Learning How to Learn

Core Principles

1. **Meta-Learning > Rote Learning**
   - Focus on *how* to learn before *what* to learn.
   - **Example**: The "3D cake cut" puzzle teaches:
     - Default thinking = 2D → fails
     - **Solution**: Visualize vertically (3D) → 8 equal slices with 3 cuts
2. **SMART Goals** (Task 1)
   - **Specific, Measurable, Achievable, Relevant, Time-bound**
   - *Cybersecurity example*:
     - "~~Learn hacking~~" → "Master SQLi via PortSwigger labs by Month 3"
3. **80/20 Rule (Pareto Principle)**
   - 20% effort → 80% results (e.g., learn core OWASP Top 10 vs. every vuln)

Learning Techniques

Technique	Retention Rate	How-To
Passive (Video)	20%	Watch → Take notes
Active (Lab)	75%	Do labs → Discuss mistakes
Feynman (Task 2)	90%	Teach imaginary student → Gaps reveal weak points

Productivity Hacks

- **Pomodoro**: 25m focus (zero distractions) → 5m break
- **Note-Taking Apps**: **Obsidian** (linked notes) > linear docs
- **Practice**: Re-do labs → Add 1 new vuln variant each time

Asking Better Questions (Task 4)

- **Bad**: "How to hack?"
- **Good**: "How to exploit FTP anonymous login on X server?"
- **Pro Tip**: Use ChatGPT to refine questions ("Act as a pentester...")

Common Pitfalls

- **Illusion of Competence**: Environment shapes thinking (e.g., 2D cake trap)
- **Distractions**: Phone = #1 focus killer.
- **Overlearning**: You *don't* need 100% mastery to start.

Internet and Networking Basics

Why this matters for cybersecurity:

- **Attack Surface**: 90% of breaches start via networks (phishing, MITM, misconfigs).
- **Defense**: Firewalls, IDS, and encryption operate at specific network layers (OSI Model).
- **Hacker POV**: You must understand traffic flow to exploit/protect systems.

Core Concepts (Linked)

1. # Hosts and IP Addresses

- **Host**: Any device with an IP (PC, server, IoT).
- **IP Address**:
  - IPv4: `192.168.1.1` (32-bit, e.g., private ranges `10.0.0.0/8`, `172.16.0.0/12`)
  - IPv6: `2001:0db8:85a3::` (128-bit, solves IPv4 exhaustion)
- **Key for Pentesting**:
  - `nmap -sn 192.168.1.0/24` → Discover live hosts.
  - Private IPs indicate internal network targets.

2. Network Devices → Hubs, Switches, Routers
3. OSI Model Explained (Layer-by-Layer)
4. Protocols Examples → DNS, DHCP, HTTP/S, etc.
5. Data Flow Step-by-Step (Interview Prep)

Subnetting Guide (Linked)

Subnetting Guide

What is Subnetting?

Subnetting is the method of dividing a single IP network into multiple smaller logical networks, called subnets. It's used to:

- Improve network performance by reducing broadcast traffic.
- Enhance security by isolating segments.
- Optimize IP address usage and management.
- Support custom network topologies.

Instead of wasting a large range of IPs on a small group of devices, subnetting helps you slice and allocate just enough for each group.

Understanding IP Addresses

An IPv4 address is 32 bits long, split into four 8-bit sections (octets), usually written in decimal:

Example:
192.168.10.5 → binary: 11000000.10101000.00001010.00000101

Every IP address has two parts:

- **Network portion**: Identifies the subnet the device belongs to.
- **Host portion**: Identifies the individual device in that subnet.

The split between those two is controlled by the **subnet mask** or **CIDR prefix**.

What is a Subnet Mask?

A subnet mask tells us **how many bits are used for the network** and **how many are left for hosts**.

For example, a subnet mask of:

- 255.255.255.0 → 24 bits for the network → `/24` in CIDR notation.
- 255.255.255.192 → 26 bits for the network → `/26` in CIDR.

The higher the CIDR value, the **smaller** the subnet (fewer hosts, more subnets).

Key Formulas

- **Usable hosts** = 2^(32 - subnet bits) - 2
  (We subtract 2: one for the network address and one for broadcast.)
- **Block size** = 256 - value of the subnet mask octet where subnetting happens
  (Used to find the step between each subnet.)

