Comprehensive Log Analysis Guide

1. General Log Analysis

Using grep

1. **Find lines with specific keywords (case-insensitive):**

   ```
   grep -i "error" application.log
   ```

   - **What It Does:** Searches for lines containing the keyword "error" (case-insensitive).
   - **Example Use Case:** Identify all error occurrences in a log file.
2. **Search for multiple keywords:**

   ```
   grep -iE "error|failed|exception" application.log
   ```

   - **What It Does:** Matches lines containing "error", "failed", or "exception".
   - **Example Use Case:** Debugging application logs for different error patterns.
3. **Count occurrences of a keyword:**

   ```
   grep -i "timeout" application.log | wc -l
   ```

   - **What It Does:** Counts all lines containing "timeout".
   - **Example Output:** 45 (indicating 45 timeout errors).
4. **Find lines with context (before/after matches):**

   ```
   grep -i "error" application.log -A 5 -B 3
   ```

   - **What It Does:** Displays 5 lines after and 3 lines before each match.
   - **Example Use Case:** Analyze logs surrounding errors for better context.
5. **Exclude lines containing specific keywords:**

   ```
   grep -i "error" application.log | grep -vi "debug"
   ```

   - **What It Does:** Filters out lines containing "debug" from the error matches.
6. **Search for lines matching specific patterns (e.g., IP addresses):**

   ```
   grep -oE "([0-9]{1,3}\.){3}[0-9]{1,3}" access.log
   ```

   - **What It Does:** Extracts IPv4 addresses from the log file.

7. **Monitor logs in real-time for specific issues:**

   ```
   tail -f system.log | grep --line-buffered -i "critical"
   ```

   - **What It Does:** Streams new log entries and filters for "critical".
8. **Search for lines within a specific date range:**

   ```
   grep -E "2025-01-05|2025-01-06" application.log | grep -i "error"
   ```

   - **What It Does:** Finds errors logged on specific dates.

Using awk

1. **Print specific fields from matched lines:**

   ```
   awk '/error/ {print $1, $2, $5}' application.log
   ```

   - **What It Does:** Prints the first, second, and fifth fields from lines containing "error".
2. **Summarize errors by hour:**

   ```
   awk '/error/ {split($2, time, ":"); hour[time[1]]++} END {for (h in hour) print h, hour[h]}' application.log
   ```

   - **What It Does:** Groups and counts errors by the hour.
   - **Example Output:**

     ```
     10 35
     11 28
     12 40
     ```
3. **Summarize errors by type:**

   ```
   awk '/error/ {type[$3]++} END {for (t in type) print t, type[t]}' application.log | sort -nr -k2
   ```

   - **What It Does:** Groups and counts errors based on the third field.
4. **Calculate average response times:**

   ```
   awk '{sum += $NF} END {print "Average response time:", sum/NR}' access.log
   ```

   - **What It Does:** Computes the average response time, assuming the last field is the time.

Using sed

1. **Highlight specific keywords:**

   ```
   sed 's/error/\x1b[31m&\x1b[0m/Ig' application.log
   ```

   - **What It Does:** Highlights "error" in red for improved readability.
2. **Extract lines between specific markers:**

   ```
   sed -n '/START_ERROR/,/END_ERROR/p' application.log
   ```

   - **What It Does:** Extracts all lines between START_ERROR and END_ERROR.
3. **Remove sensitive information (e.g., email addresses):**

   ```
   sed -E 's/[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}/[REDACTED]/g' logs.log
   ```

   - **What It Does:** Masks email addresses in the logs.

Using head and tail

1. **Display the first 20 lines of a log file:**

   ```
   head -20 application.log
   ```

   - **What It Does:** Shows the first 20 lines.
2. **Display the last 50 lines containing "error":**

   ```
   grep -i "error" application.log | tail -50
   ```

   - **What It Does:** Extracts the last 50 matches for "error".
3. **Monitor real-time logs:**

   ```
   tail -f application.log
   ```

   - **What It Does:** Streams new log entries in real-time.

---

2. Database Log Analysis

Using grep

1. **Find Oracle errors (e.g., ORA- codes):**

   ```
   grep "ORA-" oracle.log
   ```

   - **What It Does:** Matches lines with Oracle error codes.
2. **Search for slow queries:**

   ```
   grep "execution time: [5-9][0-9]*ms" db.log
   ```

   - **What It Does:** Matches SQL queries taking over 5 seconds.

Using awk

1. **Summarize Oracle errors by code:**

   ```
   awk '/ORA-/ {split($0, a, ":"); print a[1]}' oracle.log | sort | uniq -c | sort -nr
   ```

   - **What It Does:** Groups and counts occurrences of each Oracle error code.
2. **Identify largest table space usage:**

   ```
   awk '/tablespace/ {print $2, $5}' oracle.log | sort -k2 -nr | head -5
   ```

   - **What It Does:** Lists the top 5 table spaces by usage.

---

3. JVM Log Analysis

Using grep

1. **Find memory-related errors:**

   ```
   grep -i "OutOfMemoryError" jvm.log
   ```

   - **What It Does:** Matches memory-related errors in JVM logs.

2. **Search for specific exceptions:**

   grep -A 10 -B 5 "NullPointerException" jvm.log

   - **What It Does:** Displays 10 lines after and 5 lines before NullPointerException.

---

4. Kubernetes Log Analysis

Using grep

1. **Find pods in restart loops:**

   grep -i "CrashLoopBackOff" kube.log

   - **What It Does:** Matches pods stuck in CrashLoopBackOff.
2. **Detect out-of-memory kills:**

   grep -i "OOMKilled" kube.log

   - **What It Does:** Finds containers terminated due to memory issues.

Using awk

1. **Count restarts for each pod:**

   awk '/Restarted/ {pod=$2; restarts[pod]++} END {for (p in restarts) print restarts[p], p}' kube.log | sort -nr | head -10

   - **What It Does:** Lists the top 10 pods with the highest restarts.

---

5. AWS Log Analysis

Using grep

1. **Find Lambda errors:**

   ```
   grep -i "error" aws-lambda.log | grep -i "function"
   ```

   - **What It Does:** Matches errors related to AWS Lambda functions.
2. **Search for API Gateway 5xx errors:**

   ```
   grep '"status":5[0-9][0-9]' api-gateway.log
   ```

   - **What It Does:** Matches HTTP 5xx status codes.

---

6. Automating Log Analysis with Scripts

Daily Error Summary Script

```
#!/bin/bash

LOGFILE="application.log"
OUTPUT="error_summary_$(date +%F).txt"

echo "Error Summary for $(date)" > "$OUTPUT"
echo "----------------------------------" >> "$OUTPUT"

# Count total errors
echo "Total Errors:" >> "$OUTPUT"
grep -i "error" "$LOGFILE" | wc -l >> "$OUTPUT"

# Summarize errors by type
echo "Error Types:" >> "$OUTPUT"
grep -i "error" "$LOGFILE" | awk '{print $3}' | sort | uniq -c | sort -nr >> "$OUTPUT"

# Show the top 10 error lines
echo "Top 10 Errors:" >> "$OUTPUT"
grep -i "error" "$LOGFILE" | head -10 >> "$OUTPUT"

echo "Summary generated successfully at $(date)." >> "$OUTPUT"
```

---

7. Combining Tools for Advanced Use Cases

Real-Time Monitoring with Highlighting

```
tail -f system.log | grep --line-buffered -i "critical" | sed 's/critical/\x1b[31m&\x1b[0m/I'
```

- **What It Does:** Monitors new log entries, highlights "critical" in red, and displays them in real-time.
