Zadania jednorazowe: at i batch

◫◫ Poziom: podstawy

◫◫ Dotyczy: wszystkie dystrybucje (pakiet at), odpowiednik systemd zaznaczony

▯ Czas: ~14 min

`at` *uruchamia komendę* ***raz****, w podanym momencie w przyszłości, i zapomina o sobie zaraz po wykonaniu — dokładne przeciwieństwo crona, który powtarza zadanie w nieskończoność.* `batch` *robi to samo, ale czeka, aż system się odkorkuje.*

◫◫ Skróty użyte na tej stronie (4)

Skrót	Co znaczy
B2B	business to business: sprzedaż i systemy skierowane do innych firm, nie do klientów indywidualnych
CI	Continuous Integration: automatyczne budowanie i testowanie kodu po każdej zmianie
RHEL	Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa
SSH	Secure Shell: szyfrowane zdalne logowanie do serwera

Fiszka

◫◫ ZMIENIA SYSTEM

Instalacja pakietu

```
sudo dnf install at
```

Na RHEL/Rocky pakiet `at` bywa domyślnie nieobecny w minimalnej instalacji. Na Debianie/Ubuntu ta sama komenda to:

```
sudo apt install at
```

RHEL = Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa · **dnf** = Dandified YUM: menedżer pakietów RHEL/Rocky · **apt** = Advanced Package Tool: menedżer pakietów Debiana i Ubuntu

ZMIENIA SYSTEM

Uruchomienie i włączenie demona `atd`

```
sudo systemctl enable --now atd
```

`at` i `batch` to tylko nakładki na kolejkę, którą faktycznie obsługuje demon `atd` — bez niego zadania trafiają do kolejki, ale nikt ich nie uruchamia. Sprawdzenie stanu: `systemctl status atd`.

systemctl = system control: sterowanie usługami systemd

TYLKO ODCZYT

Zaplanowanie na konkretną godzinę dziś (albo jutro): `at 22:00`

```
echo "/usr/local/bin/backup.sh" | at 22:00
```

Jeśli 22:00 już minęło dzisiaj, `at` automatycznie zaplanuje zadanie na jutro — bez ostrzeżenia, po prostu wybiera najbliższy przyszły moment pasujący do podanej godziny.

Przykład:

```
job 7 at Fri Sep 18 22:00:00 2026
```

TYLKO ODCZYT

Zaplanowanie z czasem względnym: `at now + 2 hours`

```
echo "/usr/local/bin/wyslij-raport.sh" | at now + 2 hours
```

Składnia względna: `now + N minutes`, `now + N hours`, `now + N days`, `now + N weeks`. Czytelniejsze niż liczenie konkretnej godziny z głowy.

Przykład:

```
job 8 at Fri Sep 18 12:15:00 2026
```

TYLKO ODCZYT

Zaplanowanie na konkretną godzinę następnego dnia: `at 10:00 tomorrow`

```
echo "/usr/local/bin/przypomnienie.sh" | at 10:00 tomorrow
```

Jednoznaczne wskazanie „jutro", bez polegania na tym, czy podana godzina już minęła dzisiaj.

TYLKO ODCZYT

Zaplanowanie zadania z pliku: `at -f SciezkaDoSkryptu 03:00`

```
at -f /usr/local/bin/przenies-archiwum.sh 03:00
```

Zamiast pisać komendę przez `echo | at`, wskazujemy gotowy plik skryptu jako źródło zadania — czytelniejsze przy dłuższych poleceniach i łatwiejsze do wersjonowania.

Przykład:

```
job 9 at Sat Sep 19 03:00:00 2026
```

TYLKO ODCZYT

Zapis przez potok: `echo "komenda" | at midnight`

```
echo "logrotate -f /etc/logrotate.conf" | at midnight
```

`midnight` to gotowe słowo kluczowe (razem z `noon` i `teatime` — 16:00), rozpoznawane obok zwykłych godzin. Potok z `echo` to najczęstszy sposób podawania krótkiej, jednolinijkowej komendy bez otwierania interaktywnego promptu `at>`.

TYLKO ODCZYT

Wypisanie listy oczekujących zadań: `atq`

```
atq
```

Pokazujemy wszystkie zadania w kolejce, które jeszcze się nie wykonały — to odpowiednik `crontab -l`, tyle że dla zadań jednorazowych.