Subnetting Step-by-Step

Goal: Subnet 192.168.10.0/24 into smaller networks

1. **Decide how many subnets or hosts you need.**

   Example: Need 4 subnets? → 2^2 = 4 → Borrow 2 bits.
2. **Borrow bits from the host portion.**

   From /24, borrow 2 bits → /26 (i.e. now 26 bits are for the network).
3. **Calculate new subnet mask.**

   /26 → 255.255.255.192
4. **Calculate block size.**

   256 - 192 = 64 → Each subnet increases by 64.
5. **List the subnets.**

Subnet	Network Address	Host Range	Broadcast Address
1	192.168.10.0	192.168.10.1 - 192.168.10.62	192.168.10.63
2	192.168.10.64	192.168.10.65 - 192.168.10.126	192.168.10.127
3	192.168.10.128	192.168.10.129 - 192.168.10.190	192.168.10.191
4	192.168.10.192	192.168.10.193 - 192.168.10.254	192.168.10.255

```
Usable hosts per subnet = 2^(6) - 2 = 62
```

More Examples

Example 1: Subnet 10.0.0.0/24 into 8 subnets

- 8 subnets = 2^3 → borrow 3 bits → /27
- /27 = 255.255.255.224 → block size = 32
- Subnets:
  - 10.0.0.0 → 10.0.0.1 - 10.0.0.30 → broadcast 10.0.0.31
  - 10.0.0.32 → 10.0.0.33 - 10.0.0.62 → broadcast 10.0.0.63
  - ... and so on

Example 2: Find the subnet and host range of 192.168.1.77/28

- /28 = 255.255.255.240 → block size = 16
- Subnets: 192.168.1.0, 192.168.1.16, ..., 192.168.1.112
- 77 falls in the subnet starting at 192.168.1.64
- Host range: 192.168.1.65 - 192.168.1.78
- Broadcast: 192.168.1.79

CIDR & Subnet Mask Cheat Sheet

CIDR	Subnet Mask	Hosts	Block Size
/30	255.255.255.252	2	4
/29	255.255.255.248	6	8
/28	255.255.255.240	14	16
/27	255.255.255.224	30	32
/26	255.255.255.192	62	64
/25	255.255.255.128	126	128
/24	255.255.255.0	254	256
/23	255.255.254.0	510	512
/22	255.255.252.0	1022	1024

Tools You Can Use

- `ipcalc` (Linux CLI)
- `sipcalc`

- Online calculators: [Subnet Calculator (Online)](), cidr.xyz
- Practice manually for exams or certs like CCNA

Quick Tips

- Always subtract 2 from total hosts (network + broadcast).
- If subnetting feels hard, remember: it's all just binary math.
- Get comfortable converting between decimal and binary.
- Block size helps you quickly jump between subnet ranges.

Operating System Fundamentals

Quick Note

Sorry to all my windows friends, but for now I will be skipping some of this part because:

- I am not a windows user.
- I am speedrunning making those notes.
- If I need to do windows attacks or anything like that later I can google the commands I need.
  That's why for now do not expect a cover up of neither the CMD (command prompt), nor the windows power shell.
  Feel free to do this research on your own, things you'll probably need to know:

1. The windows task manager
2. The windows command prompt
3. The windows power shell
4. Windows privilege and permissions
5. Windows Defender
6. Firewall usage and making exceptions

   > This is a list that I made without looking into the things that you will actually need so keep in mind that this list might be missing some stuff.

Linux

What is Linux?

Linux is an open-source **operating system kernel** — the core program that interacts directly with your computer's hardware. But when most people say *Linux*, they’re actually referring to **Linux-based operating systems** (called **distributions** or **distros**) like:

- Ubuntu
- Fedora
- Arch Linux
- Debian
- Kali Linux

Each distro bundles the Linux kernel with tools, utilities, a shell (like `bash` or `zsh`), a package manager, and sometimes a graphical interface.

Myth: “Linux Commands”

There is no such thing as *"Linux commands"* in the sense that the commands you run aren't built into "Linux" itself.

When you open a terminal and type a command, you're actually doing one of the following:

1. Running an **external tool**

These are programs installed on your system, found in places like `/bin`, `/usr/bin`, etc.

Examples:

```
ls      # list directory contents (external tool)
cp      # copy files (external tool)
mv      # move/rename files (external tool)
```

2. Using a **shell built-in**

Some commands are built into the **shell** you're using (e.g., `bash`, `zsh`).

Examples:

```
cd      # change directory (built into the shell)
```

```
echo    # display text (also often a shell built-in)
```

3. Using **kernel syscalls** indirectly

The Linux **kernel** manages things like filesystems, processes, and memory. The commands you run interact *with the kernel* through tools or shell commands.

