Engineer

cron

the Unix scheduler that runs the commands in a crontab

What it is

cron is the Unix service that starts commands at set times, with nobody logged in to start them. Each user's schedule is kept in a crontab: a plain text table with one job per line. The same word names the `crontab` command, which installs, prints and deletes that table.

POSIX standardises the command and the table format, but not the program that reads the tables. Its rationale says so plainly: "All references to a cron daemon and to cron files have been omitted." Which daemon runs the jobs is left to each system. The cron manual pages this page draws on come from cronie, whose crontab(5) names Paul Vixie as author and calls its format a V7-style one with extensions that stay compatible.

A line has six fields separated by blanks. The first five say when: minute (0 to 59), hour (0 to 23), day of the month (1 to 31), month (1 to 12) and day of the week (0 to 6, where 0 is Sunday). Each field is an asterisk for every value, a number, a hyphen range such as `1-5`, or a comma list of those. The sixth field is the command, which `sh` runs. That is where the standard stops. Steps such as `*/15`, three-letter names such as `mon`, a `7` for Sunday, a username column in system tables and shorthands such as `@daily` are cronie extensions; a table that uses them is no longer portable.

`crontab -e` opens your table in an editor, `crontab -l` prints it and `crontab -r` removes it, while `crontab file` replaces it with the contents of a file. cronie also keeps system tables in `/etc/crontab` and `/etc/cron.d/`, where an extra field between the times and the command names the user to run as. POSIX expected as much: its rationale notes that BSD-derived systems are likely to append a user field.

Use the concept in a decision

Put “What is cron?” to work in a real decision. We can help relate the concept to the system you are building or buying. Day-to-day care of servers, mail, accounts and updates, so the things everybody relies on stay boring.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Ran 10 PostgreSQL and MS SQL servers on Ubuntu — 10 database servers deployed and kept healthy

Ran 10 PostgreSQL and MS SQL servers on Ubuntu Get in touch →

Why it matters

Much of the upkeep of a server happens while nobody is watching: old files cleared, reports mailed, databases dumped, backups taken. cron is the standard way to say “run this at three in the morning”, and the utility has been in the standard since Issue 2, as the change history of its POSIX page records. A table that sticks to the POSIX fields reads the same on any conforming system.

The weakness is built in. A scheduled job has no audience, so when it breaks, no screen turns red. POSIX says that output and errors a command leaves unredirected are mailed to the user by a method each implementation chooses. On a host where local mail goes nowhere, that is where the error goes too. Most of the trouble with cron traces back to this silence.

Two neighbours now share the work. On Linux systems run by systemd, a timer unit can start the same job and adds what cron lacks: it can catch up on a run missed while the machine was off, spread start times at random, and it never starts a second copy of a service that is still running. Inside PostgreSQL, the `pg_cron` extension reads the same five-field schedules, and a background worker in the database keeps track of the jobs. Our River job-queue project is an example: in October 2026 its page counts 20 `pg_cron` jobs beneath the queue, which prune partitions, rotate salts and refresh aggregates, housekeeping the page says belongs in the database.

What it looks like in practice

Take two jobs: a weekday clean-up and a nightly database dump. The table sets its own environment first, because a job does not inherit the settings of your login shell. The `NAME=value` lines and the `MAILTO` variable are cronie features, not POSIX ones.

```
SHELL=/bin/sh
PATH=/usr/local/bin:/usr/bin:/bin
MAILTO=ops@example.com
# min hour dom mon dow command
15 3 * * 1-5 find "$HOME" -name core -exec rm -f {} +
30 2 * * * flock -n /var/lock/dump-db dump-db >> "$HOME/dump.log" 2>&1
0 * * * * date +\%s >> "$HOME/hourly.log"
```

Read the first job as “at 03:15, Monday to Friday”. The second runs every night at 02:30, writes its output and errors to a log, and is wrapped in `flock -n`, which gives up at once instead of waiting when the lock is already held, so a dump that runs long never has a twin. The third shows the escaped percent sign: without the backslash, cron would cut the command at `%`.

Keep the table in a file under version control and install it whole, rather than editing it live:

```
crontab jobs.cron
crontab -l
```

The second command prints what cron will actually read. cronie’s `-T` option checks a file’s syntax before you install it, and the file must end with a newline.

