Shell Scripting for DevOps

🔹 **Why Shell Scripting in DevOps?**

- Automate server setups (e.g., install packages, configure services)
- Write deployment scripts
- Manage logs and backups
- Monitor system resources
- Integrate with tools like Docker, Jenkins, Kubernetes

🔹 **Basic Shell Script Structure**

```
#!/bin/bash

# This is a comment
echo "Hello, DevOps!"
```

---

1. Install Packages

```
#!/bin/bash
sudo apt update && sudo apt install -y nginx
```

Purpose: Updates system package info and installs **NGINX web server**.

---

2. Monitor Disk Usage

```
#!/bin/bash
df -h > disk_usage_report.txt
```

Purpose: Saves **disk space usage** to a file for review later.

---

3. Backup Files

```
#!/bin/bash
tar -czf backup_$(date +%F).tar.gz /path/to/directory
```

Purpose: Compresses a directory into a .tar.gz backup file with the current date.

---

4. Jenkins Job Trigger

```
#!/bin/bash
curl -X POST http://jenkins.local/job/your-job-name/build \
  --user your-user:your-api-token
```

Purpose: Triggers a Jenkins CI job remotely using a **POST request** and authentication.

---

5. Docker Container Health Check

```
#!/bin/bash
```

```
if docker ps | grep -q my_container; then
  echo "Container is running"
else
  echo "Container is down"
fi
```

Purpose: Checks if a specific Docker container (my_container) is **running**.

---

6. System Health Check

```
#!/bin/bash
echo "CPU Load:"; uptime
echo -e "\nMemory Usage:"; free -m
echo -e "\nDisk Usage:"; df -h
echo -e "\nTop 5 Memory Consuming Processes:"; ps aux --sort=-%mem | head -n
6
```

Purpose: Shows system metrics like **CPU load**, memory, disk, and top memory-consuming processes.

---

7. Service Restart on Failure

```
#!/bin/bash
SERVICE="nginx"
if ! systemctl is-active --quiet $SERVICE; then
  echo "$SERVICE is down. Restarting..."
  systemctl start $SERVICE
else
  echo "$SERVICE is running"
```

```
fi
```

Purpose: Checks if nginx service is down and **restarts** it automatically.

---

8. Log Rotation Script

```
#!/bin/bash
LOG_DIR="/var/log/myapp"
ARCHIVE_DIR="/var/log/myapp/archive"
mkdir -p $ARCHIVE_DIR
find $LOG_DIR/*.log -mtime +7 -exec mv {} $ARCHIVE_DIR \;
gzip $ARCHIVE_DIR/*.log
```

Purpose: Moves logs older than 7 days to an archive and compresses them.

---

9. Git Auto Pull

```
#!/bin/bash
cd /home/ubuntu/my-repo
git pull origin main
```

Purpose: Automatically pulls the latest code from GitHub (useful with **cron jobs**).

---

10. Docker Cleanup Script

```
#!/bin/bash
docker container prune -f
docker image prune -f
```

```
docker volume prune -f
```

Purpose: Frees disk space by removing unused Docker containers, images, and volumes.

---

11. PostgreSQL Database Backup

```
#!/bin/bash
BACKUP_DIR="/backups"
DB_NAME="mydb"
USER="postgres"
mkdir -p $BACKUP_DIR
pg_dump -U $USER $DB_NAME > $BACKUP_DIR/${DB_NAME}_$(date +%F).sql
```

Purpose: Creates a daily backup of a PostgreSQL database.

---

12. Kubernetes Pod Status Checker

```
#!/bin/bash
NAMESPACE="default"
kubectl get pods -n $NAMESPACE | grep -v Running
```

Purpose: Lists **non-running** pods in a Kubernetes namespace.

---

13. Jenkins Job Trigger with Token

```
#!/bin/bash
```

```
JENKINS_URL="http://jenkins.local"
JOB_NAME="my-job"
USER="your-user"
API_TOKEN="your-token"
curl -X POST "$JENKINS_URL/job/$JOB_NAME/build" --user
$USER:$API_TOKEN
```

Purpose: Triggers a Jenkins job using **username + token** for security.

---

14. Check Port Availability

```
#!/bin/bash
PORT=8080
if lsof -i:$PORT > /dev/null; then
  echo "Port $PORT is in use."
else
  echo "Port $PORT is free."
fi
```

Purpose: Checks if a specific port (like 8080) is being used by any process.

---

15. Simple CI Build Script

```
#!/bin/bash
echo "Starting build process..."
cd /home/ubuntu/app
git pull origin main
mvn clean install -DskipTests
if [ $? -eq 0 ]; then
  echo "Build successful!"
```

```
else
  echo "Build failed!"
  exit 1
fi
```

Purpose: A basic **CI build** script that pulls code and builds a Java project using Maven.

🔹 16. Kubernetes Rolling Restart

```
#!/bin/bash
DEPLOYMENT="myapp"
NAMESPACE="default"
kubectl rollout restart deployment $DEPLOYMENT -n $NAMESPACE
```

- **Purpose**: Triggers a **rolling restart** of a Kubernetes deployment.
- **Use**: Used to apply changes to a deployment (like new code) without downtime.

🔹 17. Check Jenkins Job Status via API

```
#!/bin/bash
JOB_NAME="my-job"
USER="admin"
API_TOKEN="xxxxxx"
JENKINS_URL="http://jenkins.local"
```

```
curl -s --user $USER:$API_TOKEN
"$JENKINS_URL/job/$JOB_NAME/lastBuild/api/json" | jq '.result'
```

- **Purpose**: Fetches the last build status of a Jenkins job using the Jenkins API.
- **Use**: Helpful for monitoring Jenkins jobs programmatically.

🔹 18. Pull Latest Docker Image and Restart Container

```
#!/bin/bash
IMAGE="myrepo/myapp:latest"
CONTAINER="myapp"
docker pull $IMAGE
docker stop $CONTAINER
docker rm $CONTAINER
docker run -d --name $CONTAINER -p 80:80 $IMAGE
```

- **Purpose**: Pulls the latest Docker image, stops the existing container, removes it, and then restarts the container with the updated image.
- **Use**: Ideal for CI/CD pipelines that require container updates.

🔹 19. Terraform Plan & Apply with Auto-Approval

```
#!/bin/bash
cd /path/to/terraform
terraform init
```

```
terraform plan -out=tfplan
terraform apply -auto-approve tfplan
```

- **Purpose**: Automates the process of running terraform plan and applying the changes without manual approval.
- **Use**: Useful for environments that require continuous infrastructure deployment.

🔹 20. Ansible Playbook Trigger

```
#!/bin/bash
ansible-playbook -i inventory.ini site.yml --limit web_servers
```

- **Purpose**: Executes an **Ansible playbook** on a set of hosts defined by web_servers in the inventory file.
- **Use**: Automates configuration management tasks like provisioning or deploying on specific servers.

🔹 21. Monitor CPU Usage and Send Alert

```
#!/bin/bash
THRESHOLD=80
CPU_LOAD=$(top -bn1 | grep "Cpu(s)" | awk '{print $2 + $4}')

if (( $(echo "$CPU_LOAD > $THRESHOLD" | bc -l) )); then
```

```
  echo "High CPU Load: $CPU_LOAD%" | mail -s "Alert: CPU Load"
admin@example.com
fi
```

- **Purpose**: Monitors the CPU usage and sends an email alert if it exceeds the threshold (80% in this case).
- **Use**: Ideal for alerting system administrators about high CPU usage.

🔹 22. Git Branch Cleanup (Delete Merged Local Branches)

```
#!/bin/bash
git branch --merged | grep -v '\*' | grep -v main | xargs -n 1 git branch -d
```

- **Purpose**: Deletes local Git branches that have already been merged into the main branch.
- **Use**: Helps keep the repository clean by removing old branches that are no longer needed.

🔹 23. Archive and Transfer Files to Remote Server

```
#!/bin/bash
tar -czf project_backup_$(date +%F).tar.gz /var/www/project/
scp project_backup_*.tar.gz user@remote:/backups/
```

- **Purpose**: Archives a project directory into a tarball and transfers it to a remote server.
- **Use**: Useful for backing up project files to a remote server.

🔹 24. SSH to Multiple Servers and Run Command

```
#!/bin/bash
SERVERS=("server1" "server2" "server3")

for HOST in "${SERVERS[@]}"
do
  ssh user@$HOST "uptime"
done
```

- **Purpose**: SSHs into multiple servers and runs the uptime command to check system load.
- **Use**: Can be extended for running various commands on multiple servers in a single operation.

🔹 25. GitHub Repo Auto Cloner

```
#!/bin/bash
REPO_LIST=("repo1" "repo2" "repo3")
ORG="your-org"

for REPO in "${REPO_LIST[@]}"; do
```

```
  git clone https://github.com/$ORG/$REPO.git
done
```

- **Purpose**: Automatically clones a list of GitHub repositories from a specific organization.
- **Use**: Useful for setting up multiple repositories quickly.

🔹 26. Jenkins Agent Disk Usage Check

```
#!/bin/bash
AGENTS=("agent1" "agent2")

for AGENT in "${AGENTS[@]}"
do
  ssh jenkins@$AGENT "df -h | grep '/$'"
done
```

- **Purpose**: Checks the disk usage on Jenkins agents and reports the root filesystem usage.
- **Use**: Helpful for monitoring available disk space on Jenkins nodes.

🔹 27. Restart All Failed Systemd Services

```
#!/bin/bash
for SERVICE in $(systemctl --failed --no-legend | awk '{print $1}'); do
```

```
  systemctl restart $SERVICE
done
```

- **Purpose**: Restarts all failed systemd services.
- **Use**: Ensures that any failed services on a Linux system are automatically restarted.

🔹 28. Pull Docker Logs for the Last 1 Hour

```
#!/bin/bash
CONTAINER="myapp"
docker logs --since 1h $CONTAINER > logs_last_hour.txt
```

- **Purpose**: Pulls logs from a Docker container for the last 1 hour and saves them to a file.
- **Use**: Useful for troubleshooting and monitoring container behavior over a recent period.

🔹 29. Clean Old Docker Images (Keep Last 2)

```
#!/bin/bash
docker image prune -af --filter "until=24h"
docker images --filter=reference='myapp*' --format "{{.ID}}" | tail -n +3 | xargs
docker rmi -f
```

- **Purpose**: Removes old and unused Docker images, keeping only the most recent ones.

- **Use**: Helps in cleaning up disk space by removing old Docker images that are no longer needed.

🔹 30. Git Pre-Commit Hook for Code Format Check

```
#!/bin/bash
FILES=$(git diff --cached --name-only --diff-filter=ACM | grep -E '\.py$')

for FILE in $FILES; do
  if ! black --check "$FILE"; then
    echo "Formatting error in $FILE. Run 'black $FILE'"
    exit 1
  fi
done
```

- **Purpose**: A pre-commit hook that checks if Python files are properly formatted using the black formatter before committing.

- **Use**: Ensures that code follows formatting standards before it is committed to the Git repository.

🔹 31. Auto-Deploy to Kubernetes (using kubectl)

Script:

```
#!/bin/bash
```

```
# Define deployment variables
DEPLOYMENT="myapp-deployment"
NAMESPACE="default"
IMAGE="myrepo/myapp:latest"

# Deploy the new image to Kubernetes
kubectl set image deployment/$DEPLOYMENT myapp=$IMAGE -n
$NAMESPACE
kubectl rollout status deployment/$DEPLOYMENT -n $NAMESPACE
```

Explanation:

- **kubectl**: This is the command-line tool used to interact with Kubernetes clusters.
- **set image**: This updates the image in the deployment to the new version.
- **rollout status**: This ensures the new deployment has been successfully rolled out.
- **New Learner Insight**: This script automates the process of deploying a new version of an application to a Kubernetes cluster.

🔹 32. Docker Compose Up and Down

Script:

```
#!/bin/bash

# Start services
docker-compose -f /path/to/docker-compose.yml up -d
```

```
# Stop services
docker-compose -f /path/to/docker-compose.yml down
```

Explanation:

- **docker-compose**: A tool for defining and running multi-container Docker applications.
- **up -d**: This starts the containers in detached mode (background).
- **down**: This stops the containers and removes the network.
- **New Learner Insight**: This script automates the start and stop process of multiple Docker containers defined in a docker-compose.yml file.

🔹 33. Clean Up Docker Volumes

Script:

```
#!/bin/bash

# List unused volumes
docker volume ls -qf dangling=true

# Remove all unused volumes
docker volume prune -f
```

Explanation:

- **docker volume ls -qf dangling=true**: This lists all volumes that are not currently in use (dangling volumes).

- **docker volume prune -f**: This removes all unused volumes to free up disk space.

- **New Learner Insight**: Docker volumes can take up space if they are not cleaned up. This script helps remove them to save storage.

🔹 34. SSH Key Generator Script

Script:

```
#!/bin/bash

echo "Generating SSH key for GitHub..."
ssh-keygen -t rsa -b 4096 -C "your_email@example.com" -f ~/.ssh/id_rsa -N ""

# Show the public key
cat ~/.ssh/id_rsa.pub
```

Explanation:

- **ssh-keygen**: This generates a new SSH key pair (public and private keys).

- **-t rsa -b 4096**: This specifies the RSA algorithm with 4096 bits of encryption strength.

- **-f ~/.ssh/id_rsa**: This saves the private key in the specified file path.

- **New Learner Insight**: SSH keys are used for secure communication, such as connecting to servers or GitHub repositories without needing a password.

🔹 35. CloudFormation Stack Status Check (AWS CLI)

Script:

```
#!/bin/bash

STACK_NAME="my-cloudformation-stack"

aws cloudformation describe-stacks --stack-name $STACK_NAME --query "Stacks[0].StackStatus"
```

Explanation:

- **aws cloudformation describe-stacks**: This AWS CLI command retrieves information about CloudFormation stacks.

- **--query "Stacks[0].StackStatus"**: This extracts the status of the specified stack.

- **New Learner Insight**: CloudFormation is a service used to automate the setup of AWS resources. This script helps check the status of a stack deployment.

🔹 36. Generate a Self-Signed SSL Certificate

Script:

```
#!/bin/bash

DOMAIN="example.com"
CERT_DIR="/etc/ssl/certs"
KEY_DIR="/etc/ssl/private"

openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout
$KEY_DIR/$DOMAIN.key -out $CERT_DIR/$DOMAIN.crt
```

Explanation:

- **openssl req -x509**: This generates a self-signed SSL certificate.
- **-newkey rsa:2048**: This creates a new RSA key with 2048 bits.
- **-keyout and -out**: These specify the file paths to save the private key and certificate.
- **New Learner Insight**: SSL certificates are essential for securing communication over the internet. This script helps generate a self-signed certificate for testing purposes.

🔹 37. Backup MySQL Database

Script:

```
#!/bin/bash

DB_NAME="mydatabase"
USER="root"
PASSWORD="password"
BACKUP_DIR="/backups"
```

```
# Backup MySQL database
mysqldump -u $USER -p$PASSWORD $DB_NAME >
$BACKUP_DIR/$DB_NAME_$(date +%F).sql
```

Explanation:

- **mysqldump**: This command creates a backup of a MySQL database.
- **-u**: Specifies the MySQL user.
- **$(date +%F)**: This appends the current date to the backup file for versioning.
- **New Learner Insight**: Regular backups are essential for data safety. This script automates the process of backing up a MySQL database.

🔹 38. Update System Packages (for Ubuntu/Debian)

Script:

```
#!/bin/bash

echo "Updating system packages..."

# Update apt repositories and upgrade packages
sudo apt update -y && sudo apt upgrade -y

# Clean up unused packages
sudo apt autoremove -y
```

Explanation:

- **sudo apt update -y**: This updates the package index, checking for available updates.

- **sudo apt upgrade -y**: This upgrades installed packages to their latest versions.

- **New Learner Insight**: Keeping system packages up to date is crucial for security and performance. This script automates system maintenance tasks.

🔹 39. Monitor Memory Usage and Trigger Alert

Script:

```
#!/bin/bash

THRESHOLD=90
MEMORY_USAGE=$(free | grep Mem | awk '{print $3/$2 * 100.0}')

if (( $(echo "$MEMORY_USAGE > $THRESHOLD" | bc -l) )); then
  echo "Memory usage is high: $MEMORY_USAGE%" | mail -s "Alert: High Memory Usage" admin@example.com
fi
```

Explanation:

- **free**: Displays memory usage statistics.

- **awk '{print $3/$2 * 100.0}'**: Calculates the percentage of used memory.

- **bc -l**: This command-line calculator is used for floating-point comparison.

- **New Learner Insight**: This script monitors memory usage on a system and sends an email alert if usage exceeds a specified threshold.

🔹 40. Automated Version Bumping for NPM Projects

Script:

```
#!/bin/bash

# Increment version using npm
npm version patch

# Push changes to Git and GitHub
git push origin main
git push origin --tags
```

Explanation:

- **npm version patch**: This increments the patch version in package.json.
- **git push origin main**: This pushes the changes to the remote repository.
- **New Learner Insight**: This script automates the process of versioning a Node.js project and pushing the new version to GitHub.

🔹 41. Check Disk Usage and Send Alert

Script:

```
#!/bin/bash
```

```
THRESHOLD=85
DISK_USAGE=$(df / | grep / | awk '{ print $5 }' | sed 's/%//g')

if [ $DISK_USAGE -gt $THRESHOLD ]; then
  echo "Disk usage is over threshold: $DISK_USAGE%" | mail -s "Disk Usage Alert" admin@example.com
fi
```

Explanation:

- **df /**: Displays disk usage of the root file system.
- **awk '{ print $5 }'**: Extracts the percentage of disk space used.
- **sed 's/%//g'**: Removes the percentage sign.
- **New Learner Insight**: This script monitors disk usage and sends an email alert if usage exceeds a specified threshold.

🔹 42. Automatically Sync Local Repository to Remote (Git)

Script:

```
#!/bin/bash

# Navigate to the local repository directory
cd /path/to/repository

# Pull the latest changes
git pull origin main
```

```
# Add new changes to git
git add .

# Commit the changes
git commit -m "Automated commit"

# Push the changes to the remote repository
git push origin main
```

Explanation:

- **git pull origin main**: This fetches and integrates the latest changes from the remote repository.
- **git add .**: This stages all modified files for committing.
- **git commit -m "Automated commit"**: This commits the changes with a message.
- **git push origin main**: This pushes the changes to the main branch on the remote repository.
- **New Learner Insight**: This script automates the process of syncing changes between a local and a remote Git repository.

🔹 43. Clean Up Old Docker Containers

Script:

```
#!/bin/bash

# List all containers (including stopped ones) and remove them
```

```
docker ps -a -q | xargs docker rm -f
```

Explanation:

- **docker ps -a -q**: This lists all containers, including those that are stopped.
- **xargs docker rm -f**: This removes each container listed by the docker ps command.
- **New Learner Insight**: Docker containers can accumulate over time, taking up unnecessary space. This script removes all containers to clean up the system.

🔹 44. Backup PostgreSQL Database

Script:

```
#!/bin/bash

# Define database credentials and backup location
DB_NAME="mydb"
USER="postgres"
BACKUP_DIR="/backups"
DATE=$(date +%F)

# Dump the PostgreSQL database
pg_dump -U $USER $DB_NAME > $BACKUP_DIR/$DB_NAME-$DATE.sql
```

Explanation:

- **pg_dump**: This command creates a backup of the specified PostgreSQL database.
- **$DATE**: The current date is added to the backup filename for versioning.
- **New Learner Insight**: Regular database backups are critical. This script automates the process of backing up a PostgreSQL database.