Przykład:

```
7	Fri Sep 18 22:00:00 2026 a adm_user
9	Sat Sep 19 03:00:00 2026 a adm_user
```

Kolumna z pojedynczą literą (`a`, `b`...) to kolejka — `a` to domyślna kolejka `at`, `b` to kolejka `batch`.

TYLKO ODCZYT

Usunięcie zadania z kolejki: `atrm`

```
atrm 7
```

Usuwamy zadanie o podanym numerze, zanim zdąży się wykonać. Numer bierzemy z kolumny pierwszej wyniku `atq`.

TYLKO ODCZYT

Podgląd treści zadania wraz ze środowiskiem: `at -c N`

```
at -c 9
```

Wypisujemy pełną treść zadania numer 9 — łącznie z całym środowiskiem (zmiennymi, katalogiem roboczym, umaskiem), które `at` zapisał w momencie tworzenia zadania. To bezpośredni dowód na kluczową różnicę względem crona opisaną w *Jak to działa*.

Przykład:

```
#!/bin/sh
# atrun uid=1000 gid=1000
# mail adm_user 0
umask 22
PATH=/home/adm_user/bin:/usr/local/bin:/usr/bin:/bin
export PATH
HOME=/home/adm_user; export HOME
cd /home/adm_user || exit 1
```

```
logrotate -f /etc/logrotate.conf
```

TYLKO ODCZYT

Uruchomienie zadania, gdy spadnie obciążenie: `batch`

```
echo "/usr/local/bin/przelicz-raporty.sh" | batch
```

Zamiast konkretnej godziny, `batch` czeka, aż średnie obciążenie systemu spadnie poniżej progu (domyślnie 1.5, konfigurowalne przy starcie `atd`), i dopiero wtedy uruchamia zadanie. Zadania trafiają do osobnej kolejki `b`, widocznej w `atq`.

Przykład:

```
job 10 at Fri Sep 18 10:32:00 2026
```

TYLKO ODCZYT

Kontrola dostępu do at: `/etc/at.allow` i `/etc/at.deny`

```
cat /etc/at.allow
```

Ten sam mechanizm co `cron.allow`/`cron.deny`: jeśli `at.allow` istnieje, tylko wymienieni w nim użytkownicy mogą planować zadania przez `at`/`batch`; w przeciwnym razie obowiązuje `at.deny` z listą zablokowanych. Brak obu plików zwykle oznacza pełny dostęp (zależnie od dystrybucji).

cat = concatenate: sklej i wypisz pliki

TYLKO ODCZYT

Zaplanowanie bez instalowania at: `systemd-run --on-active=30m`

```
systemd-run --on-active=30m --unit=jednorazowe-zadanie /usr/local/bin/skrypt.sh
```

Tworzymy tymczasową, jednorazową jednostkę timer i service w locie, bez pisania żadnych plików na dysk i bez instalowania pakietu `at` — systemd i tak już jest w systemie. `--on-active` liczy czas od teraz, dokładnie jak `at now + 30 minutes`.

Przykład:

```
Running timer as unit: jednorazowe-zadanie.timer
Will run service as unit: jednorazowe-zadanie.service
```

TYLKO ODCZYT

Zaplanowanie jednorazowego zadania na konkretną datę: `systemd-run --on-calendar=`

```
systemd-run --on-calendar="2026-12-31 23:59:00" --unit=nowy-rok
/usr/local/bin/powiadomienie.sh
```

Ten sam mechanizm co `--on-active`, ale z dokładną datą i godziną w formacie `OnCalendar` znanym ze strony o timerach systemd — wygodne, gdy liczy się konkretny moment w kalendarzu, a nie odstęp od teraz.

TYLKO ODCZYT

Wszystkie komendy rodziny `at`

Komenda	Co robi
`at CZAS`	planuje zadanie z komendami czytanymi ze standardowego wejścia
`at -f plik CZAS`	planuje zadanie z gotowego pliku skryptu
`atq`	lista oczekujących zadań
`atrm N`	usuwa zadanie o numerze N
`at -c N`	wypisuje pełną treść i środowisko zadania N
`batch`	planuje zadanie do uruchomienia, gdy spadnie obciążenie
`atd`	demon obsługujący kolejki `at` i `batch`

Uważaj

- Bez działającego `atd` zadania czekają w kolejce w nieskończoność — zawsze sprawdź `systemctl status atd` po instalacji.
- `atrm` nie pyta o potwierdzenie i nie da się cofnąć — literówka w numerze usunie inne zadanie niż zamierzone.

