# **12 Essential Bash Scripts for Streamlining Database Administration**

#### **Automated Database Backup Script (MySQL/PostgreSQL)**

#!/bin/bash

set -euo pipefail

# Configuration

DB\_TYPE="mysql" # or "postgres"

DB\_USER="${DB\_USER:-your\_db\_user}"

DB\_PASS="${DB\_PASS:-your\_db\_password}"

DB\_NAME="${DB\_NAME:-your\_database\_name}"

BACKUP\_DIR="/var/backups/databases"

TIMESTAMP=$(date +%Y%m%d\_%H%M%S)

LOG\_FILE="$BACKUP\_DIR/backup\_$DB\_TYPE\_$TIMESTAMP.log"

# Make sure required tools are available

command -v date >/dev/null || { echo "date command missing"; exit 1; }

command -v mkdir >/dev/null || { echo "mkdir command missing"; exit 1; }

# Create backup directory if it doesn't exist

mkdir -p "$BACKUP\_DIR"

# Backup logic

if [[ "$DB\_TYPE" == "mysql" ]]; then

command -v mysqldump >/dev/null || { echo "mysqldump not found"; exit 1; }

echo "Starting MySQL backup..." | tee -a "$LOG\_FILE"

mysqldump -u "$DB\_USER" -p"$DB\_PASS" "$DB\_NAME" > "$BACKUP\_DIR/$DB\_NAME-$TIMESTAMP.sql"

echo "MySQL backup created at $BACKUP\_DIR/$DB\_NAME-$TIMESTAMP.sql" | tee -a "$LOG\_FILE"

elif [[ "$DB\_TYPE" == "postgres" ]]; then

command -v pg\_dump >/dev/null || { echo "pg\_dump not found"; exit 1; }

echo "Starting PostgreSQL backup..." | tee -a "$LOG\_FILE"

PGPASSWORD="$DB\_PASS" pg\_dump -U "$DB\_USER" -h "localhost" -d "$DB\_NAME" > "$BACKUP\_DIR/$DB\_NAME-$TIMESTAMP.sql"

echo "PostgreSQL backup created at $BACKUP\_DIR/$DB\_NAME-$TIMESTAMP.sql" | tee -a "$LOG\_FILE"

else

echo "Unsupported DB\_TYPE: $DB\_TYPE" | tee -a "$LOG\_FILE"

exit 1

fi

This script automates database backups using mysqldump for MySQL and pg\_dump for PostgreSQL, storing them in a timestamped directory.

#### **2. Database Restore Script (MySQL/PostgreSQL)**

This script restores a database backup by using mysql or psql with the provided backup file.

#!/bin/bash

set -euo pipefail # Exit if any command fails

# Configuration - you should use ENV variables or secrets, espsecially in production

DB\_TYPE="${DB\_TYPE:-mysql}" # mysql or postgres

DB\_USER="${DB\_USER:-your\_db\_user}"

DB\_PASS="${DB\_PASS:-your\_db\_password}"

DB\_NAME="${DB\_NAME:-your\_database\_name}"

BACKUP\_FILE="$1"

# Log file

LOG\_FILE="/var/log/db\_restore\_$(date +%F).log"

# Ensure backup file was provided

if [[ -z "$BACKUP\_FILE" || ! -f "$BACKUP\_FILE" ]]; then

echo " Error: Backup file is missing or does not exist." | tee -a "$LOG\_FILE"

exit 1

fi

if [[ "$DB\_TYPE" == "mysql" ]]; then

echo "Starting MySQL restore..." | tee -a "$LOG\_FILE"

mysql -u "$DB\_USER" -p"$DB\_PASS" "$DB\_NAME" < "$BACKUP\_FILE"

echo "MySQL restore completed from: $BACKUP\_FILE" | tee -a "$LOG\_FILE"

elif [[ "$DB\_TYPE" == "postgres" ]]; then

echo "Starting PostgreSQL restore..." | tee -a "$LOG\_FILE"

PGPASSWORD="$DB\_PASS" psql -U "$DB\_USER" -h "localhost" -d "$DB\_NAME" -f "$BACKUP\_FILE"

echo "PostgreSQL restore completed from: $BACKUP\_FILE" | tee -a "$LOG\_FILE"

else

echo "Unsupported DB\_TYPE: $DB\_TYPE" | tee -a "$LOG\_FILE"

exit 1

fi

- It takes the backup file as an input parameter. - For MySQL, it uses < to feed the .sql file into the database. - For PostgreSQL, it uses psql -f to run the SQL file.