🔹 45. Monitor System Load and Send Alerts

Script:

```
#!/bin/bash

THRESHOLD=80
LOAD=$(uptime | awk -F'load average: ' '{ print $2 }' | cut -d, -f1 | tr -d ' ')

if (( $(echo "$LOAD > $THRESHOLD" | bc -l) )); then
  echo "High system load: $LOAD" | mail -s "Alert: High System Load"
admin@example.com
fi
```

Explanation:

- **uptime**: This command shows how long the system has been running and the system load averages.
- **awk -F'load average: '**: Extracts the load average from the output.
- **cut -d, -f1**: Gets the 1-minute load average.

- **New Learner Insight**: System load refers to the amount of computational work the system is performing. This script helps monitor load and send an alert if it exceeds a certain threshold.

🔹 46. Set Up Cron Job for Regular Tasks

Script:

```
#!/bin/bash

# Add a cron job to run the backup script every day at 2 AM
echo "0 2 * * * /path/to/backup.sh" | crontab -
```

Explanation:

- **crontab -**: This sets the cron job for the current user.
- **0 2 * * ***: This specifies the time and frequency (2 AM every day).
- **New Learner Insight**: A **cron job** is a scheduled task that automatically runs at specified intervals. This script adds a cron job to back up data at 2 AM daily.

🔹 47. Create and Configure New User on Linux

Script:

```
#!/bin/bash

# Define username and password
```

```
USER_NAME="newuser"
USER_PASSWORD="password123"

# Create the user
sudo useradd $USER_NAME

# Set the password for the new user
echo "$USER_NAME:$USER_PASSWORD" | sudo chpasswd

# Add user to sudo group
sudo usermod -aG sudo $USER_NAME
```

Explanation:

- **useradd**: This creates a new user.
- **chpasswd**: This sets the password for the new user.
- **usermod -aG sudo**: This adds the new user to the sudo group, granting administrative privileges.
- **New Learner Insight**: Automating user creation and management is common in system administration. This script sets up a user with sudo privileges.

🔹 48. Check for Security Updates and Apply Them

Script:

```
#!/bin/bash

# Update package lists
sudo apt update
```

```
# Install security updates
sudo apt upgrade -y

# Clean up unneeded packages
sudo apt autoremove -y
```

Explanation:

- **sudo apt update**: Updates the list of available packages and their versions.
- **sudo apt upgrade -y**: Upgrades the system, installing security patches and other updates.
- **sudo apt autoremove -y**: Removes any unnecessary packages that were installed as dependencies.
- **New Learner Insight**: Security updates are essential to protect systems. This script ensures the system is up-to-date and free of unneeded packages.

🔹 49. Monitor Disk Usage and Alert on Threshold

Script:

```
#!/bin/bash

THRESHOLD=90
DISK_USAGE=$(df / | grep / | awk '{ print $5 }' | sed 's/%//g')

if [ $DISK_USAGE -gt $THRESHOLD ]; then
  echo "Disk usage is over threshold: $DISK_USAGE%" | mail -s "Disk Usage Alert" admin@example.com
```

```
fi
```

Explanation:

- **df /**: Shows disk space usage for the root file system.
- **awk '{ print $5 }'**: Extracts the disk usage percentage.
- **sed 's/%//g'**: Removes the % sign from the output.
- **New Learner Insight**: Monitoring disk space helps avoid running out of space, which can cause system issues. This script sends an alert if the disk usage exceeds a threshold.

🔹 50. Rotate and Backup Logs

Script:

```
#!/bin/bash

# Define the log file
LOG_FILE="/var/log/myapp.log"

# Rotate log files (move to a new file with timestamp)
mv $LOG_FILE $LOG_FILE.$(date +%F)

# Create a new log file
touch $LOG_FILE

# Compress old log files
gzip $LOG_FILE.$(date +%F)
```

Explanation:

- **mv $LOG_FILE $LOG_FILE.$(date +%F)**: Renames the current log file by appending the current date to its name.
- **touch $LOG_FILE**: Creates a new log file.
- **gzip**: Compresses the old log file to save disk space.
- **New Learner Insight**: Log rotation helps keep system logs manageable and prevents them from growing too large. This script automatically rotates, backs up, and compresses log files.

🔹 51. Monitor System CPU Usage and Alert

Script:

```
#!/bin/bash

THRESHOLD=85
CPU_USAGE=$(top -bn1 | grep "Cpu(s)" | sed "s/.*, *\([0-9.]*\)%* id.*/\1/" | awk '{print 100 - $1}')

if [ $(echo "$CPU_USAGE > $THRESHOLD" | bc) -eq 1 ]; then
  echo "High CPU usage detected: $CPU_USAGE%" | mail -s "CPU Usage Alert" admin@example.com
fi
```

Explanation:

- **top -bn1**: Runs top in batch mode to get a one-time output.
- **grep "Cpu(s)"**: Extracts the CPU usage information.

- **awk '{print 100 - $1}'**: Calculates the CPU usage by subtracting the idle percentage from 100%.

🔹 52. Auto-deploy Application with Git Pull and Restart Service

Script:

```
#!/bin/bash

# Define the application directory and service name

APP_DIR="/path/to/app"

SERVICE_NAME="myapp.service"


# Navigate to the application directory

cd $APP_DIR


# Pull the latest changes from Git repository

git pull origin main


# Restart the application service

sudo systemctl restart $SERVICE_NAME


# Print a success message

echo "Application deployed and service restarted."
```

Explanation:

- **git pull origin main**: Fetches the latest changes from the main branch of the Git repository.
- **systemctl restart $SERVICE_NAME**: Restarts the specified service, ensuring the application runs with the latest code.
- **New Learner Insight**: This script automates the process of deploying updated code by pulling from the repository and restarting the application service.

🔹 53. Monitor Memory Usage and Send Alert

Script:

```
#!/bin/bash

# Define the threshold for memory usage

THRESHOLD=90

MEMORY_USAGE=$(free | grep Mem | awk '{print $3/$2 * 100.0}')


if (( $(echo "$MEMORY_USAGE > $THRESHOLD" | bc -l) )); then

  echo "Memory usage is over threshold: $MEMORY_USAGE%" | mail -s "Memory Usage Alert" admin@example.com

fi
```

Explanation:

- **free**: Shows the memory usage of the system.
- **awk '{print $3/$2 * 100.0}'**: Calculates the percentage of memory being used.
- **bc -l**: Performs the comparison to check if memory usage exceeds the threshold.
- **New Learner Insight**: Memory usage monitoring is important to prevent the system from running out of memory, which could cause applications to crash. This script sends an email alert if memory usage exceeds the set threshold.

🔹 **54. Backup MySQL Database**

Script:

```
#!/bin/bash


# Define MySQL credentials and backup location

DB_NAME="mydb"

USER="root"

PASSWORD="password"

BACKUP_DIR="/path/to/backups"
```

DATE=$(date +%F)

Backup MySQL database

mysqldump -u $USER -p$PASSWORD $DB_NAME > $BACKUP_DIR/$DB_NAME-$DATE.sql

Print a success message

echo "Backup completed successfully!"

Explanation:

- **mysqldump**: This command creates a backup of a MySQL database.
- **$BACKUP_DIR/$DB_NAME-$DATE.sql**: The backup file is saved with the database name and the current date.
- **New Learner Insight**: Backing up databases is crucial for data recovery. This script automates MySQL database backups.

🔹 **55. Clean Up Old Log Files**

Script:

#!/bin/bash

Define the log directory and number of days to retain logs

LOG_DIR="/var/log/myapp"

```
DAYS_TO_KEEP=30


# Find and delete log files older than the specified number of days

find $LOG_DIR -type f -name "*.log" -mtime +$DAYS_TO_KEEP -exec rm -f {} \;


# Print a success message

echo "Old log files deleted."
```

Explanation:

- **find $LOG_DIR -type f -name "*.log" -mtime +$DAYS_TO_KEEP**: Finds log files older than the specified number of days.
- **-exec rm -f {}**: Deletes each found file.
- **New Learner Insight**: Log files can accumulate and take up disk space. This script helps clean up old log files by deleting those that are older than a specified number of days.

🔹 56. Deploy Docker Containers Automatically

Script:

```
#!/bin/bash

# Define Docker image and container name
```

```
DOCKER_IMAGE="myapp:latest"

CONTAINER_NAME="myapp_container"


# Stop the existing container if it's running

docker stop $CONTAINER_NAME


# Remove the existing container

docker rm $CONTAINER_NAME


# Run the new container with the latest image

docker run -d --name $CONTAINER_NAME $DOCKER_IMAGE


# Print a success message

echo "Docker container deployed successfully."
```

Explanation:

- **docker stop $CONTAINER_NAME**: Stops the running container.
- **docker rm $CONTAINER_NAME**: Removes the container to prepare for the new one.
- **docker run -d --name $CONTAINER_NAME $DOCKER_IMAGE**: Runs the container in detached mode using the specified Docker image.

- **New Learner Insight**: This script automates the process of stopping, removing, and redeploying a Docker container, ensuring you always have the latest version of your application.

🔹 57. Auto-Rotate SSL Certificates (for Nginx)

Script:

```
#!/bin/bash


# Define the location of the SSL certificate and private key

CERT_FILE="/etc/ssl/certs/myapp.crt"

KEY_FILE="/etc/ssl/private/myapp.key"

NGINX_CONF="/etc/nginx/nginx.conf"


# Check if certificate is about to expire (within 30 days)

EXPIRY_DATE=$(openssl x509 -enddate -noout -in $CERT_FILE | sed "s/^.*=\(.*\)$/\1/")

EXPIRY_DATE_SECONDS=$(date --date="$EXPIRY_DATE" +%s)

CURRENT_DATE_SECONDS=$(date +%s)

DAYS_LEFT=$((($EXPIRY_DATE_SECONDS - $CURRENT_DATE_SECONDS) / 86400))


if [ $DAYS_LEFT -lt 30 ]; then
```

```
  # Reload Nginx configuration to apply new certificate
  sudo systemctl reload nginx
  echo "SSL certificate renewed and Nginx reloaded."
else
  echo "No certificate renewal needed. Expiry in $DAYS_LEFT days."
fi
```

Explanation:

- **openssl x509 -enddate**: Extracts the expiry date of the SSL certificate.
- **date --date="$EXPIRY_DATE" +%s**: Converts the expiry date to seconds for comparison.
- **systemctl reload nginx**: Reloads the Nginx service to apply the new certificate if needed.
- **New Learner Insight**: Regular SSL certificate renewal ensures secure communication. This script checks for certificates that are about to expire and reloads the web server with the new certificate.

🔹 **58. Sync Files Between Servers Using Rsync**

Script:

```
#!/bin/bash
```

```
# Define source and destination directories

SOURCE_DIR="/path/to/source"

DEST_DIR="user@remote_server:/path/to/destination"


# Use rsync to sync files

rsync -avz --delete $SOURCE_DIR $DEST_DIR


# Print a success message

echo "Files synced successfully."
```

Explanation:

- **rsync -avz**: The rsync command synchronizes files and directories between local and remote servers. The -a option preserves attributes, -v is for verbosity, and -z compresses data during transfer.
- **--delete**: Deletes files in the destination directory that are no longer in the source directory.
- **New Learner Insight**: rsync is a powerful tool for syncing files. This script automates file transfer between two servers, ensuring they are in sync.

🔹 59. Automatically Generate System Health Report

Script:

```
#!/bin/bash
```

```
# Define the output file for the health report
REPORT_FILE="/path/to/health_report.txt"

# Get system uptime
echo "Uptime:" >> $REPORT_FILE
uptime >> $REPORT_FILE

# Get disk usage
echo "Disk Usage:" >> $REPORT_FILE
df -h >> $REPORT_FILE

# Get memory usage
echo "Memory Usage:" >> $REPORT_FILE
free -h >> $REPORT_FILE

# Get CPU load
echo "CPU Load:" >> $REPORT_FILE
top -bn1 | grep "Cpu(s)" >> $REPORT_FILE

# Print a success message
```

```
echo "System health report generated."
```

Explanation:

- **uptime**: Displays how long the system has been running.
- **df -h**: Shows disk space usage in a human-readable format.
- **free -h**: Shows memory usage in a human-readable format.
- **top -bn1**: Provides a snapshot of the system's CPU usage.
- **New Learner Insight**: This script collects key system metrics and writes them into a health report, which can be used to monitor system health over time.

🔹 60. Automatically Scale Web Server (Example with Apache)

Script:

```
#!/bin/bash

# Define server health URL and threshold for scaling

HEALTH_CHECK_URL="http://localhost/health"

THRESHOLD=5

CURRENT_LOAD=$(curl -s $HEALTH_CHECK_URL)
```

```
if [ $CURRENT_LOAD -ge $THRESHOLD ]; then
  # Scale up the web server by adding a new instance
  echo "Scaling up the web server..."
  # Command to scale web server (e.g., launch a new instance)
  # Example: aws ec2 run-instances --image-id ami-xxxx --count 1 --instance-type t2.micro
else
  echo "Load is below threshold. No scaling needed."
fi
```

Explanation:

- **curl -s $HEALTH_CHECK_URL**: Fetches the health check data from a web server.
- **aws ec2 run-instances**: (Commented out) A command that could be used to scale up an AWS EC2 instance if the load exceeds a threshold.

🔹 61. Automated Database Migration Script

Script:

```
#!/bin/bash

# Define the database credentials and migration directory
```

```
DB_USER="dbuser"

DB_PASS="dbpass"

DB_NAME="dbname"

MIGRATION_DIR="/path/to/migrations"


# Run database migrations

cd $MIGRATION_DIR

for migration in *.sql; do

  echo "Applying migration: $migration"

  mysql -u $DB_USER -p$DB_PASS $DB_NAME < $migration

done


# Print a success message

echo "All migrations applied successfully."
```

Explanation:

- **mysql -u $DB_USER -p$DB_PASS $DB_NAME**: Executes each migration SQL script against the specified MySQL database.

- **New Learner Insight**: This script automates the process of running database migrations, which is a common task in deployment pipelines to ensure the database schema is up-to-date.

🔹 62. Clean Up Docker Images

Script:

```
#!/bin/bash

# Remove unused Docker images

docker image prune -f


# Remove all stopped containers

docker container prune -f


# Remove dangling volumes

docker volume prune -f


# Print a success message

echo "Docker cleanup completed successfully."
```

Explanation:

- **docker image prune -f**: Removes unused Docker images to free up disk space.
- **docker container prune -f**: Removes stopped containers.
- **docker volume prune -f**: Removes unused Docker volumes.

- **New Learner Insight**: Regular cleanup of Docker resources helps maintain system performance and free up disk space by removing unnecessary images, containers, and volumes.

🔹 63. Monitor Disk Space Usage and Send Alert

Script:

```
#!/bin/bash

# Set the disk usage threshold (in percentage)

THRESHOLD=80

DISK_USAGE=$(df / | grep / | awk '{ print $5 }' | sed 's/%//g')


if [ $DISK_USAGE -gt $THRESHOLD ]; then

  echo "Disk space usage is over threshold: $DISK_USAGE%" | mail -s "Disk Space Alert" admin@example.com

fi
```

Explanation:

- **df /**: Displays the disk usage statistics for the root directory.
- **awk '{ print $5 }'**: Extracts the percentage of disk space used.
- **sed 's/%//g'**: Removes the percentage symbol for easier comparison.

- **New Learner Insight**: Disk space monitoring is crucial in production environments to avoid downtime or performance issues. This script sends an alert when the disk space usage exceeds the set threshold.

🔹 64. Generate SSH Key Pair for Deployment

Script:

```
#!/bin/bash


# Define the location for the SSH key

KEY_PATH="$HOME/.ssh/deploy_key"


# Generate a new SSH key pair

ssh-keygen -t rsa -b 4096 -f $KEY_PATH -N ""


# Output the public key for deployment

cat $KEY_PATH.pub


# Print a success message

echo "SSH key pair generated successfully."
```

Explanation:

- **ssh-keygen -t rsa -b 4096**: Generates an RSA SSH key pair with a 4096-bit key.
- **-f $KEY_PATH**: Specifies the file path where the SSH key pair will be stored.
- **New Learner Insight**: SSH keys are used for secure authentication, especially in deployment scenarios. This script automates the creation of an SSH key pair for use in secure connections to remote servers.

🔹 65. Backup Files to AWS S3 Bucket

Script:

```
#!/bin/bash

# Define the source directory and S3 bucket
SOURCE_DIR="/path/to/files"
S3_BUCKET="s3://my-bucket/backup"

# Sync files to the S3 bucket
aws s3 sync $SOURCE_DIR $S3_BUCKET --delete

# Print a success message
echo "Backup to S3 completed successfully."
```

Explanation:

- **aws s3 sync**: Syncs files from the local directory to the specified S3 bucket. The --delete flag removes files from S3 that no longer exist locally.

- **New Learner Insight**: This script automates the backup of files to AWS S3, which is a commonly used cloud storage solution. Syncing ensures that the backup is up-to-date with the local directory.

🔹 66. Monitor and Restart a Service if Down

Script:

```
#!/bin/bash


# Define the service name

SERVICE_NAME="nginx"


# Check if the service is running

if ! systemctl is-active --quiet $SERVICE_NAME; then

  echo "$SERVICE_NAME is not running. Restarting the service..."

  systemctl restart $SERVICE_NAME

else

  echo "$SERVICE_NAME is running."

fi
```

Explanation:

- **systemctl is-active --quiet $SERVICE_NAME**: Checks if the service is active (running).
- **systemctl restart $SERVICE_NAME**: Restarts the service if it is found to be inactive.
- **New Learner Insight**: This script automates the monitoring and recovery of services. If a service stops unexpectedly, it will automatically be restarted to maintain system availability.

🔹 67. Automate Deployment to AWS EC2 Instance

Script:

```
#!/bin/bash

# Define the AWS EC2 instance ID and the application directory

INSTANCE_ID="i-xxxxxxxxxxxxx"

APP_DIR="/path/to/app"


# SSH into the EC2 instance and deploy the application

ssh -i "your-key.pem" ec2-user@ec2-xx-xxx-xxx-xx.compute-1.amazonaws.com
<< EOF

  cd $APP_DIR

  git pull origin main
```

```
  sudo systemctl restart myapp.service

  exit

EOF


# Print a success message

echo "Deployment completed successfully."
```

Explanation:

- **ssh -i "your-key.pem"**: SSH into the EC2 instance using the private key for authentication.
- **git pull origin main**: Pulls the latest code from the Git repository.
- **sudo systemctl restart myapp.service**: Restarts the application service on the EC2 instance.

🔹 68. Set Up Cron Jobs for Regular Tasks

Script:

```
#!/bin/bash


# Define the cron job schedule and command to run

CRON_SCHEDULE="0 3 * * *"

COMMAND="/path/to/backup_script.sh"
```

```
# Add the cron job

(crontab -l; echo "$CRON_SCHEDULE $COMMAND") | crontab -


# Print a success message

echo "Cron job added successfully."
```

Explanation:

- **crontab -l**: Lists existing cron jobs.
- **echo "$CRON_SCHEDULE $COMMAND"**: Adds a new cron job with the specified schedule and command to be executed.
- **New Learner Insight**: Cron jobs automate tasks to be executed at scheduled intervals. This script adds a cron job to run a backup script daily at 3:00 AM.

