Day-9 SRE Training

Topic: Log File Analysis and Crontab

Log File Analysis:

we explored how to extract insights from a log file using various Linux commands:

`[a-z]` → Matches any **lowercase** letter (a to z).
`[A-Z]` → Matches any **uppercase** letter (A to Z).
`[0-9]` → Matches any **digit** (0 to 9).
`[a-zA-Z0-9.]` → Matches any **letter, digit, or period (`.`)**.
`[^...]` → Matches any **character NOT inside the brackets** (negation).

`{2,}` → **At least 2 times** Matches **two or more** occurrences of the preceding character or group.

`{1,3}` → **Between 1 and 3 times** Matches **at least 1 but at most 3** occurrences.

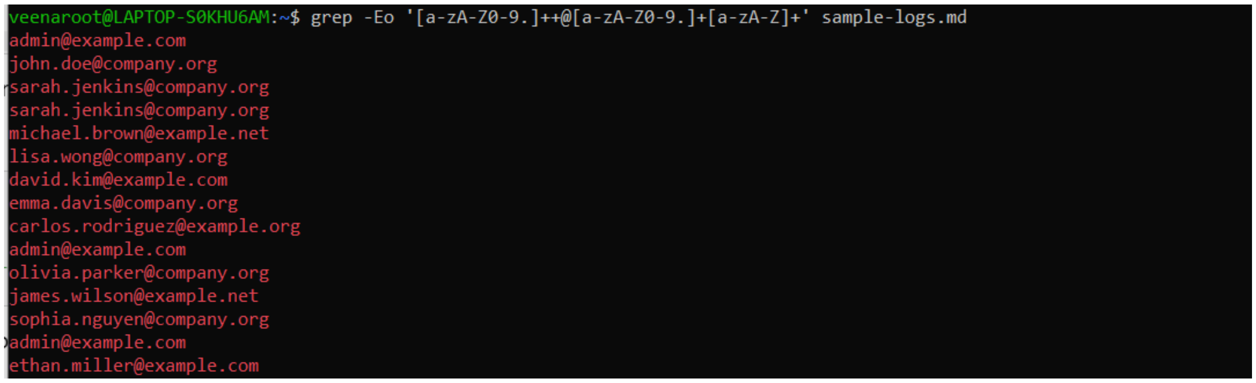
`+` → **One or more times (same as `{1,}`)** Matches the preceding character or group **at least once**.

`.` → **Matches any single character (except newline)** It is a **wildcard** that can match **any character** except a line break.

`*` → **Zero or more times** Matches the preceding character or group **any number of times (including zero)**.

```
grep -Eo '[a-zA-Z0-9.]++@[a-zA-Z0-9.]+[a-zA-Z]+' sample-logs.md
```

`grep -E` enables extended regex, and `-o` prints only matching email addresses.
The regex `[a-zA-Z0-9.]++@[a-zA-Z0-9.]+[a-zA-Z]+` extracts emails from the log file.



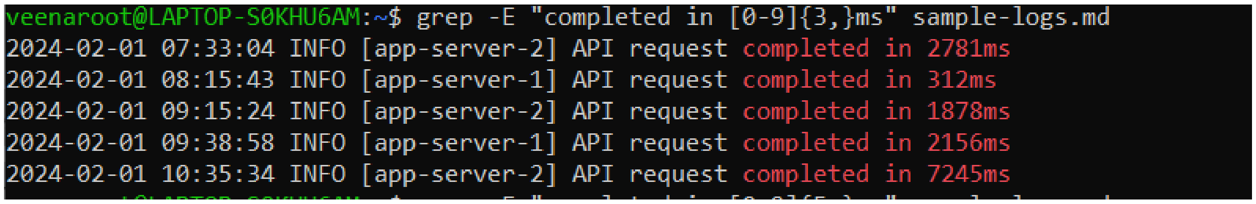
```
veenaroot@LAPTOP-S0KHU6AM:~$ grep -Eo '[a-zA-Z0-9.]++@[a-zA-Z0-9.]+[a-zA-Z]+' sample-logs.md
admin@example.com
john.doe@company.org
sarah.jenkins@company.org
sarah.jenkins@company.org
michael.brown@example.net
lisa.wong@company.org
david.kim@example.com
emma.davis@company.org
carlos.rodriguez@example.org
admin@example.com
olivia.parker@company.org
james.wilson@example.net
sophia.nguyen@company.org
admin@example.com
ethan.miller@example.com
```

```
grep -Eo '[a-zA-Z0-9.]++@[a-zA-Z0-9.]+[a-zA-Z]{3,}' sample-logs.md
```

