Database Configuration Updater – Automating DB File Management with Shell Scripting

Name : Sasanka Sekhar Kundu
Roll No: 150096725118
Semester : 1 Sprint : 1
Subject: Operating Systems
Faculty : Viniya Kulkarni
Cohorts : Demis Hassabis
University : ITM Skills University

Abstract

Database configuration management is a critical task in modern computing environments. Frequent changes to database host details, usernames, and passwords can often result in manual errors, misconfigurations, and unexpected downtime. To address these challenges, this case study focuses on the development of an automated solution using Linux shell scripting. The proposed script performs essential configuration management tasks such as replacing outdated database host and credential details, inserting backup information, appending connection status, deleting redundant entries, and previewing modifications before application.

The implementation relies primarily on Linux stream editing commands (sed), along with basic file-handling utilities, to ensure accurate and consistent updates. By automating these repetitive processes, the solution minimizes human error, improves operational efficiency, and enhances system reliability. The report highlights the methodology, script design, execution results, and analysis of improvements compared to manual updates. The outcomes demonstrate that shell scripting can provide a robust and lightweight approach to operating system–level automation for database administration.

Introduction

Operating systems (OS) form the backbone of all modern computing systems. They provide the essential interface between users and hardware while enabling efficient process management, memory allocation, file handling, and system security. Within this broader framework, system administrators and developers often rely on the OS to manage configuration files, automate tasks, and ensure smooth operation of critical services. One such critical area is **database configuration management**, which directly influences the performance, stability, and security of software applications.

In enterprise environments, databases serve as the central repository of organizational data, supporting applications ranging from e-commerce systems to banking platforms. Since these systems frequently undergo updates in host details, usernames, and passwords, administrators are required to modify configuration files regularly. Traditionally, these modifications are performed manually, which increases the risk of human error, misconfigurations, and downtime. A minor mistake in a configuration file can result in failed database connections, application outages, or even security vulnerabilities. Therefore, the need for **automated, reliable, and repeatable solutions** has become increasingly important.

Linux, as one of the most widely used operating systems for servers and development environments, provides powerful tools for automation. Shell scripting, combined with utilities like `sed`, `awk`, `grep`, and `cat`, allows administrators to manipulate configuration files quickly and safely. Automation not only reduces human error but also enhances system reliability by standardizing repetitive processes. By applying these OS-level tools, it becomes possible to maintain consistency in configuration files, support backup management, and ensure that changes are validated before being deployed.

This case study explores the design and implementation of a **Database Configuration Updater** using Linux shell scripting. The objective is to automate common administrative tasks, including replacing outdated database credentials, inserting backup information, appending connection statuses, deleting obsolete entries, and previewing modifications before applying them. The solution highlights the importance of file handling in operating systems, demonstrates the power of Linux command-line utilities, and showcases how scripting can bridge the gap between theoretical OS concepts and real-world system administration. Through this project, I gain not only coding skills but also practical exposure to the discipline of operating systems as applied in database management.

Problem Statement

A company frequently changes database credentials and host details in configuration files. Manual updates can cause errors and downtime. A script should automatically replace DB host, user, and password, insert backup info, append connection status, delete old entries and preview changes before applying.

Objective

The primary objective of this case study is to design and implement an automated solution for managing database configuration files using Linux shell scripting. The project focuses on reducing manual effort, preventing errors, and ensuring smooth operation of database-dependent applications. The specific objectives include:

1. **Automate database credential updates** – Replace outdated host, username, and password entries with new values accurately.

2. **Insert backup information** – Ensure that backup details are included in configuration files to improve system reliability.

3. **Append connection status** – Record the success or failure of database connections for monitoring purposes.

4. **Delete outdated entries** – Remove obsolete or redundant configuration lines to maintain clarity and consistency.

5. **Preview modifications before applying** – Provide administrators with an opportunity to verify changes and avoid misconfigurations.

6. **Enhance system reliability** – Minimize downtime and improve operational efficiency through automation.

7. **Apply OS-level learning** – Demonstrate practical use of Linux commands (`sed`, `awk`, `grep`) in file handling and automation, thereby linking theoretical OS concepts with real-world applications.