#### **MongoDB Backup Script**

#!/bin/bash

set -euo pipefail # Exit on error, undefined var, or pipe failure

# Configuration

DB\_NAME="${DB\_NAME:-your\_mongodb\_database}"

DB\_USER="${DB\_USER:-}" # Optional: provide via env or secure method

DB\_PASS="${DB\_PASS:-}" # Optional

AUTH\_DB="${AUTH\_DB:-admin}" # Usually 'admin'

BACKUP\_BASE\_DIR="/var/backups/mongodb"

TIMESTAMP=$(date +%Y%m%d\_%H%M%S)

BACKUP\_DIR="$BACKUP\_BASE\_DIR/$DB\_NAME-$TIMESTAMP"

LOG\_FILE="/var/log/mongodb\_backup\_$(date +%F).log"

# Ensure backup directory exists

mkdir -p "$BACKUP\_DIR"

# Build mongodump command

DUMP\_CMD="mongodump --db \"$DB\_NAME\" --out \"$BACKUP\_DIR\""

if [[ -n "$DB\_USER" && -n "$DB\_PASS" ]]; then

DUMP\_CMD+=" --username \"$DB\_USER\" --password \"$DB\_PASS\" --authenticationDatabase \"$AUTH\_DB\""

fi

# Run backup

echo "Running MongoDB backup for database: $DB\_NAME" | tee -a "$LOG\_FILE"

eval $DUMP\_CMD 2>&1 | tee -a "$LOG\_FILE"

if [[ $? -eq 0 ]]; then

echo "MongoDB backup created at: $BACKUP\_DIR" | tee -a "$LOG\_FILE"

else

echo "Backup failed for MongoDB database: $DB\_NAME" | tee -a "$LOG\_FILE"

exit 1

fi

- Uses mongodump to create a full backup of your MongoDB database. - Output is saved to a folder named with the timestamp.

#### **4. MongoDB Restore Script**

#!/bin/bash

set -euo pipefail # Exit on error, unset var, or pipe failure

# Configuration

DB\_NAME="${DB\_NAME:-your\_mongodb\_database}"

DB\_USER="${DB\_USER:-}" # Optional: provide via environment

DB\_PASS="${DB\_PASS:-}" # Optional

AUTH\_DB="${AUTH\_DB:-admin}" # Usually 'admin'

BACKUP\_DIR="${1:-}"

# Logging

LOG\_FILE="/var/log/mongodb\_restore\_$(date +%F).log"

# Validate input

if [[ -z "$BACKUP\_DIR" || ! -d "$BACKUP\_DIR" ]]; then

echo "ERROR: Backup directory not specified or does not exist." | tee -a "$LOG\_FILE"

exit 1

fi

# Build mongorestore command

RESTORE\_CMD="mongorestore --db \"$DB\_NAME\" \"$BACKUP\_DIR\""

if [[ -n "$DB\_USER" && -n "$DB\_PASS" ]]; then

RESTORE\_CMD+=" --username \"$DB\_USER\" --password \"$DB\_PASS\" --authenticationDatabase \"$AUTH\_DB\""

fi

echo "Starting MongoDB restore for database: $DB\_NAME" | tee -a "$LOG\_FILE"

eval $RESTORE\_CMD 2>&1 | tee -a "$LOG\_FILE"

if [[ $? -eq 0 ]]; then

echo "MongoDB restore completed from: $BACKUP\_DIR" | tee -a "$LOG\_FILE"

else

echo "MongoDB restore failed." | tee -a "$LOG\_FILE"

exit 1

fi

- Restores a MongoDB database from a previously created backup using mongorestore.

#### **5. Database User Creation Script (MySQL/PostgreSQL)**

#!/bin/bash

set -euo pipefail

# Configuration

DB\_TYPE="${DB\_TYPE:-mysql}" # Set via environment or default

DB\_USER="${DB\_USER:-your\_admin}"

DB\_PASS="${DB\_PASS:-your\_password}"

DB\_NAME="${DB\_NAME:-your\_database}" # Only for MySQL

NEW\_USER="${1:-}"

NEW\_PASS="${2:-}"

LOG\_FILE="/var/log/db\_user\_create\_$(date +%F).log"

