

# How to Automate Bash Scripts on Linux Using Cron and Systemd

Learn how to schedule and automate your Bash scripts on Linux using Cron jobs and Systemd services for efficient system management and task automation.

# Contents

01	Introduction	3
02	Prerequisites	3
03	Create Bash Scripts	3
04	Automate Bash Script on Linux Using Cron	6
05	Automate Bash Script on Linux as a System Service Using Systemd	7
06	Conclusion	10

# Introduction

Automating Bash scripts on Linux is essential for system administration tasks, such as installing applications and monitoring system logs. Cron is a job scheduler that runs tasks at specific intervals, while systemd serves as a service manager, handling applications through service profiles in Linux.

This article explains how to automate Bash scripts on Linux using Cron and Systemd to execute and manage system tasks automatically.

## Prerequisites

Before you begin:

- Deploy an [Ubuntu instance on Vultr](#) to use as your management workstation.
- Access the instance using SSH as a non-root sudo user.
- Install [LAMP stack](#).

## Create Bash Scripts

Bash scripts support multiple commands and functions that define how tasks run in a system. Follow the steps below to create new Bash scripts to back up critical web application files and monitor the system logs for errors.

1. Switch to your user's home directory.

```
CONSOLE
```

```
$ cd
```

2. Create a new `log-monitor.sh` script.

```
CONSOLE
```

```
$ nano log-monitor.sh
```

3. Add the following contents to the `log-monitor.sh` file.

```
BASH
#!/bin/bash

LOG_FILE="/var/log/syslog"
ERROR_LOG="/opt/data/error_log.txt"
WARNING_LOG="/opt/data/warning_log.txt"

tail -F "$LOG_FILE" | while read line; do
    if echo "$line" | grep -q "error"; then
        echo "$line" >> "$ERROR_LOG"
    elif echo "$line" | grep -q "warning"; then
        echo "$line" >> "$WARNING_LOG"
    fi
done
```

Save and close the file.

The above Bash script creates `error_log.txt` and `warning_log.txt` files in the `/opt/data` directory to store filtered result from the `/var/log/syslog` system logs file using a `while loop`. The script filters the result and writes it to the respective error or warning file every time a new error displays in the `/var/log/syslog` file.

4. Create another `backup.sh` script.

```
CONSOLE
$ nano backup.sh
```

5. Add the following contents to the file.

```
BASH
#!/bin/bash

DB_USER="user"
DB_PASSWORD="new-password"
```

```
DB_NAME="mysql"

DB_BACKUP_DIR="mysql_backup"
WEB_FILES_DIR="web_backup"

mkdir -p "$DB_BACKUP_DIR"
mkdir -p "$WEB_FILES_DIR"

DATE=$(date +"%Y%m%d_%H%M%S")

mysqldump -u"$DB_USER" -p"$DB_PASSWORD" "$DB_NAME" >
"$DB_BACKUP_DIR/db_backup_$DATE.sql"
echo "MySQL backup completed: $DB_BACKUP_DIR/
db_backup_$DATE.sql"

tar -czf "$WEB_FILES_DIR/web_backup_$DATE.tar.gz" /var/www/
html
echo "Web application backup completed: $WEB_FILES_DIR/
web_backup_$DATE.tar.gz"
```

Save and close the file.

The above Bash script connects to the MySQL database server using the `DB_...` variables. The script then creates `mysql_backup` and `web_backup` directories to store the database and web applications' backups.

6. Move the Bash script to a system-wide directory like `/opt`.

CONSOLE

```
$ sudo cp backup.sh log-monitor.sh /opt/
```

7. Create a new `data` directory under the system-wide directory you've chosen.

CONSOLE

```
$ sudo mkdir -p /opt/data
```

8. Change the directory permissions to `777` to allow all users to write files.

CONSOLE

```
$ sudo chmod 777 /opt/data
```

## Automate Bash Script on Linux Using Cron

Cron creates jobs that you can use to schedule and automatically run a Bash script at specific intervals. Follow the steps below to automate the Bash scripts you created earlier to automatically [back up the MySQL database](#) and web application files daily.

1. Open the Crontab editor.

CONSOLE

```
$ crontab -e
```

Select your desired text editor to edit the Crontab file when prompted. For example, select **1** to choose the `nano` text editor.

2. Add the following directive at the end of the file.

```
0 2 * * * /opt/backup.sh
```

Save and close the file.