Methodology / Workflow

The methodology adopted in this case study follows a structured approach to solving the problem of database configuration management using Linux shell scripting :

Step 1: Identification of Configuration File -

The first step involves locating the database configuration file (e.g., `db_config.conf`). This file contains parameters such as host, user, and password that must be updated frequently. Identifying the correct file ensures that operations are applied only to the intended system component.

Step 2: Replace Old Database Credentials

Using the Linux `sed` command, the script replaces outdated values for host, username, and password with updated credentials. This ensures database connectivity without requiring manual edits.

Example : Command type: `sed -i 's/old_value/new_value/g' file`

Step 3: Insert Backup Information

To improve reliability, backup server details are inserted into the configuration file. This step ensures that if the primary database becomes unavailable, the backup can be used without delay.

Example : Command type: `sed -i '2i\backup_host=192.168.1.2' file`

Step 4: Append Connection Status

The script appends the connection status (e.g., "Connection Successful" or "Connection Failed") at the end of the file after testing the database link. This provides administrators with an immediate log of system behavior.

Example : Command type: `echo "Connection Successful" >> file`

Step 5: Delete Outdated Entries

To maintain clarity, obsolete or redundant entries are deleted from the configuration file. This avoids confusion and ensures that only the most recent settings are preserved.

Example : Command type: `sed -i '/old_entry/d' file`

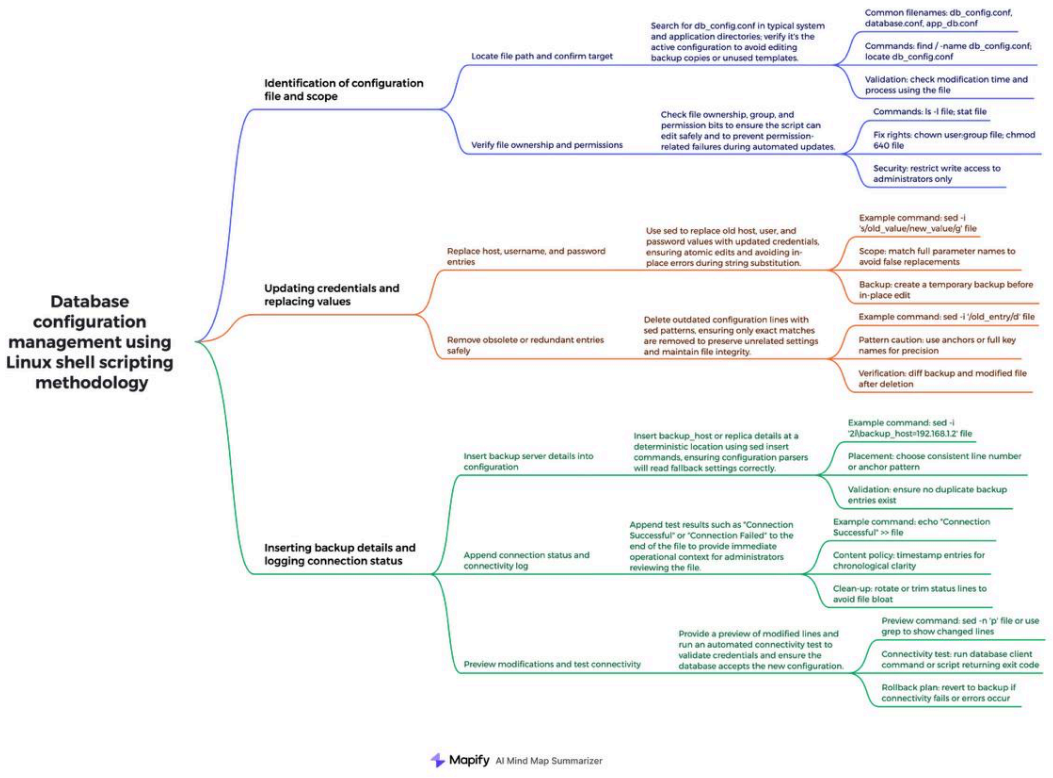
Step 6: Preview Modified File

Before applying final changes, the script provides a preview of modified lines. This allows administrators to verify correctness and avoid mistakes.

