

Job Scheduling

LinuXpert Systems

No.5, Dr. Ambedkar St, Rathinapuram,

Nanganallur, Chennai – 600 061.

Phone: 98841 65649, 90808 01382

E-mail: baskar@linuxpert.in

TABLE OF CONTENTS

1. cron 1

2. crontab 7

3. at 13

4. batch 18

1. cron

`cron` is a **time-based job scheduler** in Linux/Unix systems. It is used to **automatically run commands or scripts at a specific time, date, or interval** without manual execution.

The word **cron** comes from **chronos**, the Greek word for **time**.

Basic Concept

A **cron job** is a scheduled task managed by the **cron daemon (`crond`)**.

Examples:

- Run a backup every day at midnight
- Delete temporary files every week
- Send reports every Monday morning
- Monitor system status every 5 minutes

Check if cron service is running

```
systemctl status cron
```

or on some distributions:

```
systemctl status crond
```

Start cron service:

```
sudo systemctl start cron
```

Enable at boot:

```
sudo systemctl enable cron
```

Managing Cron Jobs

The main command used by users is:

```
crontab
```

View existing cron jobs

```
crontab -l
```

Example output:

```
0 2 * * * /home/user/backup.sh
```

Meaning:

Run `backup.sh` every day at 2:00 AM.

Edit cron jobs

```
crontab -e
```

This opens the cron table in an editor.

Example:

```
# m h dom mon dow command
```

```
0 0 * * * /home/user/backup.sh
```

Remove all cron jobs

```
crontab -r
```

This deletes all scheduled jobs for the current user.

Cron Time Format

A cron entry has **5 time fields** + **command**:

```
* * * * * command
│ │ │ │ │
│ │ │ │ └── Day of week (0-7)
│ │ │ └──── Month (1-12)
│ │ └────── Day of month (1-31)
│ └──────── Hour (0-23)
└────────── Minute (0-59)
```

Examples

1. Run a script every minute

```
* * * * * /home/user/test.sh
```

Runs every minute.

---

2. Run backup every day at 1 AM

```
0 1 * * * /home/user/backup.sh
```

---

3. Run every Sunday at midnight

```
0 0 * * 0 /home/user/weekly.sh
```

---

4. Run every 5 minutes

```
*/5 * * * * /home/user/check.sh
```

---

5. Run every month on the 1st day at 6 AM

```
0 6 1 * * /home/user/report.sh
```

Special Cron Shortcuts

Linux provides shortcuts:

Shortcut	Meaning	Equivalent
`@reboot`	Run after system startup	-
`@yearly`	Once a year	`0 0 1 1 *`
`@monthly`	Once a month	`0 0 1 * *`
`@weekly`	Once a week	`0 0 * * 0`
`@daily`	Every day	`0 0 * * *`
`@hourly`	Every hour	`0 * * * *`

Example:

```
@reboot /home/user/startup.sh
```

Runs when the system boots.

System Cron Locations

Apart from user cron jobs, Linux has system-wide cron directories:

```
/etc/crontab
```

and:

```
/etc/cron.hourly/
/etc/cron.daily/
/etc/cron.weekly/
/etc/cron.monthly/
```

Example:

Place a script in:

```
/etc/cron.daily/
```

It will run daily automatically.

Difference Between `cron` and `at`

cron	at
Repeated tasks	One-time task
Runs periodically	Runs once
Example: daily backup	Example: shutdown after 30 minutes

Example `at`:

```
echo "shutdown now" | at 11:00 PM
```

Useful Cron Commands

List another user's cron jobs (root)

```
sudo crontab -u username -l
```

Edit another user's cron jobs

```
sudo crontab -u username -e
```

View cron logs

Ubuntu/Debian:

```
grep CRON /var/log/syslog
```

RHEL/CentOS:

```
grep CRON /var/log/cron
```

Real-world Examples

Automatic database backup

```
30 2 * * * mysqldump database > /backup/db.sql
```

Runs database backup every day at 2:30 AM.

Clean temporary files

```
0 3 * * * rm -rf /tmp/*
```

Deletes temporary files daily at 3 AM.

Monitor disk usage

```
*/10 * * * * df -h > /home/user/disk_report.txt
```

Stores disk usage every 10 minutes.

Summary

Command	Purpose
`cron`	Background scheduling service
`crontab -e`	Edit scheduled jobs
`crontab -l`	List jobs
`crontab -r`	Remove jobs
`systemctl status cron`	Check cron service

In simple terms:

`cron` = Linux's automatic task scheduler that runs commands/scripts at predefined times.

2. crontab

`crontab` stands for **"cron table"**. It is a Linux command used to **schedule commands or scripts to run automatically at specific times**.