`[a-zA-Z]{3,}` → Matches the top-level domain (TLD) with at least **3 characters** (e.g., `.com`, `.info`).

```
grep -E "completed in [0-9]{3,}ms" sample-logs.md
```

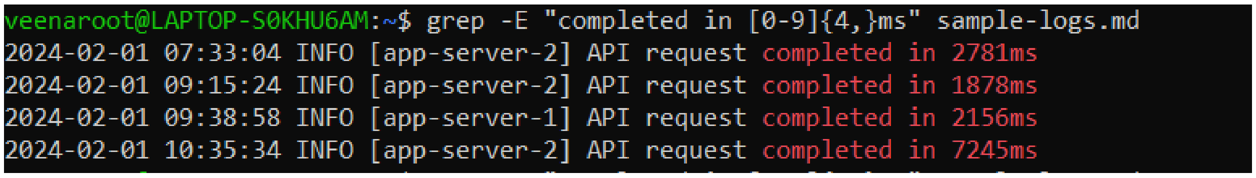
`-E` enables extended regex.
`[0-9]{3,}` matches numbers with **3 or more digits** (e.g., 100ms, 2500ms).



```
veenaroot@LAPTOP-S0KHU6AM:~$ grep -E "completed in [0-9]{3,}ms" sample-logs.md
2024-02-01 07:33:04 INFO [app-server-2] API request completed in 2781ms
2024-02-01 08:15:43 INFO [app-server-1] API request completed in 312ms
2024-02-01 09:15:24 INFO [app-server-2] API request completed in 1878ms
2024-02-01 09:38:58 INFO [app-server-1] API request completed in 2156ms
2024-02-01 10:35:34 INFO [app-server-2] API request completed in 7245ms
```

```
grep -E "completed in [0-9]{4,}ms" sample-logs.md
```

`-E` enables extended regex.
`[0-9]{4,}` matches numbers with **4 or more digits** (e.g., 1000ms, 25000ms).

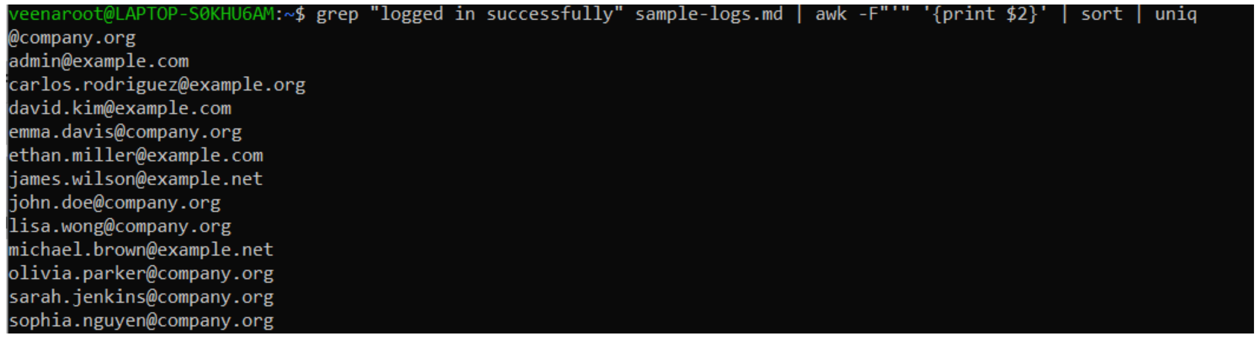


```
veenaroot@LAPTOP-S0KHU6AM:~$ grep -E "completed in [0-9]{4,}ms" sample-logs.md
2024-02-01 07:33:04 INFO [app-server-2] API request completed in 2781ms
2024-02-01 09:15:24 INFO [app-server-2] API request completed in 1878ms
2024-02-01 09:38:58 INFO [app-server-1] API request completed in 2156ms
2024-02-01 10:35:34 INFO [app-server-2] API request completed in 7245ms
```

```
grep "logged in successfully" sample-logs.md  | awk -F"'" '{print $2}' |
sort | uniq
```

`-F` in `awk` sets the **field separator**, meaning it splits the text based on the given character (here, a single quote `'`).