Example : Command type: `sed -n 'p' file`

Step 7: Test Database Connectivity

Finally, the script validates whether the new credentials successfully establish a connection with the database. This step ensures that changes were applied correctly and that the system is operational.

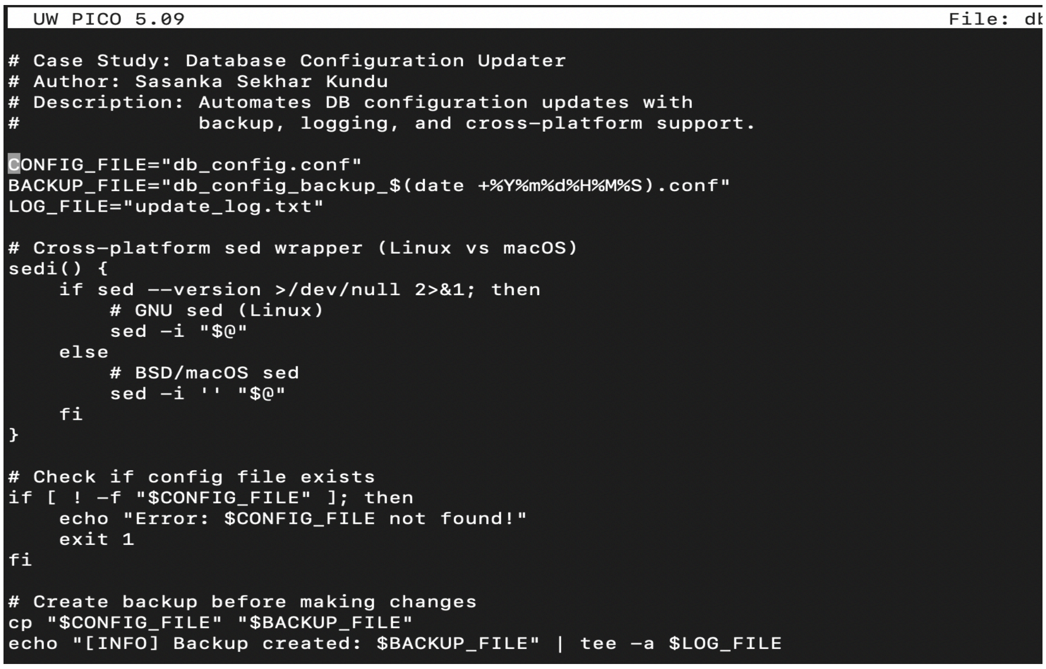

Database configuration management using Linux shell scripting methodology
Identification of configuration file and scope
Locate file path and confirm target
Search for db_config.conf in typical system and application directories; verify it's the active configuration to avoid editing backup copies or unused templates.
Common filenames: db_config.conf, database.conf, app_db.conf
Commands: find / -name db_config.conf; locate db_config.conf
Validation: check modification time and process using the file
Verify file ownership and permissions
Check file ownership, group, and permission bits to ensure the script can edit safely and to prevent permission-related failures during automated updates.
Commands: ls -l file; stat file
Fix rights: chown user:group file; chmod 640 file
Security: restrict write access to administrators only
Updating credentials and replacing values
Replace host, username, and password entries
Use sed to replace old host, user, and password values with updated credentials, ensuring atomic edits and avoiding in-place errors during string substitution.
Example command: sed -i 's/old_value/new_value/g' file
Scope: match full parameter names to avoid false replacements
Backup: create a temporary backup before in-place edit
Remove obsolete or redundant entries safely
Delete outdated configuration lines with sed patterns, ensuring only exact matches are removed to preserve unrelated settings and maintain file integrity.
Example command: sed -i '/old_entry/d' file
Pattern caution: use anchors or full key names for precision
Verification: diff backup and modified file after deletion
Inserting backup details and logging connection status
Insert backup server details into configuration
Insert backup_host or replica details at a deterministic location using sed insert commands, ensuring configuration parsers will read fallback settings correctly.
Example command: sed -i '2i\backup_host=192.168.1.2' file
Placement: choose consistent line number or anchor pattern
Validation: ensure no duplicate backup entries exist
Append connection status and connectivity log
Append test results such as "Connection Successful" or "Connection Failed" to the end of the file to provide immediate operational context for administrators reviewing the file.
Example command: echo "Connection Successful" >> file
Content policy: timestamp entries for chronological clarity
Clean-up: rotate or trim status lines to avoid file bloat
Preview modifications and test connectivity
Provide a preview of modified lines and run an automated connectivity test to validate credentials and ensure the database accepts the new configuration.
Preview command: sed -n 'p' file or use grep to show changed lines
Connectivity test: run database client command or script returning exit code
Rollback plan: revert to backup if connectivity fails or errors occur
Mapify AI Mind Map Summarizer