🔹 69. Automatically Install Software Packages

Script:

```
#!/bin/bash


# Define the list of software packages to install

PACKAGES=("git" "docker.io" "nginx")
```

```
# Install each package
for PACKAGE in "${PACKAGES[@]}"; do
  if ! dpkg -l | grep -q "$PACKAGE"; then
    sudo apt-get install -y $PACKAGE
    echo "$PACKAGE installed successfully."
  else
    echo "$PACKAGE is already installed."
  fi
done
```

Explanation:

- **dpkg -l**: Lists installed packages on a Debian-based system.
- **apt-get install -y**: Installs the package if it's not already installed.
- **New Learner Insight**: This script automates the installation of necessary software packages. It checks whether each package is already installed and installs it if necessary.

🔹 70. Schedule System Reboot

Script:

```
#!/bin/bash
```

```
# Define the reboot schedule (e.g., at midnight)
REBOOT_TIME="00:00"

# Schedule a reboot using cron
echo "$REBOOT_TIME root /sbin/bashreboot" | sudo tee -a /etc/crontab

# Print a success message
echo "System reboot scheduled for $REBOOT_TIME."
```

Explanation:

- **echo "$REBOOT_TIME root /sbin/bashreboot" | sudo tee -a /etc/crontab**: Adds a cron job to reboot the system at the specified time.
- **New Learner Insight**: This script helps automate the scheduling of system reboots. Reboots are important for applying updates or maintaining system performance.

:

🔹 71. Auto-Scale EC2 Instances Based on CPU Utilization

Script:

```
#!/bin/bash

# Define the CPU utilization threshold and Auto Scaling group name
```

```
CPU_THRESHOLD=80

AUTO_SCALING_GROUP="my-auto-scaling-group"

# Get the current CPU utilization of EC2 instances in the Auto Scaling group

CPU_UTILIZATION=$(aws cloudwatch get-metric-statistics --metric-name
CPUUtilization --start-time $(date -u -d '5 minutes ago'
+"%Y-%m-%dT%H:%M:%SZ") --end-time $(date -u
+"%Y-%m-%dT%H:%M:%SZ") --period 300 --namespace AWS/EC2 --statistics
Average --dimensions
Name=AutoScalingGroupName,Value=$AUTO_SCALING_GROUP --query
'Datapoints[0].Average' --output text)



# Check if CPU utilization exceeds threshold

if (( $(echo "$CPU_UTILIZATION > $CPU_THRESHOLD" | bc -l) )); then

  # Scale up EC2 instances

  aws autoscaling update-auto-scaling-group --auto-scaling-group-name
$AUTO_SCALING_GROUP --desired-capacity $(($CURRENT_CAPACITY +
1))

  echo "CPU utilization is high. Scaling up the Auto Scaling group."

else

  echo "CPU utilization is within acceptable limits."

fi
```

Explanation:

- **aws cloudwatch get-metric-statistics**: Retrieves CPU utilization statistics for the Auto Scaling group.
- **aws autoscaling update-auto-scaling-group**: Scales the group up when CPU utilization exceeds the threshold.
- **New Learner Insight**: This script automates scaling EC2 instances based on CPU utilization, which is useful for maintaining application performance.

🔹 72. Create Backup of Configuration Files

Script:

```
#!/bin/bash

# Define source and destination directories for the backup

SOURCE_DIR="/etc/myapp"

BACKUP_DIR="/backups/myapp/$(date +'%Y-%m-%d')"


# Create the backup directory if it doesn't exist

mkdir -p $BACKUP_DIR


# Copy configuration files to the backup directory

cp -r $SOURCE_DIR/* $BACKUP_DIR/


# Print a success message
```

```
echo "Backup of configuration files completed successfully."
```

Explanation:

- **mkdir -p $BACKUP_DIR**: Creates a backup directory with the current date.
- **cp -r $SOURCE_DIR/* $BACKUP_DIR/**: Copies all files from the source to the backup directory.
- **New Learner Insight**: Automating backups ensures important configuration files are safely stored, which is critical in case of system failure or when restoring environments.

🔹 73. Monitor System Load and Send Email Alert

Script:

```
#!/bin/bash

# Define the system load threshold
LOAD_THRESHOLD=5.0

# Get the current system load (1-minute average)
CURRENT_LOAD=$(uptime | awk '{print $10}' | sed 's/,//')

# Check if the load exceeds the threshold
```

```
if (( $(echo "$CURRENT_LOAD > $LOAD_THRESHOLD" | bc -l) )); then
  echo "System load is high: $CURRENT_LOAD" | mail -s "System Load Alert" admin@example.com
  echo "Load is above the threshold. Alert sent."
else
  echo "System load is normal: $CURRENT_LOAD"
fi
```

Explanation:

- **uptime**: Retrieves the system load information.
- **echo "$CURRENT_LOAD > $LOAD_THRESHOLD" | bc -l**: Compares the current load to the defined threshold.
- **mail -s "System Load Alert"**: Sends an email alert if the load is high.
- **New Learner Insight**: Monitoring system load is critical for detecting potential issues before they impact performance, and this script automates that process.

🔹 74. Automate Docker Container Build and Push to Docker Hub

Script:

```
#!/bin/bash

# Define Docker image name and tag
```

```
IMAGE_NAME="myapp"

IMAGE_TAG="latest"


# Build the Docker image

docker build -t $IMAGE_NAME:$IMAGE_TAG .


# Log in to Docker Hub

echo "$DOCKER_PASSWORD" | docker login --username
$DOCKER_USERNAME --password-stdin


# Push the Docker image to Docker Hub

docker push $IMAGE_NAME:$IMAGE_TAG


# Print a success message

echo "Docker image $IMAGE_NAME:$IMAGE_TAG pushed to Docker Hub
successfully."
```

Explanation:

- **docker build -t $IMAGE_NAME:$IMAGE_TAG .**: Builds a Docker image with the specified name and tag.

- **docker login**: Logs into Docker Hub using credentials.

- **docker push**: Pushes the image to Docker Hub for use in deployments.
- **New Learner Insight**: Automating Docker image builds and pushing to a registry simplifies deployment workflows and version control for applications.

🔹 75. Monitor Disk Space and Alert If Below Threshold

Script:

```
#!/bin/bash

# Set disk space threshold (in percentage)
DISK_THRESHOLD=90

# Get the current disk usage of the root filesystem
DISK_USAGE=$(df / | grep / | awk '{print $5}' | sed 's/%//')

# Check if the disk usage exceeds the threshold
if [ $DISK_USAGE -gt $DISK_THRESHOLD ]; then
  echo "Warning: Disk space usage is over $DISK_THRESHOLD%. Current usage: $DISK_USAGE%" | mail -s "Disk Space Alert" admin@example.com
```

```
  echo "Disk space usage is high. Alert sent."
else
  echo "Disk space usage is normal: $DISK_USAGE%"
fi
```

Explanation:

- **df /**: Shows the disk usage for the root filesystem.
- **awk '{print $5}'**: Extracts the percentage of disk usage.
- **mail -s "Disk Space Alert"**: Sends an email alert if the disk usage exceeds the threshold.
- **New Learner Insight**: Disk space monitoring is a vital task in system administration. This script ensures that admins are alerted when disk usage becomes critical.

🔹 76. Deploy Application Using Git and Restart the Service

Script:

```
#!/bin/bash
# Define the repository and application directory
REPO_URL="https://github.com/myorg/myapp.git"
APP_DIR="/var/www/myapp"
```

```
# Navigate to the application directory
cd $APP_DIR

# Pull the latest changes from the repository
git pull $REPO_URL

# Restart the application service
sudo systemctl restart myapp

# Print a success message
echo "Application deployed and service restarted successfully."
```

Explanation:

- **git pull $REPO_URL**: Pulls the latest code from the repository to the application directory.
- **sudo systemctl restart myapp**: Restarts the application service after the code update.
- **New Learner Insight**: Automating the deployment of code changes ensures that the application is always up-to-date without manual intervention.

🔹 **77. Clean Up Old Logs Automatically**

Script:

```
#!/bin/bash


# Define the log directory and the retention period (in days)

LOG_DIR="/var/log/myapp"

RETENTION_PERIOD=7


# Find and delete log files older than the retention period

find $LOG_DIR -type f -name "*.log" -mtime +$RETENTION_PERIOD -exec rm -f {} \;


# Print a success message

echo "Old log files older than $RETENTION_PERIOD days have been removed."
```

Explanation:

- **find $LOG_DIR -type f -name "*.log" -mtime +$RETENTION_PERIOD**: Finds log files older than the specified number of days.

- **-exec rm -f {}**: Deletes the found log files.

- **New Learner Insight**: Managing log files and ensuring they don't grow uncontrollably is important for system health. This script automates log cleanup to prevent disk space issues.

🔹 78. Automated SSL Certificate Renewal

Script:

```
#!/bin/bash

# Define the domain and certificate location

DOMAIN="example.com"

CERT_DIR="/etc/ssl/certs"


# Renew the SSL certificate using certbot

certbot renew --quiet --deploy-hook "systemctl reload nginx"


# Print a success message

echo "SSL certificate for $DOMAIN has been renewed and Nginx reloaded."
```

Explanation:

- **certbot renew**: Renews SSL certificates using the Certbot tool.
- **--deploy-hook "systemctl reload nginx"**: Reloads Nginx after the certificate is renewed to apply the new SSL certificate.
- **New Learner Insight**: SSL certificates need regular renewal. This script automates the renewal process and ensures the web server uses the updated certificate.

- **79. Monitor Network Latency and Send Alert**

Script:

```
#!/bin/bash

# Define the target host and acceptable latency threshold

TARGET_HOST="google.com"

LATENCY_THRESHOLD=100


# Get the network latency in milliseconds

LATENCY=$(ping -c 4 $TARGET_HOST | tail -1| awk -F '/' '{print $5}')


# Check if latency exceeds the threshold

if [ $(echo "$LATENCY > $LATENCY_THRESHOLD" | bc) -eq 1 ]; then

  echo "High network latency detected: $LATENCY ms" | mail -s "Network Latency Alert" admin@example.com

  echo "Latency is high. Alert sent."

else

  echo "Network latency is normal: $LATENCY ms"

fi
```

Explanation:

- **ping -c 4 $TARGET_HOST**: Pings the target host to measure network latency.
- **awk -F '/' '{print $5}'**: Extracts the average latency.
- **mail -s "Network Latency Alert"**: Sends an alert if the latency exceeds the threshold.

New Learner Insight: Monitoring network latency ensures that potential networking issues are identified early. This script automates network latency checks and alerts if the performance is degrading.

---

80. Sync Local Files to Remote Server Using rsync

Script:

```
#!/bin/bash

# Define local and remote directories
LOCAL_DIR="/var/www/myapp/"
REMOTE_DIR="user@remote.server:/var/www/myapp/"

# Sync the local directory to the remote server
rsync -avz --delete $LOCAL_DIR $REMOTE_DIR

# Print a success message
echo "Files have been successfully synced to the remote server."
```

Explanation:

- **rsync -avz**: Synchronizes files in archive mode (-a), with verbose output (-v), and compresses the transfer (-z).

- **--delete**: Removes files in the destination that no longer exist in the source.

- **New Learner Insight**: rsync is a powerful tool for syncing files between local and remote servers. This script automates the process of keeping files in sync.

🔹 81. Check and Apply Latest System Updates

Script:

```
#!/bin/bash

# Update the package list
sudo apt update

# Upgrade all packages to their latest version
sudo apt upgrade -y

# Clean up unnecessary packages
sudo apt autoremove -y

# Print a success message
echo "System has been updated and unnecessary packages removed."
```

Explanation:

- **sudo apt update**: Updates the package index with the latest information about available packages.

- **sudo apt upgrade -y**: Upgrades all installed packages to their latest versions automatically.

- **sudo apt autoremove -y**: Removes unnecessary packages that were installed as dependencies but are no longer needed.

- **New Learner Insight**: Regular system updates are essential to maintain security and performance. This script automates the process of checking for updates and cleaning up the system.

🔹 82. Deploy Docker Containers from a Docker Compose File

Script:

```
#!/bin/bash

# Define the directory containing the Docker Compose file
COMPOSE_DIR="/home/user/myapp"

# Navigate to the Docker Compose directory
cd $COMPOSE_DIR

# Pull the latest images and deploy the containers
docker-compose pull
docker-compose up -d

# Print a success message
echo "Docker containers have been deployed and are running in detached mode."
```

Explanation:

- **docker-compose pull**: Pulls the latest Docker images as defined in the docker-compose.yml file.
- **docker-compose up -d**: Deploys the containers in detached mode (-d).
- **New Learner Insight**: Docker Compose makes managing multi-container Docker applications simple. This script automates the deployment process for applications defined in docker-compose.yml.

🔹 83. Check Disk Space and Alert if Critical

Script:

```
#!/bin/bash

# Set the threshold for disk space usage
THRESHOLD=90

# Get the disk space usage percentage for the root filesystem
DISK_USAGE=$(df / | grep / | awk '{print $5}' | sed 's/%//')

# If the disk usage exceeds the threshold, send an alert
if [ $DISK_USAGE -gt $THRESHOLD ]; then
  echo "Disk space usage is critical: $DISK_USAGE%" | mail -s "Disk Space Alert" admin@example.com
  echo "Disk space is above threshold. Alert sent."
else
  echo "Disk space usage is under control: $DISK_USAGE%"
fi
```

Explanation:

- **df /**: Shows disk usage for the root filesystem.
- **awk '{print $5}'**: Extracts the percentage of disk usage.
- **mail -s "Disk Space Alert"**: Sends an email alert if disk usage exceeds the threshold.
- **New Learner Insight**: Monitoring disk space usage ensures that your system doesn't run out of space, which could cause issues. This script helps automate disk space monitoring and alerts when it becomes critical.

🔹 84. Restart a Service if It's Not Running

Script:

```
#!/bin/bash

# Define the service to monitor
SERVICE="nginx"

# Check if the service is running
if ! systemctl is-active --quiet $SERVICE; then
  # Restart the service if it's not running
  sudo systemctl restart $SERVICE
  echo "$SERVICE was not running. It has been restarted."
else
  echo "$SERVICE is running normally."
fi
```

Explanation:

- **systemctl is-active --quiet $SERVICE**: Checks whether the specified service is running.
- **sudo systemctl restart $SERVICE**: Restarts the service if it is not running.
- **New Learner Insight**: This script ensures that critical services like Nginx are always running, preventing downtime.

🔹 85. Monitor System Load and Automatically Scale EC2 Instances

Script:

```
#!/bin/bash

# Define the load threshold and scaling parameters
LOAD_THRESHOLD=80
AUTO_SCALING_GROUP="my-auto-scaling-group"
AWS_CLI_PATH="/usr/bin/bashaws"

# Get the average system load for the last 5 minutes
LOAD=$(uptime | awk -F 'load average:' '{ print $2 }' | awk '{print $1}' | sed 's/,//')

# Check if the load exceeds the threshold
if (( $(echo "$LOAD > $LOAD_THRESHOLD" | bc -l) )); then
  # Scale up the EC2 instances in the Auto Scaling group
  $AWS_CLI_PATH autoscaling update-auto-scaling-group
--auto-scaling-group-name $AUTO_SCALING_GROUP --desired-capacity 3
  echo "System load is high. Scaling up EC2 instances."
else
  echo "System load is normal."
fi
```

Explanation:

- **uptime**: Retrieves the system load average.
- **aws autoscaling update-auto-scaling-group**: Scales the EC2 instances up if the load exceeds the threshold.
- **New Learner Insight**: This script automatically scales EC2 instances based on system load, ensuring the infrastructure can handle increased traffic or compute demand.

🔹 86. Perform Database Backup Using pg_dump

Script:

```
#!/bin/bash

# Define the database name, user, and backup directory
DB_NAME="mydatabase"
DB_USER="postgres"
BACKUP_DIR="/backups/db"
BACKUP_FILE="$BACKUP_DIR/$(date +'%Y-%m-%d')-backup.sql"

# Create the backup directory if it doesn't exist
mkdir -p $BACKUP_DIR

# Perform the backup using pg_dump
pg_dump -U $DB_USER -F c -b -v -f $BACKUP_FILE $DB_NAME

# Print a success message
echo "Database backup has been created successfully at $BACKUP_FILE."
```

Explanation:

- **pg_dump**: Dumps the PostgreSQL database into a backup file.
- **-F c**: Specifies the custom format for the backup file.
- **New Learner Insight**: Automating database backups ensures data safety and simplifies recovery in case of failure. This script uses pg_dump to create a backup of a PostgreSQL database.

🔹 87. Send Custom Alerts for Specific Events Using mail

Script:

```
#!/bin/bash

# Define the event condition
ERROR_LOG="/var/log/myapp/error.log"
ALERT_EMAIL="admin@example.com"

# Check if the error log contains a specific string (e.g., 'Critical Error')
if grep -q "Critical Error" $ERROR_LOG; then
  echo "Critical error detected in the log file" | mail -s "Critical Error Alert"
$ALERT_EMAIL
  echo "Critical error detected. Alert sent."
else
  echo "No critical errors detected."
fi
```

Explanation:

- **grep -q "Critical Error" $ERROR_LOG**: Searches for the specific string in the error log.
- **mail -s "Critical Error Alert"**: Sends an email alert if the error is found.
- **New Learner Insight**: This script allows you to monitor logs for specific errors or events and send alerts, making it useful for alerting administrators about system issues.

🔹 88. Monitor and Restart Web Application if Not Responding

Script:

```
#!/bin/bash

# Define the URL to check
URL="http://localhost:8080"

# Check if the URL responds with HTTP 200 status code
HTTP_STATUS=$(curl -o /dev/null -s -w "%{http_code}" $URL)

# If the HTTP status code is not 200, restart the application
if [ $HTTP_STATUS -ne 200 ]; then
  echo "Web application is down. Restarting the application."
  sudo systemctl restart myapp
else
  echo "Web application is running normally."
fi
```

Explanation:

- **curl -o /dev/null -s -w "%{http_code}"**: Sends an HTTP request and captures the HTTP status code.

- **systemctl restart myapp**: Restarts the application if the status code is not 200 (OK).

- **New Learner Insight**: This script monitors the availability of a web application and ensures it is restarted automatically if it becomes unresponsive.

🔹 89. Automate the Installation of Dependencies for a Node.js Project

Script:

```
#!/bin/bash

# Define the directory of the Node.js project
PROJECT_DIR="/home/user/my-node-app"

# Navigate to the project directory
cd $PROJECT_DIR

# Install dependencies using npm
npm install

# Print a success message
echo "Dependencies have been successfully installed."
```

Explanation:

- **npm install**: Installs the dependencies listed in the package.json file of a Node.js project.

- **New Learner Insight**: This script automates the process of setting up the environment for a Node.js project by installing all the required dependencies.

🔹 90. Monitor and Restart Docker Containers Automatically

Script:

```
#!/bin/bash

# Define the name of the Docker container
CONTAINER_NAME="myapp-container"

# Check if the Docker container is running
if ! docker ps --filter "name=$CONTAINER_NAME" | grep -q $CONTAINER_NAME; then
  # Restart the container if it's not running
  docker restart $CONTAINER_NAME
  echo "$CONTAINER_NAME was not running. It has been restarted."
else
  echo "$CONTAINER_NAME is running normally."
fi
```

Explanation:

- **docker ps --filter "name=$CONTAINER_NAME"**: Checks if the container with the specified name is running.