The `cron` service runs in the background and checks the `crontab` files to execute scheduled tasks.

Common uses:

- Run backups automatically
- Schedule system maintenance
- Run scripts periodically
- Send reports/emails automatically
- Clean temporary files

Syntax

```
crontab [options]
```

Important `crontab` Options

1. View current user's cron jobs

```
crontab -l
```

Example output:

```
0 2 * * * /home/user/backup.sh
```

Meaning:

Run `backup.sh` every day at 2:00 AM.

2. Edit cron jobs

```
crontab -e
```

Opens the cron file in an editor.

Example:

```
crontab -e
```

Add:

```
*/5 * * * * /home/user/test.sh
```

This runs `test.sh` every 5 minutes.

---

3. Remove all cron jobs

```
crontab -r
```

Example:

```
crontab -r
```

Deletes all scheduled tasks for the current user.

> Be careful — it does not ask for confirmation.

---

4. Edit another user's crontab (root only)

```
crontab -u username -e
```

Example:

```
sudo crontab -u john -e
```

Edits John's scheduled jobs.

---

5. List another user's cron jobs

```
sudo crontab -u john -l
```

Crontab File Format

A cron entry has **5 time fields + command**:

```
minute hour day_of_month month day_of_week command
```

Example:

```
30 22 * * * /home/user/backup.sh
```

Meaning:

Run `/home/user/backup.sh` every day at **10:30 PM**.

Time Fields

Field	Allowed Values	Meaning
Minute	0-59	Minute of hour
Hour	0-23	Hour of day
Day	1-31	Day of month
Month	1-12	Month
Day of Week	0-7	Sunday-Saturday

Special Characters

* (Any value)

Example:

```
* * * * * command
```

Runs every minute.

, (Multiple values)

Example:

```
0 8,12,18 * * * command
```

Runs at:

- 8 AM
- 12 PM
- 6 PM

- (Range)

Example:

```
0 9-17 * * * command
```

Runs every hour from 9 AM to 5 PM.

/ (Step value)

Example:

```
*/10 * * * * command
```

Runs every 10 minutes.

Common Examples

1. Run a script every day at midnight

```
0 0 * * * /home/user/script.sh
```

---

2. Run every 5 minutes

```
*/5 * * * * /home/user/script.sh
```

3. Run every Sunday at 6 AM

```
0 6 * * 0 /home/user/backup.sh
```

---

4. Delete temporary files every night

```
0 1 * * * rm -rf /tmp/*
```

Runs daily at 1 AM.

---

5. Restart a service every day

```
0 3 * * * systemctl restart nginx
```

Runs at 3 AM daily.

System Cron Directories

Linux also has predefined cron directories:

`/etc/cron.hourly/`

Runs every hour.

`/etc/cron.daily/`

Runs every day.

`/etc/cron.weekly/`

Runs every week.

`/etc/cron.monthly/`

Runs every month.

Check Cron Service Status

On systems using `systemd`:

```
systemctl status cron
```

or on some distributions:

```
systemctl status crond
```

Start cron:

```
sudo systemctl start cron
```

Enable at boot:

```
sudo systemctl enable cron
```

Difference Between `cron` and `crontab`

Command	Purpose
`cron`	Background service that executes scheduled tasks
`crontab`	Command used to create/manage scheduled tasks

Real-world Example

Automatically backup `/home` every night:

Edit:

```
crontab -e
```

Add:

```
0 2 * * * tar -czf /backup/home.tar.gz /home
```

Every day at **2:00 AM**, Linux creates a compressed backup.

3. at

The **`at`** command is used to **schedule a command or task to run once at a specific future time**.

Unlike **`cron`**, which is used for **repeating tasks**, `at` is used for **one-time execution**.

Syntax

```
at [time]
```

After entering the command, you type the commands you want to run, then press **Ctrl + D** to save the job.

Basic Examples

1. Run a command after a specific time

Example: Run `backup.sh` at 10:30 PM today:

```
at 22:30
```

Then enter:

```
/home/user/backup.sh
```

Press:

```
Ctrl + D
```

The command will execute at 10:30 PM.

2. Schedule a command for tomorrow

```
at tomorrow
```

Example:

```
at tomorrow
```

Enter:

```
echo "System maintenance completed" > /tmp/status.txt
```

Press:

```
Ctrl + D
```

---

3. Schedule after a delay

Run a command after 1 hour:

```
at now + 1 hour
```

Example:

```
at now + 1 hour
```

Then:

```
shutdown -h now
```

The system will shut down after one hour.