Change of Perspective

So when people say "learning Linux commands", what they’re really learning is:

- How to use **tools** available in a Linux environment
- How to **navigate the filesystem** using a shell
- How to **combine tools** to accomplish tasks efficiently

Introduction to Linux Navigation and File Creation

You interact with your Linux system using the **terminal**, a text interface that lets you issue commands.

Navigating the Filesystem

Think of Linux’s filesystem like a tree:

```
/
├── home/
│   └── you/
├── etc/
├── bin/
├── var/
```

The root of the filesystem is `/`. All files and folders are underneath this root.

Common Navigation Commands

Command	Purpose
`pwd`	Print current directory
`cd`	Change directory
`ls`	List contents of a directory
`ls -l`	Long listing with permissions and sizes
`ls -a`	Show hidden files (start with `.`)

Examples:

```
cd /home/you        # Go to your home folder
cd ..               # Go one directory up
ls -la              # List all files in long format
```

File and Directory Creation

Create Directories

```
mkdir new_folder
mkdir -p folder/subfolder  # create nested folders

```

Create Files

```
touch file.txt             # create empty file
echo "Hello" > file.txt    # create file with content
vim file.txt              # open file in text editor
```

Copy, Move, Delete

```
cp source.txt dest.txt     # copy file
mv file.txt folder/        # move file
rm file.txt                # delete file
rm -r folder/              # delete folder recursively
```

Bonus: Understanding Paths

- **Absolute Path**: starts from root `/` `/home/you/documents/file.txt`
- **Relative Path**: from your current directory

```
./file.txt     # file in current directory
../file.txt    # file one level up
```

Wrap-Up

Learning Linux is not about memorizing magical “Linux commands.” It’s about understanding the **tools**, **shell**, and **kernel** interactions — and how to combine them effectively.

You’re learning how to talk to your machine like a pro

Tasks

Task 1: What are `awk`, `sed`, `curl`, and `wget` commands?

`awk` – Pattern Scanning and Processing

`awk` is a powerful text-processing language used to manipulate and analyze structured text (like tables, logs, and CSVs).

Basic Syntax

`awk 'pattern {action}' file`

Common Use Cases

- Print specific columns from a file: `awk '{print $1, $3}' data.txt`
- Filter lines with a condition: `awk '$2 > 50' file.txt`

- Use field separator (like commas):
  `awk -F, '{print $1}' data.csv`

Key Concepts

- `$0` = whole line
- `$1`, `$2`, ... = columns
- `NR` = current line number
- `BEGIN {}` and `END {}` blocks for setup and teardown

`sed` – Stream Editor

`sed` is a command-line utility for parsing and transforming text in a stream or file.

Basic Syntax

`sed 's/pattern/replacement/' file`

Common Use Cases

- Replace first occurrence of "cat" with "dog":
  `sed 's/cat/dog/' file.txt`
- Replace globally on each line:
  `sed 's/cat/dog/g' file.txt`
- Delete lines:
  `sed '/pattern/d' file.txt`

Notes

- `-i` modifies files in-place (be cautious):
  `sed -i 's/foo/bar/g' file.txt`

`curl` – Client URL

`curl` is a command-line tool to transfer data to or from a server using protocols like HTTP, HTTPS, FTP, etc.

Common Use Cases

- Fetch a webpage:
  `curl https://example.com`
- Save output to file:
  `curl -o index.html https://example.com`
- Send POST request:
  `curl -X POST -d "username=user&password=pass" https://site.com/login`
- Add headers:
  `curl -H "Authorization: Bearer TOKEN" https://api.com/data`

`wget` – Web Get

`wget` is used to **download files** from the web non-interactively.

Common Use Cases

- Download a file:
  `wget https://example.com/file.zip`
- Download in background:
  `wget -b https://example.com/file.zip`
- Mirror an entire website:
  `wget --mirror -p --convert-links -P ./local-dir https://site.com`

`wget` vs `curl`

- `wget` is better for bulk or recursive downloading.
- `curl` is better for APIs and fine-grained control.

Task 2: Linux File Permissions

File Permissions in Linux

Linux permissions control **who can read, write, or execute** files and directories.

Every file has **three levels of access**:

1. **Owner (user)**
2. **Group**

3. **Others (world)**

Each level has **three permission types**:

- `r` = read
- `w` = write
- `x` = execute

Viewing Permissions

Run:

```
ls -l
```

Example output:

```
-rwxr-xr-- 1 user group 1234 Apr 25 10:00 script.sh
```

Breakdown:

- `-` → regular file
- `rwx` → owner: can read/write/execute
- `r-x` → group: can read/execute
- `r--` → others: can read only

Changing Permissions

`chmod` – Change Mode

Change file permissions.