- **docker restart $CONTAINER_NAME**: Restarts the container if it's not running.

- **New Learner Insight**: Docker containers can sometimes stop unexpectedly. This script ensures the container is restarted automatically when it is down.

🔹 91. Automate Backup of MySQL Database

Script:

```
#!/bin/bash

# Define database parameters
DB_NAME="mydatabase"
DB_USER="root"
DB_PASS="password"
BACKUP_DIR="/backups/mysql"
DATE=$(date +'%Y-%m-%d')

# Create the backup directory if it doesn't exist
mkdir -p $BACKUP_DIR

# Create a backup of the MySQL database
mysqldump -u $DB_USER -p$DB_PASS $DB_NAME >
"$BACKUP_DIR/$DB_NAME-$DATE.sql"

# Print a success message
echo "Database backup has been successfully created."
```

Explanation:

- **mysqldump**: Dumps the specified MySQL database into a SQL file.

- **New Learner Insight**: This script automates the process of creating backups for a MySQL database, helping protect against data loss.

🔹 92. Automate Server Health Checks

Script:

```
#!/bin/bash

# Check for disk usage
DISK_USAGE=$(df / | grep / | awk '{ print $5 }' | sed 's/%//')
if [ $DISK_USAGE -gt 80 ]; then
  echo "Warning: Disk space is above 80%."
fi

# Check for memory usage
MEMORY_USAGE=$(free | grep Mem | awk '{print $3/$2 * 100.0}')
if (( $(echo "$MEMORY_USAGE > 80" | bc -l) )); then
  echo "Warning: Memory usage is above 80%."
fi

# Check if a specific process is running
PROCESS="nginx"
if ! pgrep -x "$PROCESS" > /dev/null; then
  echo "$PROCESS is not running!"
else
  echo "$PROCESS is running normally."
fi
```

Explanation:

- **Disk Usage**: Checks if disk usage exceeds 80% and prints a warning.
- **Memory Usage**: Checks if memory usage exceeds 80%.
- **Process Check**: Checks if a specified process (e.g., nginx) is running.
- **New Learner Insight**: This script monitors system health, including disk space, memory usage, and running processes, providing critical alerts for system administrators.

🔹 93. Automated Kubernetes Pod Restart Based on Resource Usage

Script:

```
#!/bin/bash

# Set the pod name and namespace
POD_NAME="myapp-pod"
NAMESPACE="default"

# Get the CPU and memory usage of the pod
CPU_USAGE=$(kubectl top pod $POD_NAME --namespace=$NAMESPACE |
awk 'NR==2 {print $3}' | sed 's/m//')
MEMORY_USAGE=$(kubectl top pod $POD_NAME
--namespace=$NAMESPACE | awk 'NR==2 {print $4}' | sed 's/Mi//')

# Define thresholds
CPU_THRESHOLD=500
MEMORY_THRESHOLD=200

# Restart pod if CPU or memory usage exceeds thresholds
```

```
if [ $CPU_USAGE -gt $CPU_THRESHOLD ] || [ $MEMORY_USAGE -gt $MEMORY_THRESHOLD ]; then
  kubectl delete pod $POD_NAME --namespace=$NAMESPACE
  echo "$POD_NAME is being restarted due to high resource usage."
else
  echo "$POD_NAME is under control."
fi
```

Explanation:

- **kubectl top pod**: Fetches CPU and memory usage for the specified pod.
- **Thresholds**: If the resource usage exceeds predefined thresholds, the pod is restarted.
- **New Learner Insight**: This script automates resource monitoring in Kubernetes and restarts the pod if it is consuming too much CPU or memory.

🔹 94. Automate Rollback of a Deployment in Kubernetes

Script:

```
#!/bin/bash

# Set the deployment name and namespace
DEPLOYMENT_NAME="myapp-deployment"
NAMESPACE="default"

# Check if the current deployment is successful
DEPLOYMENT_STATUS=$(kubectl rollout status deployment/$DEPLOYMENT_NAME --namespace=$NAMESPACE)
```

```
if [[ "$DEPLOYMENT_STATUS" != *"successfully rolled out"* ]]; then
  echo "Deployment failed. Rolling back to the previous version."
  kubectl rollout undo deployment/$DEPLOYMENT_NAME
--namespace=$NAMESPACE
else
  echo "Deployment was successful."
fi
```

Explanation:

- **kubectl rollout status**: Checks the status of the deployment.
- **kubectl rollout undo**: Rolls back the deployment to the previous version if the deployment fails.
- **New Learner Insight**: This script helps automate the rollback process when a deployment fails, ensuring quick recovery in Kubernetes environments.

🔹 95. Rotate Logs and Clean Up Old Logs Automatically

Script:

```
#!/bin/bash

# Define log directory and retention period (e.g., 7 days)
LOG_DIR="/var/log/myapp"
RETENTION_DAYS=7

# Find and remove logs older than the retention period
find $LOG_DIR -type f -name "*.log" -mtime +$RETENTION_DAYS -exec rm -f
{} \;
```

```
# Print a success message
echo "Old logs older than $RETENTION_DAYS days have been deleted."
```

Explanation:

- **find**: Finds logs older than the specified retention period and removes them.
- **New Learner Insight**: Log rotation and cleanup are crucial for maintaining disk space. This script automates the deletion of old logs to avoid filling up disk space with unnecessary files.

🔹 96. Automate the Setup of a New Server (Install Common Packages)

Script:

```
#!/bin/bash

# Update package list
sudo apt update

# Install common packages
sudo apt install -y git vim curl wget unzip

# Print a success message
echo "Common packages have been installed."
```

Explanation:

- **sudo apt install -y**: Installs the specified packages (e.g., Git, Vim, Curl, Wget, Unzip).

- **New Learner Insight**: This script is helpful for automating the setup of a new server or environment by installing essential tools commonly used by developers and administrators.

🔹 97. Automate Docker Image Build and Push to Docker Hub

Script:

```
#!/bin/bash

# Define the image name and tag
IMAGE_NAME="myapp"
TAG="latest"
DOCKER_USERNAME="user"

# Build the Docker image
docker build -t $DOCKER_USERNAME/$IMAGE_NAME:$TAG .

# Log in to Docker Hub
echo "Please enter your Docker Hub password:"
docker login

# Push the image to Docker Hub
docker push $DOCKER_USERNAME/$IMAGE_NAME:$TAG

# Print a success message
echo "Docker image has been built and pushed to Docker Hub."
```

Explanation:

- **docker build**: Builds the Docker image.

- **docker login**: Logs into Docker Hub using the provided credentials.
- **docker push**: Pushes the built image to Docker Hub.
- **New Learner Insight**: This script automates the process of building and pushing a Docker image to Docker Hub, streamlining the workflow for developers.

🔹 98. Clean Up Old Docker Containers and Images

Script:

```
#!/bin/bash

# Remove stopped containers
docker container prune -f

# Remove unused images
docker image prune -a -f

# Remove unused volumes
docker volume prune -f

# Print a success message
echo "Old Docker containers, images, and volumes have been cleaned up."
```

Explanation:

- **docker container prune**: Removes all stopped containers.
- **docker image prune -a**: Removes all unused images, not just dangling ones.

- **docker volume prune**: Removes all unused Docker volumes.

- **New Learner Insight**: This script helps free up disk space by removing unused Docker resources (containers, images, and volumes), which can accumulate over time.

🔹 **99. Automate the Deployment of a Web Application with Git**

Script:

```
#!/bin/bash

# Define project directory and Git repository URL
PROJECT_DIR="/var/www/myapp"
GIT_REPO_URL="https://github.com/user/myapp.git"

# Navigate to the project directory
cd $PROJECT_DIR

# Pull the latest code from the Git repository
git pull $GIT_REPO_URL

# Restart the web server (e.g., Apache or Nginx)
systemctl restart apache2

# Print a success message
echo "Web application has been updated and the web server has been restarted."
```

Explanation:

- **git pull**: Fetches and integrates the latest changes from the specified Git repository.

- **systemctl restart apache2**: Restarts the Apache web server to apply changes.

- **New Learner Insight**: This script automates the deployment process by updating the local codebase with the latest version from the Git repository and restarting the web server to reflect changes.

🔹 100. Perform System Update and Security Patches

Script:

```
#!/bin/bash

# Update the package list
sudo apt update

# Upgrade all installed packages to the latest version
sudo apt upgrade -y

# Install security updates automatically
sudo apt install unattended-upgrades

# Run unattended upgrades
sudo unattended-upgrade -d

# Print a success message
echo "System and security updates have been successfully applied."
```

Explanation:

- **sudo apt update**: Updates the list of available packages.
- **sudo apt upgrade**: Upgrades all installed packages to their latest version.
- **sudo unattended-upgrade**: Installs and runs unattended security updates.
- **New Learner Insight**: This script helps ensure the system is up to date and secure by automatically installing the latest updates and security patches.

🔹 101. Automate the Creation of a New User on a Server

Script:

```
#!/bin/bash

# Get the username and password for the new user
read -p "Enter username: " USERNAME
read -sp "Enter password: " PASSWORD

# Create a new user
sudo useradd -m $USERNAME

# Set the password for the new user
echo "$USERNAME:$PASSWORD" | sudo chpasswd

# Add the new user to the sudo group (optional)
sudo usermod -aG sudo $USERNAME

# Print a success message
echo "User $USERNAME has been successfully created."
```

Explanation:

- **useradd -m**: Creates a new user with a home directory.
- **chpasswd**: Sets the password for the new user.
- **usermod -aG sudo**: Adds the user to the sudo group, allowing them to execute administrative commands.
- **New Learner Insight**: This script automates the process of adding new users to a server and optionally grants them administrative privileges.

🔹 102. Deploy Application Code to Multiple Servers (Parallel Execution)

Script:

```
#!/bin/bash

# List of server IPs
SERVERS=("192.168.1.1" "192.168.1.2" "192.168.1.3")

# Path to the deployment directory
DEPLOY_DIR="/var/www/myapp"

# Git repository URL
GIT_REPO_URL="https://github.com/user/myapp.git"

# Function to deploy code to a server
deploy_to_server() {
  SERVER=$1
  ssh user@$SERVER "cd $DEPLOY_DIR && git pull $GIT_REPO_URL && systemctl restart apache2"
  echo "Code deployed to $SERVER."
}
```

```
# Deploy to all servers in parallel
for SERVER in "${SERVERS[@]}"; do
  deploy_to_server $SERVER &
done

# Wait for all background jobs to complete
wait

echo "Deployment completed on all servers."
```

Explanation:

- **ssh user@$SERVER**: SSH into each server to deploy the code.
- **git pull**: Pulls the latest changes from the Git repository.
- **systemctl restart apache2**: Restarts the Apache web server to apply the updates.
- **&**: Runs each deployment in the background, allowing parallel execution.
- **New Learner Insight**: This script automates code deployment to multiple servers in parallel, saving time when managing a large infrastructure.

🔹 103. Backup and Compress Files Automatically

Script:

```
#!/bin/bash

# Define the directory to back up
```

```
SOURCE_DIR="/home/user/data"

# Define the backup directory
BACKUP_DIR="/backups"

# Define the backup file name with timestamp
BACKUP_FILE="$BACKUP_DIR/backup-$(date +'%Y%m%d%H%M').tar.gz"

# Create a compressed backup of the source directory
tar -czf $BACKUP_FILE $SOURCE_DIR

# Print a success message
echo "Backup has been successfully created: $BACKUP_FILE"
```

Explanation:

- **tar -czf**: Creates a compressed tarball archive of the specified directory.
- **$(date +'%Y%m%d%H%M')**: Appends a timestamp to the backup file name to make it unique.
- **New Learner Insight**: This script automates the process of creating compressed backups of a specified directory, which is useful for disaster recovery.

🔹 104. Rotate Logs and Archive Them Automatically

Script:

```
#!/bin/bash

# Define log directory and archive directory
```

```
LOG_DIR="/var/log/myapp"
ARCHIVE_DIR="/var/log/archive"

# Define the number of days after which to archive the logs
RETENTION_DAYS=30

# Find and archive logs older than the retention period
find $LOG_DIR -type f -name "*.log" -mtime +$RETENTION_DAYS -exec tar
-czf $ARCHIVE_DIR/$(basename {} .log)-$(date +'%Y%m%d').tar.gz {} \;

# Remove logs older than retention period
find $LOG_DIR -type f -name "*.log" -mtime +$RETENTION_DAYS -exec rm -f
{} \;

# Print a success message
echo "Old logs have been archived and removed."
```

Explanation:

- **find**: Finds logs older than a specified retention period and archives them.
- **tar -czf**: Creates a compressed archive of old log files.
- **rm -f**: Removes logs older than the retention period after archiving them.
- **New Learner Insight**: This script automates log rotation, archiving, and deletion to ensure logs don't take up too much disk space.

🔹 **105. Automate the Removal of Temporary Files**

Script:

```
#!/bin/bash

# Define the directory to clean up
TEMP_DIR="/tmp"

# Find and remove temporary files older than 7 days
find $TEMP_DIR -type f -mtime +7 -exec rm -f {} \;

# Print a success message
echo "Temporary files older than 7 days have been removed."
```

Explanation:

- **find**: Searches for files in the specified directory that are older than 7 days.
- **rm -f**: Removes the identified files.
- **New Learner Insight**: This script helps clean up old temporary files that may accumulate and occupy valuable disk space.

🔹 106. Monitor Disk Space Usage and Alert If Threshold is Exceeded

Script:

```
#!/bin/bash

# Define the disk usage threshold (e.g., 80%)
THRESHOLD=80

# Get the current disk usage percentage for the root directory
USAGE=$(df / | grep / | awk '{ print $5 }' | sed 's/%//g')
```

```
# Compare the disk usage with the threshold
if [ $USAGE -gt $THRESHOLD ]; then
  # Send an alert (you can replace this with an actual email or other notification)
  echo "Disk usage is at ${USAGE}% which exceeds the threshold of
${THRESHOLD}%" | mail -s "Disk Space Alert" user@example.com
  echo "Alert sent: Disk usage exceeded ${THRESHOLD}%"
else
  echo "Disk usage is under control: ${USAGE}%"
fi
```

Explanation:

- **df /**: Checks the disk usage of the root directory.
- **awk '{ print $5 }'**: Extracts the disk usage percentage.
- **if [$USAGE -gt $THRESHOLD]**: Compares the current disk usage with the defined threshold.
- **mail**: Sends an email alert if the disk usage exceeds the threshold.
- **New Learner Insight**: This script monitors disk space and alerts you if usage exceeds a set limit. It's helpful for ensuring your server doesn't run out of space unexpectedly.

🔹 107. Automatically Restart a Service If It Stops

Script:

```
#!/bin/bash

# Define the service name
```

```
SERVICE_NAME="apache2"

# Check if the service is running
if ! systemctl is-active --quiet $SERVICE_NAME; then
  # If not, restart the service
  systemctl restart $SERVICE_NAME
  echo "$SERVICE_NAME service was down and has been restarted."
else
  echo "$SERVICE_NAME service is running."
fi
```

Explanation:

- **systemctl is-active --quiet**: Checks if the service is running.
- **systemctl restart**: Restarts the service if it's not active.
- **New Learner Insight**: This script ensures that critical services like web servers are always running by restarting them if they stop unexpectedly.

🔹 108. Sync Files Between Servers Using rsync

Script:

```
#!/bin/bash

# Define source and destination directories
SOURCE_DIR="/var/www/myapp"
DEST_SERVER="user@192.168.1.10"
DEST_DIR="/var/www/myapp"

# Sync files using rsync
```

```
rsync -avz $SOURCE_DIR/ $DEST_SERVER:$DEST_DIR

# Print a success message
echo "Files have been successfully synchronized."
```

Explanation:

- **rsync**: Synchronizes files from the source to the destination directory on another server.
- **-avz**: Options for archiving, verbose output, and compressing data during transfer.
- **New Learner Insight**: This script uses rsync to efficiently sync files between servers. It's great for ensuring that files are consistently updated across multiple servers.

🔹 109. Automate Backup of Database Using mysqldump

Script:

```
#!/bin/bash

# Define the database credentials and backup path
DB_USER="root"
DB_PASS="password"
DB_NAME="mydatabase"
BACKUP_DIR="/backups"
DATE=$(date +'%Y%m%d%H%M')

# Create the backup using mysqldump
```

```
mysqldump -u $DB_USER -p$DB_PASS $DB_NAME > $BACKUP_DIR/$DB_NAME-$DATE.sql

# Print a success message
echo "Database backup has been successfully created."
```

Explanation:

- **mysqldump**: Creates a backup of the specified MySQL database.
- **$DATE**: Adds a timestamp to the backup file name.
- **New Learner Insight**: This script automates the backup of a MySQL database, which is essential for disaster recovery and data safety.

🔹 110. Monitor System Load and Notify If It Exceeds Threshold

Script:

```
#!/bin/bash

# Define the CPU load threshold (e.g., 80%)
THRESHOLD=80

# Get the current system load (1-minute average)
LOAD=$(uptime | awk '{print $10}' | sed 's/,//')

# Compare the load with the threshold
if (( $(echo "$LOAD > $THRESHOLD" | bc -l) )); then
  # Send an alert (can replace with actual notification mechanism)
  echo "System load is $LOAD, which exceeds the threshold of $THRESHOLD%" | mail -s "High Load Alert" user@example.com
```

```
  echo "Alert sent: System load exceeded $THRESHOLD%"
else
  echo "System load is within the acceptable range: $LOAD"
fi
```

Explanation:

- **uptime**: Displays the system load averages.
- **awk '{print $10}'**: Extracts the 1-minute load average.
- **bc -l**: Used for floating-point comparison in shell scripting.
- **New Learner Insight**: This script helps monitor system load and sends an alert if the load exceeds a specified threshold, ensuring you can take corrective action if needed.

🔹 111. Automated Server Health Check (Disk, Memory, Load)

Script:

```
#!/bin/bash

# Define thresholds
DISK_THRESHOLD=90
MEMORY_THRESHOLD=80
LOAD_THRESHOLD=80

# Check disk usage
DISK_USAGE=$(df / | grep / | awk '{ print $5 }' | sed 's/%//g')
if [ $DISK_USAGE -gt $DISK_THRESHOLD ]; then
```

```
  echo "Disk usage is $DISK_USAGE%, which exceeds the threshold of
$DISK_THRESHOLD%" | mail -s "Disk Usage Alert" user@example.com
fi

# Check memory usage
MEMORY_USAGE=$(free | grep Mem | awk '{print $3/$2 * 100.0}')
if [ $(echo "$MEMORY_USAGE > $MEMORY_THRESHOLD" | bc) -eq 1 ];
then
  echo "Memory usage is $MEMORY_USAGE%, which exceeds the threshold of
$MEMORY_THRESHOLD%" | mail -s "Memory Usage Alert"
user@example.com
fi

# Check system load
LOAD=$(uptime | awk '{print $10}' | sed 's/,//')
if (( $(echo "$LOAD > $LOAD_THRESHOLD" | bc -l) )); then
  echo "System load is $LOAD, which exceeds the threshold of
$LOAD_THRESHOLD%" | mail -s "Load Alert" user@example.com
fi

echo "Health check completed."
```

Explanation:

- This script performs a health check for disk usage, memory usage, and system load.
- **df /**: Checks disk usage.
- **free**: Displays memory usage.
- **uptime**: Displays system load averages.