- Format czasu `at` jest tolerancyjny, ale niejednoznaczny zapis (sama godzina bez `tomorrow`/`today`) po północy potrafi zaskoczyć — zawsze weryfikuj wynikiem `job N at ...` zaraz po utworzeniu.

Z życia wzięte

◫◫ Z ŻYCIA WZIĘTE

Wtorek po Patch Tuesday, 14:00: Ola chce przeliczyć raporty sklepu B2B na app01 dopiero po godzinach szczytu i zleca `echo "/opt/app/przelicz.sh" | at now + 2 hours`. Zadanie wciąż wisi w `atq` długo po szesnastej, bo na tym serwerze nikt nigdy nie uruchomił demona `atd`. Dopiero `systemctl enable --now atd` i ponowne zlecenie tego samego polecenia sprawiają, że skrypt faktycznie startuje.

Po co to istnieje

Pojęcie: kolejka zadań jednorazowych. Struktura danych utrzymywana przez demon `atd` (zwykle w katalogu `/var/spool/at/`), w której każde zadanie utworzone przez `at` albo `batch` czeka na swoją kolej do wykonania, po czym jest z tej kolejki usuwane raz na zawsze. W przeciwieństwie do crontaba, który jest trwałym planem powtarzanym w kółko, kolejka `at` jest jednorazowa z natury: zadanie wykonuje się dokładnie raz i znika, chyba że samo zaplanuje swoje kolejne wywołanie.

Cron i systemd timers świetnie radzą sobie z powtarzalnością, ale nie każde zadanie ma być powtarzalne. „Wyślij mi przypomnienie za dwie godziny", „zrestartuj tę usługę o 3 w nocy, bo dostawca ogłosił okno serwisowe", „uruchom ten skrypt migracyjny raz, gdy będzie mniej ruchu" — to zadania z natury jednorazowe, dla których pisanie crontaba i pamiętanie o jego usunięciu po wykonaniu byłoby przesadą. `at` istnieje dokładnie po to.

Analogia

`at` to **kartka przyklejona na lodówce z dopiskiem „zrób to o 22:00, dziś"**.

Nikt nie planuje w ten sposób cotygodniowych zakupów — od tego jest stały grafik przypięty na tablicy (`cron`). Kartka na lodówce służy do jednej konkretnej rzeczy w jednym konkretnym

momencie: „wyjmij ciasto z piekarnika o 18:30". Po wykonaniu zrywasz kartkę — nikt nie zostawia jej tam, żeby przypominała o tym samym ciastku jutro.

`batch` to wariant tej samej kartki z dopiskiem „zrób to, jak już nie będzie tłoku w kuchni" — nie interesuje Cię dokładna godzina, tylko to, żeby zadanie nie kolidowało z czymś ważniejszym, co akurat się dzieje.

Jak to działa

Rozwiń: jak to działa pod spodem

> **Pojęcie: obciążenie jako warunek uruchomienia.** `batch` nie sprawdza zegara — sprawdza tzw. load average (średnie obciążenie systemu z ostatnich minut, tę samą liczbę, którą pokazuje `uptime`) i uruchamia zadanie dopiero, gdy spadnie poniżej progu skonfigurowanego dla `atd` (domyślnie 1.5, modyfikowalne parametrem `-l` przy starcie demona). To odwraca zwykłe pytanie „o której godzinie" na „czy teraz jest dobry moment" — przydatne dla zadań, które same są ciężkie (przeliczenia, indeksowanie, kompresja) i nie powinny startować w środku szczytu.

`at` **zapamiętuje całe środowisko z momentu utworzenia zadania — to kluczowa różnica względem crona.** Kiedy tworzysz zadanie przez `at`, program w tej właśnie chwili zrzuca do pliku zadania kopię Twojego bieżącego środowiska: `PATH`, katalog roboczy, `umask`, zmienne shellowe — dokładnie to, co widać w wyniku `at -c N` w Fiszce. Zadanie uruchomione później przez `atd` odtwarza to środowisko tak, jakbyś sam wpisał komendę w tamtym terminalu, o tamtej porze. Cron robi coś dokładnie odwrotnego — ignoruje środowisko, w którym go skonfigurowano, i zawsze startuje z własnym, ubogim zestawem zmiennych opisanym na stronie o cronie. Skutek praktyczny: skrypt z aliasami i rozszerzonym `PATH`, który przez `at` zadziała bez poprawek, w cronie może wymagać ścieżek bezwzględnych.

