**AWK – 15 Practical & Detailed Use Cases**

📌 awk is a powerful text processing tool used in the Linux/Unix shell.

It’s perfect for:

* 📊 Reading and processing structured text
* 🔎 Extracting columns (fields) from a file
* 📋 Analyzing and filtering records based on conditions
* 🧮 Performing operations like sum, average, count, min/max

✅ **Think of awk as:**

A mini-programming language for processing text line by line, especially where data is structured in columns (space/tab/comma-separated).

🔧 Common Syntax:

awk 'pattern { action }' filename

* pattern: (optional) Condition to check for each line.
* action: What to do if the condition matches (usually print, sum, etc.)

**🧪 Sample File: data.txt**

1 John Manager 5000 Active

2 Alice Developer 4200 Inactive

3 Bob Tester 3000 Active

4 Charlie Manager 5200 Active

5 David Developer 4500 Inactive

**✅ 1. Print the entire file**

awk '{ print }' data.txt

📤 Output: Same as the content.

🔍 Explanation:

* { print }: Default behavior—print every line.
* Useful for testing awk line-by-line processing.

🧠 Use Case: When reading line by line and planning to apply further logic later.

**✅ 2. Print specific fields**

awk '{ print $2, $3 }' data.txt

📤 Output:

John Manager

Alice Developer

Bob Tester

Charlie Manager

David Developer

🔍 Explanation:

* $2 = Name, $3 = Role
* AWK splits lines by spaces by default.

🧠 Use Case: Extract selected columns (e.g., name and designation).

**✅ 3. Print line numbers with content**

awk '{ print NR ": " $0 }' data.txt

📤 Output:

1: 1 John Manager 5000 Active

2: 2 Alice Developer 4200 Inactive

...

🔍 Explanation:

* NR = Current line number.
* $0 = Entire line.

🧠 Use Case: Helpful in debugging or creating numbered logs.

**✅ 4. Print lines where salary > 4500**

awk '$4 > 4500 { print $2, $4 }' data.txt

📤 Output:

Charlie 5200

🔍 Explanation:

* $4 > 4500: Applies condition.
* Prints name and salary only if the condition matches.

🧠 Use Case: Salary filtering, analytics, or HR tools.

**✅ 5. Count total salary**

awk '{ sum += $4 } END { print "Total Salary:", sum }' data.txt

📤 Output:

Total Salary: 21900

🔍 Explanation:

* In each line, $4 (salary) is added to sum.
* END block runs once after all lines are processed.

🧠 Use Case: Summing up numerical data across rows.

**✅ 6. Average salary of all employees**

awk '{ sum += $4; count++ } END { print "Average:", sum/count }' data.txt

🧠 Use Case: HR dashboards, resource cost calculations.

**✅ 7. Filter lines where role is "Manager"**

awk '$3 == "Manager" { print $2, $3 }' data.txt

📤 Output:

John Manager

Charlie Manager

🔍 Explanation:

* $3 == "Manager": Exact string match.

🧠 Use Case: Role-based reports, targeted selections.

**✅ 8. Change a field value**

awk '$3 == "Developer" { $3 = "Engineer" } { print }' data.txt

📤 Output: Lines where "Developer" is replaced by "Engineer".

🧠 Use Case: Updating records without using sed.

**✅ 9. Print last column (status)**

awk '{ print $NF }' data.txt

📤 Output:

Active

Inactive

Active

Active

Inactive

🔍 Explanation:

* NF = Number of fields
* $NF = Value of the last field

🧠 Use Case: When you don’t know how many fields are there, but want the last one (common in logs).

**✅ 10. Count how many employees are Active**

awk '$5 == "Active" { count++ } END { print count }' data.txt

📤 Output:

3

🧠 Use Case: Dynamic status tracking, monitoring users/systems.

**✅ 11. Use different field separator (CSV example)**

🧪 File: info.csv

1,John,Manager,5000,Active

2,Alice,Developer,4200,Inactive

awk -F, '{ print $2, $4 }' info.csv

📤 Output:

John 5000

Alice 4200

🔍 Explanation:

* -F,: Sets delimiter to comma for CSVs.

🧠 Use Case: Parsing CSV files for automation scripts.

**✅ 12. Print employees with odd IDs**

awk '$1 % 2 == 1 { print $2 }' data.txt

🧠 Use Case: Alternate selection logic or group segmentation.

**✅ 13. Align output in formatted table**

awk '{ printf "%-10s %-10s %-6s\n", $2, $3, $4 }' data.txt

📤 Output:

John Manager 5000

Alice Developer 4200

...

🧠 Use Case: Creating neat reports for CLI tools.

**✅ 14. Group count by designation**

awk '{ roles[$3]++ } END { for (r in roles) print r, roles[r] }' data.txt

📤 Output:

Developer 2

Tester 1

Manager 2

🧠 Use Case: Role statistics, grouping and frequency analysis.

**✅ 15. Find max salary**

awk 'BEGIN { max = 0 } { if ($4 > max) max = $4 } END { print "Max Salary:", max }' data.txt

📤 Output:

Max Salary: 5200

🧠 Use Case: Business analytics, top-N logic, dashboards.