- If any of these metrics exceed the predefined thresholds, an alert email is sent.

- **New Learner Insight**: This script automates system health checks and helps proactively identify performance issues.

🔹 112. Create and Send a System Resource Report

Script:

```
#!/bin/bash

# Define the report file
REPORT_FILE="/tmp/system_report.txt"

# Collect system information
echo "System Resource Report - $(date)" > $REPORT_FILE
echo "------------------------------" >> $REPORT_FILE
echo "Disk Usage:" >> $REPORT_FILE
df -h >> $REPORT_FILE
echo "" >> $REPORT_FILE
echo "Memory Usage:" >> $REPORT_FILE
free -h >> $REPORT_FILE
echo "" >> $REPORT_FILE
echo "CPU Usage:" >> $REPORT_FILE
top -n 1 | grep "Cpu(s)" >> $REPORT_FILE
echo "" >> $REPORT_FILE

# Send the report via email
mail -s "System Resource Report" user@example.com < $REPORT_FILE

# Clean up
```

```
rm $REPORT_FILE

echo "System resource report has been sent."
```

Explanation:

- **df -h**: Provides a human-readable disk usage report.
- **free -h**: Shows memory usage in a human-readable format.
- **top -n 1**: Captures CPU usage from the top command.
- **New Learner Insight**: This script generates a report of system resources (disk, memory, CPU usage) and sends it via email, helping you monitor the health of your server.

🔹 113. Automated Cleanup of Old Log Files

Script:

```
#!/bin/bash

# Define the log directory
LOG_DIR="/var/log/myapp"

# Define the retention period (in days)
RETENTION_PERIOD=30

# Find and remove log files older than the retention period
find $LOG_DIR -type f -name "*.log" -mtime +$RETENTION_PERIOD -exec rm -f {} \;
```

```
# Print a success message
echo "Old log files older than $RETENTION_PERIOD days have been removed."
```

Explanation:

- **find $LOG_DIR**: Finds log files in the specified directory.
- **-mtime +$RETENTION_PERIOD**: Searches for files older than the retention period.
- **rm -f**: Deletes those old log files.
- **New Learner Insight**: This script helps prevent logs from consuming too much disk space by automatically deleting files older than a set period.

🔹 114. Backup System Files Using tar

Script:

```
#!/bin/bash

# Define the source directories and the backup file path
SOURCE_DIRS=("/etc" "/var/www" "/home/user")
BACKUP_DIR="/backups"
DATE=$(date +'%Y%m%d%H%M')

# Create a backup using tar
tar -czf $BACKUP_DIR/system_backup-$DATE.tar.gz ${SOURCE_DIRS[@]}

# Print success message
echo "Backup of system files completed successfully."
```

Explanation:

- **tar -czf**: Creates a compressed .tar.gz archive.
- **${SOURCE_DIRS[@]}**: Array that lists directories to be backed up.
- **New Learner Insight**: This script automates the process of backing up important system directories, which is crucial for disaster recovery.

🔹 **115. Monitor and Restart a Web Service (e.g., Nginx)**

Script:

```
#!/bin/bash

# Define the service name
SERVICE="nginx"

# Check if the service is running
if ! systemctl is-active --quiet $SERVICE; then
  echo "$SERVICE is not running. Restarting it..."
  systemctl restart $SERVICE
else
  echo "$SERVICE is running fine."
fi
```

Explanation:

- **systemctl is-active --quiet**: Checks if the service is running.
- **systemctl restart**: Restarts the service if it's not running.

- **New Learner Insight**: This script ensures critical services like Nginx are always up and running, and it automatically restarts them if they fail.

🔹 116. Install Required Software Packages Automatically

Script:

```
#!/bin/bash

# List of packages to be installed
PACKAGES=("curl" "git" "vim" "docker")

# Install each package
for PACKAGE in "${PACKAGES[@]}"; do
  if ! dpkg -l | grep -q $PACKAGE; then
    echo "Installing $PACKAGE..."
    apt-get install -y $PACKAGE
  else
    echo "$PACKAGE is already installed."
  fi
done
```

Explanation:

- **dpkg -l**: Lists all installed packages on Debian-based systems.
- **apt-get install -y**: Installs packages without asking for confirmation.
- **New Learner Insight**: This script automates the installation of necessary software, ensuring the system has the tools needed for development and operations.

🔹 117. Set Up a Simple Cron Job

Script:

```
#!/bin/bash

# Define the cron job schedule and the script to run
CRON_SCHEDULE="0 2 * * *"
SCRIPT_PATH="/home/user/scripts/backup.sh"

# Add the cron job to the crontab
(crontab -l ; echo "$CRON_SCHEDULE $SCRIPT_PATH") | crontab -

# Print confirmation
echo "Cron job has been set up to run at $CRON_SCHEDULE"
```

Explanation:

- **crontab -l**: Lists existing cron jobs.
- **echo "$CRON_SCHEDULE $SCRIPT_PATH"**: Adds a new cron job to run a script.
- **New Learner Insight**: This script automates the setup of cron jobs, which can be used to schedule tasks like backups, system health checks, or other maintenance tasks.

🔹 118. Sync Files Between Local and Remote Servers Using rsync

Script:

```
#!/bin/bash

# Define source and destination
SOURCE_DIR="/var/www/myapp"
DEST_SERVER="user@192.168.1.20"
DEST_DIR="/var/www/myapp"

# Perform rsync to sync files
rsync -avz --delete $SOURCE_DIR/ $DEST_SERVER:$DEST_DIR

# Print success message
echo "Files synchronized between local and remote server."
```

Explanation:

- **rsync -avz**: Synchronizes files between local and remote servers, preserving attributes and compressing data.
- **--delete**: Deletes files from the destination if they no longer exist on the source.
- **New Learner Insight**: This script ensures that files are consistently synced between a local server and a remote server, which is useful for deployment and backups.

🔹 119. Monitor System Memory Usage and Send Alert

Script:

```
#!/bin/bash

# Set memory usage threshold
```

```
THRESHOLD=80

# Get current memory usage (percentage)
MEMORY_USAGE=$(free | grep Mem | awk '{print $3/$2 * 100.0}')

# Compare memory usage with threshold
if (( $(echo "$MEMORY_USAGE > $THRESHOLD" | bc -l) )); then
  echo "Memory usage is $MEMORY_USAGE%, exceeding threshold of
$THRESHOLD%" | mail -s "Memory Usage Alert" user@example.com
  echo "Memory usage alert sent."
else
  echo "Memory usage is under control: $MEMORY_USAGE%"
fi
```

Explanation:

- **free**: Displays memory usage.
- **awk '{print $3/$2 * 100.0}'**: Calculates the percentage of used memory.
- **bc -l**: Compares floating-point values.
- **New Learner Insight**: This script monitors memory usage and sends an alert if usage exceeds the threshold, which is useful for ensuring that a system doesn't run out of memory unexpectedly.

🔹 **120. Backup a Directory and Upload to Cloud Storage**

Script:

```
#!/bin/bash
```

```
# Define directories and cloud storage path
SOURCE_DIR="/var/www/myapp"
BACKUP_DIR="/backups"
CLOUD_BUCKET="gs://my-cloud-storage-backup"

# Create a timestamped backup file
DATE=$(date +'%Y%m%d%H%M')
tar -czf $BACKUP_DIR/myapp-backup-$DATE.tar.gz $SOURCE_DIR

# Upload backup to Google Cloud Storage
gsutil cp $BACKUP_DIR/myapp-backup-$DATE.tar.gz $CLOUD_BUCKET

# Clean up local backup
rm $BACKUP_DIR/myapp-backup-$DATE.tar.gz

echo "Backup uploaded to cloud storage."
```

Explanation:

- **tar -czf**: Creates a compressed backup of the source directory.
- **gsutil cp**: Uploads the backup to Google Cloud Storage.
- **rm**: Removes the local backup after uploading to the cloud.
- **New Learner Insight**: This script automates the backup and cloud storage upload process, ensuring that critical data is backed up securely in the cloud.

🔹 **121. Automate Software Updates on a Server**

Script:

```
#!/bin/bash

# Update package list
echo "Updating package list..."
apt-get update -y

# Upgrade all packages
echo "Upgrading packages..."
apt-get upgrade -y

# Remove unnecessary packages
echo "Removing unnecessary packages..."
apt-get autoremove -y

# Print success message
echo "System software has been updated successfully."
```

Explanation:

- **apt-get update -y**: Updates the list of available packages.
- **apt-get upgrade -y**: Installs updates for all installed packages.
- **apt-get autoremove -y**: Removes unused packages that are no longer needed.
- **New Learner Insight**: This script ensures your system is always up to date, improving security and performance by automatically applying updates.

🔹 **122. Archive Old Logs for Future Reference**

Script:

```
#!/bin/bash

# Define log directory and archive path
LOG_DIR="/var/log/myapp"
ARCHIVE_DIR="/backups/logs"
DATE=$(date +'%Y%m%d')

# Create a tarball of old logs
tar -czf $ARCHIVE_DIR/old_logs-$DATE.tar.gz $LOG_DIR/*.log

# Print success message
echo "Old logs have been archived."
```

Explanation:

- **tar -czf**: Compresses log files into an archive.
- **$LOG_DIR/*.log**: Specifies the log files to archive.
- **New Learner Insight**: This script helps you archive older logs for future reference, reducing disk space usage while keeping logs available for troubleshooting or auditing purposes.

🔹 123. Automate Database Backup Using mysqldump

Script:

```
#!/bin/bash

# Define MySQL credentials and backup directory
```

```
DB_USER="root"
DB_PASS="password"
DB_NAME="mydb"
BACKUP_DIR="/backups"
DATE=$(date +'%Y%m%d%H%M')

# Perform database backup using mysqldump
mysqldump -u $DB_USER -p$DB_PASS $DB_NAME >
$BACKUP_DIR/db_backup-$DATE.sql

# Print success message
echo "Database backup for $DB_NAME completed successfully."
```

Explanation:

- **mysqldump**: Creates a backup of the MySQL database.
- **$BACKUP_DIR/db_backup-$DATE.sql**: Specifies the backup file with a timestamp.
- **New Learner Insight**: This script automates the backup of a MySQL database, which is crucial for data protection.

🔹 124. Check Disk Space and Send Alert if Below Threshold

Script:

```
#!/bin/bash

# Set the threshold for disk space (in percentage)
THRESHOLD=80
```

```
# Get the current disk usage percentage
DISK_USAGE=$(df / | grep / | awk '{ print $5 }' | sed 's/%//g')

# Check if disk usage exceeds the threshold
if [ $DISK_USAGE -gt $THRESHOLD ]; then
  echo "Disk usage is above threshold: ${DISK_USAGE}%" | mail -s "Disk Usage Alert" user@example.com
  echo "Disk usage alert sent."
else
  echo "Disk usage is under control: ${DISK_USAGE}%"
fi
```

Explanation:

- **df /**: Checks disk usage of the root file system.
- **awk '{ print $5 }'**: Extracts the percentage of disk usage.
- **sed 's/%//g'**: Removes the percentage sign.
- **New Learner Insight**: This script monitors disk usage and sends an alert if it exceeds a defined threshold. Monitoring disk space is essential to prevent servers from running out of space.

🔹 125. Monitor CPU Usage and Restart a Service if It Exceeds Threshold

Script:

```
#!/bin/bash

# Set the CPU usage threshold (in percentage)
THRESHOLD=90
```

```
# Get the current CPU usage
CPU_USAGE=$(top -bn1 | grep "Cpu(s)" | sed "s/.*, *\([0-9.]*\)%* id.*/\1/" | awk '{print 100 - $1}')

# Check if CPU usage exceeds the threshold
if (( $(echo "$CPU_USAGE > $THRESHOLD" | bc -l) )); then
  echo "CPU usage is above threshold: ${CPU_USAGE}%" | mail -s "High CPU Usage Alert" user@example.com
  echo "Restarting service..."
  systemctl restart nginx  # Replace with the actual service name
else
  echo "CPU usage is under control: ${CPU_USAGE}%"
fi
```

Explanation:

- **top -bn1**: Gets the current CPU usage using top command.
- **bc -l**: Compares floating-point values.
- **systemctl restart nginx**: Restarts the Nginx service if CPU usage exceeds the threshold.
- **New Learner Insight**: This script is useful for automating the response to high CPU usage, helping to maintain system performance.

🔹 **126. Update Git Repository and Push Changes**

Script:

```
#!/bin/bash
```

```
# Define the repository path
REPO_DIR="/path/to/repo"

# Navigate to the repository directory
cd $REPO_DIR

# Pull latest changes from remote
git pull origin main

# Add any changes to the staging area
git add .

# Commit changes with a message
git commit -m "Automated commit with latest changes"

# Push changes to remote repository
git push origin main

# Print success message
echo "Changes pushed to the repository."
```

Explanation:

- **git pull origin main**: Fetches the latest changes from the remote repository.
- **git add .**: Stages changes for commit.
- **git commit -m**: Commits the changes with a message.
- **git push origin main**: Pushes changes to the remote repository.

- **New Learner Insight**: This script automates the process of updating a Git repository, which is a common task in CI/CD pipelines.

🔹 127. Run a Security Update on a Linux Server

Script:

```
#!/bin/bash

# Update package list
apt-get update -y

# Install security updates
apt-get upgrade --only-upgrade -y

# Print success message
echo "Security updates applied successfully."
```

Explanation:

- **apt-get update -y**: Updates the list of available packages.
- **apt-get upgrade --only-upgrade -y**: Installs only security updates.
- **New Learner Insight**: This script helps ensure your server is secure by installing only the necessary security patches.

🔹 128. Clean Up Old Log Files Automatically

Script:

```
#!/bin/bash

# Define the log directory and retention period (in days)
LOG_DIR="/var/log/myapp"
RETENTION_PERIOD=30

# Find and delete log files older than the retention period
find $LOG_DIR -type f -name "*.log" -mtime +$RETENTION_PERIOD -exec rm {} \;

# Print success message
echo "Old log files deleted successfully."
```

Explanation:

- **find**: Searches for log files older than the specified retention period.
- **-exec rm {}**: Deletes the found log files.
- **New Learner Insight**: This script helps maintain server performance by removing old log files that are no longer needed.

🔹 129. Automate Docker Container Restart on Failure

Script:

```
#!/bin/bash

# Define the Docker container name
CONTAINER_NAME="myapp_container"
```

```
# Check if the container is running
if ! docker ps --filter "name=$CONTAINER_NAME" --format "{{.Names}}" |
grep -q $CONTAINER_NAME; then
  echo "Container $CONTAINER_NAME is not running. Restarting..."
  docker restart $CONTAINER_NAME
else
  echo "Container $CONTAINER_NAME is running."
fi
```

Explanation:

- **docker ps**: Lists running Docker containers.
- **docker restart $CONTAINER_NAME**: Restarts the container if it's not running.
- **New Learner Insight**: This script ensures that critical Docker containers are always running by automatically restarting them if they fail.

🔹 130. Check and Alert on Service Status (e.g., Apache)

Script:

```
#!/bin/bash

# Define the service name
SERVICE="apache2"

# Check the status of the service
if systemctl is-active --quiet $SERVICE; then
  echo "$SERVICE is running."
else
```

```
  echo "$SERVICE is not running. Sending an alert..."
  echo "$SERVICE is down on $(hostname)" | mail -s "$SERVICE Service Alert"
admin@example.com
fi
```

Explanation:

- **systemctl is-active --quiet**: Checks if the service is running.
- **mail -s**: Sends an email alert if the service is down.
- **New Learner Insight**: This script helps monitor the status of services and sends alerts if any critical service like Apache is down.

🔹 131. Set Up a New User Account and Grant Permissions

Script:

```
#!/bin/bash

# Define username and group
USER_NAME="newuser"
USER_GROUP="developers"

# Add a new user to the system
useradd -m -g $USER_GROUP $USER_NAME

# Set a password for the new user
echo "$USER_NAME:password" | chpasswd

# Grant sudo privileges
usermod -aG sudo $USER_NAME
```

```
# Print success message
echo "User $USER_NAME has been added and granted sudo privileges."
```

Explanation:

- **useradd -m -g $USER_GROUP $USER_NAME**: Adds a new user and assigns them to a group.
- **chpasswd**: Sets the password for the user.
- **usermod -aG sudo**: Adds the user to the sudo group, granting administrative privileges.
- **New Learner Insight**: This script automates user creation, which is useful for managing users in a server environment.

🔹 132. Monitor System Load and Send Alerts if Too High

Script:

```
#!/bin/bash

# Set the maximum load average threshold (1-minute load)
THRESHOLD=5.0

# Get the 1-minute load average
LOAD=$(uptime | awk '{print $10}' | sed 's/,//')

# Compare the load with the threshold
if (( $(echo "$LOAD > $THRESHOLD" | bc -l) )); then
  echo "System load is high: $LOAD" | mail -s "High System Load Alert"
admin@example.com
  echo "Alert sent due to high system load."
```

```
else
  echo "System load is normal: $LOAD"
fi
```

Explanation:

- **uptime**: Retrieves the system load averages.
- **awk '{print $10}'**: Extracts the 1-minute load average.
- **bc -l**: Compares the load with the threshold.
- **New Learner Insight**: This script monitors system load and alerts the administrator if the load is too high, which is important for system performance management.

🔹 133. Automate File Permissions Update for New Files

Script:

```
#!/bin/bash

# Define the directory to monitor
DIR="/path/to/directory"

# Set the desired permissions for new files
PERMISSIONS="644"

# Find new files and update permissions
find $DIR -type f -mmin -10 -exec chmod $PERMISSIONS {} \;

# Print success message
```

```
echo "Permissions updated for new files."
```

Explanation:

- **find $DIR -type f -mmin -10**: Finds files modified within the last 10 minutes.
- **chmod $PERMISSIONS {}**: Changes the file permissions to 644.
- **New Learner Insight**: This script ensures that new files in a directory have the appropriate permissions set automatically.

🔹 134. Automated Jenkins Job Trigger Script

Script:

```
#!/bin/bash

# Define Jenkins job parameters
JENKINS_URL="http://localhost:8080"
JOB_NAME="my-job"
JENKINS_USER="admin"
JENKINS_TOKEN="your_jenkins_token"

# Trigger the Jenkins job
curl -X POST "$JENKINS_URL/job/$JOB_NAME/build" --user
"$JENKINS_USER:$JENKINS_TOKEN"

# Print success message
echo "Jenkins job $JOB_NAME triggered successfully."
```

Explanation:

- **curl -X POST**: Sends a POST request to trigger a Jenkins job.
- **--user "$JENKINS_USER:$JENKINS_TOKEN"**: Authenticates using the Jenkins user and token.
- **New Learner Insight**: This script automates triggering Jenkins jobs, which is common in CI/CD pipelines.

🔹 135. Automate GitHub Repository Deployment Using SSH

Script:

```
#!/bin/bash

# Define repository and remote server details
REPO_URL="git@github.com:username/repository.git"
DEPLOY_DIR="/var/www/myapp"

# Navigate to the deployment directory
cd $DEPLOY_DIR

# Pull the latest changes from the repository
git pull $REPO_URL

# Restart the application service
systemctl restart myapp

# Print success message
echo "Deployment successful."
```

Explanation:

- **git pull $REPO_URL**: Pulls the latest changes from the GitHub repository.
- **systemctl restart myapp**: Restarts the application after deployment.
- **New Learner Insight**: This script automates the process of pulling the latest changes from a GitHub repository and deploying the updates to a server.

