# 🕒 Linux Job Scheduling with Cron & Scripts

Automating tasks is one of Linux’s biggest strengths. You can:

* write **shell scripts** to bundle multiple commands together,
* use **cron** to run those scripts or single commands on a schedule.

This guide walks through every piece step by step.

## 1️⃣ Understanding Crontab

crontab (“cron table”) is a per-user file that lists jobs for cron to execute.

### Basic Commands

|  |  |
| --- | --- |
| Command | Purpose |
| crontab -e | Edit your crontab (opens editor) |
| crontab -l | List your scheduled jobs |
| crontab -r | Remove all jobs for your user |
| sudo crontab -u <user> -l | (root only) View another user’s jobs |

**Example session:**

$ crontab -l  
no crontab for deepak  
$ crontab -e # add jobs

### The Crontab Line Format

Each job line:

minute hour day\_of\_month month day\_of\_week command\_to\_run

Five time fields + one command. The asterisk \* means “every possible value.”

|  |  |  |  |
| --- | --- | --- | --- |
| Field | Allowed Values | Example Value | Meaning |
| Minute | 0–59 | 0 | start of the hour |
| Hour | 0–23 | 18 | 6 PM |
| Day of Month | 1–31 | 10 | 10th day |
| Month | 1–12 | 12 | December |
| Day of Week | 0–6 | 0 or 7 for Sunday | Sunday |

### Scheduling Examples

|  |  |  |
| --- | --- | --- |
| **Schedule** | **Crontab Line** | **What Happens** |
| Every minute | \* \* \* \* \* /home/deepak/logtime.sh | Script runs every minute |
| Daily at 6 PM | 0 18 \* \* \* /home/deepak/sysinfo.sh | At 18:00 |
| On the 10th of each month at 6 PM | 0 18 10 \* \* /home/deepak/backup.sh | At 6 PM, day 10 |
| Every Sunday | \* \* \* \* 0 /home/deepak/weekly.sh | All day Sunday |

**Tip:** Always use full paths (/home/deepak/script.sh) and full paths to commands inside scripts (/usr/bin/find, /usr/bin/tar) when running from cron. Cron doesn’t load your normal shell’s PATH.

## 2️⃣ User Permissions & Root Control in Cron

Cron is powerful, but not every user should always have permission to schedule jobs. Linux has a built-in system to **control who can and cannot use cron**.

### 🔹 Default Behavior

1. **Every user can have their own crontab file.**
   * Example: If user **ajay** runs crontab -e, they get their own cron table.
   * Cron will execute their jobs under **ajay’s account**, not as root.
2. **The root user has full control.**
   * Root can **view, edit, or delete** the cron jobs of any user.

For example, if root wants to **see what cron jobs ajay has scheduled**, they can run:

sudo crontab -u ajay -l

👉 This will list all jobs in ajay’s crontab. 👉 Similarly, root can **edit** ajay’s crontab with:

sudo crontab -u ajay -e

### 🔹 Restricting Users

Sometimes you don’t want certain users to be able to schedule cron jobs (for security or resource reasons).

There are **two special files** that control access:

1. /etc/cron.allow → If this file exists, **only users listed inside it** can use cron.
2. /etc/cron.deny → Users listed inside this file are **blocked from using cron**.

📌 Most Linux systems use /etc/cron.deny by default.

### 🔹 Example: Denying a User

Suppose we don’t want user **neeta** to use cron. As root, run:

echo "neeta" | sudo tee -a /etc/cron.deny

👉 Let’s break this down:

* echo "neeta" → Prints the word **neeta**.
* | (pipe) → Sends that output into the next command.
* sudo tee -a /etc/cron.deny →
  + tee writes input into a file.
  + -a means **append** (don’t overwrite existing content).
  + /etc/cron.deny is the file where we’re adding her name.

✅ Effect: The word **neeta** gets added as a new line inside /etc/cron.deny.

Now, if neeta tries to run crontab -e, she will see:

You (neeta) are not allowed to use this program (crontab)

👉 This means she can no longer create or edit cron jobs.

### 🔹 Example: Allowing Only Specific Users

If you want **only a few users** to be able to use cron, create /etc/cron.allow and list their names inside:

sudo nano /etc/cron.allow

Add:

ajay  
deepak

👉 Now, **only ajay and deepak** can use cron. 👉 All other users will be blocked automatically.

💡 **Real-World Example:** Imagine a system with many users, but only the admins should schedule jobs.

* You’d create /etc/cron.allow and list only the admins.
* This prevents mistakes like a random user scheduling a heavy script every minute, which could overload the server.

## 3. Writing Shell Scripts for Cron Jobs

When cron jobs become longer or involve multiple commands, it’s better to put them inside a **shell script** instead of writing everything directly in crontab. This makes the job easier to manage, debug, and reuse.

