cron i crontab: zadania cykliczne

Poziom: podstawy

Dotyczy: wszystkie dystrybucje (Vixie cron / cronie), różnice Debian/Ubuntu zaznaczone

Czas: ~18 min

`cron` *to demon, który co minutę sprawdza, czy nie nadszedł czas uruchomić którejś z zapisanych komend, i jeśli tak — uruchamia ją, niezależnie od tego, czy ktoś jest zalogowany. To najstarszy i wciąż najczęściej spotykany mechanizm zadań cyklicznych w Linuksie.*

Skróty użyte na tej stronie (7)

Skrót	Co znaczy
CPU	Central Processing Unit: procesor
MTA	Mail Transfer Agent: serwer przekazujący pocztę dalej, np. Postfix
NFS	Network File System: udostępnianie katalogów po sieci w świecie Linuksa
RHEL	Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa
RPM	RPM Package Manager: format paczek i narzędzie instalujące oprogramowanie w RHEL/Rocky, dnf korzysta z niego pod spodem
SLES	SUSE Linux Enterprise Server: płatna dystrybucja Linuksa firmy SUSE, konkurent Rocky/RHEL
SSH	Secure Shell: szyfrowane zdalne logowanie do serwera

Fiszka

TYLKO ODCZYT

Wypisanie własnego harmonogramu: `crontab -l`

```
crontab -l
```

Pokazujemy zawartość crontaba zalogowanego użytkownika — dokładnie te linie, które demon `cron` odczytuje co minutę. Jeśli użytkownik nigdy nie miał crontaba, komenda kończy się komunikatem błędu, nie pustym wynikiem.

Przykład:

```
no crontab for adm_user
```

Jeśli crontab już istnieje, wygląda mniej więcej tak:

Przykład:

```
SHELL=/bin/bash
MAILTO=""
0 2 * * * /usr/local/bin/backup.sh > /var/log/backup.log 2>&1
*/15 * * * * /usr/local/bin/check-disk.sh
```

crontab = cron table: tabela zadań cyklicznych

TYLKO ODCZYT

Edycja własnego harmonogramu: `crontab -e`

```
crontab -e
```

Otwieramy crontab w edytorze wskazanym przez zmienną `$EDITOR` (domyślnie zwykle `vi`); po zapisaniu polecenie **samo sprawdza składnię** — błędną linię odrzuci z komunikatem i pozwoli poprawić, zanim cokolwiek nadpisze. To jedyny bezpieczny sposób edycji; ręczna edycja pliku pod `/var/spool/cron/` pomija tę walidację.

Przykład:

```
crontab: installing new crontab
```

crontab = cron table: tabela zadań cyklicznych

TYLKO ODCZYT

Usunięcie całego harmonogramu: `crontab -r`

```
crontab -r
```

Ostrzeżenie: kasuje cały crontab natychmiast, bez potwierdzenia i bez kopii zapasowej. Nie ma opcji „na pewno?", nie ma powiadomienia — jedna literówka zamiast `-e` i zadania, które ktoś pielęgnował miesiącami, znikają bezpowrotnie. Przed skasowaniem warto zrobić `crontab -l > backup-crontab.txt`.

crontab = cron table: tabela zadań cyklicznych

TYLKO ODCZYT

Podgląd cudzego crontaba: `crontab -l -u nazwa_usera`

```
sudo crontab -l -u nazwa_usera
```

Tylko root (albo ktoś z uprawnieniami sudo) może zobaczyć crontab innego użytkownika. Przydaje się przy diagnozowaniu „coś się uruchamia w nocy, ale nie wiadomo, z czyjego konta".

Przykład:

```
30 3 * * * /opt/app/scripts/nightly-sync.sh
```

crontab = cron table: tabela zadań cyklicznych

TYLKO ODCZYT

Wczytanie crontaba z pliku

```
crontab plik-z-zadaniami.txt
```

Zamiast otwierać edytor, podstawiamy gotowy plik — **cały dotychczasowy crontab zostaje nim nadpisany**, nie dopisany. Wygodne przy wdrażaniu tego samego zestawu zadań przez automatyzację (Ansible, skrypt provisioningowy).

crontab = cron table: tabela zadań cyklicznych

TYLKO ODCZYT

Pięć pól harmonogramu

Przykład:

```
*  *  *  *  *  komenda
|  |  |  |  |
|  |  |  |  └─ dzień tygodnia (0-7, 0 i 7 to niedziela)
|  |  |  └──── miesiąc (1-12)
|  |  └─────── dzień miesiąca (1-31)
|  └────────── godzina (0-23)
└───────────── minuta (0-59)
```

Każde pole akceptuje pojedynczą wartość, gwiazdkę (dowolna wartość), listę, zakres albo krok — opisane niżej. Gdy w obu tych polach jednocześnie poda się coś innego niż gwiazdkę — dzień miesiąca i dzień tygodnia — cron traktuje je jako **alternatywę** (albo–albo), nie koniunkcję — klasyczna pułapka opisana w *Częste błędy*.