`systemd-run` **jako alternatywa bez instalowania niczego.** Na maszynie, gdzie `at` nie jest zainstalowany (a systemd i tak jest, bo to standardowy init), `systemd-run --on-active=` albo `--on-calendar=` tworzy tymczasową jednostkę timer w locie, bez zapisywania plików na dysk i bez demona `atd`. Traci przy tym jedną rzecz, którą ma `at`: `systemd-run` nie odtwarza automatycznie całego środowiska powłoki z momentu utworzenia w taki sam sposób — uruchomiona jednostka dziedziczy środowisko podobne do zwykłej usługi systemd, nie pełny zrzut Twojej sesji.

Krok po kroku

1. Sprawdź, czy `at` jest zainstalowane i czy demon działa.

```
rpm -q at || sudo dnf install at
systemctl enable --now atd
systemctl status atd
```

```
● atd.service - Job spooling tools
     Active: active (running)
```

2. Zaplanuj zadanie testowe za dwie minuty.

```
echo "date >> /tmp/test-at.log" | at now + 2 minutes
```

```
job 11 at Fri Sep 18 09:32:00 2026
```

3. Sprawdź kolejkę, zanim zadanie się wykona.

```
atq
```

```
11	Fri Sep 18 09:32:00 2026 a adm_user
```

4. Poczekaj i zweryfikuj wynik.

```
sleep 130 && cat /tmp/test-at.log
atq
```

```
Fri Sep 18 09:32:00 CEST 2026
```

Pusty wynik drugiego `atq` potwierdza, że zadanie wykonało się i zniknęło z kolejki — samo się nie powtórzy.

Częste błędy

Zadanie utworzone, ale nigdy się nie wykonuje.

```
job 12 at Fri Sep 18 22:00:00 2026
```

`atq` pokazuje zadanie w kolejce godzinami po wyznaczonej porze. Przyczyna niemal zawsze jest ta sama: demon `atd` nie działa. Sprawdzenie i naprawa:

```
systemctl enable --now atd
```

Literówka w numerze przy `atrm` kasuje złe zadanie.

Dwa zadania w kolejce, administrator chce usunąć nowsze, wpisuje numer starszego z pamięci zamiast sprawdzić `atq` — zadanie znika bez potwierdzenia i bez możliwości cofnięcia. Naprawa nawykowa: zawsze `atq` bezpośrednio przed `atrm`, nigdy z pamięci.

Skrypt zakłada środowisko, którego `at` nie miało w momencie tworzenia zadania.

```
sh: 1: terraform: not found
```

Zadanie utworzone z sesji SSH bez wczytanego `.bashrc` (np. z automatyzacji CI) ma inny, uboższy `PATH` niż interaktywna sesja administratora. Ponieważ `at` zamraża środowisko z chwili tworzenia zadania, a nie z chwili jego wykonania, problem nie ujawnia się od razu — dopiero przy zadaniu utworzonym z innego kontekstu. Naprawa: sprawdzić zamrożone środowisko przez `at -c N` przed zatwierdzeniem, użyć ścieżek bezwzględnych w skrypcie.

Zadanie `batch` czeka bez końca na maszynie stale obciążonej.

Serwer bazodanowy ma load average stale powyżej progu 1.5, więc zadanie zlecone przez `batch` nigdy nie dostaje okazji do uruchomienia. `batch` nie ma limitu czasu oczekiwania — czeka, aż warunek się spełni, nawet jeśli to nigdy nie nastąpi. Naprawa: dla zadań z twardym terminem używać `at` z konkretną godziną, zostawiając `batch` wyłącznie zadaniom bez presji czasu.

Pomylenie kolejki `at` (`a`) z kolejką `batch` (`b`) przy czytaniu `atq`.

```
9	Fri Sep 18 03:00:00 2026 a adm_user
10	Fri Sep 18 09:32:00 2026 b adm_user
```

Druga litera w wierszu `atq` to nazwa kolejki, nie przypadkowy znak — zadanie numer 10 zostało zlecone przez `batch` i uruchomi się dopiero po spadku obciążenia, niezależnie od widocznej tam godziny utworzenia (ta kolumna to czas zlecenia, nie planowanego wykonania, dla kolejki `b`).

□□ Ćwiczenia

> Wszystko poniżej wykonujesz na własnej maszynie, w katalogu `/tmp`. Instalacja pakietu `at` wymaga uprawnień administratora; reszta ćwiczeń nie.