The whole command extracts unique usernames (or values) from log entries containing **"logged in successfully"** by filtering, splitting, sorting, and removing duplicates.

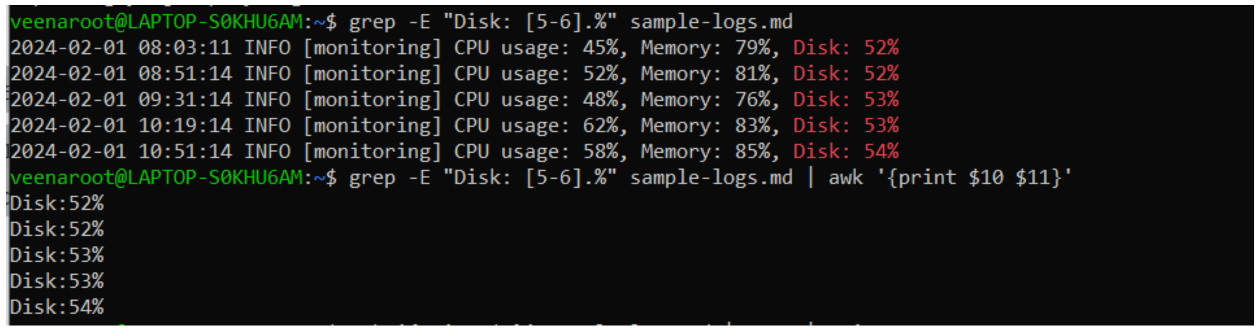


```
veenaroot@LAPTOP-S0KHU6AM:~$ grep "logged in successfully" sample-logs.md | awk -F"'" '{print $2}' | sort | uniq
@company.org
admin@example.com
carlos.rodriguez@example.org
david.kim@example.com
emma.davis@company.org
ethan.miller@example.com
james.wilson@example.net
john.doe@company.org
lisa.wong@company.org
michael.brown@example.net
olivia.parker@company.org
sarah.jenkins@company.org
sophia.nguyen@company.org
```

```
grep -E "Disk: [5-6].%" sample-logs.md | awk '{print $10 $11}'
```

`grep -E "Disk: [5-6].%"` filters lines where disk usage is between **50% and 69%**.

`awk '{print $10 $11}'` extracts and joins the **10th and 11th** fields from those lines, likely showing disk usage details.

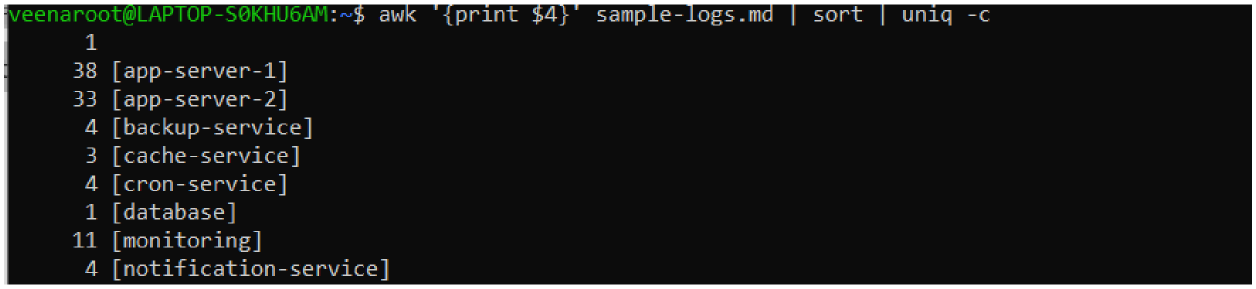


```
veenaroot@LAPTOP-S0KHU6AM:~$ grep -E "Disk: [5-6].%" sample-logs.md
2024-02-01 08:03:11 INFO [monitoring] CPU usage: 45%, Memory: 79%, Disk: 52%
2024-02-01 08:51:14 INFO [monitoring] CPU usage: 52%, Memory: 81%, Disk: 52%
2024-02-01 09:31:14 INFO [monitoring] CPU usage: 48%, Memory: 76%, Disk: 53%
2024-02-01 10:19:14 INFO [monitoring] CPU usage: 62%, Memory: 83%, Disk: 53%
2024-02-01 10:51:14 INFO [monitoring] CPU usage: 58%, Memory: 85%, Disk: 54%
veenaroot@LAPTOP-S0KHU6AM:~$ grep -E "Disk: [5-6].%" sample-logs.md | awk '{print $10 $11}'
Disk:52%
Disk:52%
Disk:53%
Disk:53%
Disk:54%
```

```
awk '{print $4}' sample-logs.md | sort | uniq -c
```

