Digital Forensics

Topic 3: Technical Concepts
Lab 03: Linux Commands & Artefacts

Master Exam-Ready Study Notes — 2025/26

Contents

1 Operating Systems (OS)

1.1 What Is an Operating System?

Definition – Operating System An **Operating System (OS)** is a set of computer programs that manage the hardware and software resources of a computer. Think of it as the "manager" that sits between you (the user) and the physical machine.

Two types of interface:

- **Command Line Interface (CLI)** – you type text commands (e.g. Linux terminal).
- **Graphical User Interface (GUI)** – you click icons and windows (e.g. Windows desktop).

Five Core Functions of an OS (Exam Favourite)

1. Partitions and formats storage devices.
2. Creates a standard for naming files and folders.
3. Maintains the integrity of files and folders.
4. Error recovery.
5. Security of the file system.

Why does this matter for Digital Forensics (DF)?
The OS provides access to data storage. A DF investigator uses OS-level and DF-specific tools to view, recover, and analyse data stored by the OS.

1.2 OS Data – User Data

User data is anything *created by or for the user*: documents, photos, emails, downloads, etc.

Category	What it includes
Content Data	File/directory names, media files, word docs, web downloads, emails
OS Metadata	Creation/modification times, file permissions, physical location on disk
App Metadata	Geographical info (e.g. GPS in photos), name of file creator
Data Services	Encryption provided by the OS

DF Caution When a DF investigator views a file using normal OS tools, those tools can **accidentally modify** the file's metadata (e.g. update the access time). This is why dedicated DF tools and write-blockers are used.

Deleted files: When you "delete" a file, the OS usually only removes the *link/pointer* to the data. The actual data may still sit on the disk until it is overwritten. DF tools can recover such data.

1.3 OS Data – System Data

System data is used by the OS and its applications, not directly by the user.

Type	What It Is	DF Relevance
Configuration Data	Info about the OS, users, network settings, installed software	Shows who had access, what resources were available, what commands were run
Log Data	Records of system events: logins, errors, application behaviour	Attributing actions to users, building timelines
Process Data	Info about running programs managed by the OS	Shows what applications were running; only available via *live forensics* (RAM only)

1.3.1 System Configuration Data – Key Locations

OS	Location / Command	What It Shows
Linux	`/etc/` directory	OS config files
Linux	`cat /etc/os-release`	Platform info (OS name, version)
Linux	`cat /etc/passwd | column -t -s :`	User accounts on the system
Linux	`cat /etc/group`	Groups on the system
Windows	`systeminfo`	System configuration summary
Windows	Windows Registry (database)	Comprehensive system config

1.3.2 Log Data – Key Locations

OS	Log Location
Linux	`/var/log/` (e.g. `auth.log` for login attempts)
Windows	`C:\Windows\System32\winevt\Logs` (viewed via Event Viewer)

1.3.3 Process Data – Key Locations

OS	Location / Command	Notes
Linux	`/proc/` directory, `ps` command	Files have size 0 (pseudo-filesystem)
Windows	`tasklist`, `get-process`, Process Explorer	

EXAM KEY POINT – Process Data Files in `/proc/` have content you can read, but the `ls` command reports their size as **0 bytes**. This is because `/proc/` is a **pseudo-filesystem** — it exists only in RAM, not on disk. Therefore, process data is **NOT available from a disk image** and requires **live forensics** to capture.

1.4 Data Representation

Users see files as readable text, images, etc. Internally, computers store everything as **bits** (0 or 1).

- 1 **bit** = 0 or 1
- 1 **byte** = 8 bits
- 1 **Kilobyte (KB)** = 1024 bytes = 2^{10} bytes

1.4.1 ASCII Table

Definition – ASCII **ASCII** (American Standard Code for Information Interchange) is a character encoding standard. Every letter, number, and symbol you type is mapped to a number.

- Standard ASCII: 128 characters, encoded in 7 bits ($2^7 = 128$).
- Extended ASCII: 256 characters, encoded in 8 bits ($2^8 = 256$).
- Includes "non-printable" characters like tab, line feed, escape, delete.
- 95 of the 128 standard characters are printable.