🔹 136. Automate Backup of Docker Volumes

Script:

```
#!/bin/bash

# Define the Docker volume and backup directory
VOLUME_NAME="myvolume"
BACKUP_DIR="/backups"
DATE=$(date +'%Y%m%d%H%M')

# Create a backup of the Docker volume
docker run --rm -v $VOLUME_NAME:/volume -v $BACKUP_DIR:/backup
alpine \
  tar czf /backup/backup-$DATE.tar.gz /volume

# Print success message
echo "Backup of Docker volume $VOLUME_NAME completed."
```

Explanation:

- **docker run --rm -v $VOLUME_NAME:/volume -v $BACKUP_DIR:/backup alpine**: Uses a temporary Alpine container to mount the Docker volume and backup directory.

- **tar czf**: Creates a compressed archive of the volume's contents.

- **New Learner Insight**: This script automates the backup of Docker volumes, which is crucial for preserving application data in containerized environments.

🔹 137. Clean Up Old Docker Containers and Images

Script:

```
#!/bin/bash

# Remove all stopped containers
docker container prune -f

# Remove all unused images
docker image prune -a -f

# Print success message
echo "Old containers and images cleaned up."
```

Explanation:

- **docker container prune -f**: Removes all stopped Docker containers.

- **docker image prune -a -f**: Removes unused Docker images to free up space.

- **New Learner Insight**: This script helps manage Docker resources by cleaning up containers and images, ensuring that the system doesn’t run out of space.

- **138. Check SSL Certificate Expiry and Send Alert**

Script:

```
#!/bin/bash

# Define the domain and alert email
DOMAIN="example.com"
ALERT_EMAIL="admin@example.com"

# Check SSL certificate expiration date
EXPIRY_DATE=$(echo | openssl s_client -connect $DOMAIN:443 -servername
$DOMAIN 2>/dev/null | openssl x509 -noout -dates | grep 'notAfter' | sed
's/notAfter=//')

# Convert the expiry date to seconds since epoch
EXPIRY_EPOCH=$(date -d "$EXPIRY_DATE" +%s)
CURRENT_EPOCH=$(date +%s)

# Calculate the difference in days
DAYS_LEFT=$(( ($EXPIRY_EPOCH - $CURRENT_EPOCH) / 86400 ))

# Check if the certificate expires within 30 days
if [ $DAYS_LEFT -le 30 ]; then
  echo "SSL certificate for $DOMAIN will expire in $DAYS_LEFT days. Please
renew it." | mail -s "SSL Certificate Expiry Warning" $ALERT_EMAIL
  echo "Alert sent about SSL certificate expiration."
else
  echo "SSL certificate for $DOMAIN is valid for $DAYS_LEFT more days."
fi
```

Explanation:

- **openssl s_client**: Connects to the server and fetches the SSL certificate.
- **openssl x509 -noout -dates**: Extracts certificate expiration information.
- **date -d "$EXPIRY_DATE" +%s**: Converts the expiration date to seconds since epoch for comparison.
- **New Learner Insight**: This script automates SSL certificate expiry checks and alerts the administrator if renewal is needed, which is crucial for maintaining secure communication.

🔹 139. Update Local Repository and Push Changes in One Command

Script:

```
#!/bin/bash

# Define the repository path
REPO_PATH="/path/to/local/repo"

# Navigate to the repository
cd $REPO_PATH

# Pull the latest changes
git pull origin main

# Add changes and commit
git add .
git commit -m "Automated commit message"

# Push changes to remote repository
git push origin main
```

```
# Print success message
echo "Repository updated and changes pushed successfully."
```

Explanation:

- **git pull origin main**: Fetches the latest changes from the remote repository.
- **git add .**: Stages any new or modified files.
- **git commit -m**: Commits changes with a message.
- **New Learner Insight**: This script automates the process of updating a local Git repository and pushing changes back to the remote server.

🔹 140. Monitor and Auto-Restart a Docker Service if It Stops

Script:

```
#!/bin/bash

# Define the Docker container name
CONTAINER_NAME="myapp_container"

# Check if the container is running
if ! docker ps --filter "name=$CONTAINER_NAME" --format "{{.Names}}" |
grep -q $CONTAINER_NAME; then
  echo "Container $CONTAINER_NAME is not running. Restarting..."
  docker restart $CONTAINER_NAME
else
  echo "Container $CONTAINER_NAME is running."
fi
```

Explanation:

- **docker ps --filter "name=$CONTAINER_NAME"**: Checks if the specified container is running.
- **docker restart $CONTAINER_NAME**: Restarts the container if it's not running.
- **New Learner Insight**: This script ensures that a critical Docker service is always running by checking its status and restarting it if necessary.

---

141. Automate Disk Space Check and Alert

Script:

```
#!/bin/bash

# Define the threshold for disk space usage (in percentage)
THRESHOLD=80

# Get the disk usage percentage for the root directory
DISK_USAGE=$(df / | grep / | awk '{ print $5 }' | sed 's/%//g')

# Check if disk usage exceeds the threshold
if [ $DISK_USAGE -gt $THRESHOLD ]; then
  echo "Disk space usage is critical: ${DISK_USAGE}%" | mail -s "Disk Space Alert" admin@example.com
  echo "Disk space usage exceeded threshold. Alert sent."
else
  echo "Disk space usage is normal: ${DISK_USAGE}%"
fi
```

Explanation:

- **df /**: Retrieves disk space usage information.
- **awk '{ print $5 }'**: Extracts the percentage of disk usage.
- **sed 's/%//g'**: Removes the percentage symbol for numeric comparison.
- **New Learner Insight**: This script checks disk space usage and sends an alert if it exceeds a defined threshold, which helps in managing server resources.

🔹 142. Perform Regular System Update and Reboot if Necessary

Script:

```
#!/bin/bash

# Perform system update
echo "Updating system packages..."
sudo apt update && sudo apt upgrade -y

# Check if a reboot is needed
if [ -f /var/run/reboot-required ]; then
  echo "Reboot is required to complete updates. Rebooting now..."
  sudo reboot
else
  echo "System is up to date, no reboot required."
fi
```

Explanation:

- **sudo apt update && sudo apt upgrade -y**: Updates all system packages to the latest versions.
- **[-f /var/run/reboot-required]**: Checks if a reboot is required after the update.
- **sudo reboot**: Reboots the system if necessary.
- **New Learner Insight**: This script ensures that the system is always up-to-date and reboots it if required after an update, automating system maintenance tasks.

🔹 143. Cleanup Old Log Files

Script:

```
#!/bin/bash

# Define the log directory and retention period (in days)
LOG_DIR="/var/log"
RETENTION_PERIOD=30

# Find and delete log files older than the retention period
find $LOG_DIR -type f -name "*.log" -mtime +$RETENTION_PERIOD -exec rm -f {} \;

# Print success message
echo "Old log files cleaned up."
```

Explanation:

- **find $LOG_DIR -type f -name "*.log"**: Finds all .log files in the specified directory.
- **-mtime +$RETENTION_PERIOD**: Filters files older than the specified number of days.
- **rm -f {}**: Deletes the log files.
- **New Learner Insight**: This script automatically cleans up old log files, ensuring that logs do not consume excessive disk space over time.

🔹 144. Automatically Pull Docker Images and Restart Containers

Script:

```
#!/bin/bash

# Define the Docker container name and image name
CONTAINER_NAME="myapp_container"
IMAGE_NAME="myapp_image:latest"

# Pull the latest Docker image
docker pull $IMAGE_NAME

# Stop and remove the running container
docker stop $CONTAINER_NAME
docker rm $CONTAINER_NAME

# Run a new container with the latest image
docker run -d --name $CONTAINER_NAME $IMAGE_NAME

# Print success message
echo "Docker container $CONTAINER_NAME updated and restarted."
```

Explanation:

- **docker pull $IMAGE_NAME**: Pulls the latest version of the Docker image.
- **docker stop $CONTAINER_NAME and docker rm $CONTAINER_NAME**: Stops and removes the currently running container.
- **docker run -d --name $CONTAINER_NAME**: Starts a new container with the latest image.
- **New Learner Insight**: This script automates the process of updating Docker containers, ensuring that the latest image is always deployed.

🔹 145. Monitor Web Server Health and Restart if Down

Script:

```
#!/bin/bash

# Define the web server URL
URL="http://localhost:80"

# Check if the web server is reachable
HTTP_STATUS=$(curl -s -o /dev/null -w "%{http_code}" $URL)

# If the HTTP status code is not 200, restart the web server
if [ "$HTTP_STATUS" -ne 200 ]; then
  echo "Web server is down. HTTP Status: $HTTP_STATUS. Restarting server..."
  sudo systemctl restart apache2
```

```
else
  echo "Web server is running fine. HTTP Status: $HTTP_STATUS."
fi
```

Explanation:

- **curl -s -o /dev/null -w "%{http_code}" $URL**: Sends an HTTP request to the web server and retrieves the HTTP status code.
- **sudo systemctl restart apache2**: Restarts the Apache web server if it's down.
- **New Learner Insight**: This script ensures that the web server is always up by monitoring its health and automatically restarting it if it's not responsive.

🔹 146. Automate User Backup and Restore

Script:

```
#!/bin/bash

# Define the user to backup and restore
USER_NAME="user1"
BACKUP_DIR="/home/backups"
BACKUP_FILE="$BACKUP_DIR/$USER_NAME-backup.tar.gz"

# Backup user home directory
tar -czf $BACKUP_FILE /home/$USER_NAME

# Print success message
echo "Backup of user $USER_NAME completed successfully."
```

```
# If you want to restore:
# tar -xzf $BACKUP_FILE -C /home/$USER_NAME
# echo "User $USER_NAME restored from backup."
```

Explanation:

- **tar -czf $BACKUP_FILE /home/$USER_NAME**: Compresses and backs up the user's home directory.
- **New Learner Insight**: This script automates the backup of a user's home directory, which can be restored later if needed.

🔹 147. Schedule Regular Database Backup Using Cron

Script:

```
#!/bin/bash

# Define the database name, user, and backup directory
DB_NAME="mydb"
DB_USER="root"
BACKUP_DIR="/backups"

# Create a timestamp for the backup file
DATE=$(date +'%Y%m%d%H%M')

# Perform the database backup
mysqldump -u $DB_USER -p $DB_NAME >
$BACKUP_DIR/$DB_NAME-backup-$DATE.sql

# Print success message
echo "Database backup completed successfully."
```

Explanation:

- **mysqldump**: Performs the backup of the MySQL database.
- **date +'%Y%m%d%H%M'**: Creates a timestamp for the backup file name.
- **New Learner Insight**: This script automates MySQL database backups, which can be scheduled in cron for regular execution.

🔹 148. Rotate Application Logs and Compress Them

Script:

```
#!/bin/bash

# Define log directory and retention period
LOG_DIR="/var/log/app"
ARCHIVE_DIR="/var/log/archive"

# Rotate logs: move current log to archive directory
mv $LOG_DIR/app.log $ARCHIVE_DIR/app.log-$(date +'%Y%m%d%H%M')

# Compress the archived log file
gzip $ARCHIVE_DIR/app.log-$(date +'%Y%m%d%H%M')

# Print success message
echo "Log rotation and compression completed."
```

Explanation:

- **mv $LOG_DIR/app.log**: Moves the current log file to the archive directory.
- **gzip**: Compresses the archived log file.
- **New Learner Insight**: This script automates log rotation and compression, which helps manage log file growth and storage.

🔹 149. Sync Files Between Servers Using Rsync

Script:

```
#!/bin/bash

# Define source and destination directories
SRC_DIR="/path/to/source/"
DEST_DIR="user@remote_server:/path/to/destination/"

# Sync files using rsync
rsync -avz $SRC_DIR $DEST_DIR

# Print success message
echo "Files synced successfully."
```

Explanation:

- **rsync -avz**: Synchronizes files between local and remote directories.
- **New Learner Insight**: This script automates file synchronization between servers, ensuring that both locations have the same files.

🔹 150. Backup a Directory to a Remote Server Using SCP

Script:

```
#!/bin/bash

# Define the source directory and remote server destination
SOURCE_DIR="/path/to/local/directory"
REMOTE_SERVER="user@remote_server"
REMOTE_DIR="/path/to/remote/directory"

# Copy the directory to the remote server using SCP
scp -r $SOURCE_DIR $REMOTE_SERVER:$REMOTE_DIR

# Print success message
echo "Directory backup to remote server completed successfully."
```

Explanation:

- **scp -r**: Securely copies the entire directory to a remote server.
- **New Learner Insight**: This script automates the process of backing up a local directory to a remote server via SCP.

🔹 151. Monitor System Resources and Send Alert if High Usage Detected

Script:

```
#!/bin/bash

# Define resource thresholds
CPU_THRESHOLD=80
```

```
MEMORY_THRESHOLD=80

# Get current CPU and Memory usage
CPU_USAGE=$(top -bn1 | grep "Cpu(s)" | sed "s/.*, *\([0-9.]*\)%* id.*/\1/" | awk
'{print 100 - $1}')
MEMORY_USAGE=$(free | grep Mem | awk '{print $3/$2 * 100.0}')

# Check if CPU usage exceeds threshold
if (( $(echo "$CPU_USAGE > $CPU_THRESHOLD" | bc -l) )); then
  echo "CPU usage is high: ${CPU_USAGE}%" | mail -s "CPU Usage Alert"
admin@example.com
fi

# Check if memory usage exceeds threshold
if (( $(echo "$MEMORY_USAGE > $MEMORY_THRESHOLD" | bc -l) )); then
  echo "Memory usage is high: ${MEMORY_USAGE}%" | mail -s "Memory
Usage Alert" admin@example.com
fi

echo "System resource usage check completed."
```

Explanation:

- **top -bn1**: Gets CPU usage from the top command.
- **free**: Retrieves memory usage information.
- **echo "$CPU_USAGE > $CPU_THRESHOLD" | bc -l**: Compares CPU usage against a defined threshold using bc.
- **New Learner Insight**: This script checks the CPU and memory usage on the system and sends an email alert if usage exceeds the defined thresholds.

🔹 152. Set Up a Cron Job to Run a Script Daily

Script:

```
#!/bin/bash

# Create a cron job to run the script daily at 2 AM
CRON_JOB="0 2 * * * /path/to/script.sh"

# Add the cron job to the crontab
(crontab -l; echo "$CRON_JOB") | crontab -

# Print success message
echo "Cron job set up successfully to run script.sh daily at 2 AM."
```

Explanation:

- **crontab -l**: Lists the current cron jobs.
- **echo "$CRON_JOB" | crontab -**: Adds a new cron job to the crontab.
- **New Learner Insight**: This script schedules a shell script to run automatically at a specific time every day, saving time on manual execution.

🔹 153. Automate System Shutdown After Backup

Script:

```
#!/bin/bash

# Define the backup directory and log file
BACKUP_DIR="/path/to/backup"
```

```
LOG_FILE="/path/to/backup/log.txt"

# Perform backup (example: using tar)
tar -czf $BACKUP_DIR/backup_$(date +'%Y%m%d').tar.gz /path/to/data

# Log the success
echo "Backup completed on $(date)" >> $LOG_FILE

# Shutdown the system after backup is complete
shutdown -h now

echo "System will shut down after backup completion."
```

Explanation:

- **tar -czf**: Creates a compressed backup of the specified directory.
- **shutdown -h now**: Shuts down the system immediately after the backup is completed.
- **New Learner Insight**: This script combines backup automation with system shutdown, which is useful when the system needs to be powered off after important tasks.

🔹 154. Clean Up Old Docker Containers and Images

Script:

```
#!/bin/bash

# Stop and remove all running containers
```

```
docker stop $(docker ps -aq)
docker rm $(docker ps -aq)

# Remove all unused Docker images
docker rmi $(docker images -q)

# Print success message
echo "Old Docker containers and images removed successfully."
```

Explanation:

- **docker ps -aq**: Lists all container IDs.
- **docker stop and docker rm**: Stop and remove all containers.
- **docker rmi**: Removes all unused Docker images.
- **New Learner Insight**: This script automates Docker container and image cleanup, freeing up disk space and keeping your Docker environment tidy.

🔹 155. Automate Log File Upload to S3

Script:

```
#!/bin/bash

# Define log file and S3 bucket
LOG_FILE="/var/log/app.log"
S3_BUCKET="s3://my-backup-bucket/logs/"

# Upload log file to S3
aws s3 cp $LOG_FILE $S3_BUCKET
```

```
# Print success message
echo "Log file uploaded to S3 successfully."
```

Explanation:

- **aws s3 cp**: Uses the AWS CLI to upload the log file to an S3 bucket.
- **New Learner Insight**: This script uploads important log files to AWS S3 for long-term storage, ensuring that logs are not lost and can be accessed later.

🔹 156. Install and Configure Nginx Web Server

Script:

```
#!/bin/bash

# Update package list
sudo apt update

# Install Nginx web server
sudo apt install -y nginx

# Start and enable Nginx to start on boot
sudo systemctl start nginx
sudo systemctl enable nginx

# Print success message
echo "Nginx web server installed and started successfully."
```

Explanation:

- **sudo apt install -y nginx**: Installs the Nginx web server.
- **sudo systemctl start nginx**: Starts the Nginx service.
- **sudo systemctl enable nginx**: Ensures Nginx starts automatically on system boot.
- **New Learner Insight**: This script automates the process of installing and configuring a web server, which is often a key part of deploying web applications.

🔹 157. Automate Server Health Check and Restart Apache if Down

Script:

```
#!/bin/bash

# Define the Apache server status URL
APACHE_STATUS_URL="http://localhost/server-status"

# Check if Apache is running by requesting server status
HTTP_STATUS=$(curl -s -o /dev/null -w "%{http_code}"
$APACHE_STATUS_URL)

# If the server is down (status not 200), restart Apache
if [ "$HTTP_STATUS" -ne 200 ]; then
  echo "Apache server is down. HTTP Status: $HTTP_STATUS. Restarting
Apache..."
  sudo systemctl restart apache2
else
  echo "Apache server is running fine. HTTP Status: $HTTP_STATUS."
fi
```

Explanation:

- **curl**: Sends a request to the Apache server's status page and checks if it's up.
- **sudo systemctl restart apache2**: Restarts the Apache server if it's down.
- **New Learner Insight**: This script helps in maintaining the uptime of critical services, automating checks, and restarts when necessary.

🔹 158. Automate SSL Certificate Renewal Using Certbot

Script:

```
#!/bin/bash

# Define the domain for SSL certificate renewal
DOMAIN="example.com"

# Renew the SSL certificate using Certbot
sudo certbot renew --quiet --post-hook "systemctl reload nginx"

# Print success message
echo "SSL certificate for $DOMAIN renewed and Nginx reloaded."
```

Explanation:

- **sudo certbot renew**: Renews SSL certificates automatically.
- **--post-hook "systemctl reload nginx"**: Reloads the Nginx server after the certificate is renewed.

- **New Learner Insight**: This script automates SSL certificate renewal for your website, ensuring secure HTTPS connections without manual intervention.

🔹 159. Synchronize System Time Using NTP

Script:

```
#!/bin/bash

# Install NTP package if not installed
sudo apt install -y ntp

# Start and enable NTP service
sudo systemctl start ntp
sudo systemctl enable ntp

# Print success message
echo "System time synchronized using NTP."
```

Explanation:

- **sudo apt install -y ntp**: Installs the NTP (Network Time Protocol) service to synchronize the system clock.