`awk '{print $4}' sample-logs.md` extracts the **4th field** from each line in the log file.

`sort | uniq -c` sorts the extracted values and counts occurrences of each unique value.

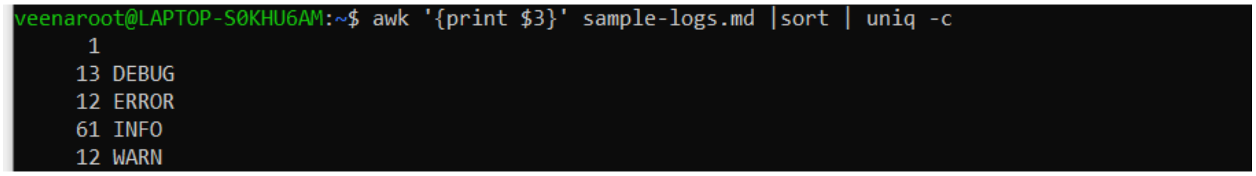


```
veenaroot@LAPTOP-S0KHU6AM:~$ awk '{print $4}' sample-logs.md | sort | uniq -c
      1
     38 [app-server-1]
     33 [app-server-2]
      4 [backup-service]
      3 [cache-service]
      4 [cron-service]
      1 [database]
     11 [monitoring]
      4 [notification-service]
```

```
awk '{print $3}' sample-logs.md |sort | uniq -c
```

`awk '{print $3}' sample-logs.md` extracts the **3rd field** from each line in the log file.

`sort | uniq -c` sorts these values and counts how many times each unique value appears.

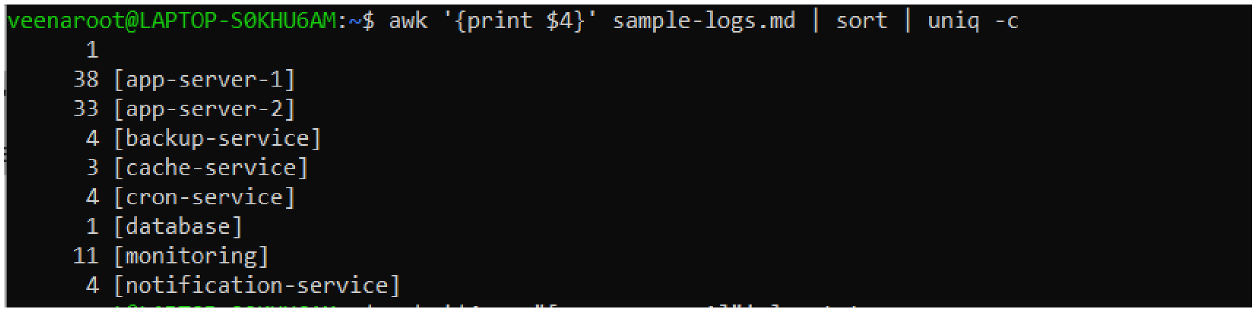


```
veenaroot@LAPTOP-S0KHU6AM:~$ awk '{print $3}' sample-logs.md |sort | uniq -c
      1
     13 DEBUG
     12 ERROR
     61 INFO
     12 WARN
```

```
grep -E "Memory: [5-8].%" sample-logs.md | awk '{print $8 $9}'
```

`grep -E "Memory: [5-8].%" sample-logs.md` filters lines where memory usage is between **50% and 89%**.