DF tools like `xxd` show file contents in hexadecimal alongside their ASCII representation. This "hex dump" view is extremely common in forensic analysis.

2 Data Storage

2.1 Types of Computer Memory

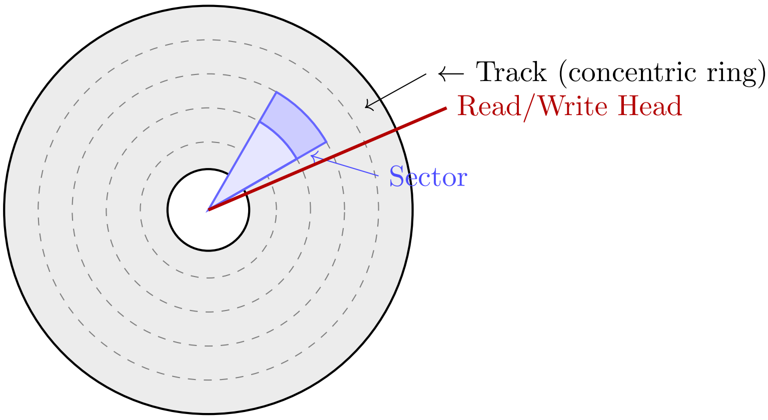
	Primary Memory	Secondary Storage
Examples	RAM, Cache	HDD, SSD, CD/DVD, USB
Volatile?	Yes (data lost when power off)	No (data persists)
Speed	Very fast	Slower
DF Note	Data can linger briefly after power loss (cold boot attack)	Main target for forensic imaging

Physical storage methods:

1. **Electromagnetism** – Hard Disk Drives (HDD).
2. **Microscopic electrical transistors (flash)** – SSDs, USB drives.
3. **Reflecting light** – CDs, DVDs.

2.2 Hard Disk Drive (HDD) Structure

2.2.1 Physical Components



Platter (top view)

Key HDD Terminology

Term	Meaning
Platter	Circular disk coated with magnetic material; data stored on top and bottom surfaces
Track	A narrow concentric ring on a platter surface
Cylinder	A vertical stack of the same track across all platters
Sector	A subdivision of a track; the **smallest addressable storage unit**, typically **512 bytes**
Cluster / Block	A group of sectors; the **minimum space allocated** to store a file. Called *cluster* in Windows, *block* in Linux
Head	The read/write mechanism; one per platter surface

2.2.2 How an HDD Works (4 Steps)

1. A circuit board controls the head actuator and a small motor.

2. The motor spins the platters continuously while the computer is on.
3. When an application requests data, the read/write heads consult the FAT/NTFS to find the data location.
4. The head actuator positions the read/write heads over the correct location to read or write.

2.3 Storage Interfaces

Interface	Used For	Notes
SAS	Enterprise HDDs	Serial Attached SCSI
SATA	Consumer HDDs & some SSDs	Serial ATA
PCIe/NVMe	Modern SSDs	Fastest
M.2	Modern SSDs (form factor)	Uses NVMe or SATA protocol
mSATA	Older compact SSDs	

2.4 Disk Capacity Formula

Disk Capacity Formula

$$\boxed{\text{Disk Capacity} = \text{Cylinders} \times \text{Heads} \times \text{Sectors per Track} \times 512 \text{ bytes}}$$

In plain English: Multiply the number of cylinders by the number of read/write heads by the number of sectors on each track by 512 bytes (the size of one sector).

2.5 Disk Formatting

Level	What It Does
Low-Level (Physical)	Creates the physical structure: divides platters into tracks and sectors. Done at the factory.
Logical (Partitioning)	Divides the disk into separate partitions; each acts like its own drive. Creates file system structures and marks all sectors as free.
High-Level	Sets up the specific file system (FAT, NTFS, EXT, etc.) within each partition. Creates the tables used to locate files.

2.6 Slack Space

Definition – Slack Space **Slack space** is the unused space within a data unit (cluster/block) that occurs when a file's size is **not an exact multiple** of the data unit size. A file must occupy at least one full data unit, even if it only uses a fraction of it.