TYLKO ODCZYT

Ustawienie kroku wykonania: `*/5`

Przykład:

```
*/5 * * * * /usr/local/bin/check-disk.sh
```

Uruchamiamy zadanie co 5 minut: 00, 05, 10, 15... `*/N` w polu godzin oznacza co N godzin od północy, nie „co N godzin od teraz".

TYLKO ODCZYT

Ustawienie zakresu: `1-5`

Przykład:

```
0 9 * * 1-5 /usr/local/bin/raport-dzienny.sh
```

Zakres w polu dnia tygodnia — tu: od poniedziałku (1) do piątku (5), czyli dni robocze. Zakresy łączy się z listami, np. `1-3,5`.

TYLKO ODCZYT

Ustawienie listy wartości: `1,15`

Przykład:

```
0 3 1,15 * * /usr/local/bin/rozliczenie.sh
```

Zadanie uruchamia się dwa razy w miesiącu, pierwszego i piętnastego dnia, o 3:00. Wartości w liście rozdziela przecinek, bez spacji.

□□ ZMIENIA SYSTEM

Uruchomienie po starcie systemu: `@reboot`

Przykład:

```
@reboot /opt/vendor/client/bin/mount-helper -d bkp01.zielona.example -s backup_share
/mnt/backup
```

Zamiast pięciu pól, skrót `@reboot` uruchamia komendę raz, przy każdym starcie demona `cron` (praktycznie przy każdym uruchomieniu systemu). Realny przykład z produkcji: klient zasobu NFS trzeba było zamontować jako konkretny, zwykły użytkownik, a nie jako root — `/etc/fstab` montuje zawsze na prawach roota, a autofs w tamtej konfiguracji nie działał, więc `@reboot` w crontabie tego użytkownika okazał się najprostszym rozwiązaniem, które faktycznie zadziałało. Mało eleganckie, ale sprawdzone w produkcji.

□□ **NFS** = Network File System: udostępnianie katalogów po sieci w świecie Linuksa

□□ TYLKO ODCZYT

Skróty harmonogramu: `@daily`, `@weekly` i pozostałe

Przykład:

```
@daily   /usr/local/bin/backup.sh
@weekly  /usr/local/bin/czyszczenie-tmp.sh
```

> **Pojęcie: skróty harmonogramu.** Cron rozumie sześć gotowych nazw zamiast pięciu pól: `@reboot` (przy starcie), `@yearly`/`@annually` (`0 0 1 1 *`), `@monthly` (`0 0 1 * *`), `@weekly` (`0 0 * * 0`), `@daily`/`@midnight` (`0 0 * * *`) i `@hourly` (`0 * * * *`). Są czytelniejsze niż pięć gwiazdek i zer, ale nie dają kontroli nad dokładną godziną — `@daily` zawsze znaczy północ.

TYLKO ODCZYT

Crontab systemowy: `/etc/crontab`

```
cat /etc/crontab
```

Przykład:

```
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root


# przykład: uruchom o 6:25 każdego dnia jako user "root"
25 6 * * * root run-parts /etc/cron.daily
```

Plik systemowy różni się od crontaba użytkownika jednym polem: zaraz po piątej kolumnie harmonogramu następuje **nazwa użytkownika**, na którego prawach zadanie ma się wykonać. To jedyny crontab, który nie należy do konkretnej osoby, tylko do całego systemu, i edytuje się go zwykłym edytorem (nie przez `crontab -e`).

cat = concatenate: sklej i wypisz pliki

TYLKO ODCZYT

Dodatkowe crontaby systemowe: `/etc/cron.d/`

```
ls /etc/cron.d/
cat /etc/cron.d/sysstat
```

Przykład:

```
*/10 * * * * root /usr/lib64/sa/sa1 1 1
```

Katalog na crontaby instalowane przez pakiety RPM/DEB (np. `sysstat`, `logrotate` na niektórych dystrybucjach) — każdy plik ma ten sam sześciopolowy format co `/etc/crontab`, z użytkownikiem po godzinach. Pakiet może dopisać tu swój harmonogram przy instalacji i usunąć go przy deinstalacji, bez ingerencji w cudze crontaby.