- **Symbolic mode**:

~~~
chmod u+x file.sh   # add execute for owner
chmod g-w file.sh   # remove write for group
chmod o=r file.sh   # others can only read

  ```

- **Numeric mode**:
~~~

~~~
    - `r = 4`, `w = 2`, `x = 1`

    - Add values to get total permission

    `chmod 755 file.sh`

    Meaning:

    - Owner: 7 (4+2+1 = rwx)

    - Group: 5 (4+0+1 = r-x)

    - Others: 5 (4+0+1 = r-x)


---

## Changing Ownership

### `chown` - Change Owner

```bash
chown user file.txt           # Change owner
chown user:group file.txt     # Change owner and group
~~~

`chgrp` – Change Group

```
chgrp group file.txt
```

Directory Permissions

- `r` → list contents (`ls`)
- `w` → create/delete files
- `x` → access (enter the directory)

Example:

```
chmod 700 private_dir
```

Only the owner can read/write/enter.

Summary Table

Symbol	Meaning	Binary	Decimal
`r`	Read	100	4
`w`	Write	010	2
`x`	Execute	001	1
`-`	No permission	000	0

Role	Description
u	User (owner)
g	Group
o	Others
a	All (user + group + others)

Servers, Web Apps, Apks

Servers

Quick List of Services

- DNS
- DHCP
- HTTP
- HTTPS
- FTP

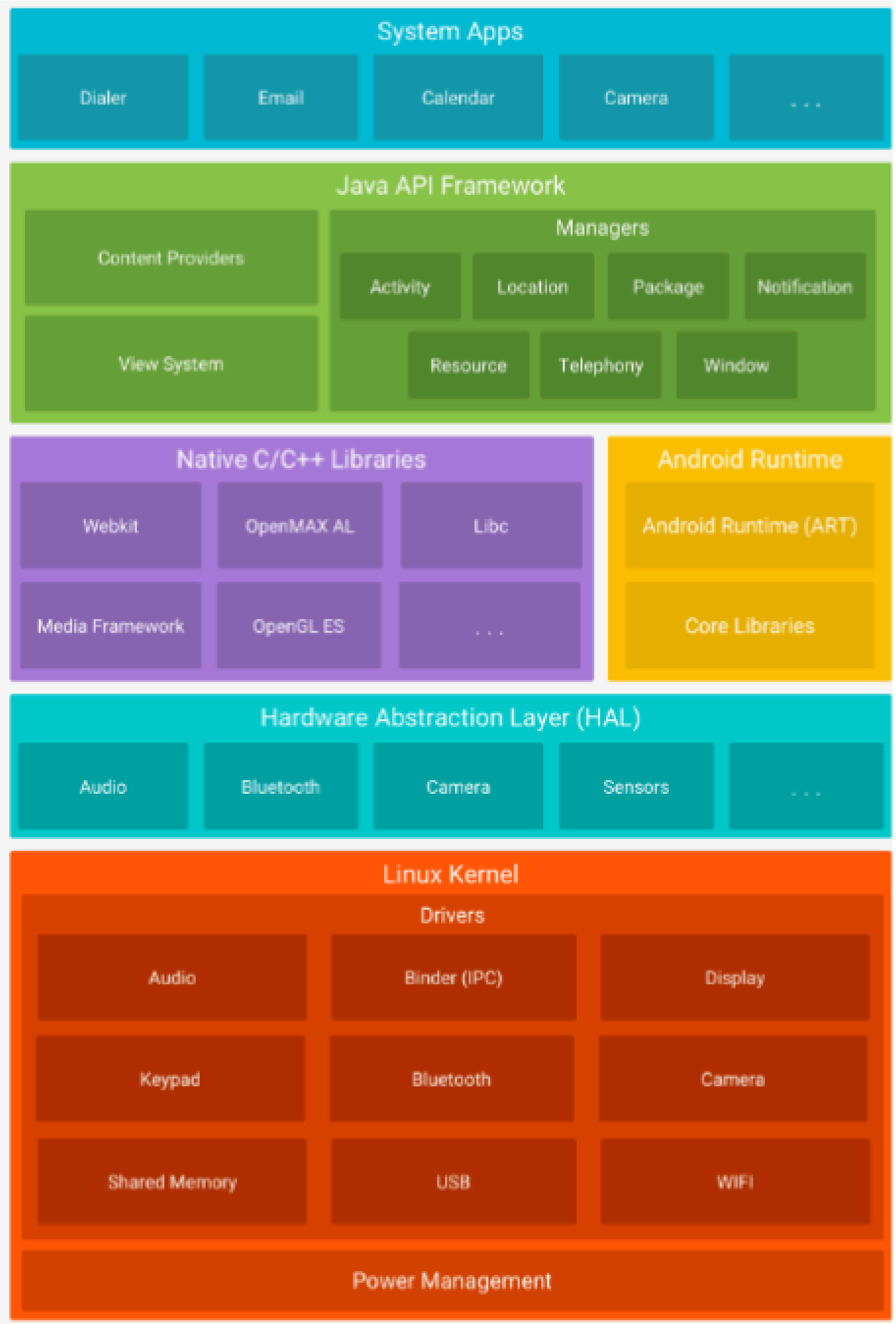
Web Apps

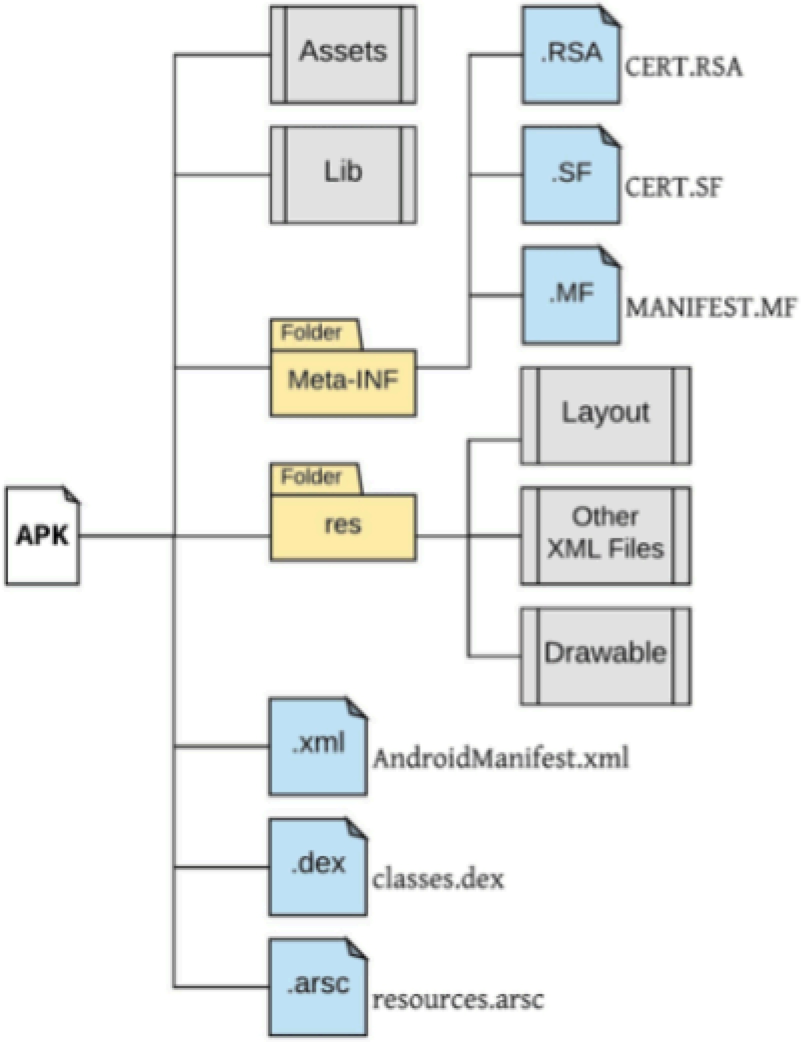
Understand how http requests work.
Quick list:

- GET
- POST
- PUT

- DELETE
- etc.

APKs


System Apps
Dialer
Email
Calendar
Camera
. . .
Java API Framework
Content Providers
View System
Managers
Activity
Location
Package
Notification
Resource
Telephony
Window
Native C/C++ Libraries
Webkit
OpenMAX AL
Libc
Media Framework
OpenGL ES
. . .
Android Runtime
Android Runtime (ART)
Core Libraries
Hardware Abstraction Layer (HAL)
Audio
Bluetooth
Camera
Sensors
. . .
Linux Kernel
Drivers
Audio
Binder (IPC)
Display
Keypad
Bluetooth
Camera
Shared Memory
USB
WIFI
Power Management



Tasks

Last Module Tasks

This is also quickly thrown together, will fix later.

Task 1

Burp Suite is a software security application used for penetration testing of web applications.

Task 2

Jadex:
From what I know should be de-compiler, basically gets source code from apk.