2.6.1 Worked Example – Slack Space Calculation

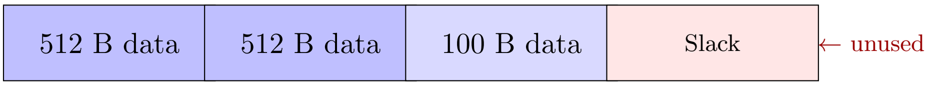
Exam-Style Calculation **Given:** A cluster has 4 sectors, each sector is 512 bytes. A file is 800 bytes.
Step 1: Cluster size $= 4 \times 512 = 2048$ bytes.
Step 2: The file uses:

- Sector 1: 512 bytes of data (full).
- Sector 2: $800 - 512 = 288$ bytes of data $+ 512 - 288 = 224$ bytes unused.
- Sectors 3 & 4: completely empty $= 512 \times 2 = 1024$ bytes unused.

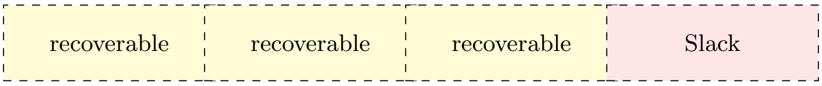
Step 3: Total slack space = $224 + 1024 = \mathbf{1248}$ bytes.
DF Relevance: Slack space may contain remnants of previously stored data, making it a goldmine for forensic recovery.

2.6.2 File Slack Diagram

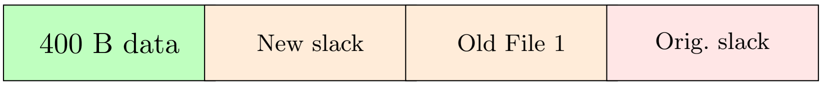
File 1 added: 1124 bytes



File 1 "deleted" – data may remain!



File 2 added: 400 bytes



EXAM KEY POINT – Deletion & Recovery
- "Deleting" a file usually only removes the OS pointers/links to it, **not the actual data**.
- The original data remains until **overwritten** by new data.
- Even after a shorter file overwrites part of the space, **remnants of the old file may survive** in the new slack space.

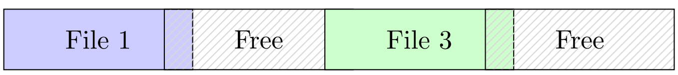
2.7 File Fragmentation

Definition – File Fragmentation **Fragmentation** occurs when a file is stored across **non-consecutive** data units (clusters/blocks) on disk. This happens when contiguous free space is not available.

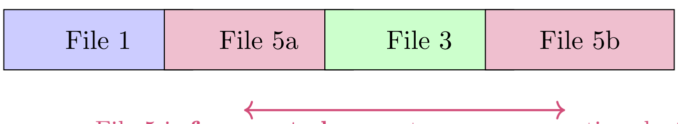
Why it matters for DF: If a fragmented file is "deleted" (pointers removed), recovery is harder because the pieces are scattered. DF tools use partial OS metadata, statistical analysis, and pattern matching to reassemble fragments.

2.7.1 Fragmentation Example (Diagram)

After Files 2 & 4 deleted:



File 5 added (bigger than one free gap):



File 5 is **fragmented** across two non-consecutive clusters

2.7.2 The `filefrag` Command

The **`filefrag`** command shows how a file is stored on disk:

filefrag examples

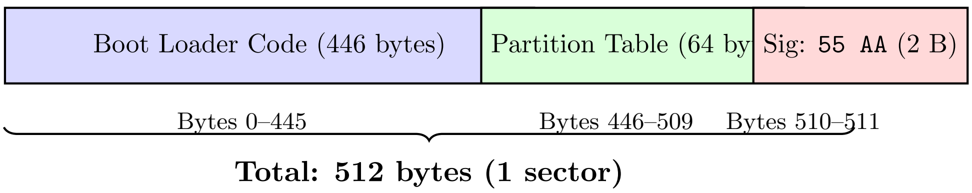
```
# Small 6-byte file: uses 1 extent (1 block group)
$ filefrag -b512 -v temp.txt
File size of temp.txt is 6 (8 blocks of 512 bytes)
ext: logical: physical:  length: flags:
  0:  0..7:  1167344..1167351: 8: last,eof
temp.txt: 1 extent found

# Large file: fragmented into 2 extents
$ filefrag -b512 -v bigTemp.pdf
File size of bigTemp.pdf is 2821185 (5512 blocks of 512 bytes)
ext: logical:    physical:    length: expected: flags:
  0: 0..4095:    29945856..:  4096:
  1: 4096..5511: 29999104..:  1416: 29949952: last,eof
bigTemp.pdf: 2 extents found
```

Key observation: Even a tiny 6-byte file "reserves" an entire data unit (8 sectors = 4096 bytes). The rest is slack space.

2.8 Disk Data Structures: MBR & Partitions

2.8.1 Master Boot Record (MBR)

Definition – MBR The **Master Boot Record** occupies the **first 512 bytes** (Sector 0) of a disk. It defines how the disk is partitioned and how the OS boots.



The **Partition Table** has space for **4 entries** (16 bytes each = 64 bytes total).

2.8.2 Partition Table Entry Structure (16 bytes)

Offset (bytes)	Length	Contents
0	1	Boot Indicator (`0x80` = active/bootable)
1–3	3	Starting CHS values
4	1	Partition-type descriptor (FAT, NTFS, etc.)
5–7	3	Ending CHS values
8–11	4	Starting Sector (**little-endian**)
12–15	4	Partition Size in sectors (**little-endian**)

2.8.3 Worked Example – Partition Size Calculation

Exam-Style: Hex to Partition Size **Given:** Partition size field (bytes 12–15) reads: `4B 34 41 00`

Step 1 – Reverse for Little-Endian:
Read the bytes from right to left: `00 41 34 4B`

Step 2 – Convert Hex to Decimal:

$$\texttt{00 41 34 4B}_{16} = 0\times16^7 + 0\times16^6 + 4\times16^5 + 1\times16^4 + 3\times16^3 + 4\times16^2 + 4\times16^1 + 11\times16^0$$
$$= 0 + 0 + 4{,}194{,}304 + 65{,}536 + 12{,}288 + 1{,}024 + 64 + 11 = 4{,}273{,}227 \text{ sectors}$$

Step 3 – Convert sectors to bytes:
$4{,}273{,}227 \times 512 = 2{,}187{,}892{,}224$ bytes

Step 4 – Convert to human-readable:
$2{,}187{,}892{,}224 \div 1{,}024 = 2{,}136{,}613$ KB
$2{,}136{,}613 \div 1{,}024 \approx 2{,}086$ MB $\approx$ **2 GB**

2.9 SSD Drives

Feature	Detail
Storage method	Flash memory (no spinning disks or heads)
Data organisation	Data stored in **pages**; pages grouped into **blocks** (loosely analogous to HDD sectors)
Write behaviour	Always writes to **empty pages**; data stored “randomly”
Partition scheme	Uses **GPT** (GUID Partition Table) instead of MBR
GPT advantages	Supports volumes > 2 TB; up to 128 partitions
Wear levelling	Distributes writes evenly; can impact data recovery

EXAM KEY POINT – TRIM Command

- **TRIM** tells the SSD which blocks are no longer in use (deleted).
- The SSD then **zeroes out** those blocks, making data **unrecoverable**.
- TRIM is **enabled by default** on internal SATA/eSATA drives.
- TRIM is **NOT enabled** on: RAID arrays, external SSDs, and some older OSes.
- **DF impact:** TRIM makes recovery of deleted files from SSDs **much more difficult** than from HDDs.

3 File Systems

Definition – File System A **file system** manages how data is stored and retrieved on a storage device. It defines the rules for organising files, directories, and metadata.