RPM = RPM Package Manager: format paczek i narzędzie instalujące oprogramowanie w RHEL/Rocky, dnf korzysta z niego pod spodem · **ls** = list: wypisz zawartość katalogu · **cat** = concatenate: sklej i wypisz pliki

TYLKO ODCZYT

Katalogi skryptów cyklicznych: `/etc/cron.{hourly,daily,weekly,monthly}`

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

Przykład:

```
logrotate
man-db.cron
```

Zamiast pojedynczej linii harmonogramu, te cztery katalogi zawierają **skrypty wykonywalne**, które coś je uruchamia (patrz `run-parts` niżej) raz na godzinę, dzień, tydzień lub miesiąc. Wrzucenie własnego skryptu do `/etc/cron.daily/` (z prawem wykonywania!) wystarcza, żeby zaczął się uruchamiać codziennie — bez pisania jednej linijki harmonogramu.

ls = list: wypisz zawartość katalogu

TYLKO ODCZYT

Mechanizm za katalogami cron.*: `run-parts`

```
run-parts --test /etc/cron.daily
```

To właśnie `run-parts`, wywoływane z `/etc/crontab` (linia wyżej: `run-parts /etc/cron.daily`), przechodzi po katalogu i uruchamia po kolei każdy plik wykonywalny, którego nazwa nie zawiera kropki ani typowego rozszerzenia po menedżerze pakietów (`.rpmsave`, `.dpkg-old`). `--test` tylko wypisuje, co by uruchomił, bez wykonania — bezpieczny podgląd przed zmianą.

Przykład:

```
/etc/cron.daily/logrotate
/etc/cron.daily/man-db.cron
```

TYLKO ODCZYT

Kontrola dostępu do crontab: `/etc/cron.allow` i `/etc/cron.deny`

```
cat /etc/cron.allow
```

Jeśli plik `/etc/cron.allow` istnieje, tylko wymienieni w nim użytkownicy mogą używać `crontab` ; wszyscy pozostali dostają odmowę, nawet jeśli technicznie mają konto w systemie. Jeśli `cron.allow` nie istnieje, obowiązuje `/etc/cron.deny` — wymienieni tam użytkownicy są zablokowani, reszta ma dostęp. Brak obu plików zwykle oznacza „każdy może" (zależnie od dystrybucji i kompilacji demona).

cat = concatenate: sklej i wypisz pliki

TYLKO ODCZYT

Przegląd logów crona (RHEL/Rocky, SLES): `journalctl -u crond`

```
journalctl -u crond --since today
```

Na systemach z `crond` jako usługą systemd (RHEL/Rocky, SLES) to pierwsze miejsce, w którym szukamy potwierdzenia, że zadanie w ogóle wystartowało.

Przykład:

```
wrz 18 02:00:01 bkp01 crond[812]: (adm_user) CMD (/usr/local/bin/backup.sh > /var/log/backup.log 2>&1)
```

RHEL = Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa · **SLES** = SUSE Linux Enterprise Server: płatna dystrybucja Linuksa firmy SUSE, konkurent Rocky/RHEL · **journalctl** = journal control: czytanie dziennika systemd

TYLKO ODCZYT

Przegląd logów crona w stylu klasycznym (Debian, Ubuntu): `grep CRON /var/log/cron`

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

Debian i Ubuntu domyślnie nie mają osobnego pliku `/var/log/cron` — wpisy demona `cron` lądują w `/var/log/syslog` razem z resztą systemu, stąd filtr po słowie `CRON`. Na RHEL-owej rodzinie ten sam wpis trafia albo do dziennika systemd (`journalctl -u crond`), albo do `/var/log/cron`, jeśli rsyslog jest tak skonfigurowany.