# Validate input

if [[ -z "$NEW\_USER" || -z "$NEW\_PASS" ]]; then

echo "Error: Usage: $0 <username> <password>" | tee -a "$LOG\_FILE"

exit 1

fi

# MySQL User Creation

if [[ "$DB\_TYPE" == "mysql" ]]; then

echo "Creating MySQL user '$NEW\_USER'..." | tee -a "$LOG\_FILE"

mysql -u "$DB\_USER" -p"$DB\_PASS" -e "CREATE USER IF NOT EXISTS '$NEW\_USER'@'localhost' IDENTIFIED BY '$NEW\_PASS'; GRANT ALL PRIVILEGES ON \`${DB\_NAME//\`/}\`.\* TO '$NEW\_USER'@'localhost'; FLUSH PRIVILEGES;" \

&& echo "MySQL user '$NEW\_USER' created and granted access to $DB\_NAME." | tee -a "$LOG\_FILE" \

|| { echo "Failed to create MySQL user '$NEW\_USER'." | tee -a "$LOG\_FILE"; exit 1; }

# PostgreSQL User Creation

elif [[ "$DB\_TYPE" == "postgres" ]]; then

echo "Creating PostgreSQL user '$NEW\_USER'..." | tee -a "$LOG\_FILE"

PGPASSWORD="$DB\_PASS" psql -U "$DB\_USER" -d "postgres" -c "DO \$\$ BEGIN IF NOT EXISTS (SELECT FROM pg\_roles WHERE rolname = '$NEW\_USER') THEN CREATE USER \"$NEW\_USER\" WITH PASSWORD '$NEW\_PASS'; END IF; END \$\$;" \

&& echo "PostgreSQL user '$NEW\_USER' created." | tee -a "$LOG\_FILE" \

|| { echo "Failed to create PostgreSQL user '$NEW\_USER'." | tee -a "$LOG\_FILE"; exit 1; }

else

echo "Error: Unsupported DB\_TYPE '$DB\_TYPE'" | tee -a "$LOG\_FILE"

exit 1

fi

- For MySQL, it creates the user and grants full access. - For PostgreSQL, it uses PL/pgSQL to avoid duplicate users.

#### **6. Database Size Check Script (MySQL/PostgreSQL)**

Reports how much space your database is using.

#!/bin/bash

set -euo pipefail

# Configuration (use env vars or secure secrets store in production)

DB\_TYPE="${DB\_TYPE:-mysql}"

DB\_USER="${DB\_USER:-your\_db\_user}"

DB\_PASS="${DB\_PASS:-your\_db\_password}"

DB\_NAME="${DB\_NAME:-your\_database\_name}"

DB\_HOST="${DB\_HOST:-localhost}"

LOG\_FILE="/var/log/db\_size\_check\_$(date +%F).log"

# Validate database name

if [[ -z "$DB\_NAME" ]]; then

echo "Error: Database name is not set." | tee -a "$LOG\_FILE"

exit 1

fi

# Check size for MySQL

if [[ "$DB\_TYPE" == "mysql" ]]; then

echo "Checking MySQL database size for '$DB\_NAME'..." | tee -a "$LOG\_FILE"

mysql -u "$DB\_USER" -p"$DB\_PASS" -h "$DB\_HOST" -e "

SELECT table\_schema AS 'Database Name',

ROUND(SUM(data\_length + index\_length) / 1024 / 1024, 2) AS 'Database Size (MB)'

FROM information\_schema.TABLES

WHERE table\_schema = '$DB\_NAME'

GROUP BY table\_schema;" | tee -a "$LOG\_FILE"

# Check size for PostgreSQL

elif [[ "$DB\_TYPE" == "postgres" ]]; then

echo "Checking PostgreSQL database size for '$DB\_NAME'..." | tee -a "$LOG\_FILE"

PGPASSWORD="$DB\_PASS" psql -U "$DB\_USER" -h "$DB\_HOST" -d "$DB\_NAME" -c "

SELECT pg\_size\_pretty(pg\_database\_size('$DB\_NAME')) AS \"Database Size\";" | tee -a "$LOG\_FILE"

else

echo "Error: Unsupported DB\_TYPE '$DB\_TYPE'. Use 'mysql' or 'postgres'." | tee -a "$LOG\_FILE"

exit 1

fi

- MySQL calculates from table sizes, PostgreSQL uses pg\_database\_size.