3.1 File System Layers

Logical Layer	User-level API: read, write, chmod, etc.
Virtual Layer (optional)	Allows access to multiple physical file systems simultaneously
Physical Layer	Hardware interaction: block management, device drivers

3.2 File System Terminology

Term	Definition
Sector	Smallest addressable section of memory (512 / 2048 / 4096 bytes)
INode	A data structure containing metadata, pointers, and structures about a file
Data Unit	Standard-sized container for content data. Called *cluster* (Windows) or *block* (Linux)
Physical Sector Size	Actual sector size on hardware (typically 4096 bytes on modern systems)
Logical Sector Size	Smaller size reported by OS for backwards compatibility (often 512 bytes)

Checking sector sizes in Linux

```
$ lsblk -t /dev/sda
NAME  PHY-SEC  LOG-SEC  ROTA
sda     4096     512      1      # PHY-SEC=4096, LOG-SEC=512
```

3.3 File System Data Categories

Category	What It Contains
File System	Overview data about the file system itself
Content	Actual file contents, organised in data units
Metadata	Descriptions of files: access times, sizes, owners
File Name	Human-readable name (mapped to metadata address)
Application	Special features: quota data, journalling

3.4 File System Architectures

FS	Full Name	Default OS	Notes
FAT32	File Allocation Table (32-bit)	Removable media	Simple; supported by Windows & Linux; used for SD cards & USB
NTFS	New Technology File System	Windows	Default since Windows XP era
Ext2/3/4	Extended File System	Linux	Ext4 is current default

3.4.1 FAT32 Key Structures

- Data units called **clusters**.
- **File Allocation Table (FAT):** Stores the next cluster for each file; holds allocation status.
- **Directory Entries:** One per file/directory, containing: file name, size, starting cluster address, and metadata.

3.4.2 Ext2 Key Features

- Partition content split into **block groups**.
- **Advantages of block groups:**
  - Reduced file access times (sectors physically close).
  - Reduced fragmentation (files stored within single block groups).
  - Redundancy (super block info repeated across groups).
  - Stability (errors localised to individual block groups).
- Reduced fragmentation and redundancy are **advantageous for DF recovery**.

4 Numbering Systems

4.1 Overview of Number Systems

System	Base	Symbols	Humans?	DF Experts?	Computer Storage?
Decimal	10	0–9	Yes	Yes	No
Binary	2	0, 1	No	Yes	Yes
Octal	8	0–7	No	Yes	No
Hexadecimal	16	0–9, A–F	No	Yes	No

Why do we care? Computers store data in binary. Hex and octal are compact, human-readable ways to represent binary. DF tools display data in hex constantly.
Significant digits: In any base, the **leftmost** digit is the *Most Significant Digit (MSD)* and the **rightmost** is the *Least Significant Digit (LSD)*.
Hex prefix: Hex numbers are often prefixed with `0x` for clarity, e.g. `0x19` = hex 19.

4.2 Key Relationship: Powers of 2

Why Octal and Hex Are "Easy"

- Octal: base $8 = 2^3$, so each octal digit = exactly **3 binary bits**.
- Hex: base $16 = 2^4$, so each hex digit = exactly **4 binary bits**.
- This makes conversion between binary ↔ octal/hex trivial (just group bits).
- Decimal (base 10) is NOT a power of 2, so conversion requires division.

4.3 Converting TO Decimal

Universal Method: Multiply each digit by base$^{\text{position}}$ (position starts at 0 from the right), then add.

4.3.1 Binary → Decimal

Worked Example: $11001_2 \to$ Decimal

$$\underbrace{1}_{pos\,4}\ \underbrace{1}_{pos\,3}\ \underbrace{0}_{pos\,2}\ \underbrace{0}_{pos\,1}\ \underbrace{1}_{pos\,0}$$

$$= 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$
$$= 16 + 8 + 0 + 0 + 1 = \boxed{25}$$