Przykład:

```
Sep 18 02:00:01 bkp01 CRON[9821]: (adm_user) CMD (/usr/local/bin/backup.sh > /var/log/backup.log 2>&1)
```

grep = global regular expression print: szukaj wzorca w tekście

TYLKO ODCZYT

Obsługa maszyn, które nie chodzą non-stop: `anacron` i `/etc/anacrontab`

```
cat /etc/anacrontab
```

Przykład:

```
# okres w dniach   opóźnienie w minutach   identyfikator zadania   komenda
1                  5                       cron.daily              run-parts
/etc/cron.daily
7                  25                      cron.weekly             run-parts
/etc/cron.weekly
```

> **Pojęcie: anacron.** Uzupełnienie crona dla maszyn, które nie działają dwadzieścia cztery godziny na dobę — laptopy, stacje robocze, serwery gaszone na noc. Zwykły cron pomija uruchomienie, którego termin minął, gdy komputer był wyłączony: zadanie na 3:00 w nocy nigdy się nie wykona, jeśli maszyna wtedy spała. `anacron` liczy w **dniach**, nie minutach — przy starcie sprawdza, czy od ostatniego uruchomienia danego zadania minął zadeklarowany okres (kolumna pierwsza), i jeśli tak, uruchamia je po odczekaniu opóźnienia z kolumny drugiej. Na większości dystrybucji RPM-owych i DEB-owych `anacron` jest już wpięty jako wykonawca katalogów `cron.daily`/`weekly`/`monthly` zamiast samego crona — dlatego zadania dzienne na laptopie i tak w końcu się wykonują.

cat = concatenate: sklej i wypisz pliki

TYLKO ODCZYT

Wszystkie flagi `crontab`

Flaga	Co robi
`-l`	wypisuje bieżący crontab
`-e`	edytuje crontab (z walidacją składni przy zapisie)
`-r`	usuwa cały crontab bez pytania

Flaga	Co robi
`-u user`	działa na crontabie wskazanego użytkownika (wymaga uprawnień)
`plik`	wczytuje crontab z pliku, nadpisując bieżący
`-i`	jak `-r`, ale pyta o potwierdzenie przed usunięciem

Uważaj

- `crontab -r` nie ma odpowiednika `-i` domyślnie włączonego — literówka kasuje wszystko bez ostrzeżenia.
- Crontab systemowy (`/etc/crontab`, `/etc/cron.d/*`) ma **sześć** pól (z użytkownikiem), crontab osobisty (`crontab -e`) ma **pięć** — pomylenie formatów to najczęstszy powód, dla którego zadanie w ogóle się nie ładuje.
- Znak `%` w komendzie crontaba jest specjalny — patrz *Częste błędy*.

Z życia wzięte

Z ŻYCIA WZIĘTE

Piątek, 16:00: Tomek chce dopisać do crontaba na bkp01 zadanie zapisujące datę w nazwie pliku i wpisuje `date +%Y-%m-%d` bez escapowania procentów. Cron tnie linię na znaku procenta, resztę traktuje jako dane wejściowe komendy, a zaplanowany backup w ogóle nie startuje. Dopiero `journalctl -u crond` pokazuje, że komenda w crontabie wygląda zupełnie inaczej, niż Tomek myślał, że ją zapisał.

Po co to istnieje

Pojęcie: demon cron. Proces działający w tle od startu systemu do wyłączenia, bez własnego terminala i bez interakcji z użytkownikiem — dokładnie w tym sensie, w jakim strona o systemd definiuje demona. `cron` budzi się raz na minutę, czyta crontaby wszystkich użytkowników oraz pliki systemowe, porównuje bieżący czas z zapisanymi harmonogramami i uruchamia każde dopasowane zadanie jako osobny proces potomny. Nie czeka, aż poprzednie uruchomienie tego samego zadania się skończy — jeśli zadanie trwa dłużej niż odstęp między uruchomieniami, dostaniesz nakładające się kopie (rozwiązanie: `flock`, temat strony o diagnostyce).

Zanim istniał systemd z jednostkami typu timer, `cron` był jedynym standardowym sposobem uruchamiania czegokolwiek o określonej porze: kopie zapasowe w nocy, rotacja logów, wysyłka raportów, czyszczenie plików tymczasowych. Wciąż pozostaje najprostszym narzędziem do zadań bez zależności między usługami — nowocześniejszą alternatywę opisuje kolejna strona rozdziału.

