12 Essential Bash Scripts for Streamlining Database Administration

simplify your DBA workflows for MySQL, PostgreSQL, and MongoDB

Obafemi

Follow

DevOps.dev a11y-light ~10 min read · July 15, 2025 (Updated: July 15, 2025) · Free: No

1. Automated Database Backup Script (MySQL/PostgreSQL)

Copy

```bash
#!/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"
```

```
    command -v mysqldump >/dev/null || { echo "mysqldump not found"; exi

    echo "Starting MySQL backup..." | tee -a "$LOG_FILE"
    mysqldump -u "$DB_USER" -p"$DB_PASS" "$DB_NAME" > "$BACKUP_DIR/$DB_N
    echo "MySQL backup created at $BACKUP_DIR/$DB_NAME-$TIMESTAMP.sql" |

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_N
    echo "PostgreSQL backup created at $BACKUP_DIR/$DB_NAME-$TIMESTAMP.s

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.

Copy

```
#!/bin/bash

set -euo pipefail  # Exit if any command fails

# Configuration - you should use ENV variables or secrets, espsecially i
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"
```

```
# 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 "$LO
  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"
  echo "PostgreSQL restore completed from: $BACKUP_FILE" | tee -a "$LOG_

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.

3. MongoDB Backup Script

Copy

```
#!/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 metl
DB_PASS="${DB_PASS:-}"         # Optional
AUTH_DB="${AUTH_DB:-admin}"    # Usually 'admin'
BACKUP_BASE_DIR="/var/backups/mongodb"
```

```
LOG_FILE="/var/log/mongodb_backup_$(date +%Y).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\" --authenti
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_FIL
  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

Copy

```
#!/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:-}"
```

Freedium

```
# Validate input
if [[ -z "$BACKUP_DIR" || ! -d "$BACKUP_DIR" ]]; then
  echo "ERROR: Backup directory not specified or does not exist." | tee
  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\" --auther
fi

echo "Starting MongoDB restore for database: $DB_NAME" | tee -a "$LOG_FI
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)

Copy

```
#!/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:-}"
```

```
# 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_US
    && echo "MySQL user '$NEW_USER' created and granted access to $DB_NAM
    || { echo "Failed to create MySQL user '$NEW_USER'." | tee -a "$LOG_

# PostgreSQL User Creation
elif [[ "$DB_TYPE" == "postgres" ]]; then
  echo "Creating PostgreSQL user '$NEW_USER'..." | tee -a "$LOG_FILE"
  PGPASSWORD="$DB_PASS" psql -U "$DB_USER" -d "postgres" -c "DO \$\$ BEGI
    && echo "PostgreSQL user '$NEW_USER' created." | tee -a "$LOG_FILE"
    || { echo "Failed to create PostgreSQL user '$NEW_USER'." | tee -a "

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.

Copy

```
#!/bin/bash

set -euo pipefail

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

Freedium

```
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 "$LOG_F
  mysql -u "$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 'Da
    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 "$L
  PGPASSWORD="$DB_PASS" psql -U "$DB_USER" -h "$DB_HOST" -d "$DB_NAME" -c
    SELECT pg_size_pretty(pg_database_size('$DB_NAME')) AS \"Database Siz

else
  echo "Error: Unsupported DB_TYPE '$DB_TYPE'. Use 'mysql' or 'postgres'
  exit 1
fi
```

- MySQL calculates from table sizes, PostgreSQL uses `pg_database_size`.

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

Copy

Freedium

```
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
```

```
  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.

Copy

```
#!/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
```

```
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 \"$DB_HOS
  [[ -n "$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"` .

Freedium

Copy

```
#!/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 '$USER_TO_DELETE'? [y/N]: '
if [[ ! "$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 PI

  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
```

```
  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:

Copy

```
#!/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
```

```
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

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" "$db" > "$B
    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
find "$BACKUP_DIR" -type f -name "*.sql" -mtime +$RETENTION_DAYS -exec rm

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:

Copy

```
#!/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}"
```

Freedium

```
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_FI

# Perform backup, compression, and encryption
if mysqldump -u "$USER" -p"$PASSWORD" "$DB_NAME" | gzip | openssl enc -a
  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 $RETENTION_DAYS days..." | tee -a "
find "$BACKUP_DIR" -name "*.sql.gz.enc" -mtime +$RETENTION_DAYS -exec rm

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:

Copy

```
#!/bin/bash

set -euo pipefail

# Configuration
LOG_FILE="/var/log/db_backup_master.log"
TIMESTAMP=$(date +"%F %T")
```

```
# 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 "
```

- Runs all backup scripts and cleanup in one go.

Automate with Cron:

Add this to crontab to run at midnight:

Copy

```
0 0 * * * /path/to/master_backup.sh
```

write clean, reusable, modular scripts and drastically simplify complex workflows

medium.com

15 Essential Helm Charts for Kubernetes Deployments

and how to create your own custom chart

medium.com

Why spend hours manually hardening servers? *With this* ***Security Hardening Kit****, containing production-tested Bash scripts, you can automate it in 1 hour . Grab it here* 👉

#scripting #bash-script #bash #devops #programming