4.3.2 Octal → Decimal

Worked Example: $31_8 \to$ Decimal

$$= 3 \times 8^1 + 1 \times 8^0 = 24 + 1 = \boxed{25}$$

4.3.3 Hexadecimal → Decimal

Worked Example: $19_{16} \to$ Decimal

$$= 1 \times 16^1 + 9 \times 16^0 = 16 + 9 = \boxed{25}$$

Summary: $25_{10} = 11001_2 = 31_8 = 19_{16}$

4.4 Converting TO Binary

4.4.1 Decimal → Binary (Repeated Division by 2)

Worked Example: $25_{10} \to$ Binary Divide by 2 repeatedly, record the **remainder** each time. Read remainders from **bottom to top**.

$$\begin{aligned} 25 \div 2 &= 12 \quad \text{remainder } \mathbf{1} \text{ (LSB)} \\ 12 \div 2 &= 6 \quad \text{remainder } \mathbf{0} \\ 6 \div 2 &= 3 \quad \text{remainder } \mathbf{0} \\ 3 \div 2 &= 1 \quad \text{remainder } \mathbf{1} \\ 1 \div 2 &= 0 \quad \text{remainder } \mathbf{1} \text{ (MSB)} \end{aligned}$$

Read bottom-to-top: $\boxed{11001_2}$

4.4.2 Octal → Binary (3-bit groups)

Each octal digit becomes exactly 3 binary bits.

Worked Example: $705_8 \to$ Binary

$$7 \to 111 \qquad 0 \to 000 \qquad 5 \to 101$$

$$705_8 = \boxed{111\ 000\ 101_2}$$

4.4.3 Hex → Binary (4-bit groups)

Each hex digit becomes exactly 4 binary bits.

Worked Example: $10AF_{16} \to$ Binary

$$1 \to 0001 \quad 0 \to 0000 \quad A(=10) \to 1010 \quad F(=15) \to 1111$$

$$10AF_{16} = \boxed{0001\ 0000\ 1010\ 1111_2}$$

4.5 Converting TO Octal

4.5.1 Decimal → Octal (Repeated Division by 8)

Worked Example: $25_{10} \to$ Octal

$$\begin{aligned} 25 \div 8 &= 3 \quad \text{remainder } \mathbf{1} \text{ (LSD)} \\ 3 \div 8 &= 0 \quad \text{remainder } \mathbf{3} \text{ (MSD)} \end{aligned}$$

Read bottom-to-top: $\boxed{31_8}$

4.5.2 Binary → Octal (Group in 3s from right)

Worked Example: 1011010111_2 → Octal Group from the right in sets of 3 (pad with leading zeros if needed):

$$\underbrace{1}_{1}\ \underbrace{011}_{3}\ \underbrace{010}_{2}\ \underbrace{111}_{7}$$

$$1011010111_2 = \boxed{1327_8}$$

4.5.3 Hex → Octal (via Binary)

Worked Example: $1F0C_{16}$ → Octal **Step 1:** Hex → Binary (4-bit groups):

$$1 \to 0001 \quad F \to 1111 \quad 0 \to 0000 \quad C \to 1100$$

Binary: 0001 1111 0000 1100
Step 2: Regroup into 3-bit groups from the right:

$$\underbrace{1}_{1}\ \underbrace{111}_{7}\ \underbrace{100}_{4}\ \underbrace{001}_{1}\ \underbrace{100}_{4}$$

Wait — let me regroup: 0 001 111 100 001 100

$$\underbrace{000}_{0}\ \underbrace{011}_{3}\ \underbrace{111}_{7}\ \underbrace{000}_{0}\ \underbrace{011}_{3}\ \underbrace{00}_{?}$$

Let me be precise. The binary is: 0001111100001100. That's 16 bits. Group in 3s from right:

$$\underbrace{0}\ \underbrace{001}_{1}\ \underbrace{111}_{7}\ \underbrace{100}_{4}\ \underbrace{001}_{1}\ \underbrace{100}_{4}$$

Pad the leftmost group: $\underbrace{000}_{0}$ 001 111 100 001 100 – no, let me just do it carefully:

0001 1111 0000 1100 = 16 bits.
Group from right: 0|001|111|100|001|100
$= 0, 1, 7, 4, 1, 4$
$= \boxed{17414_8}$