Analogia

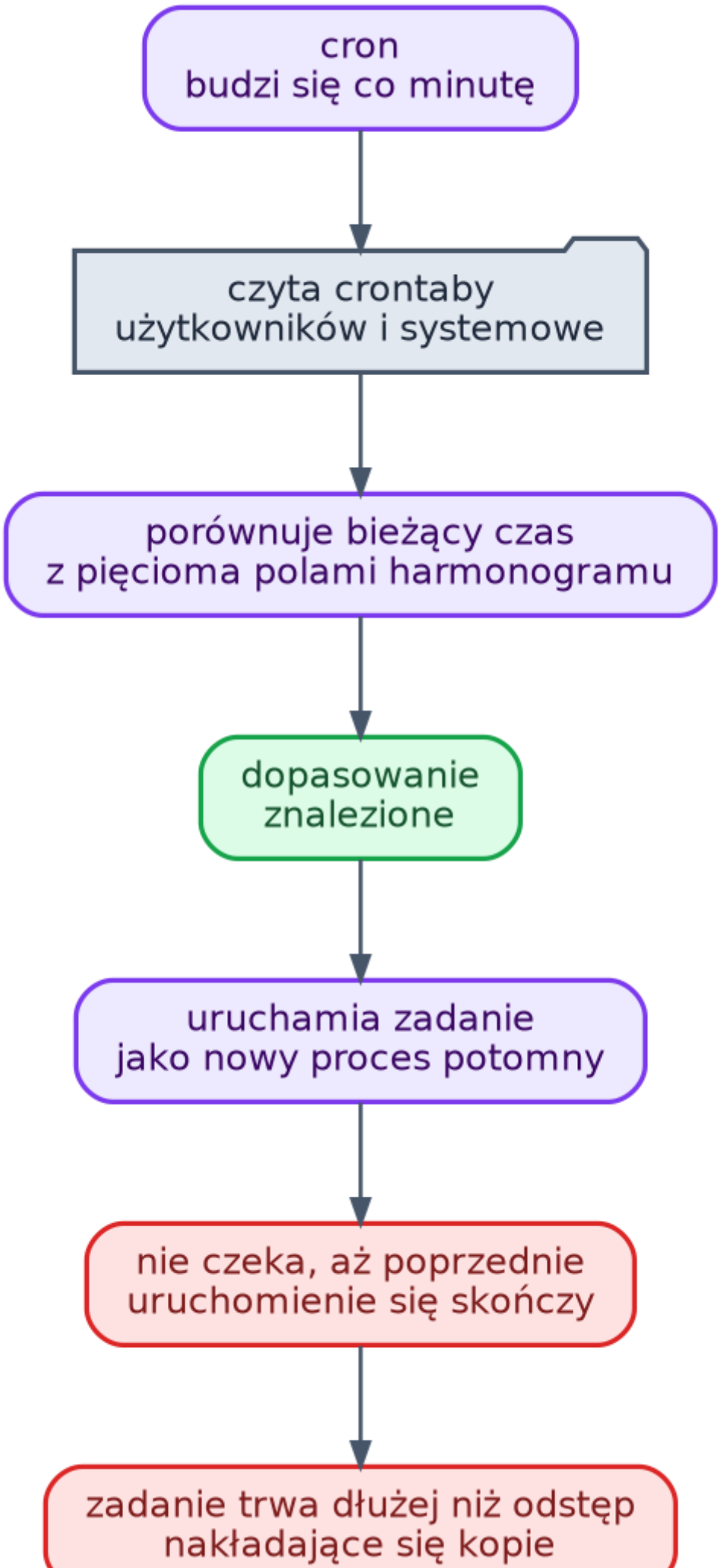
`cron` to **portier z listą dyżurów przypiętą do tablicy**. Co minutę podchodzi do tablicy, sprawdza godzinę i porównuje ją z listą wpisów: „o 2:00 wynieś śmieci", „w poniedziałek o 6:00 podlej kwiaty". Jeśli wpis pasuje, robi dokładnie to, co napisano — i nic więcej. Nie sprawdza, czy poprzednie zadanie się udało, nie czeka, aż ktoś skończy poprzednie, nie zna kontekstu poza tym, co dosłownie zapisano na kartce.

Stąd najczęstsza pułapka crona: portier nie zna Twoich prywatnych zwyczajów. Ty wiesz, gdzie stoi wiadro, bo masz w pamięci całą rutynę dnia. Portier zna tylko regulamin dyżuru — i dokładnie tak samo `cron` uruchamia zadanie w skąpym środowisku, bez Twojego `.bashrc`, bez aliasów, czasem z inną ścieżką do programów niż ta w Twoim terminalu.