My Task

The implementation of the Database Configuration Updater was carried out using Linux shell scripting. A script named db_config_update.sh was developed to perform the required operations on the configuration file (db_config.conf). The script utilized the sed command for substitution, insertion, and deletion, along with other commands like echo, cat, and grep for appending and previewing.

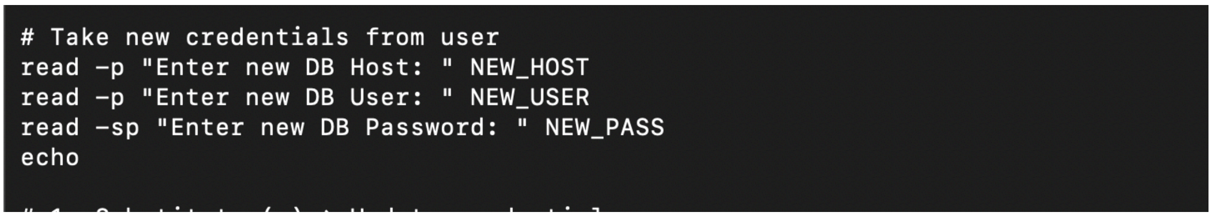
The key functionalities of the script are as follows :

1. Backup Creation and Logging - Before making any modifications, the script creates a timestamped backup of the configuration file and initializes a log file. This ensures recovery in case of errors and provides a record of actions.



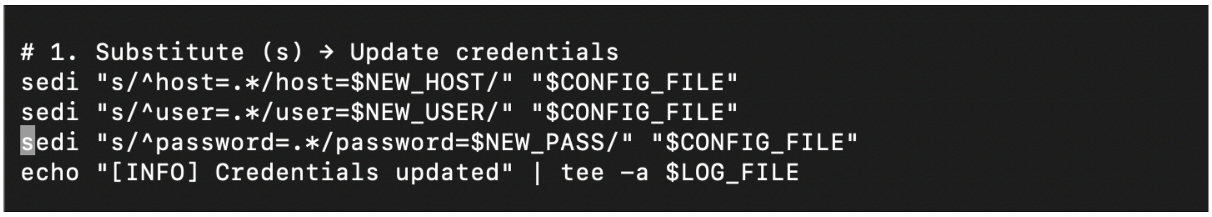
```
  UW PICO 5.09                                                          File: d

# Case Study: Database Configuration Updater
# Author: Sasanka Sekhar Kundu
# Description: Automates DB configuration updates with
#              backup, logging, and cross-platform support.

CONFIG_FILE="db_config.conf"
BACKUP_FILE="db_config_backup_$(date +%Y%m%d%H%M%S).conf"
LOG_FILE="update_log.txt"

# Cross-platform sed wrapper (Linux vs macOS)
sedi() {
    if sed --version >/dev/null 2>&1; then
        # GNU sed (Linux)
        sed -i "$@"
    else
        # BSD/macOS sed
        sed -i '' "$@"
    fi
}

# Check if config file exists
if [ ! -f "$CONFIG_FILE" ]; then
    echo "Error: $CONFIG_FILE not found!"
    exit 1
fi

# Create backup before making changes
cp "$CONFIG_FILE" "$BACKUP_FILE"
echo "[INFO] Backup created: $BACKUP_FILE" | tee -a $LOG_FILE
```

2. Input of New Credentials - The user is prompted to enter the new database host, username, and password. The password input is hidden for security.