4.6 Converting TO Hexadecimal

4.6.1 Decimal → Hex (Repeated Division by 16)

Worked Example: 422_{10} → Hex

$$\begin{aligned} 422 \div 16 &= 26 && \text{remainder } \mathbf{6} \text{ (LSD)} \\ 26 \div 16 &= 1 && \text{remainder } \mathbf{10} = \mathbf{A} \\ 1 \div 16 &= 0 && \text{remainder } \mathbf{1} \text{ (MSD)} \end{aligned}$$

Read bottom-to-top: $\boxed{1A6_{16}}$
Verify: $1 \times 16^2 + 10 \times 16^1 + 6 \times 16^0 = 256 + 160 + 6 = 422$ ✓

4.6.2 Binary → Hex (Group in 4s from right)

Worked Example: 1010111011_2 → Hex Group from right in sets of 4 (pad left with zeros):

$$\underbrace{0010}_{2}\ \underbrace{1011}_{B}\ \underbrace{1011}_{B}$$

$$1010111011_2 = \boxed{2BB_{16}}$$

4.6.3 Octal → Hex (via Binary)

Worked Example: 1076_8 → Hex **Step 1:** Octal → Binary (3-bit groups):

$$1 \to 001 \quad 0 \to 000 \quad 7 \to 111 \quad 6 \to 110$$

Binary: 001 000 111 110
Step 2: Regroup into 4-bit groups from right:

$$\underbrace{0010}_{2}\ \underbrace{0011}_{3}\ \underbrace{1110}_{E}$$

$$1076_8 = \boxed{23E_{16}}$$

4.7 Quick-Reference: Conversion Cheat Sheet

Conversion	Method
Any → Decimal	Multiply each digit by base$^{\text{position}}$, sum all
Decimal → Any base b	Repeated division by b, read remainders bottom-up
Binary ↔ Octal	Group/split in **3-bit** chunks
Binary ↔ Hex	Group/split in **4-bit** chunks
Octal ↔ Hex	Convert to binary first, then regroup

Decimal	**Binary**	**Octal**	**Hex**
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10

5 Lab 03: Linux Commands & Artefacts

5.1 Lab 3.1 – File Timestamps (MAC Times)

Linux records **four timestamps** for every file. These are critical for building a forensic timeline.

Time	Abbrev.	Meaning	Command to View
Access time	atime	Last time file **content was read**	`ls -lu file`
Modification time	mtime	Last time file **content was changed**	`ls -l file` (default)
Change time	ctime	Last time file **metadata was changed** (name, permissions, etc.)	`ls -lc file`
Creation time	crtime	Time the file was **first created** ("birth")	`ls --time=birth file`

EXAM KEY POINT

- `ls -l` shows **modification time (mtime)** by default.
- The `stat file` command shows **all four timestamps** at once.
- `crtime` (creation/birth time) is only available on newer file systems and may not be supported everywhere.

Example commands

```
$ ls -l  fileA       # Shows modification time (mtime)
$ ls -lu fileA       # Shows access time (atime)
$ ls -lc fileA       # Shows change time (ctime)
$ ls --time=birth fileA  # Shows creation time (crtime)
$ stat fileA         # Shows ALL times + other metadata
```

5.2 Lab 3.2 – Log Files & Processes

5.2.1 Log Files

File / Directory	Purpose
`/var/log/`	Main directory for all Linux log files
`/var/log/auth.log`	Records all login/authentication attempts, including `sudo` commands
`~/.bash_history`	History of commands typed by the user in the terminal

Viewing authentication logs

```
$ sudo ls /var/log         # List all log files
$ tail /var/log/auth.log   # View last 10 authentication entries
```

What `auth.log` shows for DF:

- Date/time of the authentication attempt.
- Whether a sudo session was initiated.
- Which user initiated the attempt.
- Duration of the session.

- Failed authentication attempts.

EXAM Q&A – Can a user delete `~/.bash_history`? **Yes.** The file is owned by the user, and the user has write (`w`) permission. Therefore, they can delete or modify it. This is important because a suspect could tamper with their command history.