Jak to działa

Rozwiń: jak to działa pod spodem

> **Pojęcie: definicja harmonogramu.** Każda linia crontaba (poza komentarzami od `#` i przypisaniami zmiennych typu `MAILTO=`) to definicja harmonogramu: pięć pól czasu z Fiszki, po których następuje komenda do wykonania. Cron parsuje te pięć pól osobno dla każdej linii i co minutę sprawdza, czy bieżący czas pasuje do wszystkich naraz (z jednym wyjątkiem — patrz niżej). Kolejność pól jest sztywna: minuta, godzina, dzień miesiąca, miesiąc, dzień tygodnia. Pole dnia tygodnia liczy niedzielę jako `0` **albo** `7` — obie wartości są równoważne, co bywa mylące przy zakresach obejmujących weekend.


cron
budzi się co minutę
czyta crontaby
użytkowników i systemowe
porównuje bieżący czas
z pięcioma polami harmonogramu
dopasowanie
znalezione
uruchamia zadanie
jako nowy proces potomny
nie czeka, aż poprzednie
uruchomienie się skończy
zadanie trwa dłużej niż odstęp
nakładające się kopie

Cron budzi się co minutę, porównuje czas z harmonogramem i uruchamia zadanie jako nowy proces potomny, nie czekając na poprzednie (kliknięcie otwiera pełny rozmiar)

Pułapka koniunkcji dnia miesiąca i dnia tygodnia. Gdy oba te pola są ustawione na coś innego niż gwiazdka, cron traktuje je jako **alternatywę**, nie iloczyn logiczny: `0 9 1 * 1 komenda` nie znaczy „pierwszego dnia miesiąca, jeśli to poniedziałek", tylko **pierwszego dnia każdego miesiąca ORAZ w każdy poniedziałek** — dwa niezależne warunki połączone „lub". Prawdziwe „i" wymaga warunku wewnątrz skryptu (np. sprawdzenie `date +%u` na starcie).

> **Pojęcie: crontab użytkownika kontra crontab systemowy.** Linux rozróżnia dwa rodzaje crontabów, podobnych z wyglądu, ale o innym formacie i miejscu przechowywania. Crontab **użytkownika** — edytowany przez `crontab -e`, trzymany w `/var/spool/cron/` (RHEL/SLES) albo `/var/spool/cron/crontabs/` (Debian/Ubuntu) — ma pięć pól czasu i uruchamia komendę na prawach właściciela pliku; nie ma pola z nazwą użytkownika, bo właściciel pliku *jest* tym użytkownikiem. Crontab **systemowy** — `/etc/crontab` i pliki w `/etc/cron.d/` — ma sześć pól: te same pięć plus kolumna z nazwą użytkownika, bo ten plik może uruchamiać zadania na cudzych prawach i musi jawnie powiedzieć, na czyich. Wrzucenie linii sześciopolowej do crontaba użytkownika (albo pięciopolowej do `/etc/crontab`) kończy się tym, że cron potraktuje fragment czasu jako początek komendy — zadanie się nie uruchomi albo uruchomi w złym momencie.

Ubogie środowisko procesu — przyczyna numer jeden „u mnie działa, z crona nie". Logowanie przez SSH czy otwarcie terminala wczytuje `.bashrc`/`.bash_profile`, definiuje aliasy, rozszerza `PATH` o `~/bin`, `/usr/local/bin` czy ścieżki narzędzi z `nvm`, `rbenv`, `pyenv`. `cron` nic z tego nie robi — uruchamia zadanie z ubogim, minimalnym środowiskiem: krótkim `PATH` (często tylko `/usr/bin:/bin`), bez zmiennych z plików startowych powłoki, i z `SHELL` domyślnie ustawionym na `/bin/sh`, nie na interaktywny bash. Skrypt, który w terminalu znajduje `docker` czy `node`, z crona dostaje `command not found`.

Trzy sposoby na to, żeby zadanie zachowywało się tak samo z crona jak z terminala:

1. **Ścieżki bezwzględne wszędzie** — `/usr/local/bin/node skrypt.js`, nie `node skrypt.js`.
2. `PATH=` **na górze crontaba** — jedna linia przed zadaniami: `PATH=/usr/local/bin:/usr/bin:/bin` ustawia PATH dla wszystkich zadań w tym crontabie.
3. `bash -lc` — owijasz komendę tak, żeby uruchomiła się jako login shell i wczytała pliki startowe: `bash -lc '/opt/app/deploy.sh'`.

Znak `%` jest specjalny. W crontabie procent nieoznaczony ukośnikiem oznacza znak nowej linii wewnątrz komendy — pułapka, na którą trafiają wszyscy przy formatowaniu daty:

```
0 2 * * * /usr/local/bin/backup.sh > /var/log/backup-%Y-%m-%d.log
```

Cron zinterpretuje to jako `.../backup-` z resztą (`m-%d.log`) przekazaną na standardowe wejście komendy. Poprawka to escapowanie każdego procentu ukośnikiem odwrotnym: `date +\%Y-\%m-\%d`.