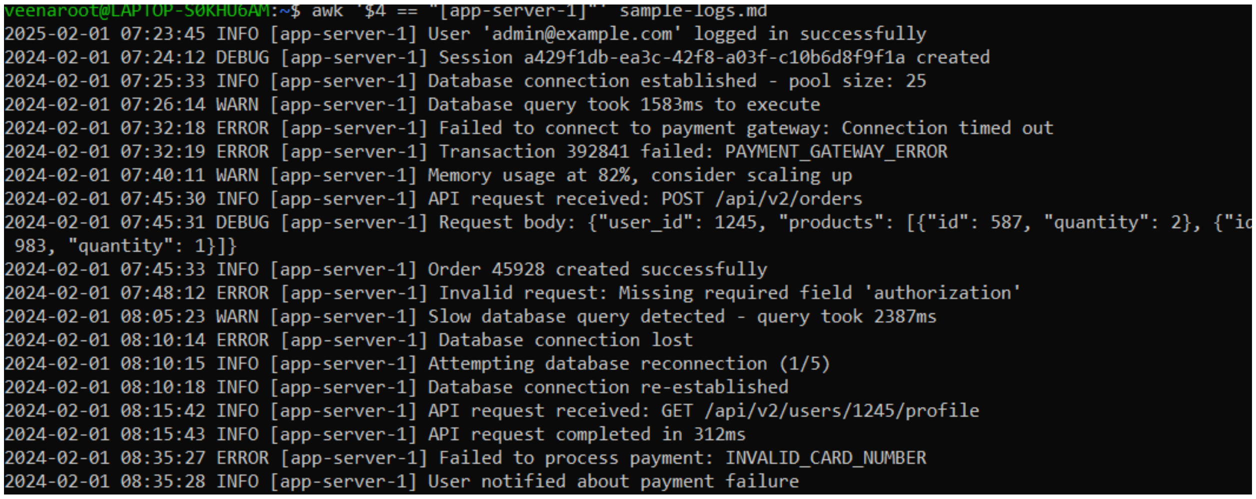
`awk '{print $8 $9}'` extracts and prints the **8th and 9th fields** from those lines, combining them without a space.



```
veenaroot@LAPTOP-S0KHU6AM:~$ awk '{print $4}' sample-logs.md | sort | uniq -c
      1
     38 [app-server-1]
     33 [app-server-2]
      4 [backup-service]
      3 [cache-service]
      4 [cron-service]
      1 [database]
     11 [monitoring]
      4 [notification-service]
```

```
awk '$4 == "[app-server-1]"' sample-logs.md
```

`awk '$4 == "[app-server-1]"' sample-logs.md` filters and displays only the lines from `sample-logs.md` where the **4th field** is exactly `"[app-server-1]"`.