5.2.2 Process Data (`/proc/`)

Working with processes

```
$ echo abcd > file.txt        # Create a file
$ gedit file.txt &            # Open in editor (& = background)
$ ps -a                       # List processes, find PID (e.g., 1234)
$ ls /proc/1234/              # View files for that process
$ xxd /proc/1234/cmdline      # View the command that started it
```

EXAM KEY POINT – `/proc/` Pseudo-Filesystem

- Files in `/proc/` all report **size = 0 bytes** (via `ls`).
- Despite size 0, they **do contain data** that can be read.
- This is because `/proc/` is a **pseudo-filesystem**: data exists only in **RAM**, not on disk.
- **DF implication:** These files would **NOT** appear in a static disk image. They are only accessible during **live forensics**.

5.3 Lab 3.3 – File Blocks & Fragmentation

5.3.1 File Sizes

Command	Unit Shown	Notes
`ls -l`	Bytes	Default; a value of 5 means 5 bytes
`ls -lh`	Human-readable (K, M, G)	Uses 1K = 1024 bytes (not 1000!)

Worked Example: File Size Conversion **Given:** `ls -l` shows a file as 246,779 bytes. `ls -lh` shows 241K. Why?

Step-by-step:

$$\frac{246{,}779 \text{ bytes}}{1024 \text{ bytes/KB}} = 240.995... \approx 241 \text{ K}$$

Key: 1 Kilobyte = 1024 bytes ($= 2^{10}$), NOT 1000 bytes.

5.3.2 Block Size & Block Count

Determining block size

```
$ stat -fc %s .               # Shows block size (likely 4096 bytes)
$ filefrag -b4096 -v file     # Shows number of blocks used by file
```

Worked Example: How Many Blocks? **Given:** File size = 246,779 bytes. Block size = 4096 bytes.

Step 1: Divide file size by block size:

$$\frac{246{,}779}{4{,}096} = 60.248...$$

Step 2: You can't use a fraction of a block, so round **up**: $\lceil 60.248 \rceil = \mathbf{61}$ blocks.
Step 3: Verify — 61 blocks = $61 \times 4096 = 249{,}856$ bytes of space allocated.
Step 4: Slack space in last block = $249{,}856 - 246{,}779 = 3{,}077$ bytes.
Step 5: Sanity check — $3{,}077 < 4{,}096$ (less than one block), confirming 60 blocks would be too few.

6 Exam Quick-Reference Summary

6.1 Must-Know Formulas

What	Formula
Disk Capacity	Cylinders × Heads × Sectors/Track × 512
Cluster/Block Size	Sectors per cluster × Sector size
Number of Blocks	$\lceil$File size ÷ Block size$\rceil$ (round up)
Slack Space	(Blocks × Block size) − File size
KB from bytes	Bytes ÷ 1024
MB from KB	KB ÷ 1024
Base b → Decimal	$\sum d_i \times b^i$ (position-weighted sum)
Decimal → base b	Repeated division by b, read remainders bottom-up

6.2 Must-Know Comparison: HDD vs SSD

Feature	HDD	SSD
Technology	Magnetic platters + spinning heads	Flash memory
Data units	Tracks → Sectors → Clusters	Pages → Blocks
Partition scheme	MBR (4 partitions, ≤2TB)	GPT (128 partitions, >2TB)
Deleted data recovery	Generally possible	Harder (TRIM zeroes blocks)
TRIM	N/A	Enabled by default (internal)

6.3 Must-Know: Key Linux Paths for DF

Path	Forensic Value
`/etc/`	System configuration (users, groups, OS info)
`/var/log/`	System and application logs
`/var/log/auth.log`	Authentication attempts (who logged in, sudo use)
`~/.bash_history`	User's command history (deletable by user!)
`/proc/`	Running process info (RAM only, size 0, live forensics only)

6.4 Must-Know: MBR Structure

Section	**Size**	**Byte Offsets**
Boot Loader Code	446 bytes	0–445
Partition Table (4 × 16 bytes)	64 bytes	446–509
Signature (`55 AA`)	2 bytes	510–511
Total MBR	**512 bytes**	**Sector 0**

End of Master Study Notes – Topic 3 + Lab 03