Przekierowanie wyjścia, bo inaczej dostajesz maila. Domyślnie cron wysyła standardowy wynik i błąd zadania e-mailem do właściciela crontaba (albo pod adres z `MAILTO=`), przez lokalny MTA. Bez skonfigurowanej wysyłki poczty (typowa sytuacja) maile piętrzą się nieodczytane w `/var/spool/mail/`, aż zapchają dysk w `/var`. Zawsze przekierowuj wyjście jawnie: `> /var/log/zadanie.log 2>&1`.

Krok po kroku

1. Sprawdź, czy demon w ogóle działa (na Debianie/Ubuntu usługa nazywa się `cron`, nie `crond`).

```
systemctl status crond
```

```
● crond.service - Command Scheduler
     Loaded: loaded (/usr/lib/systemd/system/crond.service; enabled)
     Active: active (running) since Thu 2026-09-18 06:00:12 CEST
```

2. Dodaj zadanie testowe co minutę — `crontab -e`, dopisz linię, zapisz:

```
* * * * * date >> /tmp/test-cron.log
```

3. Poczekaj dwie minuty i sprawdź wynik.

```
sleep 120 && cat /tmp/test-cron.log
```

```
Fri Sep 18 08:41:00 CEST 2026
Fri Sep 18 08:42:00 CEST 2026
```

Dwa wpisy potwierdzają regularne uruchamianie co minutę.

4. Wyczyść po sobie.

```
crontab -r
rm /tmp/test-cron.log
```

Częste błędy

Skrypt działa ręcznie, ale z crona zwraca `command not found`.

```
/bin/sh: line 1: node: command not found
```

Cron uruchamia zadania z minimalnym `PATH`, bez katalogów dodanych przez menedżery wersji (`nvm`, `pyenv`). Naprawa: ścieżka bezwzględna do binarki albo linia `PATH=...` na górze crontaba.

Procent w dacie ucina komendę i wysyła maila z resztą jako danymi wejściowymi.

```
/bin/sh: -c: line 1: unexpected EOF while looking for matching `"'
```

Nieescapowany `%` w `date +%Y-%m-%d` wewnątrz crontaba jest interpretowany jako znak nowej linii. Naprawa: `date +\%Y-\%m-\%d` albo przeniesienie logiki formatowania daty do osobnego skryptu, gdzie procent nie jest specjalny.

Zadanie nakłada się samo na siebie.

Zadanie trwa 90 sekund, ale harmonogram uruchamia je co minutę — po godzinie działają już dziesiątki nałożonych na siebie kopii, obciążając dysk i CPU. Cron nie sprawdza, czy poprzednie uruchomienie się skończyło. Naprawa: `flock -n /var/lock/zadanie.lock /usr/local/bin/skrypt.sh` (pełne omówienie na stronie o diagnostyce).

Skrzynka pocztowa roota puchnie do granic dysku.

```
No space left on device
```

Bez przekierowania wyjścia każde uruchomienie wysyła maila z pełnym stdout/stderr do lokalnej skrzynki, a bez skonfigurowanego MTA maile tylko się kumulują w `/var/spool/mail/root`. Naprawa: `> /var/log/zadanie.log 2>&1` przy każdej linii albo `MAILTO=""` na górze pliku.

Pomylenie formatu crontaba użytkownika z systemowym.

```
crontab: syntax error, unexpected end of file
```

Wklejenie linii z `/etc/crontab` (sześć pól, z użytkownikiem) do crontaba edytowanego przez `crontab -e` (pięć pól) powoduje, że nazwa użytkownika jest odczytywana jako początek komendy, a `crontab -e`

po zapisie zgłasza błąd składni i odrzuca zmianę. Naprawa: usuń szóste pole (nazwę użytkownika) przy przenoszeniu wpisu do crontaba osobistego.

□□ Ćwiczenia

> Wszystko poniżej wykonujesz na własnej maszynie, na własnym koncie użytkownika. Nic nie wymaga uprawnień administratora poza jednym zaznaczonym miejscem.

Ćwiczenie 1 — środowisko crona kontra środowisko terminala

```
cd /tmp
crontab -l > /tmp/backup-crontab.txt 2>/dev/null
(crontab -l 2>/dev/null; echo "* * * * * env > /tmp/cron-env.txt") | crontab -
sleep 90
env > /tmp/shell-env.txt
diff /tmp/cron-env.txt /tmp/shell-env.txt
```

Które zmienne są w Twojej powłoce, a których brakuje w środowisku crona? Sprzątnij po sobie (`crontab -r` albo przywróć `/tmp/backup-crontab.txt` przez `crontab plik`).