- **sudo systemctl start ntp**: Starts the NTP service.

- **New Learner Insight**: This script ensures that your system time is accurate, which is important for logging and synchronization in distributed systems.

160. Automate the Deployment of a Docker Container

Script:

```
#!/bin/bash

# Define image name and container name
IMAGE_NAME="my-app-image"
CONTAINER_NAME="my-app-container"

# Pull the latest Docker image
docker pull $IMAGE_NAME

# Stop and remove the existing container if it's running
docker stop $CONTAINER_NAME
docker rm $CONTAINER_NAME

# Run the container with the latest image
docker run -d --name $CONTAINER_NAME $IMAGE_NAME

# Print success message
echo "Deployment of the Docker container completed successfully."
```

Explanation:

- **docker pull**: Pulls the latest image from Docker Hub.
- **docker stop and docker rm**: Stops and removes the existing container if it exists.
- **docker run -d**: Runs the new container in detached mode.
- **New Learner Insight**: This script automates the process of deploying a Docker container by pulling the latest image, stopping and removing old containers, and launching a new one.

🔹 161. Generate and Download a Backup of a MySQL Database

Script:

```
#!/bin/bash

# Define MySQL credentials and backup path
MYSQL_USER="root"
MYSQL_PASSWORD="password"
DB_NAME="my_database"
BACKUP_PATH="/path/to/backup"
DATE=$(date +'%Y%m%d_%H%M%S')

# Create a backup of the database
mysqldump -u $MYSQL_USER -p$MYSQL_PASSWORD $DB_NAME > $BACKUP_PATH/db_backup_$DATE.sql

# Compress the backup file to save space
gzip $BACKUP_PATH/db_backup_$DATE.sql

# Print success message
echo "MySQL database backup completed and compressed."
```

Explanation:

- **mysqldump**: Dumps the database content into a .sql file.
- **gzip**: Compresses the backup file to save storage space.
- **New Learner Insight**: This script automates MySQL database backups and compresses them, making it easier to manage backups over time.

🔹 162. Check Disk Space and Send Alert if Low

Script:

```
#!/bin/bash

# Set the disk space threshold (in percentage)
THRESHOLD=90

# Get the current disk usage percentage for the root filesystem
DISK_USAGE=$(df / | grep / | awk '{ print $5 }' | sed 's/%//g')

# If disk usage is higher than the threshold, send an alert
if [ $DISK_USAGE -gt $THRESHOLD ]; then
  echo "Warning: Disk space is above threshold. Current usage: $DISK_USAGE%"
| mail -s "Disk Space Alert" admin@example.com
fi

# Print success message
echo "Disk space check completed."
```

Explanation:

- **df**: Checks disk space usage for the specified filesystem (root in this case).
- **awk and sed**: Parse the disk usage value.
- **mail**: Sends an email if disk usage exceeds the threshold.
- **New Learner Insight**: This script monitors disk space and automatically sends an alert if disk usage exceeds the defined threshold.

🔹 163. Automatically Restart a Service if It Crashes

Script:

```
#!/bin/bash

# Define the service name
SERVICE_NAME="my_service"

# Check if the service is active
SERVICE_STATUS=$(systemctl is-active $SERVICE_NAME)

# If the service is not active, restart it
if [ "$SERVICE_STATUS" != "active" ]; then
  echo "$SERVICE_NAME is not running. Restarting..."
  sudo systemctl restart $SERVICE_NAME
else
  echo "$SERVICE_NAME is running fine."
fi
```

Explanation:

- **systemctl is-active**: Checks the current status of the service.
- **systemctl restart**: Restarts the service if it's not running.
- **New Learner Insight**: This script ensures the uptime of a service by checking if it's running and restarting it automatically if necessary.

🔹 164. Automate System Updates and Reboots

Script:

```
#!/bin/bash

# Update package list
sudo apt update -y

# Upgrade all installed packages
sudo apt upgrade -y

# Reboot the system
sudo reboot

# Print success message
echo "System update and reboot initiated."
```

Explanation:

- **sudo apt update**: Updates the list of available packages.
- **sudo apt upgrade**: Upgrades all installed packages to their latest versions.
- **sudo reboot**: Reboots the system after updates are installed.
- **New Learner Insight**: This script automates system maintenance tasks such as updating packages and rebooting the system, reducing the need for manual intervention.

🔹 **165. Monitor Log Files for Errors and Send Alert**

Script:

```
#!/bin/bash

# Define log file and error pattern
LOG_FILE="/var/log/app.log"
ERROR_PATTERN="ERROR"

# Search for the error pattern in the log file
grep "$ERROR_PATTERN" $LOG_FILE > /dev/null

# If errors are found, send an alert
if [ $? -eq 0 ]; then
  echo "Error found in the log file!" | mail -s "Error Alert" admin@example.com
else
  echo "No errors found in the log file."
fi
```

Explanation:

- **grep**: Searches the log file for the specified error pattern.
- **mail**: Sends an email if an error is found in the log.
- **New Learner Insight**: This script automates log file monitoring for specific error patterns, making it easier to track issues in your applications.

🔹 166. Backup a MongoDB Database

Script:

```
#!/bin/bash

# Define MongoDB connection details
```

```
MONGO_USER="admin"
MONGO_PASSWORD="password"
DB_NAME="my_database"
BACKUP_PATH="/path/to/backup"
DATE=$(date +'%Y%m%d_%H%M%S')

# Backup the MongoDB database
mongodump --username $MONGO_USER --password $MONGO_PASSWORD
--db $DB_NAME --out $BACKUP_PATH/mongo_backup_$DATE

# Print success message
echo "MongoDB database backup completed successfully."
```

Explanation:

- **mongodump**: Creates a backup of the specified MongoDB database.
- **New Learner Insight**: This script automates MongoDB database backups, making it easier to manage your database backups on a regular basis.

🔹 167. Automate Docker Image Cleanup

Script:

```
#!/bin/bash

# Remove all stopped containers
docker container prune -f

# Remove unused Docker images
docker image prune -a -f
```

```
# Remove unused Docker volumes
docker volume prune -f

# Print success message
echo "Old Docker containers, images, and volumes cleaned up."
```

Explanation:

- **docker container prune**: Removes all stopped containers.
- **docker image prune**: Removes all unused images.
- **docker volume prune**: Removes unused Docker volumes.
- **New Learner Insight**: This script automates Docker cleanup, ensuring that old resources are removed and your Docker environment is tidy.

🔹 168. Check for Updates in Installed Packages and Auto-Upgrade

Script:

```
#!/bin/bash

# Check for available updates
sudo apt update -y

# Automatically upgrade packages if updates are available
sudo apt upgrade -y

# Print success message
echo "System packages upgraded successfully."
```

Explanation:

- **sudo apt update**: Fetches the list of available updates for installed packages.
- **sudo apt upgrade**: Automatically upgrades all the installed packages that have new versions available.
- **New Learner Insight**: This script automates the process of keeping your system packages up to date.

🔹 169. Rotate Logs Automatically

Script:

```
#!/bin/bash

# Define log file and archive directory
LOG_FILE="/var/log/myapp.log"
ARCHIVE_DIR="/var/log/archive"
DATE=$(date +'%Y%m%d')

# Rotate log file by renaming it
mv $LOG_FILE $ARCHIVE_DIR/myapp_$DATE.log

# Create a new empty log file
touch $LOG_FILE

# Print success message
echo "Log rotation completed. New log file created."
```

Explanation:

- **mv**: Renames the log file, effectively rotating it.
- **touch**: Creates a new empty log file for future log entries.
- **New Learner Insight**: This script ensures that your log files don't grow indefinitely, by rotating them and creating new ones.

🔹 170. Check System Resource Usage and Kill Resource-Heavy Processes

Script:

```
#!/bin/bash

# Get the top 5 resource-heavy processes
TOP_PROCESSES=$(ps aux --sort=-%cpu | head -n 6)

# Print the top processes
echo "Top 5 resource-heavy processes:"
echo "$TOP_PROCESSES"

# Kill the highest resource-heavy process (if CPU > 80%)
CPU_USAGE=$(echo "$TOP_PROCESSES" | head -n 2 | tail -n 1 | awk '{print $3}')
PID=$(echo "$TOP_PROCESSES" | head -n 2 | tail -n 1 | awk '{print $2}')

if (( $(echo "$CPU_USAGE > 80" | bc -l) )); then
  echo "Killing process with PID $PID due to high CPU usage ($CPU_USAGE%)"
  kill -9 $PID
else
  echo "No process exceeds CPU usage threshold."
fi
```

Explanation:

- **ps aux**: Lists all running processes with resource usage details.
- **kill -9**: Forcefully kills a process based on its PID.
- **New Learner Insight**: This script identifies and terminates processes that consume too much CPU, helping to free up system resources.

🔹 171. Install and Configure Nginx Web Server

Script:

```
#!/bin/bash

# Update the package list
sudo apt update -y

# Install Nginx
sudo apt install nginx -y

# Start and enable Nginx service
sudo systemctl start nginx
sudo systemctl enable nginx

# Check the status of Nginx
sudo systemctl status nginx

# Print success message
echo "Nginx web server installed and started successfully."
```

Explanation:

- **apt update**: Updates the package list for Ubuntu-based systems.
- **apt install nginx**: Installs the Nginx web server.
- **systemctl**: Starts and enables Nginx, ensuring it runs on system boot.
- **New Learner Insight**: This script installs and starts the Nginx web server, essential for serving static content and reverse proxying.

🔹 172. Monitor Server Load and Send an Email Alert

Script:

```
#!/bin/bash

# Set the load threshold (load average over the last 1 minute)
THRESHOLD=2.0

# Get the current load average
LOAD=$(uptime | awk '{print $10}' | sed 's/,//')

# Check if the load is greater than the threshold
if (( $(echo "$LOAD > $THRESHOLD" | bc -l) )); then
  echo "Server load is high: $LOAD" | mail -s "High Server Load Alert"
admin@example.com
  echo "Alert sent."
else
  echo "Server load is normal: $LOAD"
fi
```

Explanation:

- **uptime**: Displays system uptime and load averages.
- **awk** and **sed**: Extract and clean the load average value.
- **mail**: Sends an email alert if the load exceeds the defined threshold.
- **New Learner Insight**: This script monitors server load and sends an alert if it exceeds a threshold, helping maintain server performance.

🔹 173. Automate the Installation of Multiple Packages

Script:

```
#!/bin/bash

# Define a list of packages to install
PACKAGES=("curl" "git" "vim" "htop")

# Install each package
for PACKAGE in "${PACKAGES[@]}"
do
  echo "Installing $PACKAGE..."
  sudo apt install -y $PACKAGE
done

# Print success message
echo "All packages installed successfully."
```

Explanation:

- **PACKAGES**: An array that holds the names of packages to install.
- **for loop**: Iterates over the array and installs each package.
- **New Learner Insight**: This script simplifies the process of installing multiple packages, saving time and effort.

🔹 174. Automatically Rotate Application Logs

Script:

```
#!/bin/bash

# Define log file and archive directory
LOG_FILE="/var/log/myapp.log"
ARCHIVE_DIR="/var/log/archive"
DATE=$(date +'%Y%m%d')

# Check if the log file exists
if [ -f $LOG_FILE ]; then
  # Rotate the log file
  mv $LOG_FILE $ARCHIVE_DIR/myapp_$DATE.log

  # Create a new empty log file
  touch $LOG_FILE

  # Print success message
  echo "Log rotation completed."
else
  echo "Log file does not exist."
fi
```

Explanation:

- **mv**: Renames the log file to rotate it.
- **touch**: Creates a new empty log file for future entries.
- **New Learner Insight**: This script rotates logs by renaming the current log file and creating a new one, helping to manage large log files.

🔹 175. Backup PostgreSQL Database

Script:

```
#!/bin/bash

# Define PostgreSQL credentials and backup path
PG_USER="postgres"
PG_DB="mydb"
BACKUP_PATH="/path/to/backup"
DATE=$(date +'%Y%m%d_%H%M%S')

# Create a backup of the PostgreSQL database
pg_dump -U $PG_USER -d $PG_DB > $BACKUP_PATH/pg_backup_$DATE.sql

# Print success message
echo "PostgreSQL database backup completed successfully."
```

Explanation:

- **pg_dump**: Dumps the content of the PostgreSQL database into a .sql file.

- **New Learner Insight**: This script automates the process of backing up a PostgreSQL database, which is essential for data recovery and migration.

🔹 176. Monitor Memory Usage and Send Alert

Script:

```
#!/bin/bash

# Set the memory usage threshold (in percentage)
THRESHOLD=80

# Get the current memory usage percentage
MEMORY_USAGE=$(free | grep Mem | awk '{print $3/$2 * 100.0}')

# Check if memory usage exceeds the threshold
if (( $(echo "$MEMORY_USAGE > $THRESHOLD" | bc -l) )); then
  echo "Memory usage is high: $MEMORY_USAGE%" | mail -s "High Memory Usage Alert" admin@example.com
  echo "Alert sent."
else
  echo "Memory usage is normal: $MEMORY_USAGE%"
fi
```

Explanation:

- **free**: Displays memory usage statistics.
- **awk**: Calculates memory usage as a percentage.
- **mail**: Sends an email alert if memory usage exceeds the threshold.

- **New Learner Insight**: This script helps monitor memory usage and automatically alerts the admin when the system is running out of memory.

🔹 177. Clean Up Old Docker Containers, Images, and Volumes

Script:

```
#!/bin/bash

# Remove all stopped containers
docker container prune -f

# Remove unused images
docker image prune -a -f

# Remove unused volumes
docker volume prune -f

# Print success message
echo "Docker cleanup completed."
```

Explanation:

- **docker container prune**: Removes all stopped containers.
- **docker image prune -a**: Removes all unused images, freeing up space.
- **docker volume prune**: Removes unused volumes.
- **New Learner Insight**: This script automates Docker cleanup tasks, ensuring that unused resources are removed, which helps maintain a clean Docker

environment.

🔹 178. Monitor Disk Space and Send Alert

Script:

```
#!/bin/bash

# Set the disk usage threshold (in percentage)
THRESHOLD=90

# Get the current disk usage percentage for the root filesystem
DISK_USAGE=$(df / | grep / | awk '{ print $5 }' | sed 's/%//g')

# If disk usage is higher than the threshold, send an alert
if [ $DISK_USAGE -gt $THRESHOLD ]; then
  echo "Warning: Disk space is above threshold. Current usage: $DISK_USAGE%"
| mail -s "Disk Space Alert" admin@example.com
fi

# Print success message
echo "Disk space check completed."
```

Explanation:

- **df**: Checks disk space usage for the specified filesystem (root in this case).
- **awk and sed**: Extract the percentage of disk usage.
- **mail**: Sends an alert if disk usage exceeds the threshold.

- **New Learner Insight**: This script monitors disk space usage and sends an email if disk usage exceeds the specified threshold.

🔹 179. Automatically Install and Configure Git

Script:

```
#!/bin/bash

# Install Git
sudo apt update -y
sudo apt install git -y

# Set global Git configurations
git config --global user.name "Your Name"
git config --global user.email "youremail@example.com"
git config --global core.editor "vim"

# Print success message
echo "Git installed and configured successfully."
```

Explanation:

- **apt install git**: Installs Git on the system.

- **git config**: Sets global configurations for the Git user name, email, and editor.

- **New Learner Insight**: This script automates the installation and basic configuration of Git, which is essential for version control.

🔹 180. Automate the Backup of a Docker Volume

Script:

```
#!/bin/bash

# Define the Docker volume and backup path
VOLUME_NAME="my_volume"
BACKUP_PATH="/path/to/backup"
DATE=$(date +'%Y%m%d_%H%M%S')

# Create a backup of the Docker volume
docker run --rm -v $VOLUME_NAME:/volume -v $BACKUP_PATH:/backup
busybox tar czf /backup/volume_backup_$DATE.tar.gz -C / volume

# Print success message
echo "Docker volume backup completed successfully."
```

Explanation:

- **docker run**: Runs a temporary container to back up the specified Docker volume.
- **tar czf**: Compresses the volume into a .tar.gz archive.
- **New Learner Insight**: This script automates the backup of a Docker volume, ensuring that data stored in volumes is safely backed up.

🔹 181. Automate System Update and Upgrade

Script:

```
#!/bin/bash

# Update package list
echo "Updating package list..."
sudo apt update -y

# Upgrade all packages
echo "Upgrading packages..."
sudo apt upgrade -y

# Upgrade distribution
echo "Upgrading distribution..."
sudo apt dist-upgrade -y

# Remove unused packages
echo "Removing unused packages..."
sudo apt autoremove -y

# Print completion message
echo "System update and upgrade completed successfully."
```

Explanation:

- **apt update**: Updates the list of available packages.
- **apt upgrade**: Upgrades all installed packages to the latest versions.
- **apt dist-upgrade**: Upgrades the entire distribution, handling package dependencies.
- **apt autoremove**: Removes unnecessary packages to free up space.
- **New Learner Insight**: This script automates system maintenance by updating and upgrading all packages, ensuring the system is up to date and

secure.

🔹 182. Create a New User and Assign Sudo Privileges

Script:

```
#!/bin/bash

# Define the new user
USER_NAME="newuser"

# Create the new user
echo "Creating user $USER_NAME..."
sudo useradd -m $USER_NAME

# Set a password for the user
echo "Setting password for $USER_NAME..."
echo "$USER_NAME:password" | sudo chpasswd

# Add the user to the sudo group
echo "Granting sudo privileges to $USER_NAME..."
sudo usermod -aG sudo $USER_NAME

# Print success message
echo "User $USER_NAME created and granted sudo privileges."
```

Explanation:

- **useradd -m**: Creates a new user with a home directory.
- **chpasswd**: Sets the password for the newly created user.

- **usermod -aG sudo**: Adds the user to the sudo group, granting them administrative privileges.

- **New Learner Insight**: This script automates the user creation process, which is commonly needed in cloud environments or server configurations.

🔹 183. Monitor Disk Usage and Delete Old Logs

Script:

```
#!/bin/bash

# Set the disk usage threshold (in percentage)
THRESHOLD=90

# Get the disk usage percentage for root
DISK_USAGE=$(df / | grep / | awk '{print $5}' | sed 's/%//g')

# Check if disk usage exceeds the threshold
if [ $DISK_USAGE -gt $THRESHOLD ]; then
  echo "Disk usage is over threshold: $DISK_USAGE%. Deleting old logs..."
  sudo find /var/log -type f -name "*.log" -exec rm -f {} \;
else
  echo "Disk usage is within limits: $DISK_USAGE%"
fi
```

Explanation:

- **df**: Checks disk space usage.

- **find**: Searches for log files in the /var/log directory and deletes them.

- **New Learner Insight**: This script helps to clear old log files when disk space usage exceeds a threshold, preventing disk full errors.

🔹 184. Schedule Daily Backup with Cron Jobs

Script:

```
#!/bin/bash

# Define the backup directory and destination
BACKUP_DIR="/home/user/data"
BACKUP_DEST="/backup"

# Get the current date
DATE=$(date +'%Y%m%d')

# Create a tarball of the directory
tar -czf $BACKUP_DEST/backup_$DATE.tar.gz -C $BACKUP_DIR .

# Add a cron job to schedule the backup daily at 2 AM
(crontab -l 2>/dev/null; echo "0 2 * * * /path/to/backup_script.sh") | crontab -

# Print success message
echo "Backup completed and scheduled."
```

Explanation:

- **tar -czf**: Compresses the specified directory into a .tar.gz file.
- **crontab**: Adds a cron job to run the backup script daily at 2 AM.

