43.1 --port 9024  # And "iperf3 -sp 9024" on 10.42.43.1
nmap -sn 10.42.43.1/24  # List all clients on the LAN
```

Learn Networking the Hard Way

The software team at ARVP relies on an extensive set of services and connections to get things done. Other subteams often need to interact with this network as well, which means software must make connections accessible to those with limited networking experience.

This guide provides a comprehensive tutorial on learning practical networking. Traditional networking courses start with the TCP/IP 7-layer model and the more “electrical” fundamentals. Here we skip all of that. Our focus is to teach you how to reproducibly setup, debug, and take full advantage of networking capabilities.

We focus on a hands-on learning experience, so you'll be ready to deal with real networks. That also means you'll need real networking equipment. Generally, this means computers with Linux and some basic networking equipment. The non-computer equipment shouldn't cost more than $30. Each lesson will begin by listing the required equipment, along with an estimated timeframe for how long the lesson should take. Lessons often build on each other, so it's recommended you go through these sequentially.

I'm proud to say this guide is extremely comprehensive, exceeding anything available online by cutting to the chase and teaching networking from the bottom up. However, you're signing up to learn the hard way. Unless you're strongly proficient at Linux, it's recommended you ask someone who is to go through this guide with you. Having a mentor for this will make it feel much easier.

1 - Networking Background

Equipment Requirements: None
Estimated Time: 10 minutes

Before we get into the real lessons, let's look at some background. Networking is a way to send signals between computers. Some core typical networking equipment includes:

- **Ethernet cables**: These come in different grades of CAT. CAT5e and up support 1gbps for a distance of at least 100m. CAT6 is currently the standard, with CAT7 and CAT8 only being useful in data centers. You can easily cut these cables and crimp them to any length you want, unlike more complex standards like Thunderbolt and USB.
- **Network Switch**: Imagine taking a single ethernet cable and splitting it in several forks, all of which are sending the same signal. That's what an unmanaged switch does. They're usually quite cheap, around $10-20.
- **WIFI Access Point**: Similar to a switch, but instead of plugging in ethernet cables, you connect using a WIFI receiver.
- **Router**: Broadly speaking, this is any computer with a routing table. We will be using a Linux computer as a router. As stand-alone devices, these are often sold with a WIFI Access Point built into them
- **Modem**: Typically a router that has a connection to Fiber or Coaxial. These are only required when an ISP gives you internet access and in data centers. We won't be using one.

There are currently two standards for IP. IPv4 and IPv6. IPv4 was invented in the early 1980s, almost a decade before the web and well before personal computing became the norm. IPv4 uses 32 bits for an address, giving 2^32 (about 4.3 billion) addresses. Each computer was meant to have exactly one address on Earth, that way we can identify that computer over the internet. However, around the mid 90s, networking professionals noticed a huge problem: the sudden uptick of personal computing meant an increase in the number of computers. They

noticed we will run out of IPv4 addresses in a few decades. This happened earlier than expected, thanks to the iPhone, in 2010.

To alleviate this IPv4 address space exhaustion, two methods were proposed in 1996. NAT and IPv6. NAT was intended as a short-term bandage on the problem, essentially making each address up to 65k addresses by appending the port number to the packet and counting on the router to figure things out. IPv6 was the real solution, which extended the address space to 2^128 addresses (essentially infinity). Due to some geopolitical reasons, western countries ended up with *far* more IPv4 addresses per person than eastern ones. This has led to an odd situation, where the east is running almost entirely using IPv6 and most western ISPs still lack IPv6 support outside their premium plans. Unfortunately, this guide is written from a western perspective, so we'll focus on using NAT with IPv4, as that's much more practical here currently. That said, most of the concepts translate pretty directly to IPv6, so you won't be missing out.

IPv4 address example: `142.250.217.78`
IPv6 address example: `[2607:f8b0:400a:80a::200e]`

An IPv4 address consists of 4 numbers from 0-255 inclusive. That's 8 bits per number, making 32 bits. IPv6 uses 8 4-digit hexadecimal numbers, with and `::` being filled with zeros. Both addresses are sometimes appended with a `/24` or `/64`. This delimits the subnet, which is your local network. For example, `10.42.43.1/24` delimits a 24-bit subnet. That means we have 32 - 24 = 8 bits for devices on the local networking, meaning we can have 256 devices on the local network. You might see odd numbers like `10.42.43.20/17` which is a 17 bit subnet, allowing 2^(32 - 17) devices to be on the local network.

2 - Address and Routing Tables

Equipment Requirements:

- Two Linux laptops/desktops, each with an ethernet port OR a USB to ethernet adapter. This will be easier if both computers are running a systemd-based distro.
- An unmanaged switch (like this)
- Two ethernet cables

Estimated Time: 70 minutes

Networks send packets which include the IP address of where they're trying to get to and some data. Routers then figure out where to send each incoming packet. There are 2 tables involved in this process that we'll be setting up here today:

- **The Address Table**: This contains the IP and MAC addresses of each interface on your computer. This includes wireless and wired interfaces. Note that a single interface can have multiple IPs and multiple interfaces can have the same IPs.
- **The Routing Table**: This specifies what interface is connected to what subnet. Specifically, if an incoming packet attempts to reach another computer on one of the subnets of one of your interfaces, the routing table directs the packet to that interface.

Start by booting up both computers. Run `ip a`. If your interfaces have IPv4 addresses, you'll need to disable your network manager and reboot. You'll be able to check by running `ip a` and getting an address table without IPv4 addresses. Some common ones to disable include:

```
Unset
systemctl disable NetworkManager.service
systemctl disable systemd-networkd.service

# Then reboot
systemctl reboot
```

Connect both computers to the switch. In `ip a`, you'll need to identify which interface you just connected. Often this will be named something like `eth0` for built-in interfaces, and `enp0s1u1` for USB adapters.

`ip a` is showing us the address table (it's short for `ip address`). Add an IPv4 address to both on the same subnet to the connected interfaces on both computers. For example, for subnet `10.42.43.0/24` we can add:

```
#On computer A
ip a add 10.42.43.10/24 dev eth0
#On computer B
ip a add 10.42.43.128/24 dev eth0
```

This may or may not add an entry to the routing table. Check with `ip r` which is short for `ip route`. You should see any entry like `10.42.43.10/24 dev eth0` in both tables. If not, manually add the route:

```
ip r add 10.42.43.0/24 dev eth0
```

Remember that since our subnet is 24 bits, the last 8 bits in the route are irrelevant. This will route the same way if we put `10.42.43.111/24`. At this point, we've theoretically connected the two computers. Next up we test the connection.

3 - Testing Connections

Equipment Requirements: Same equipment as from lesson 2

- make sure `curl`, `iperf3`, `nmap` are installed

Estimated Time: 30 minutes

We have multiple tools to diagnose different aspects of the network connection. Here we cover all of them, despite our network not being setup.

`ping` is a basic tool invented in the 1960s and bundled with virtually every operating system today, including Windows. It uses the ICMP protocol as opposed to the TCP protocol everything else uses. This means in special scenarios, especially during virtualization, `ping` can fail despite there being a network connecting. That said, it's usually the best first choice for testing.

Try pinging computer B from computer A. Assuming computer B has IP `10.42.43.128` we run the following from computer A:

```
Unset
ping 10.42.43.128
```

If this is not reporting a connection, it's time to check your routing and address tables from lesson 2. Computer B should also be able to ping computer A in a similar manner.

OpenSSH, usually referred to as `ssh`, is a tool for logging into remote computers. It uses TCP, but requires a daemon running on the "server" computer. Start by enabling the SSH daemon on computer B, then attempt to connect from computer A:

```
Unset
systemctl start sshd.service
ssh <username>@10.42.43.128
```

Depending on the SSH daemon's setup, you may be denied authentication, but getting that far implies a connection between the two systems. Try the same steps to connect from computer B to computer A.

`iperf3` is a tool built to test maximum throughput, often referred to as the bandwidth of a network. It's popular for testing Ethernet cables. Like SSH, it operates on a server-client model. Enable the server on one laptop and connect with another:

```
Unset
iperf3 --server --port 9004
iperf3 --client 10.42.43.128 --port 9004
```

Assuming you have a typical 1gbps switch and CAT6 cables, you'll see around 940mbps reported by `iperf3`.

`nmap` is a somewhat dated tool that can be used to discover al, the computers on a given subnet. This can be helpful if you don't have access to the address table of a computer, but still want to connect to it. Run the following to scan the subnet:

```
Unset
nmap -sn 10.42.43.1/24
```

`curl` is a widely available tool to test your TCP and DNS capabilities. If `curl` works, you can expect browsers and most apps to work with the network. Currently we don't have an internet connection, so this will fail, but you'll need this to test later:

```
Unset
curl google.com
```

4 - Internet Forwarding

Equipment Requirements: Same equipment as from lessons 2 and 3

- A hotspot or wifi network with internet access
- make sure `nftables` is installed (call with `nft`)

Estimated Time: 60 minutes

Currently computers A and B can talk with each other. However, neither can communicate with any other computer. Here, we'll set up computer A as a router with internet access, and use NAT to give computer B internet access through the local network we've set up.

To achieve this, we'll need to set up firewall rules. `iptables` was traditionally the choice of low-level firewall control, but it's considered legacy. We will use `nftables` instead, which is the successor to `iptables`. Start up the daemon with `systemctl start nftables.service` on computer A.

Start by connecting computer A to the internet. You should be able to run the `curl` command from lesson 3 on this computer now. Make sure it's still connected to the internal network (this requires 2 interfaces).

Enable forwarding in the kernel:

```
Unset
sysctl -w net.ipv4.ip_forward=1
```

Set up a forwarding table:

```
Unset
# Make a new table
nft add table inet forwarding-table

# Put in rules to the table. It will now do NAT
nft add chain inet forwarding-table postrouting { type nat hook postrouting
priority 100 \; }
nft add rule inet forwarding-table postrouting -o "$WANIF" counter masquerade

# Forward packets from internal -> external network and in reverse
nft add chain inet filter forward { type filter hook forward priority 0 \; }
nft add rule inet filter forward ct state {established, related} accept
nft add rule inet filter forward iifname "$LANIF" oifname "$WANIF" accept
nft add rule inet filter forward iifname "$WANIF" oifname "$LANIF" accept
```

Now computer A is acting as a router. We need computer B to contact computer A when it wants internet access. Set computer A as the default gateway, for example:

```
Unset
ip r add default via 10.42.43.10 dev eth0
```

Now you should be able to `ping 1.1` and `ping 8.8.8.8` from computer B.

5 - WIFI and DNS

6 - DHCP

DHCP provides the zero-conf experience most people expect when connecting to the internet. DHCP will automatically respond with an IP address, default gateway, and DNS servers for your computer to use on this network.

7 - Network Managers

- Systemd-networkd
- Iwd
- NetworkManager
- MacOS and Windows basics

9 - Alternative Connection Methods (VPN, SDWAN…)

Equipment Requirements: Same equipment as from lessons 2 and 3

- Access to your internet router
- Wireguard installed or dockerized
- Nebula
- Zerotier installed
- Tailscale installed
- Rathole installed
- Caddy installed
- A DigitalOcean droplet (smallest size will be fine)

Estimated Time: 180 minutes

So far, we've always assumed our "server" computer has a global static IPv4. This is quite rarely the case, especially as most non-commercial internet plans do not include a static IPv4 address. In this section we'll extensively dive into alternative ways to set up connections on the internet.

VPN over SSH

A Virtual Private Network (VPN) simulates a local network on a larger network, usually the internet. It acts as if all computers are on a LAN, despite them being connected over a WAN. This can be achieved through numerous software tricks, the easiest being SSH.

VPN

A traditional VPN can be set up using Wireguard. This requires one central server to have a static global IPv4 address, to which all clients connect to. The server acts as a router for the entire VPN over WAN.

- VPN (Wireguard, Netmaker)
- Mesh VPN (Nebula, tailscale?)
- SDWAN (Zerotier, tailscale?)
- Reverse Proxies (rathole, Caddy)

Software-Driven WAN

All the solutions above still require one central server to act as a router for connections. If the server goes down, the entire VPN is down. This can be (almost) overcome using SDWAN, which uses peer-to-peer connections for decreased latency and independence of a central router.

Reverse Proxies

All the solutions above continue to require special software installed on all clients. This is a terrible option if you need non-technical people to easily connect to your network. While reverse proxies don't strictly create local networks, they are the standard for public servers and clouds.

A reverse proxy is a computer that routes traffic to other computers. They're important for load-balancing and security

10 - Firewalld

OLD STUFF

Here we'll look into setting up a local network, similarly to what we do at pool tests.

You will need:
- Two computers running Linux, with ethernet ports or adapters
- A network switch
- Two ethernet cables

```
Unset
ip link set eth0 up
ip r add 10.42.43.0/24 dev eth0
ip r del 10.42.43.0/24 dev eth0
ip r add default via 10.42.43.99 dev eth0
ip r del default via 10.42.43.99 dev eth0
```

```
resolvectl dns
resolvectl dns eth0 9.9.9.10

apt update && apt install iwd
systemctl start iwd
iwctl station wlan0 scan
iwctl station wlan0 get-networks
iwctl station wlan0 connect fishy-net
```

We use the `10.42.43.1/24` ip address range for all our local networks. We currently only use ipv4, with no ipv6 support at all. Ip addresses are tracked on the Orca Network spreadsheet.

Once both computers are connected to the switch, you'll need to manually assign them both ip addresses. First, identify the ethernet interface with `ip a`. Then assign a static ip to that interface with something like:

```
Unset
sudo ip a add 10.42.43.1/24 dev eth0
```

Ensure each computer can ping the other. You can also identify all ip addresses on the local network with:

```
Unset
sudo nmap -sn 10.42.43.1/24
```

Remember, each interface can have multiple IP addresses.

Internet forwarding allows one computer with an external internet to give all other devices on the local network internet. We use this for all our networks, but in this specific example, we'll be reviewing how we set it up for pool tests. This means we'll be forwarding data from a phone to the entire local network.

You will need:

- Everything from the Local Network Setup lesson
- A phone (preferably iOS) with data sharing enabled
- A cable to connect your phone to your computer, or a hotspot
- A copy of the at_everything repository from Gitlab

This picks up where the previous lesson left off, with computers A and B connected to a switch and able to ping each other.

Connect the phone to computer A. Make sure data sharing is working with `ping 1.1` or any other method. Make sure computer A has the ip address `10.42.43.1`.

Run `./scripts/forward_internet.sh` from the at_everything repository. It will tell you to name your WAN and LAN interface. Find these using `ip a` and `ip r`. The WAN interface is whatever is connected to the phone. The LAN interface is the one with the `10.42.43.1` ip address.

Now rerun the script with these interfaces. For example, if my WAN interface is `wlan0` and my LAN interface is `eth0`, then we run:

```
Unset
./scripts/forward_internet.sh wlan0 eth0
```

Now check that computer B can connect to the internet. If not, debug.

Why does computer B not need to set a gateway??? Something is wrong about this section These are both conveniences we need, especially to make sure we don't run out of ethernet cables/ports and ease of access for non-software team members. Warning, this lesson is much harder than the previous ones.

5 - Reverse Proxies

6 - Peer2Peer Connections

7 - Firewalld

8 - Orca's Services