Common Time Formats

Command	Meaning
`at 5:00 PM`	Today at 5 PM
`at 23:30`	Today at 11:30 PM
`at tomorrow`	Tomorrow
`at noon`	12 PM
`at midnight`	12 AM
`at now + 10 minutes`	After 10 minutes
`at now + 2 hours`	After 2 hours
`at now + 3 days`	After 3 days

Viewing Scheduled `at` Jobs

Use:

```
atq
```

or:

```
at -l
```

Example output:

```
3   Mon Jul 27 18:30:00 2026 a user
4   Tue Jul 28 09:00:00 2026 a user
```

Here:

- 3 and 4 are job IDs
- The time shows when they will execute

Viewing a Scheduled Job

Use:

```
at -c job_id
```

Example:

```
at -c 3
```

This displays the commands stored in job number 3.

Removing a Scheduled Job

Use:

```
atrm job_id
```

Example:

```
atrm 3
```

Deletes job number 3.

Example: Schedule System Shutdown

Schedule shutdown at 11 PM:

```
at 23:00
```

Enter:

```
shutdown -h now
```

Press:

```
Ctrl + D
```

The shutdown command will run at 11 PM.

`at` vs `cron`

Feature	`at`	`cron`
Purpose	One-time jobs	Repeated jobs
Execution	Once	Periodic
Configuration	Command line	Crontab file
Example	Shutdown tomorrow	Daily backup

Example:

One-time backup:

```
at 18:00
```

Daily backup:

```
crontab -e
```

Add:

```
0 18 * * * /home/user/backup.sh
```

Managing the `at` Service

On Ubuntu/Debian:

Check status:

```
systemctl status atd
```

Start service:

```
sudo systemctl start atd
```

Enable at boot:

```
sudo systemctl enable atd
```

The background service that executes `at` jobs is called **`atd` (at daemon)**.

Common Uses of `at`

- Schedule system shutdown/reboot
- Run a script later
- Send delayed emails
- Perform maintenance tasks
- Run a one-time backup
- Execute commands after a long-running process finishes

In short:

`at` = "Run this command once at a specified future time."

4. batch

What is `batch`?

`batch` is a Linux command used to **schedule a command or job to run later when the system load is low**.

It is similar to the `at` command, but instead of running at a fixed time, `batch` waits until the system has enough free resources (low CPU load) before executing the job.

It is part of the **at daemon (`atd`)** package.

Syntax

```
batch
```

or

```
batch < command_file
```

or

```
echo "command" | batch
```

How it works

When you run:

```
batch
```

the command opens an input prompt where you type commands that should be executed later.

Example:

```
$ batch
warning: commands will be executed using /bin/sh
at> tar -czf backup.tar.gz /home/user
at> <Ctrl+D>
job 5 at Mon Jul 27 15:50:00 2026
```

The backup command will run automatically when the system is not busy.

Examples

1. Run a command when the system is idle

```
echo "updatedb" | batch
```

This runs the `updatedb` command when the system load is low.

---

2. Schedule a backup

```
echo "tar -czf /backup/home.tar.gz /home" | batch
```

The backup will execute when the machine has spare CPU capacity.

---

3. Run multiple commands

Create a file:

```
nano job.txt
```

Add:

```
date
df -h
du -sh /home
```

Submit it:

```
batch < job.txt
```

Check scheduled batch jobs

Use:

```
atq
```

Example:

```
5   Mon Jul 27 15:50:00 2026 a user
```

The letter `a` indicates an `at` job.

Remove a batch job

Find the job number:

```
atq
```

Remove it:

```
atrm 5
```

Difference between `at`, `batch`, and `cron`

Command	Purpose	Execution time
`at`	Run a command once at a specific time	Fixed time
`batch`	Run a command once when system is idle	When load is low
`cron`	Run commands repeatedly	Scheduled intervals

Example:

`at`

Run backup at 2 AM:

```
at 02:00
```

`batch`

Run backup whenever the system is free:

```
echo "backup.sh" | batch
```

`cron`

Run backup every day at 2 AM:

```
0 2 * * * backup.sh
```

Check if `atd` service is running

`batch` requires the **at daemon**:

```
systemctl status atd
```

Start it:

```
sudo systemctl start atd
```

Enable at boot:

```
sudo systemctl enable atd
```

Useful options

List jobs

```
atq
```

Remove a job

```
atrm job_number
```

Show job details

```
at -c job_number
```

Real-world use cases

- Running large backups when the server is idle
- Compressing large log files
- Database maintenance tasks
- Generating reports without affecting users
- Running CPU-intensive scripts during low activity periods

In short:

`batch` = "Run this command later when the computer is less busy."