#### **7. Database Connection Test Script (MySQL/PostgreSQL/MongoDB)**

#!/bin/bash

set -euo pipefail

# Configuration

DB\_TYPE="${DB\_TYPE:-mysql}"

DB\_USER="${DB\_USER:-your\_db\_user}"

DB\_PASS="${DB\_PASS:-your\_db\_password}"

DB\_NAME="${DB\_NAME:-your\_database\_name}"

DB\_HOST="${DB\_HOST:-localhost}"

DB\_PORT="${DB\_PORT:-}" # Optional

LOG\_FILE="/var/log/db\_connection\_check\_$(date +%F).log"

echo " Testing $DB\_TYPE connection..." | tee -a "$LOG\_FILE"

# MySQL

if [[ "$DB\_TYPE" == "mysql" ]]; then

CMD="mysql -u \"$DB\_USER\" -p\"$DB\_PASS\" -h \"$DB\_HOST\""

[[ -n "$DB\_PORT" ]] && CMD+=" -P \"$DB\_PORT\""

CMD+=" -e \"SELECT 1;\""

if eval $CMD &>/dev/null; then

echo "MySQL connection successful." | tee -a "$LOG\_FILE"

exit 0

else

echo "MySQL connection failed." | tee -a "$LOG\_FILE"

exit 1

fi

# PostgreSQL

elif [[ "$DB\_TYPE" == "postgres" ]]; then

export PGPASSWORD="$DB\_PASS"

CMD="psql -U \"$DB\_USER\" -d \"$DB\_NAME\" -h \"$DB\_HOST\""

[[ -n "$DB\_PORT" ]] && CMD+=" -p \"$DB\_PORT\""

CMD+=" -c \"SELECT 1;\""

if eval $CMD &>/dev/null; then

echo "PostgreSQL connection successful." | tee -a "$LOG\_FILE"

exit 0

else

echo "PostgreSQL connection failed." | tee -a "$LOG\_FILE"

exit 1

fi

# MongoDB

elif [[ "$DB\_TYPE" == "mongodb" ]]; then

CMD="mongo --host \"$DB\_HOST\""

[[ -n "$DB\_PORT" ]] && CMD+=" --port \"$DB\_PORT\""

CMD+=" --eval \"db.stats()\""

if eval $CMD &>/dev/null; then

echo "MongoDB connection successful." | tee -a "$LOG\_FILE"

exit 0

else

echo "MongoDB connection failed." | tee -a "$LOG\_FILE"

exit 1

fi

else

echo "Unsupported DB\_TYPE: $DB\_TYPE" | tee -a "$LOG\_FILE"

exit 1

fi

- Checks if a connection to the database is successful. - If it fails, it prints an error.

#### **8. Database Query Execution Script (MySQL/PostgreSQL)**

Executes any SQL query you pass as a command-line argument.

#!/bin/bash

set -euo pipefail

# Configuration

DB\_TYPE="${DB\_TYPE:-mysql}"

DB\_USER="${DB\_USER:-your\_db\_user}"

DB\_PASS="${DB\_PASS:-your\_db\_password}"

DB\_NAME="${DB\_NAME:-your\_database\_name}"

DB\_HOST="${DB\_HOST:-localhost}"

DB\_PORT="${DB\_PORT:-}"

QUERY="${1:-}"

LOG\_FILE="/var/log/db\_query\_exec\_$(date +%F).log"

# Validate query input

if [[ -z "$QUERY" ]]; then

echo "Error: No SQL query provided. Usage: $0 \"SELECT \* FROM table;\"" | tee -a "$LOG\_FILE"

exit 1

fi

echo "Executing query on $DB\_TYPE..." | tee -a "$LOG\_FILE"

# MySQL Execution

if [[ "$DB\_TYPE" == "mysql" ]]; then

CMD="mysql -u \"$DB\_USER\" -p\"$DB\_PASS\" -D \"$DB\_NAME\" -h \"$DB\_HOST\""

[[ -n "$DB\_PORT" ]] && CMD+=" -P \"$DB\_PORT\""

CMD+=" -e \"$QUERY\""

if eval $CMD; then

echo "MySQL query executed successfully." | tee -a "$LOG\_FILE"

else

echo "MySQL query failed." | tee -a "$LOG\_FILE"

exit 1

fi

# PostgreSQL Execution

elif [[ "$DB\_TYPE" == "postgres" ]]; then

export PGPASSWORD="$DB\_PASS"

CMD="psql -U \"$DB\_USER\" -h \"$DB\_HOST\" -d \"$DB\_NAME\""

[[ -n "$DB\_PORT" ]] && CMD+=" -p \"$DB\_PORT\""

CMD+=" -c \"$QUERY\""

if eval $CMD; then

echo "PostgreSQL query executed successfully." | tee -a "$LOG\_FILE"

else

echo "PostgreSQL query failed." | tee -a "$LOG\_FILE"

exit 1

fi

else

echo " Unsupported DB\_TYPE: $DB\_TYPE" | tee -a "$LOG\_FILE"

exit 1

fi

- This script executes an SQL query provided as an argument. - For MySQL, it uses mysql -e "$QUERY", and for PostgreSQL, it uses psql -c "$QUERY".