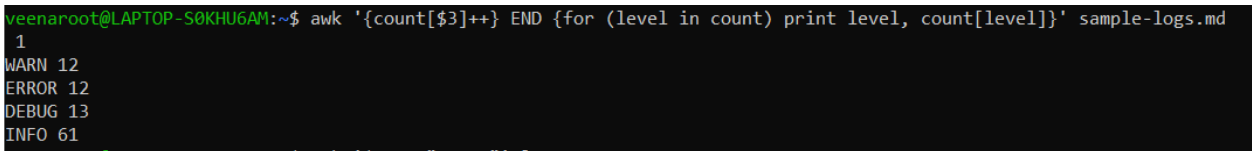


```
veenaroot@LAPTOP-S0KHU6AM:~$ awk '$4 == "[app-server-1]"' sample-logs.md
2025-02-01 07:23:45 INFO [app-server-1] User 'admin@example.com' logged in successfully
2024-02-01 07:24:12 DEBUG [app-server-1] Session a429f1db-ea3c-42f8-a03f-c10b6d8f9f1a created
2024-02-01 07:25:33 INFO [app-server-1] Database connection established - pool size: 25
2024-02-01 07:26:14 WARN [app-server-1] Database query took 1583ms to execute
2024-02-01 07:32:18 ERROR [app-server-1] Failed to connect to payment gateway: Connection timed out
2024-02-01 07:32:19 ERROR [app-server-1] Transaction 392841 failed: PAYMENT_GATEWAY_ERROR
2024-02-01 07:40:11 WARN [app-server-1] Memory usage at 82%, consider scaling up
2024-02-01 07:45:30 INFO [app-server-1] API request received: POST /api/v2/orders
2024-02-01 07:45:31 DEBUG [app-server-1] Request body: {"user_id": 1245, "products": [{"id": 587, "quantity": 2}, {"i
 983, "quantity": 1}]}
2024-02-01 07:45:33 INFO [app-server-1] Order 45928 created successfully
2024-02-01 07:48:12 ERROR [app-server-1] Invalid request: Missing required field 'authorization'
2024-02-01 08:05:23 WARN [app-server-1] Slow database query detected - query took 2387ms
2024-02-01 08:10:14 ERROR [app-server-1] Database connection lost
2024-02-01 08:10:15 INFO [app-server-1] Attempting database reconnection (1/5)
2024-02-01 08:10:18 INFO [app-server-1] Database connection re-established
2024-02-01 08:15:42 INFO [app-server-1] API request received: GET /api/v2/users/1245/profile
2024-02-01 08:15:43 INFO [app-server-1] API request completed in 312ms
2024-02-01 08:35:27 ERROR [app-server-1] Failed to process payment: INVALID_CARD_NUMBER
2024-02-01 08:35:28 INFO [app-server-1] User notified about payment failure
```

```
awk '{count[$3]++} END {for (level in count) print level, count[level]}'
sample-logs.md
```

`count[$3]++` → Increments the count for each unique value in the 3rd column.

`END {for (level in count) print level, count[level]}` → After processing all lines, prints each unique value and its count.

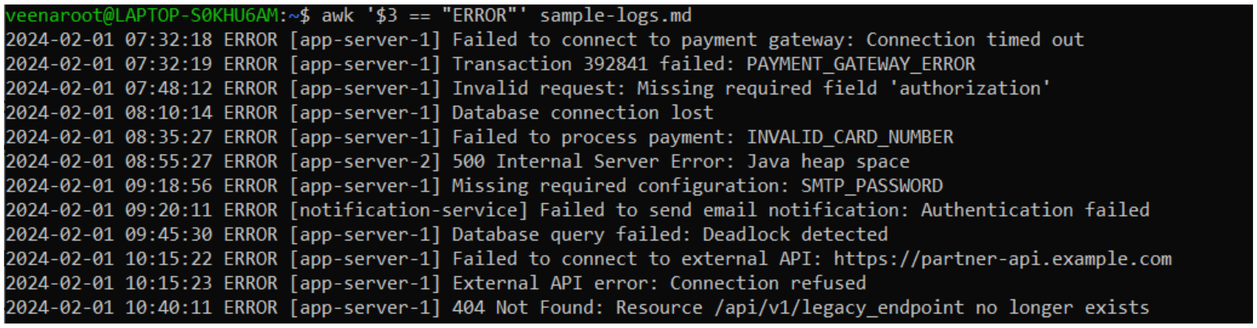


```
veenaroot@LAPTOP-S0KHU6AM:~$ awk '{count[$3]++} END {for (level in count) print level, count[level]}' sample-logs.md
 1
WARN 12
ERROR 12
DEBUG 13
INFO 61
```

```
awk '$3 == "ERROR"' sample-logs.md
```

`$3 == "ERROR"` → Checks if the 3rd column equals `"ERROR"`.

If true, the line is displayed.