### Step 1 – Create a Script

Open a new file for your script:

vi /home/deepak/job1.sh

Add the following content inside:

#!/bin/bash  
echo "Backup started at $(date)" >> /home/deepak/backup.log

🔎 **Explanation:**

* #!/bin/bash → Tells Linux to run the file using the Bash shell.
* echo "Backup started at $(date)" → Prints a message with the current date & time.
* >> /home/deepak/backup.log → Appends the message to a log file instead of showing it on screen.

### Step 2 – Make the Script Executable

chmod +x /home/deepak/job1.sh

This allows Linux to **run the file as a program**. Without this, cron won’t execute it.

### Step 3 – Add It to Crontab

Edit your crontab:

crontab -e

Add the line:

\* \* \* \* \* /home/deepak/job1.sh

🔎 **Explanation:**

* \* \* \* \* \* → Run every minute.
* /home/deepak/job1.sh → Full path to your script.

### Step 4 – Check If It’s Running

Look inside the log file:

tail -f /home/deepak/backup.log

You’ll see entries like:

Backup started at Tue Sep 17 10:00:01 IST 2025  
Backup started at Tue Sep 17 10:01:01 IST 2025

✅ Congratulations! You’ve automated your first task with cron + scripts.

## 4️⃣ Practical Script Examples

Now let’s explore some **real-world scripts** that can be scheduled using cron.

### Example 1 – Log Current Date & Time Every Minute

Create script /home/deepak/logtime.sh:

#!/bin/bash  
echo "Current time: $(date)" >> /home/deepak/time.log

Add to crontab:

\* \* \* \* \* /home/deepak/logtime.sh

Check log file:

tail -f /home/deepak/time.log

Output:

Current time: Tue Sep 17 10:02:01 IST 2025  
Current time: Tue Sep 17 10:03:01 IST 2025

### Example 2 – Collect System Information Daily at 6 PM

Create script /home/deepak/sysinfo.sh:

#!/bin/bash  
{  
echo "----- $(date) -----"  
/bin/hostname  
/bin/uname -r  
/usr/bin/free -m  
/usr/bin/last -n 5  
echo "-------------------"  
} >> /home/deepak/result.txt

Schedule in crontab:

0 18 \* \* \* /home/deepak/sysinfo.sh

At **6 PM every day**, it will save:

----- Tue Sep 17 18:00:00 IST 2025 -----  
myserver  
6.6.32-xyz  
 total used free  
Mem: 1994 350 1644  
deepak pts/0 2025-09-17 17:50  
...  
-------------------

### Example 3 – Complex Admin Tasks (Once a Year)

Script /home/deepak/admin\_tasks.sh:

#!/bin/bash  
# Copy all files owned by 'sara' to /backup  
/usr/bin/find / -user sara -exec /bin/cp {} /backup/ \;  
  
# Delete all files owned by 'harry'  
/usr/bin/find / -user harry -exec /bin/rm {} \;  
  
# Archive and compress /etc directory  
/usr/bin/tar -cvJf /backup/etc.tar.xz /etc

Schedule on **10 December at 6:30 PM**:

30 18 10 12 \* /home/deepak/admin\_tasks.sh

At that time:

* Files owned by sara are copied.
* Files owned by harry are removed.
* /etc is archived into /backup/etc.tar.xz.

## 5️⃣ Debugging Cron Jobs

Cron runs silently, so you won’t see errors on your screen.

### Check System Log

sudo tail -f /var/log/cron

Example output:

Sep 17 10:00:01 myserver CRON[2489]: (deepak) CMD (/home/deepak/job1.sh)  
Sep 17 10:01:01 myserver CRON[2512]: (deepak) CMD (/home/deepak/job1.sh)

If a command is missing, you’ll see command not found here.

### Capture Errors in a Log File

Redirect both output and errors to a log:

\* \* \* \* \* /home/deepak/job1.sh >> /home/deepak/job1.log 2>&1

* >> → Append normal output.
* 2>&1 → Capture errors (stderr) too.

## 6️⃣ Recap Table

|  |  |  |
| --- | --- | --- |
| Task | Command | What It Does |
| Edit your cron | crontab -e | Opens editor to add jobs |
| List jobs | crontab -l | Shows your current schedule |
| Remove all jobs | crontab -r | Deletes all your cron jobs |
| View another user’s jobs (root) | sudo crontab -u ajay -l | Show ajay’s cron tasks |
| Monitor log | sudo tail -f /var/log/cron | Watch cron jobs as they run |

👉 This way, you can use **scripts + cron** to automate almost anything: logging, backups, monitoring, or system administration.