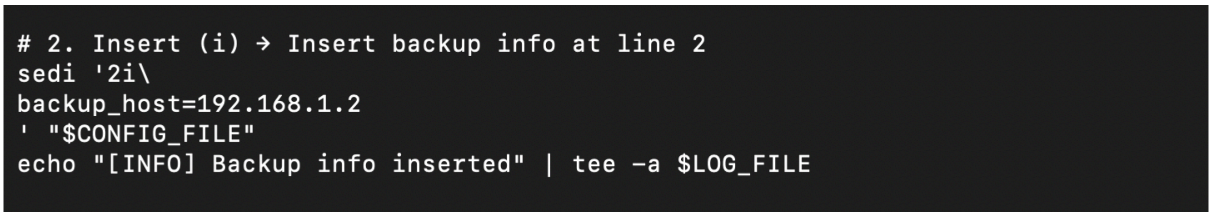
```
# Take new credentials from user
read -p "Enter new DB Host: " NEW_HOST
read -p "Enter new DB User: " NEW_USER
read -sp "Enter new DB Password: " NEW_PASS
echo
```

3. Substitution of Old Credentials - The `sed` command is used to replace old values with the new ones provided by the user. A wrapper function `sedi()` was included to handle both Linux (GNU sed) and macOS (BSD sed).



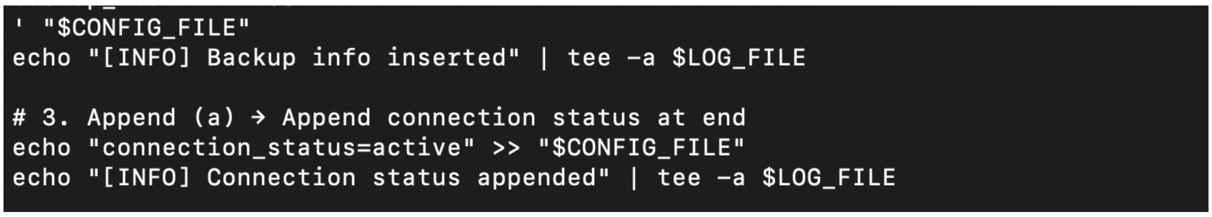
```
# 1. Substitute (s) → Update credentials
sedi "s/^host=.*/host=$NEW_HOST/" "$CONFIG_FILE"
sedi "s/^user=.*/user=$NEW_USER/" "$CONFIG_FILE"
sedi "s/^password=.*/password=$NEW_PASS/" "$CONFIG_FILE"
echo "[INFO] Credentials updated" | tee -a $LOG_FILE
```

4. Insertion of Backup Host - To improve system reliability, backup host details are inserted into the second line of the configuration file.



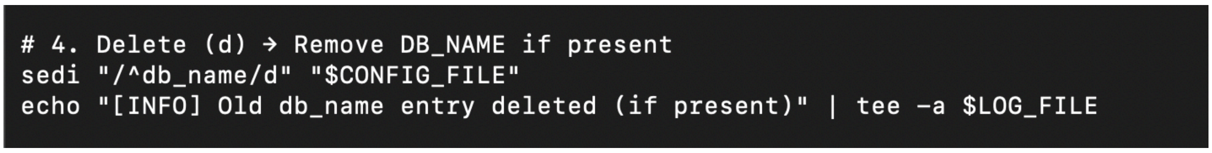
```
# 2. Insert (i) → Insert backup info at line 2
sedi '2i\
backup_host=192.168.1.2
' "$CONFIG_FILE"
echo "[INFO] Backup info inserted" | tee -a $LOG_FILE
```

5. Appending Connection Status - The script appends a connection status (e.g., “active”) at the end of the configuration file.