Ćwiczenie 1 — środowisko zamrożone przez `at`

```
cd /tmp
export MOJA_ZMIENNA="test-at"
echo "env > /tmp/at-env.txt" | at now + 1 minutes
sleep 70
grep MOJA_ZMIENNA /tmp/at-env.txt
```

Czy zmienna `MOJA_ZMIENNA` pojawia się w pliku wynikowym? Porównaj z Ćwiczeniem 1 ze strony o cronie, w którym testowałeś to samo dla crontaba.

Odpowiedź

Tak, `MOJA_ZMIENNA` jest widoczna w `/tmp/at-env.txt` — `at` zrzuca całe bieżące środowisko powłoki w momencie tworzenia zadania, łącznie ze zmiennymi wyeksportowanymi ręcznie w tej sesji. To odwrotność crona, który startuje zadania z własnym, minimalnym środowiskiem niezależnym od tego, co ustawiłeś w terminalu.

Ćwiczenie 2 — `at` kontra `systemd-run` dla tego samego zadania

```
cd /tmp
echo "date +%s >> /tmp/porownanie.log" | at now + 1 minutes
systemd-run --on-active=1min --unit=test-systemd-run bash -c 'date +%s >> /tmp/porownanie.log'
sleep 70
cat /tmp/porownanie.log
atq
systemctl list-timers | grep test-systemd-run
```

Który z dwóch mechanizmów zostawia po sobie ślad w `systemctl list-timers`, a który w `atq`? Posprzątaj: `systemctl stop test-systemd-run.timer 2>/dev/null`.

Odpowiedź

Zadanie utworzone przez `at` jest widoczne w `atq` przed wykonaniem i znika z niej po uruchomieniu; nigdy nie pojawia się w `systemctl list-timers`. Zadanie utworzone przez `systemd-run --on-active` tworzy tymczasową jednostkę `.timer`/`.service` widoczną w `systemctl list-timers` do czasu wykonania (i pozostającą jako nieaktywna jednostka jeszcze chwilę po). Oba zapiszą wynik w tym samym pliku, bo obie komendy w końcu robią to samo — różni je tylko mechanizm i miejsce, w którym widać stan oczekującego zadania.

Pytania kontrolne

1. Czym `at` różni się fundamentalnie od `cron` pod względem powtarzalności?

Odpowiedź

`at` planuje zadanie do jednorazowego wykonania — po uruchomieniu znika z kolejki i nie powtarza się. `cron` (i systemd timer) planuje zadanie do wykonywania cyklicznego, zgodnie z harmonogramem, w nieskończoność, aż ktoś je usunie.

2. Dlaczego zadanie utworzone przez `at` może się zachowywać inaczej niż to samo polecenie w crontabie, mimo identycznej treści komendy?

Odpowiedź

Bo `at` zapamiętuje całe środowisko z momentu tworzenia zadania (PATH, zmienne, katalog roboczy), a cron zawsze uruchamia zadania z własnym, ubogim środowiskiem niezależnym od tego, co było ustawione w terminalu administratora. To samo polecenie może więc znaleźć potrzebne programy przez `at`, a nie znaleźć ich przez cron.

3. Kiedy wybierzesz `batch` zamiast `at` z konkretną godziną?

Odpowiedź

Gdy zadanie samo jest obciążające dla systemu (przeliczenia, indeksowanie, kompresja) i nie ma twardego terminu wykonania — `batch` odczeka, aż load average spadnie poniżej progu, zamiast uruchamiać zadanie o sztywnej porze, potencjalnie kolidując z innym obciążeniem systemu.

4. Chcesz uruchomić jednorazowe zadanie za 45 minut na serwerze, na którym pakiet `at` nie jest zainstalowany, a instalacja czegokolwiek wymaga zgłoszenia zmiany. Co zrobisz?

Odpowiedź

`systemd-run --on-active=45min --unit=nazwa komenda` — systemd jest już częścią systemu, więc to podejście nie wymaga instalowania żadnego dodatkowego pakietu ani zgłoszenia zmiany.

Powiązane: cron i crontab: zadania cykliczne · systemd timers: nowocześniejsza alternatywa dla crona · Dlaczego zadanie nie zadziałało: diagnostyka crona i timerów

Źródła: `man 1 at` · `man 1 atq` · `man 1 atrm` · `man 1 batch` · `man 8 atd` · `man 1 systemd-run`

Revision #15
Created 2026-09-18 06:57:07 UTC by Tomasz Konieczny
Updated 2026-10-03 19:09:45 UTC by Tomasz Konieczny