#### **9. Database User Deletion Script (MySQL/PostgreSQL)**

#!/bin/bash

set -euo pipefail

# Configuration

DB\_TYPE="${DB\_TYPE:-mysql}"

DB\_USER="${DB\_USER:-your\_db\_admin\_user}"

DB\_PASS="${DB\_PASS:-your\_db\_admin\_password}"

DB\_HOST="${DB\_HOST:-localhost}"

DB\_PORT="${DB\_PORT:-}"

USER\_TO\_DELETE="${1:-}"

# Validate input

if [[ -z "$USER\_TO\_DELETE" ]]; then

echo "Error: No username provided. Usage: $0 <username\_to\_delete>"

exit 1

fi

# Confirm deletion (optional safeguard)

read -p "Are you sure you want to delete user '$USER\_TO\_DELETE'? [y/N]: " CONFIRM

if [[ ! "$CONFIRM" =~ ^[Yy]$ ]]; then

echo "Operation cancelled."

exit 0

fi

# Delete MySQL user

if [[ "$DB\_TYPE" == "mysql" ]]; then

echo " Attempting to delete MySQL user '$USER\_TO\_DELETE'..."

CMD="mysql -u \"$DB\_USER\" -p\"$DB\_PASS\" -h \"$DB\_HOST\""

[[ -n "$DB\_PORT" ]] && CMD+=" -P \"$DB\_PORT\""

CMD+=" -e \"DROP USER IF EXISTS '$USER\_TO\_DELETE'@'localhost'; FLUSH PRIVILEGES;\""

if eval $CMD; then

echo "MySQL user '$USER\_TO\_DELETE' deleted."

else

echo "Failed to delete MySQL user '$USER\_TO\_DELETE'."

exit 1

fi

# Delete PostgreSQL user

elif [[ "$DB\_TYPE" == "postgres" ]]; then

echo " Attempting to delete PostgreSQL user '$USER\_TO\_DELETE'..."

export PGPASSWORD="$DB\_PASS"

CMD="psql -U \"$DB\_USER\" -h \"$DB\_HOST\" -d postgres"

[[ -n "$DB\_PORT" ]] && CMD+=" -p \"$DB\_PORT\""

CMD+=" -c \"DROP ROLE IF EXISTS $USER\_TO\_DELETE;\""

if eval $CMD; then

echo "PostgreSQL user '$USER\_TO\_DELETE' deleted."

else

echo " Failed to delete PostgreSQL user '$USER\_TO\_DELETE'."

exit 1

fi

else

echo " Unsupported DB\_TYPE: $DB\_TYPE"

exit 1

fi

- Deletes a user account from the database. - Make sure the user isn't connected or using the DB before deletion.

#### **10. Automate Backups for Multiple MySQL Databases**

For systems with multiple databases, use this script:

#!/bin/bash

set -euo pipefail

# Configuration

USER="${MYSQL\_USER:-your\_username}"

PASSWORD="${MYSQL\_PASS:-your\_password}"

HOST="${MYSQL\_HOST:-localhost}"

PORT="${MYSQL\_PORT:-3306}"

BACKUP\_DIR="/backups"

TIMESTAMP=$(date +"%F-%H-%M-%S")

LOG\_FILE="$BACKUP\_DIR/backup\_log\_$TIMESTAMP.log"

RETENTION\_DAYS=7

# Create backup directory

mkdir -p "$BACKUP\_DIR"

echo "Starting MySQL backup at $TIMESTAMP" | tee -a "$LOG\_FILE"