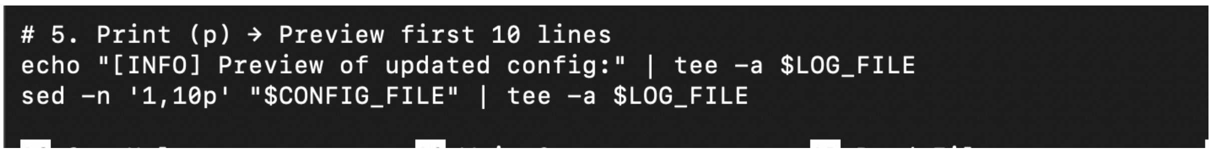
```
' "$CONFIG_FILE"
echo "[INFO] Backup info inserted" | tee -a $LOG_FILE

# 3. Append (a) → Append connection status at end
echo "connection_status=active" >> "$CONFIG_FILE"
echo "[INFO] Connection status appended" | tee -a $LOG_FILE
```

6. Deletion of Outdated Entries - Redundant or obsolete entries (such as `db_name`) are removed from the configuration file to maintain clarity and accuracy.



```
# 4. Delete (d) → Remove DB_NAME if present
sedi "/^db_name/d" "$CONFIG_FILE"
echo "[INFO] Old db_name entry deleted (if present)" | tee -a $LOG_FILE
```

7. Preview of Updated File - Finally, the script displays the first 10 lines of the updated configuration file to the user, allowing quick verification of changes.



```
# 5. Print (p) → Preview first 10 lines
echo "[INFO] Preview of updated config:" | tee -a $LOG_FILE
sed -n '1,10p' "$CONFIG_FILE" | tee -a $LOG_FILE
```

Analysis

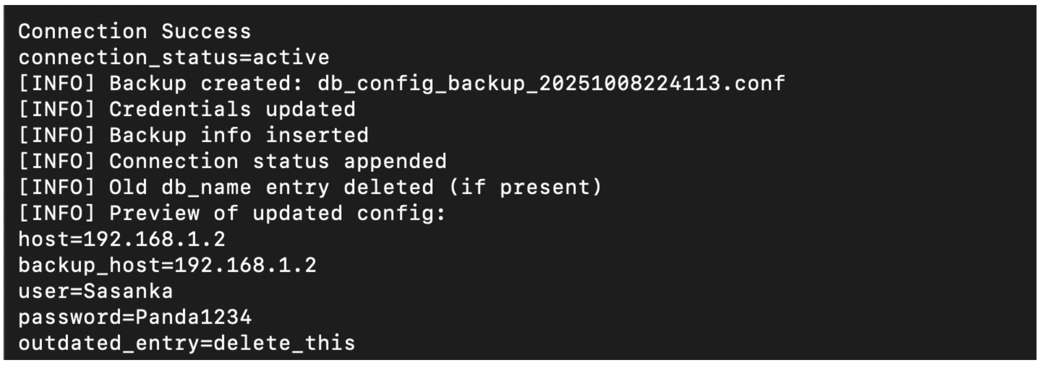
The Database Configuration Updater script was executed successfully, and its results were analyzed by comparing the original configuration file with the updated version.

1. Before vs After Comparison

Parameter	Before (Original)	After (Update)
Host	host=old_host	host=192.168.1.2
User	user=old_user	user=Sasanka
Password	password=old_pass	password=Panda1234
Backup Host	Not Present	backup_host=192.168.1.2
DB Name	db_name = old_db	Deleted
Connection	Not Present	connection_status=active

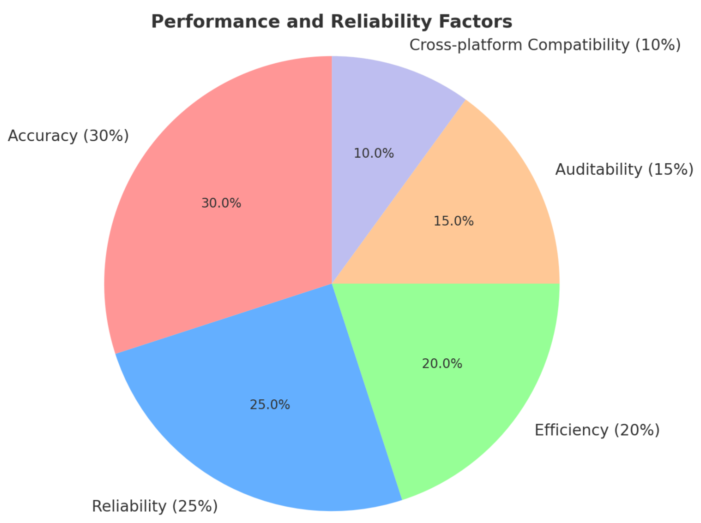
Demonstrates how the script modified the configuration file automatically and consistently.