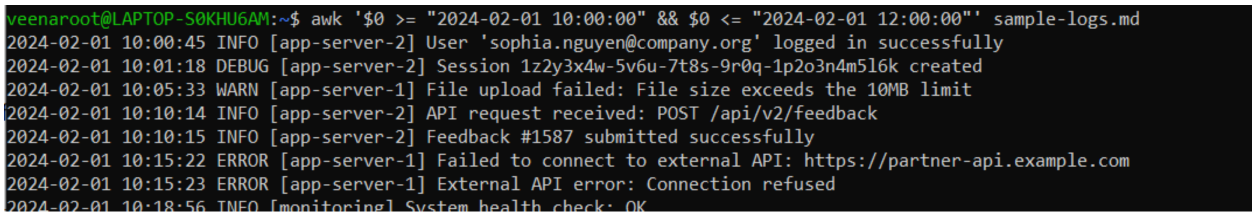


```
veenaroot@LAPTOP-S0KHU6AM:~$ awk '$3 == "ERROR"' sample-logs.md
2024-02-01 07:32:18 ERROR [app-server-1] Failed to connect to payment gateway: Connection timed out
2024-02-01 07:32:19 ERROR [app-server-1] Transaction 392841 failed: PAYMENT_GATEWAY_ERROR
2024-02-01 07:48:12 ERROR [app-server-1] Invalid request: Missing required field 'authorization'
2024-02-01 08:10:14 ERROR [app-server-1] Database connection lost
2024-02-01 08:35:27 ERROR [app-server-1] Failed to process payment: INVALID_CARD_NUMBER
2024-02-01 08:55:27 ERROR [app-server-2] 500 Internal Server Error: Java heap space
2024-02-01 09:18:56 ERROR [app-server-1] Missing required configuration: SMTP_PASSWORD
2024-02-01 09:20:11 ERROR [notification-service] Failed to send email notification: Authentication failed
2024-02-01 09:45:30 ERROR [app-server-1] Database query failed: Deadlock detected
2024-02-01 10:15:22 ERROR [app-server-1] Failed to connect to external API: https://partner-api.example.com
2024-02-01 10:15:23 ERROR [app-server-1] External API error: Connection refused
2024-02-01 10:40:11 ERROR [app-server-1] 404 Not Found: Resource /api/v1/legacy_endpoint no longer exists
```

```
awk '$0 >= "2024-02-01 10:00:00" && $0 <= "2024-02-01 12:00:00"'
sample-logs.md
```

This `awk` command filters log entries from `sample-logs.md` that fall within the timestamp range **2024-02-01 10:00:00 to 2024-02-01 12:00:00**.

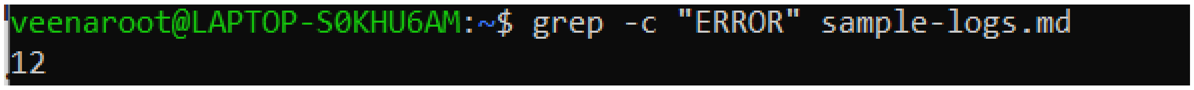
- `$0` represents the **entire line** (assuming logs start with a timestamp).
- It prints lines where the timestamp falls within the given range.



```
veenaroot@LAPTOP-S0KHU6AM:~$ awk '$0 >= "2024-02-01 10:00:00" && $0 <= "2024-02-01 12:00:00"' sample-logs.md
2024-02-01 10:00:45 INFO [app-server-2] User 'sophia.nguyen@company.org' logged in successfully
2024-02-01 10:01:18 DEBUG [app-server-2] Session 1z2y3x4w-5v6u-7t8s-9r0q-1p2o3n4m5l6k created
2024-02-01 10:05:33 WARN [app-server-1] File upload failed: File size exceeds the 10MB limit
2024-02-01 10:10:14 INFO [app-server-2] API request received: POST /api/v2/feedback
2024-02-01 10:10:15 INFO [app-server-2] Feedback #1587 submitted successfully
2024-02-01 10:15:22 ERROR [app-server-1] Failed to connect to external API: https://partner-api.example.com
2024-02-01 10:15:23 ERROR [app-server-1] External API error: Connection refused
2024-02-01 10:18:56 INFO [monitoring] System health check: OK
```

```
grep -c "ERROR" sample-logs.md
```