Odpowiedź

`diff` pokaże, że `PATH` z crona jest znacznie krótszy niż w interaktywnej powłoce — zwykle tylko `/usr/bin:/bin` — i brakuje rozszerzeń dodanych przez `.bashrc`. `SHELL` w crontabie to `/bin/sh`, o ile nie ustawiono inaczej. To potwierdza, dlaczego skrypty odwołujące się do programów spoza standardowych katalogów zawodzą wyłącznie z crona.

Przywrócenie oryginalnego crontaba:

```
crontab /tmp/backup-crontab.txt
```

Ćwiczenie 2 — pułapka procenta w dacie

```
cd /tmp
(crontab -l 2>/dev/null; echo '* * * * * date +\%Y-\%m-\%d >> /tmp/data-ok.log') | crontab -
sleep 90
cat /tmp/data-ok.log
crontab -r
```

Zamień w powyższym poleceniu `\%` na goły `%` (bez escapowania) i powtórz eksperyment na osobnym pliku `/tmp/data-zle.log`. Co się różni w wyniku?

Odpowiedź

Z escapowaniem plik zapełnia się poprawnymi datami w formacie `2026-09-18`. Bez escapowania cron interpretuje `%` jako znak nowej linii wewnątrz komendy — realnie wykonywana komenda staje się `date +` z resztą tekstu przekazaną jako dane wejściowe do `date`, co zwykle kończy się błędem składni albo pustym/nieoczekiwanym wynikiem w logu.

Pytania kontrolne

1. Czym różni się format linii w `/etc/crontab` od formatu linii w crontabie edytowanym przez `crontab -e`?

Odpowiedź

`/etc/crontab` (i pliki w `/etc/cron.d/`) mają sześć pól: pięć pól czasu plus nazwę użytkownika, na którego prawach uruchomić komendę. Crontab osobisty ma pięć pól — bez użytkownika, bo właścicielem zadania jest zawsze właściciel pliku crontaba.

2. Zapisujesz `0 9 1 * 1 komenda`. Kiedy faktycznie się uruchomi?

Odpowiedź

Pierwszego dnia każdego miesiąca ORAZ w każdy poniedziałek — cron traktuje ustawione jednocześnie pole dnia miesiąca i dnia tygodnia jako alternatywę („lub"), nie jako iloczyn logiczny („i").

3. Zadanie w crontabie działa poprawnie po ręcznym uruchomieniu w terminalu, ale z crona zwraca `command not found`. Jakie są dwie najbardziej prawdopodobne przyczyny i jak je naprawić?

Odpowiedź

Najczęściej to skąpy `PATH` crona (naprawa: ścieżka bezwzględna do programu albo `PATH=` na górze crontaba) albo brak zmiennych z `.bashrc` / `.bash_profile`, których skrypt oczekuje (naprawa: `bash -lc 'komenda'`, żeby wymusić wczytanie plików startowych powłoki logowania).

4. Masz laptop, który bywa wyłączony w nocy. Zadanie zaplanowane w `cron.daily` na porę, gdy laptop zwykle śpi, i tak się wykonuje następnego dnia rano. Co to umożliwia?

Odpowiedź

`anacron`, który na desktopowych i laptopowych dystrybucjach zwykle jest wpięty jako wykonawca katalogów `/etc/cron.daily` / `weekly` / `monthly` zamiast samego crona. Sprawdza w dniach, nie minutach, czy dane zadanie wykonało się w wymaganym okresie, i jeśli nie — uruchamia je po starcie systemu, z zadanym opóźnieniem, zamiast bezpowrotnie pomijać przegapiony termin.

Powiązane: systemd timers: nowocześniejsza alternatywa dla crona · Zadania jednorazowe: at i batch · Dlaczego zadanie nie zadziałało: diagnostyka crona i timerów · systemd i systemctl: zarządzanie usługami

Źródła: `man 5 crontab` · `man 1 crontab` · `man 8 cron` · `man 8 run-parts` · `man 8 anacron` · `man 5 anacrontab` · crontab.guru

---

Revision #15
Created 2026-09-18 06:57:04 UTC by Tomasz Konieczny
Updated 2026-10-03 19:09:42 UTC by Tomasz Konieczny