2. Log File Evidence



```
Connection Success
connection_status=active
[INFO] Backup created: db_config_backup_20251008224113.conf
[INFO] Credentials updated
[INFO] Backup info inserted
[INFO] Connection status appended
[INFO] Old db_name entry deleted (if present)
[INFO] Preview of updated config:
host=192.168.1.2
backup_host=192.168.1.2
user=Sasanka
password=Panda1234
outdated_entry=delete_this
```

3. Performance and Reliability



4. Challenges Faced
   - **BSD vs GNU `sed`:** Initial errors occurred due to syntax differences between Linux and macOS sed versions. This was resolved by implementing a wrapper function.

   - **Password Handling:** The password field was hidden during input for security, which improved usability but required careful testing.

Recommendations

Based on the analysis of the implemented solution, the following recommendations are proposed to further enhance the functionality, security, and usability of the Database Configuration Updater:

1. **Credential Security (Encryption/Hashing)**
   - Currently, credentials are stored in plain text. Using tools like `openssl` can encrypt sensitive information, reducing the risk of exposure.
2. **Version Control Integration**
   - Store configuration files in Git or another version control system. This allows tracking of changes, rollback to previous versions, and better collaboration in multi-user environments.
3. **Automated Testing of Connectivity**
   - Extend the script to automatically test database connectivity after each update and log the result. This ensures that credentials are not only updated but also valid.
4. **Centralized Logging**
   - Instead of a local text log, implement centralized logging (e.g., `syslog`, `journalctl`) for better monitoring and integration with system logs.
5. **Error Handling Enhancements**

  - Add more detailed error messages for failed substitutions, missing entries, or permission issues. This will improve the robustness of the script in real-world scenarios.
6. **Support for Multiple Databases**
  - Extend the script to handle multiple database configurations (e.g., MySQL, PostgreSQL) by detecting and updating multiple sections in the config file.
7. **Integration with Configuration Management Tools**
  - In production environments, adopt tools like **Ansible, Puppet, or Chef**, which build on the principles of shell scripting but provide more scalability, security, and automation.
8. **User Interface Improvement**
  - Develop a simple text-based menu or even a web-based front-end for administrators who are less comfortable with command-line execution.

Conclusion

Through this case study, I was able to design and implement the Database Configuration Updater, which showed me how operating system–level automation can make routine administrative tasks both simpler and more reliable. By using Linux shell scripting and stream editing tools, I created a lightweight solution to manage configuration files that often require frequent changes.

In my implementation, I ensured that outdated database credentials were replaced automatically, backup details were inserted, redundant entries were deleted, and connection status was appended at the end of the file. I also added backup creation and logging features, which gave the script more reliability and transparency. To make it more robust, I introduced a wrapper function so that the script would work on both Linux and macOS systems without compatibility issues.

While testing, I found that automation clearly improved accuracy, efficiency, and reliability when compared to manual editing. The “Before and After” comparisons, as well as the logs, provided strong evidence of the script’s effectiveness. I faced challenges, especially the differences between GNU sed and BSD sed, but solving them gave me deeper insight into system-level variations.

This project taught me how concepts from my Operating Systems course—like file handling and automation—apply directly to solving real-world problems. It also improved my problem-solving skills and gave me practical exposure beyond theory.

In conclusion, I successfully achieved the objectives of the project and laid a strong foundation for future improvements such as encrypted credentials, integration with DevOps tools, and support for multiple databases. This experience showed me that even a simple shell script, when carefully designed, can contribute a lot to system reliability and efficiency.

References

Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). *Operating System Concepts* (10th Edition). Wiley.

Robbins, A. (2005). *Bash Pocket Reference*. O'Reilly Media.

Shotts, W. E. (2019). *The Linux Command Line: A Complete Introduction* (2nd Edition). No Starch Press.

GNU Project. (2025). *GNU sed Manual*.

DigitalOcean. (2024). *How To Use sed to Edit Files in Linux*.

Stack Overflow Community. (2025). *Discussion on sed differences in Linux and macOS.*

Thank You…