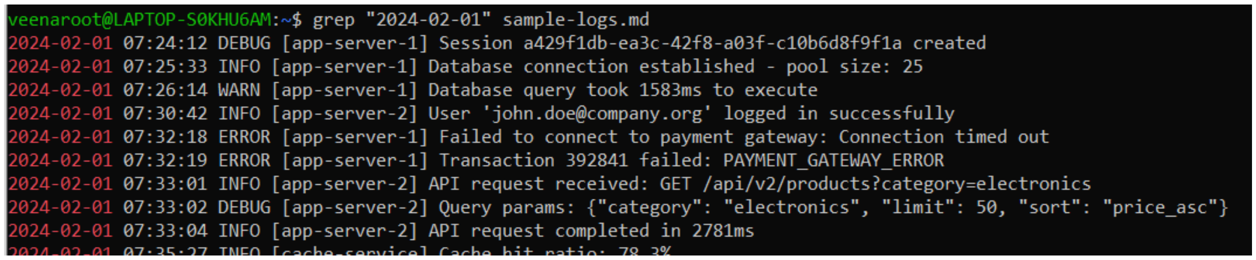
`grep -c "ERROR" sample-logs.md` counts the number of lines in `sample-logs.md` that contain the word **"ERROR"** and prints the count.



```
veenaroot@LAPTOP-S0KHU6AM:~$ grep -c "ERROR" sample-logs.md
12
```

```
grep "2024-02-01" sample-logs.md
```

`grep "2024-02-01" sample-logs.md` searches for all lines in `sample-logs.md` that contain the date **"2024-02-01"** and prints those matching lines.



```
veenaroot@LAPTOP-S0KHU6AM:~$ grep "2024-02-01" sample-logs.md
2024-02-01 07:24:12 DEBUG [app-server-1] Session a429f1db-ea3c-42f8-a03f-c10b6d8f9f1a created
2024-02-01 07:25:33 INFO [app-server-1] Database connection established - pool size: 25
2024-02-01 07:26:14 WARN [app-server-1] Database query took 1583ms to execute
2024-02-01 07:30:42 INFO [app-server-2] User 'john.doe@company.org' logged in successfully
2024-02-01 07:32:18 ERROR [app-server-1] Failed to connect to payment gateway: Connection timed out
2024-02-01 07:32:19 ERROR [app-server-1] Transaction 392841 failed: PAYMENT_GATEWAY_ERROR
2024-02-01 07:33:01 INFO [app-server-2] API request received: GET /api/v2/products?category=electronics
2024-02-01 07:33:02 DEBUG [app-server-2] Query params: {"category": "electronics", "limit": 50, "sort": "price_asc"}
2024-02-01 07:33:04 INFO [app-server-2] API request completed in 2781ms
2024-02-01 07:35:27 INFO [cache-service] Cache hit ratio: 78.3%
```

```
awk 'BEGIN {OFS=","} {print $1, $2, $3}' sample-logs.md >> output.csv
```

`awk 'BEGIN {OFS=","} {print $1, $2, $3}' sample-logs.md >> output.csv`
→ Extracts the first three fields (assumed to be Date, Level, and Server) from `sample-logs.md`, separates them with commas, and appends the output to `output.csv`.

Crontab

`crontab` (short for "cron table") is a command in Linux used to schedule and manage recurring tasks (cron jobs). It allows users to automate commands or scripts at specific time intervals (e.g., every minute, daily, weekly).

`crontab -e` → Opens the user's crontab file for editing to schedule recurring tasks.

`crontab -l` → Lists the currently scheduled cron jobs.

`(crontab -l 2>/dev/null; echo "* * * * * echo \"helloworld\"") | crontab -`

- Adds a cron job that **runs every minute (`* * * * *`) and echoes "helloworld"**.
- `crontab -l 2>/dev/null` lists existing cron jobs (redirecting errors if no jobs exist).
- `echo "* * * * * echo \"helloworld\""` appends a new job.
- `| crontab -` updates the crontab.

`crontab -l | grep -v "echo \"helloworld\"" | crontab -`

- Removes the cron job that echoes "helloworld".
- `crontab -l` lists jobs.
- `grep -v "echo \"helloworld\""` filters out the line containing "helloworld".
- `| crontab -` updates the crontab without that job.

```
* * * * * echo "Hello, World!" >> /home/user/log.txt
```

Runs every minute and appends "Hello, World!" to `log.txt`

```
echo "0 6 * * * /path/to/script.sh" | crontab -
```

Adds a cron job to run `/path/to/script.sh` every day at 6 AM without opening the editor.

```
crontab -r
```

Deletes all scheduled cron jobs for the current user.