- **New Learner Insight**: This script automates both the backup process and its scheduling with cron jobs, ensuring regular backups.

🔹 185. Install Docker and Docker Compose

Script:

```
#!/bin/bash

# Update the package list
echo "Updating package list..."
sudo apt update -y

# Install required dependencies
echo "Installing dependencies..."
sudo apt install -y apt-transport-https ca-certificates curl
software-properties-common

# Add Docker's official GPG key
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

# Add Docker repository
sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"

# Install Docker
echo "Installing Docker..."
sudo apt update -y
sudo apt install -y docker-ce

# Install Docker Compose
echo "Installing Docker Compose..."
```

```
sudo curl -L
"https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(
uname -s)-$(uname -m)" -o /usr/local/bin/bashdocker-compose
sudo chmod +x /usr/local/bin/bashdocker-compose

# Start Docker and enable it on boot
sudo systemctl start docker
sudo systemctl enable docker

# Print success message
echo "Docker and Docker Compose installed successfully."
```

Explanation:

- **Install Docker**: The script installs Docker by adding its repository, then installing the package.
- **Install Docker Compose**: Downloads and installs Docker Compose to manage multi-container Docker applications.
- **New Learner Insight**: This script simplifies the process of setting up Docker and Docker Compose on a system.

🔹 186. Check CPU Temperature and Send an Alert

Script:

```
#!/bin/bash

# Get the CPU temperature
CPU_TEMP=$(cat /sys/class/thermal/thermal_zone0/temp)
CPU_TEMP_C=$(($CPU_TEMP / 1000))
```

```
# Define the threshold temperature (in Celsius)
THRESHOLD=75

# Check if the CPU temperature exceeds the threshold
if [ $CPU_TEMP_C -gt $THRESHOLD ]; then
  echo "Warning: High CPU temperature detected: $CPU_TEMP_C°C" | mail -s
"High CPU Temperature Alert" admin@example.com
else
  echo "CPU temperature is normal: $CPU_TEMP_C°C"
fi
```

Explanation:

- **/sys/class/thermal/thermal_zone0/temp**: Reads the system's CPU temperature.
- **mail**: Sends an alert if the temperature exceeds the threshold.
- **New Learner Insight**: This script helps monitor system health by alerting administrators if the CPU temperature is too high, which can prevent overheating and potential hardware failure.

🔹 187. Automatically Sync Files to Remote Server Using Rsync

Script:

```
#!/bin/bash

# Define the source and destination paths
SOURCE_DIR="/path/to/source"
DEST_SERVER="user@remote_server:/path/to/destination"
```

```
# Sync files from source to remote destination using rsync
rsync -avz $SOURCE_DIR $DEST_SERVER

# Print success message
echo "Files synchronized successfully."
```

Explanation:

- **rsync**: Efficiently syncs files from a local directory to a remote server.
- **-avz**: Flags for archiving, verbose output, and compressing data during transfer.
- **New Learner Insight**: This script automates file synchronization between local and remote servers, useful for backups and data migration.

🔹 188. Install and Configure Apache Web Server

Script:

```
#!/bin/bash

# Update package list
echo "Updating package list..."
sudo apt update -y

# Install Apache web server
echo "Installing Apache..."
sudo apt install apache2 -y

# Start Apache and enable it on boot
```

```
sudo systemctl start apache2
sudo systemctl enable apache2

# Check Apache status
sudo systemctl status apache2

# Print success message
echo "Apache web server installed and started."
```

Explanation:

- **apt install apache2**: Installs the Apache HTTP web server.
- **systemctl**: Starts and enables Apache to run at boot.
- **New Learner Insight**: This script automates the installation and configuration of Apache, commonly used for serving dynamic and static web content.

🔹 189. Install and Configure Nginx Web Server

Script:

```
#!/bin/bash

# Update package list
echo "Updating package list..."
sudo apt update -y

# Install Nginx
echo "Installing Nginx..."
sudo apt install nginx -y
```

```
# Start Nginx and enable it on boot
echo "Starting Nginx..."
sudo systemctl start nginx
sudo systemctl enable nginx

# Check Nginx status
echo "Checking Nginx status..."
sudo systemctl status nginx

# Print success message
echo "Nginx web server installed and started."
```

Explanation:

- **apt install nginx**: Installs the Nginx web server.
- **systemctl**: Starts and enables Nginx to run at boot.
- **New Learner Insight**: Nginx is a popular web server used for hosting websites and web applications. This script automates its installation and configuration.

🔹 190. Backup MySQL Database Using mysqldump

Script:

```
#!/bin/bash

# Define database credentials and backup location
DB_NAME="my_database"
DB_USER="root"
```

```
DB_PASS="password"
BACKUP_DIR="/backups"
DATE=$(date +'%Y%m%d')

# Create a backup using mysqldump
echo "Creating backup of $DB_NAME..."
mysqldump -u $DB_USER -p$DB_PASS $DB_NAME >
$BACKUP_DIR/$DB_NAME-$DATE.sql

# Check if the backup was successful
if [ $? -eq 0 ]; then
  echo "Backup completed successfully."
else
  echo "Backup failed."
fi
```

Explanation:

- **mysqldump**: A tool to create backups of MySQL databases.
- **$?**: Checks the exit status of the previous command to determine if it was successful.
- **New Learner Insight**: This script automates database backups, which are crucial for disaster recovery and data safety.

🔹 191. Monitor System Load and Send Alerts

Script:

```
#!/bin/bash
```

```
# Set the threshold for CPU load
LOAD_THRESHOLD=75

# Get the current system load
SYSTEM_LOAD=$(uptime | awk '{print $10}' | sed 's/,//')

# Check if the load exceeds the threshold
if (( $(echo "$SYSTEM_LOAD > $LOAD_THRESHOLD" | bc -l) )); then
  echo "Warning: High system load detected ($SYSTEM_LOAD)" | mail -s "High System Load Alert" admin@example.com
else
  echo "System load is normal: $SYSTEM_LOAD"
fi
```

Explanation:

- **uptime**: Displays the system's load averages.
- **bc -l**: Performs floating-point comparison between the load and the threshold.
- **New Learner Insight**: This script monitors the system load and sends an alert if the load exceeds a specified threshold. It's useful for identifying when the system might be under stress.

🔹 192. Clean Up Docker Images and Containers

Script:

```
#!/bin/bash

# Remove stopped containers
```

```
echo "Removing stopped containers..."
docker container prune -f

# Remove dangling images
echo "Removing dangling images..."
docker image prune -f

# Remove unused volumes
echo "Removing unused volumes..."
docker volume prune -f

# Remove unused networks
echo "Removing unused networks..."
docker network prune -f

# Print success message
echo "Docker cleanup completed."
```

Explanation:

- **docker container prune**: Removes all stopped containers.
- **docker image prune**: Removes unused images that are no longer referenced by any containers.
- **docker volume prune**: Cleans up unused Docker volumes.
- **New Learner Insight**: This script helps to maintain a clean Docker environment by removing unused resources, freeing up disk space.

🔹 **193. Automate SSL Certificate Renewal with Certbot**

Script:

```
#!/bin/bash

# Renew SSL certificates using Certbot
echo "Renewing SSL certificates..."
sudo certbot renew

# Restart Nginx to apply the new certificates
echo "Restarting Nginx..."
sudo systemctl restart nginx

# Print success message
echo "SSL certificate renewed and Nginx restarted."
```

Explanation:

- **certbot renew**: Renew SSL certificates for all domains that Certbot manages.
- **systemctl restart nginx**: Restarts Nginx to apply the new SSL certificates.
- **New Learner Insight**: SSL certificates need periodic renewal to ensure secure communication over the web. This script automates the process for you.

🔹 194. Rotate Logs Using logrotate

Script:

```
#!/bin/bash
```

```
# Create a custom logrotate configuration file
echo "Creating logrotate configuration..."
cat <<EOL > /etc/logrotate.d/myapp
/path/to/myapp/logs/*.log {
    daily
    rotate 7
    compress
    missingok
    notifempty
    create 0640 root root
}
EOL

# Manually trigger logrotate for testing
echo "Triggering logrotate..."
sudo logrotate -f /etc/logrotate.d/myapp

# Print success message
echo "Log rotation configured and triggered."
```

Explanation:

- **logrotate**: A tool for managing log file rotation, compression, and removal.
- **/etc/logrotate.d/**: The directory where logrotate configuration files are stored.
- **New Learner Insight**: This script configures automatic log rotation, which is essential for managing log file sizes and avoiding disk space issues.

🔹 **195. Clean Up Old Backup Files**

Script:

```
#!/bin/bash

# Define backup directory and retention period (in days)
BACKUP_DIR="/backups"
RETENTION_PERIOD=30

# Find and delete backup files older than the retention period
echo "Cleaning up old backup files..."
find $BACKUP_DIR -type f -name "*.tar.gz" -mtime +$RETENTION_PERIOD
-exec rm -f {} \;

# Print success message
echo "Old backup files cleaned up."
```

Explanation:

- **find**: Searches for files older than a specified number of days (-mtime +30 finds files older than 30 days).
- **rm**: Deletes the identified files.
- **New Learner Insight**: This script helps keep backup directories organized by deleting old backup files that are no longer needed.

🔹 196. Set Up a Cron Job to Monitor Disk Usage

Script:

```
#!/bin/bash
```

```
# Define the threshold for disk usage
DISK_THRESHOLD=80

# Get the disk usage percentage
DISK_USAGE=$(df / | grep / | awk '{print $5}' | sed 's/%//g')

# Check if disk usage exceeds the threshold
if [ $DISK_USAGE -gt $DISK_THRESHOLD ]; then
  echo "Warning: Disk usage exceeded threshold ($DISK_USAGE%)" | mail -s
"Disk Usage Alert" admin@example.com
else
  echo "Disk usage is under control: $DISK_USAGE%"
fi
```

Explanation:

- **df**: Displays disk space usage.
- **awk**: Extracts the percentage of disk usage.
- **mail**: Sends an alert when the disk usage exceeds a specified threshold.
- **New Learner Insight**: This script checks disk usage and sends an alert if it exceeds the defined threshold, ensuring you can take action before running out of space.

🔹 **197. Check and Notify When a Service is Down**

Script:

```
#!/bin/bash
```

```
# Define the service name
SERVICE_NAME="nginx"

# Check if the service is running
SERVICE_STATUS=$(systemctl is-active $SERVICE_NAME)

# If the service is inactive, send an alert
if [ "$SERVICE_STATUS" != "active" ]; then
  echo "$SERVICE_NAME service is down!" | mail -s "$SERVICE_NAME
Service Alert" admin@example.com
else
  echo "$SERVICE_NAME service is running fine."
fi
```

Explanation:

- **systemctl is-active**: Checks the status of a service.
- **mail**: Sends an email notification if the service is not active.
- **New Learner Insight**: This script ensures that essential services are always up and running by notifying administrators if a service goes down.

🔹 198. Sync Files Between Two Directories

Script:

```
#!/bin/bash

# Define source and destination directories
SOURCE_DIR="/path/to/source"
DEST_DIR="/path/to/destination"
```

```
# Sync files from source to destination
echo "Syncing files..."
rsync -av --delete $SOURCE_DIR/ $DEST_DIR/

# Print success message
echo "Files synced successfully."
```

Explanation:

- **rsync**: Syncs files from one directory to another, with the --delete flag to remove files in the destination that no longer exist in the source.

- **New Learner Insight**: This script ensures that two directories stay synchronized, useful for backup or file mirroring.

🔹 199. Install and Configure Apache Web Server

Script:

```
#!/bin/bash

# Update the package list
echo "Updating package list..."
sudo apt update -y

# Install Apache web server
echo "Installing Apache..."
sudo apt install apache2 -y

# Start Apache and enable it on boot
echo "Starting Apache..."
```

```
sudo systemctl start apache2
sudo systemctl enable apache2

# Check Apache status
echo "Checking Apache status..."
sudo systemctl status apache2

# Print success message
echo "Apache web server installed and started."
```

Explanation:

- **sudo apt install apache2**: Installs the Apache web server.
- **systemctl**: Starts and enables Apache to run automatically on system boot.
- **New Learner Insight**: Apache is one of the most widely used web servers for hosting static and dynamic websites. This script automates its installation and configuration.

🔹 200. Automate Application Deployment with Git Pull

Script:

```
#!/bin/bash

# Define application directory
APP_DIR="/var/www/myapp"

# Navigate to the application directory
cd $APP_DIR
```

```
# Pull the latest changes from GitHub
echo "Pulling latest code from Git repository..."
git pull origin main

# Restart the application service
echo "Restarting application..."
sudo systemctl restart myapp.service

# Print success message
echo "Application updated and restarted."
```

Explanation:

- **git pull origin main**: Pulls the latest changes from the main branch of the repository.
- **systemctl restart myapp.service**: Restarts the service to reflect the new changes.
- **New Learner Insight**: This script is useful for automating code deployments and ensuring the latest version of an application is running.

🔹 201. Schedule Regular Backup with Cron Job

Script:

```
#!/bin/bash

# Define backup directory and source
SOURCE_DIR="/home/user/data"
BACKUP_DIR="/backups/$(date +'%Y%m%d')"
```

```
# Create backup directory
mkdir -p $BACKUP_DIR

# Copy files to backup directory
cp -r $SOURCE_DIR/* $BACKUP_DIR/

# Print success message
echo "Backup completed successfully. Backup stored in $BACKUP_DIR."
```

Explanation:

- **mkdir -p**: Creates the backup directory with the current date in its name.
- **cp -r**: Copies all files from the source directory to the backup directory.
- **New Learner Insight**: Backups are crucial for disaster recovery. This script ensures data is backed up regularly by scheduling it through a cron job.

🔹 202. Automate SSL Certificate Renewal for Multiple Domains

Script:

```
#!/bin/bash

# List of domains
DOMAINS=("domain1.com" "domain2.com" "domain3.com")

# Renew SSL certificates for each domain
for DOMAIN in "${DOMAINS[@]}"; do
  echo "Renewing SSL certificate for $DOMAIN..."
  sudo certbot certonly --standalone -d $DOMAIN
```

```
  # Restart web server to apply new certificates
  echo "Restarting Apache for $DOMAIN..."
  sudo systemctl restart apache2
done

# Print success message
echo "SSL certificates renewed and web server restarted for all domains."
```

Explanation:

- **certbot certonly --standalone**: Renews SSL certificates using Certbot.
- **systemctl restart apache2**: Restarts the Apache server to load the renewed certificates.
- **New Learner Insight**: This script automates the process of renewing SSL certificates for multiple domains, which is a routine task for maintaining secure connections.

🔹 **203. Check Disk Usage and Send Alert if Full**

Script:

```
#!/bin/bash

# Set disk usage threshold (in percentage)
THRESHOLD=90

# Get disk usage of root directory
DISK_USAGE=$(df / | grep / | awk '{print $5}' | sed 's/%//g')

# Check if disk usage exceeds the threshold
```

```
if [ $DISK_USAGE -ge $THRESHOLD ]; then
  echo "Warning: Disk usage is ${DISK_USAGE}% on /" | mail -s "Disk Usage Alert" admin@example.com
else
  echo "Disk usage is normal: ${DISK_USAGE}%"
fi
```

Explanation:

- **df /**: Displays the disk usage of the root directory.
- **awk and sed**: Extract the percentage of disk usage from the output.
- **mail**: Sends an alert if disk usage exceeds the threshold.
- **New Learner Insight**: This script helps in monitoring disk space usage and sends an alert when it crosses a critical level, preventing unexpected failures due to lack of space.

🔹 **204. Install and Configure Jenkins**

Script:

```
#!/bin/bash

# Update package list
echo "Updating package list..."
sudo apt update -y

# Install dependencies for Jenkins
echo "Installing dependencies..."
sudo apt install -y openjdk-11-jdk
```

```
# Add Jenkins repository and key
echo "Adding Jenkins repository..."
wget -q -O - https://pkg.jenkins.io/jenkins.io.key | sudo tee
/usr/share/keyrings/jenkins.asc
sudo sh -c 'echo deb http://pkg.jenkins.io/debian/ stable main >
/etc/apt/sources.list.d/jenkins.list'

# Install Jenkins
echo "Installing Jenkins..."
sudo apt update -y
sudo apt install jenkins -y

# Start Jenkins and enable it on boot
echo "Starting Jenkins..."
sudo systemctl start jenkins
sudo systemctl enable jenkins

# Print Jenkins status
echo "Jenkins installation complete."
sudo systemctl status jenkins
```

Explanation:

- **openjdk-11-jdk**: Installs the Java Development Kit required by Jenkins.
- **wget**: Downloads the Jenkins repository key and adds it to the system's list of trusted keys.
- **systemctl start jenkins**: Starts the Jenkins service.
- **New Learner Insight**: Jenkins is a popular automation tool for continuous integration and delivery. This script automates the installation and startup of

Jenkins.

🔹 **205. Automate Docker Container Deployment**

Script:

```
#!/bin/bash

# Define Docker image and container name
DOCKER_IMAGE="nginx:latest"
CONTAINER_NAME="my_nginx_container"

# Pull the latest Docker image
echo "Pulling latest Docker image $DOCKER_IMAGE..."
docker pull $DOCKER_IMAGE

# Run the Docker container
echo "Running Docker container $CONTAINER_NAME..."
docker run -d --name $CONTAINER_NAME -p 80:80 $DOCKER_IMAGE

# Print success message
echo "Docker container $CONTAINER_NAME deployed successfully."
```

Explanation:

- **docker pull**: Pulls the latest version of the Docker image.
- **docker run**: Runs the Docker container with the specified name and exposed port.
- **New Learner Insight**: Docker containers encapsulate applications and their dependencies, making deployments fast and consistent. This script

automates the process of pulling and running a Docker container.

🔹 206. Schedule Cron Job for System Updates

Script:

```
#!/bin/bash

# Define the log file for updates
LOG_FILE="/var/log/system_update.log"

# Update the system and log output
echo "Starting system update..." | tee -a $LOG_FILE
sudo apt update -y | tee -a $LOG_FILE
sudo apt upgrade -y | tee -a $LOG_FILE

# Print success message
echo "System update completed." | tee -a $LOG_FILE
```

Explanation:

- **tee -a**: Appends the output to the specified log file.
- **apt update and apt upgrade**: Updates the system packages to the latest version.
- **New Learner Insight**: Regular system updates are essential for security and stability. This script automates system updates and logs the process for tracking.

🔹 **207. Monitor Website Availability**

Script:

```
#!/bin/bash

# Define website URL
WEBSITE="https://www.example.com"

# Check if the website is accessible
echo "Checking availability of $WEBSITE..."
HTTP_STATUS=$(curl -s -o /dev/null -w "%{http_code}" $WEBSITE)

# Send alert if the website is down
if [ $HTTP_STATUS -ne 200 ]; then
  echo "Website $WEBSITE is down (HTTP Status: $HTTP_STATUS)" | mail -s "Website Down Alert" admin@example.com
else
  echo "Website $WEBSITE is up and running."
fi
```

Explanation:

- **curl -s -o /dev/null -w "%{http_code}"**: Sends a request to the website and retrieves the HTTP status code.
- **mail**: Sends an email alert if the website is down.
- **New Learner Insight**: This script checks the availability of a website and sends an alert if it's down, which is essential for uptime monitoring in DevOps environments.