The same nightly dump as a systemd timer takes two units. The service says what to run, the timer says when:

```
# /etc/systemd/system/dump-db.service
[Service]
Type=oneshot
ExecStart=/usr/local/bin/dump-db

# /etc/systemd/system/dump-db.timer
[Timer]
OnCalendar=*-*-* 02:30:00
Persistent=true
RandomizedDelaySec=10min

[Install]
WantedBy=timers.target
```

`Persistent=true` stores the time of the last run on disk and starts a missed run as soon as the timer comes back. `RandomizedDelaySec=` adds a random delay of up to ten minutes, so a fleet of machines with the same file does not hit the database server in the same second. `systemd-analyze calendar '*-*-* 02:30:00'` prints the normalised expression and when it will next fire, which is the quickest way to test a schedule before trusting it.

A schedule says when a job starts, not whether it did its work. On a 464 MB host of ours, the dead-man's switch turns that round: a timer starts a script four times an hour, with jitter, and the script pings an outside service only when neither memory nor disk has fallen below its threshold. When the pings stop, the alarm goes off.

Where it goes wrong

The command works in your shell and not in cron. POSIX promises a job only `HOME`, `LOGNAME`, `PATH` and `SHELL`, and the `PATH` need only find the standard utilities. Nothing from your profile is loaded. The fix is to use absolute paths or set `PATH` at the top of the table, and to test the job with that same sparse environment.

Errors vanish. Unredirected output is mailed. cronie sends it to the owner of the table, to the address in `MAILTO`, or nowhere at all when `MAILTO=""`. The fix is to redirect output to a log someone reads and, for jobs that matter, to have the job report success to something that complains when the reports stop, as the dead-man's switch above does.

A percent sign splits the command. In the command field `%` becomes a newline, only the first line runs and the rest is fed to it as standard input, so `date +%s` breaks. The fix is to escape it as `\%`, or to move the command into a script and call that.

Two day fields mean either, not both. When both the day of the month and the day of the week are restricted, a day matching either runs the job. POSIX's own example, `0 0 1,15 * 1`, fires on the 1st, the 15th and every Monday. The fix is to restrict only one of the two, or use a systemd calendar, where `Mon *-*-01..07 09:00` matches only when the weekday and the date both fit: the first Monday of the month.

Steps restart every hour. cronie evaluates a step only within its own field, so `*/35` in the minutes runs at :00 and :35, not every 35 minutes, and `*/23` in the hours runs at midnight and 23:00. The fix is a step that divides the range evenly, or a systemd timer with `OnUnitActiveSec=`, which counts from the last time the service was started.

Clocks change. cronie documents that times skipped by a daylight-saving change never match, so those jobs do not run, and that repeated times make jobs run twice. The fix is to schedule outside the hours your time zone shifts, or to set `CRON_TZ` to a zone without daylight saving.

The last line has no newline. cronie wants every entry to end in a newline; when the final one does not, it treats the table as at least partly broken and writes a warning to syslog. The fix is a trailing newline, which an editor setting or a check in review can guarantee.

Runs overlap. The crontab format has no field for what should happen when the previous run is still going. A systemd timer leaves a running service alone. If the timer's next scheduled time passed while the service was still running, the timer starts the service again the moment it finishes, unless `DeferReactivation=` is set; `pg_cron` makes a second instance wait until the first one completes. The fix for plain cron is a lock, such as the `flock -n` in the example above.

The machine was off. A crontab line names a moment, and the format has no way to say "run it later if that moment passed". The fix, where systemd runs the machine, is `Persistent=true` on an `OnCalendar=` timer.

Every machine at once. A table copied to a hundred hosts starts a hundred jobs in the same minute. The fix is `RandomizedDelaySec=` on a systemd timer, or in cronie the `RANDOM_DELAY` variable or a tilde range such as `6~15`, which picks one random minute when the table is read.

A bare `crontab` wipes the table. Typed with no argument, the command waits for a new table on standard input, and an end-of-file (usually Ctrl-D) replaces the current one with nothing. The fix is to press the interrupt key instead, and to keep the table in a file you install with `crontab file`.

Further reading

- crontab, POSIX.1-2024 — the standard's six fields, default environment, mail rule and rationale.
- crontab(5), cronie — steps, `MAILTO`, `CRON_TZ`, `RANDOM_DELAY`, daylight saving and caveats.
- systemd.timer(5) — `OnCalendar=`, `Persistent=`, `RandomizedDelaySec=` and running services.
- systemd.time(7) — the calendar event syntax, its shorthands and how to validate it.
- flock(1) — wrapping a command in a lock, with `-n` to give up when it is held.
- pg_cron — cron schedules for SQL jobs, kept inside PostgreSQL as an extension.

Glossary

Choose for your actual constraints

Bring the deployment path, the operational risks and the people on call. We can scope a change with checks, a recovery route and clear ownership.

See what it costs Get in touch →

Make the terminology practical

Ask how "What is cron?" applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/cron/