The above cron job runs the `backup.sh` script daily at 2:00 A.M. to back up the database and web application files. In the above cron job:

- `0`: Runs the script at the start of a minute. Set a value between `0` and `59`.
- `2`: Runs the Bash script at 2:00 A.M. Specify an hour between `0` and `23`.
- `*`: Runs the Bash script every day of the month. Specify a day between `1` and `31`.
- `*`: Runs the Bash script every month. Specify a month between `1` and `12`.
- `*`: Runs the bash script daily. Specify a day of the week between `0` and `7`.

3. List all Cron jobs and verify that the new Bash script task is available.

CONSOLE

```
$ crontab -l
```

Output:

```
#  
# For more information see the manual pages of crontab(5) and cron(8)  
#  
# m h dom mon dow   command  
  
0 2 * * * ~/backup.sh
```

## Automate Bash Script on Linux as a System Service Using Systemd

Systemd starts system services automatically at boot and offers options for stopping and restarting the services. Follow the steps below to set up the monitoring Bash script you created earlier as a new system service and manage the service using systemd.

1. Create a new system service file, such as `logmonitor.service`.

CONSOLE

```
$ sudo nano /etc/systemd/system/logmonitor.service
```

2. Add the following contents to the `logmonitor.service` file.

BASH

```
[Unit]  
Description= Automatic System Logs Monitoring and Filtering  
After=network.target
```

```
[Service]
ExecStart=bash /opt/log-monitor.sh
Restart=always

[Install]
WantedBy=multi-user.target
```

Save and close the file.

The above service configuration runs the `log-monitor.sh` bash script in the `/opt` directory to monitor and filter log entries in the `/var/log/syslog` file.

3. Reload systemd to apply the service file changes.

CONSOLE

```
$ sudo systemctl daemon-reload
```

4. Enable the new system service to automatically start at boot.

CONSOLE

```
$ sudo systemctl enable logmonitor.service
```

Output:

```
Created symlink /etc/systemd/system/multi-user.target.wants/logmonitor.service
→ /etc/systemd/system/logmonitor.service.
```

5. Start the system service.

CONSOLE

```
$ sudo systemctl start logmonitor
```

6. View the system service status and verify it's running.

CONSOLE

```
$ sudo systemctl status logmonitor
```

## Output:

```
● logmonitor.service - Automatic System Logs Monitoring and Filtering
   Loaded: loaded (/etc/systemd/system/logmonitor.service; enabled; preset:
   enabled)
   Active: active (running) since Wed 2024-11-06 13:20:07 UTC; 11s ago
 Main PID: 14293 (bash)
    Tasks: 3 (limit: 1062)
   Memory: 784.0K (peak: 1.5M)
      CPU: 40ms
   CGroup: /system.slice/logmonitor.service
           └─14293 bash /opt/log-monitor.sh
           └─14308 tail -F /var/log/syslog
           └─14309 bash /opt/log-monitor.sh

Nov 06 13:20:07 Linux-server systemd[1]: Started logmonitor.service - Automatic
System Logs Monitoring and Filtering.
```

7. Run the following commands to simulate new error and warning entries in the `/var/log/syslog` file.

## CONSOLE

```
$ logger -p user.err "This is a simulated error log message
from MySQL"
$ logger -p user.err
"This is a simulated warning log message from Apache"
$ logger -p user.err "This is a simulated error log message
from PHP"
$ logger -p user.err
"This is a simulated Warning log message from MySQL"
```

8. List all files in the `/opt/data` directory and verify that the `error_log.txt` and `warning_log.txt` files are available.

## CONSOLE

```
$ ls /opt/data
```

## Output:

```
error_log.txt warning_log.txt
```

9. View the `error_log.txt` file and verify that only the error entries are available in the file.

```
CONSOLE
```

```
$ cat error.log
```

Output:

```
2024-11-06T13:33:48.010096+00:00 Server linuxuser: This is a simulated error log
message from MySQL
2024-11-06T13:33:48.010096+00:00 Server linuxuser: This is a simulated error log
message from PHP
```

## Conclusion

You have automated Bash scripts on a Linux server using Cron and Systemd. Automating Bash scripts allows you to run common system administration tasks, such as backups and error monitoring. To streamline system management, create multiple Bash scripts and Cron jobs, specifying the necessary commands for each.



VULTR