# Get database list, excluding system DBs

databases=$(mysql -u "$USER" -p"$PASSWORD" -h "$HOST" -P "$PORT" -e "SHOW DATABASES;" | grep -Ev "^(Database|information\_schema|performance\_schema|mysql|sys)$")

for db in $databases; do

BACKUP\_PATH="$BACKUP\_DIR/${db}\_$TIMESTAMP.sql"

echo "Backing up database: $db" | tee -a "$LOG\_FILE"

if mysqldump -u "$USER" -p"$PASSWORD" -h "$HOST" -P "$PORT" "$db" > "$BACKUP\_PATH"; then

echo "Backup successful: $BACKUP\_PATH" | tee -a "$LOG\_FILE"

else

echo "Backup failed for database: $db" | tee -a "$LOG\_FILE"

rm -f "$BACKUP\_PATH" # Remove incomplete backup

fi

done

# Optional cleanup

echo "Deleting backups older than $RETENTION\_DAYS days..." | tee -a "$LOG\_FILE"

find "$BACKUP\_DIR" -type f -name "\*.sql" -mtime +$RETENTION\_DAYS -exec rm -f {} \;

echo "All backups completed. Logs saved to $LOG\_FILE"

- It lists all user-created databases and backs them up one by one.

#### **11. Compress and Encrypt Backups**

To save space and secure backups, use gzip and openssl:

#!/bin/bash

set -euo pipefail

# Configuration

DB\_NAME="${DB\_NAME:-your\_database}"

USER="${MYSQL\_USER:-your\_username}"

PASSWORD="${MYSQL\_PASS:-your\_password}"

BACKUP\_DIR="${BACKUP\_DIR:-/backups}"

ENCRYPTION\_KEY="${ENCRYPTION\_KEY:-your\_secure\_password}"

TIMESTAMP=$(date +"%F-%H-%M-%S")

BACKUP\_FILE="$BACKUP\_DIR/$DB\_NAME-$TIMESTAMP.sql.gz.enc"

LOG\_FILE="$BACKUP\_DIR/backup\_log\_$(date +%F).log"

# Create backup directory

mkdir -p "$BACKUP\_DIR"

echo "Starting backup for '$DB\_NAME' at $TIMESTAMP..." | tee -a "$LOG\_FILE"

# Perform backup, compression, and encryption

if mysqldump -u "$USER" -p"$PASSWORD" "$DB\_NAME" | gzip | openssl enc -aes-256-cbc -salt -out "$BACKUP\_FILE" -pass pass:"$ENCRYPTION\_KEY"; then

echo "Backup completed: $BACKUP\_FILE" | tee -a "$LOG\_FILE"

else

echo "Backup failed for $DB\_NAME" | tee -a "$LOG\_FILE"

rm -f "$BACKUP\_FILE"

exit 1

fi

# Optional: Cleanup old backups

RETENTION\_DAYS=7

echo "Cleaning up backups older than $RETENTION\_DAYS days..." | tee -a "$LOG\_FILE"

find "$BACKUP\_DIR" -name "\*.sql.gz.enc" -mtime +$RETENTION\_DAYS -exec rm -f {} \;

echo "Finished backup routine." | tee -a "$LOG\_FILE"

- Dumps the database. - Compresses it with gzip. - Encrypts it with openssl.

#### **12. Schedule All Backups with a Single Script**

Create a master backup script for all databases:

#!/bin/bash

set -euo pipefail

# Configuration

LOG\_FILE="/var/log/db\_backup\_master.log"

TIMESTAMP=$(date +"%F %T")

echo " [$TIMESTAMP] Starting backup process..." | tee -a "$LOG\_FILE"

# Function to run a script with logging and error handling

run\_script() {

local script="$1"

echo "Running $script..." | tee -a "$LOG\_FILE"

if bash "$script" >> "$LOG\_FILE" 2>&1; then

echo " $script completed successfully." | tee -a "$LOG\_FILE"

else

echo " $script failed. Check logs." | tee -a "$LOG\_FILE"

exit 1

fi

}

# Paths to backup scripts

MYSQL\_SCRIPT="/opt/db-scripts/backup\_mysql.sh"

POSTGRES\_SCRIPT="/opt/db-scripts/backup\_postgres.sh"

MONGO\_SCRIPT="/opt/db-scripts/backup\_mongo.sh"

CLEANUP\_SCRIPT="/opt/db-scripts/cleanup\_backups.sh"

# Execute backup scripts

run\_script "$MYSQL\_SCRIPT"

run\_script "$POSTGRES\_SCRIPT"

run\_script "$MONGO\_SCRIPT"

run\_script "$CLEANUP\_SCRIPT"

echo "All backups completed successfully at $(date +"%F %T")" | tee -a "$LOG\_FILE"

- Runs all backup scripts and cleanup in one go.

#### **Automate with Cron:**

Add this to crontab to run at midnight:

Copy0 0 \* \* \* /path/to/master\_backup.sh