logrotate: rotacja logów

Poziom: średnio

Dotyczy: wszystkie dystrybucje (logrotate, uruchamiany przez cron albo timer systemd)

Czas: ~18 min

`logrotate` *przenosi rosnący plik logu na bok, zaczyna nowy od zera i po pewnym czasie kasuje najstarsze kopie — bez tego mechanizmu każdy gadatliwy demon prędzej czy później zapełni cały dysk jednym, wiecznie rosnącym plikiem.*

Fiszka

TYLKO ODCZYT

Wypisanie konfiguracji globalnej: `cat /etc/logrotate.conf`

```
cat /etc/logrotate.conf
```

Ustawienia domyślne, dziedziczone przez wszystkie konfiguracje, które ich jawnie nie nadpiszą, plus dyrektywa `include`, która wciąga zawartość katalogu z konfiguracjami per-aplikacja.

Przykład:

```
weekly
rotate 4
create
dateext
compress

include /etc/logrotate.d
```

cat = concatenate: sklej i wypisz pliki

TYLKO ODCZYT

Przegląd plików w `/etc/logrotate.d/`

```
ls /etc/logrotate.d/
cat /etc/logrotate.d/nginx
```

Każdy pakiet, który generuje własne logi (nginx, httpd, mysql), instaluje tu osobny plik z regułami dopasowanymi do swoich logów — zamiast dopisywać się do wspólnego `logrotate.conf` i ryzykować konflikt przy aktualizacji pakietu.

Przykład:

```
/var/log/nginx/*.log {
    weekly
    rotate 12
    compress
    delaycompress
    missingok
    notifempty
    create 0640 nginx adm
    sharedscripts
    postrotate
        systemctl reload nginx > /dev/null 2>&1 || true
    endscript
}
```

ls = list: wypisz zawartość katalogu · **cat** = concatenate: sklej i wypisz pliki

ZMIENIA SYSTEM

Kompletny przykład, dyrektywa po dyrektywie

```
sudo tee /etc/logrotate.d/mojaapp <<'EOF'
/var/log/mojaapp/*.log {
    daily
    rotate 14
    size 100M
    compress
    delaycompress
    missingok
    notifempty
```

```
    create 0640 mojaapp adm
    dateext
    olddir /var/log/mojaapp/archiwum
    sharedscripts
    postrotate
        systemctl reload mojaapp > /dev/null 2>&1 || true
    endscript
}
EOF
```

`daily` — sprawdzamy i rotujemy raz dziennie (alternatywy: `weekly`, `monthly`). `rotate 14` — trzymamy 14 starych kopii, starsze kasujemy. `size 100M` — rotujemy też wcześniej niż wyznacza `daily`, jeśli plik przekroczy 100 MB (przydatne przy nagłym zalewie logów). `compress` — kompresujemy rotowane pliki gzipem. `delaycompress` — nie kompresujemy **najświeższej** rotowanej kopii, dopiero następnej — patrz „Koniecznie" niżej, dlaczego to ważne. `missingok` — nie zgłaszamy błędu, jeśli plik logu akurat nie istnieje. `notifempty` — pomijamy rotację, jeśli plik jest pusty (nie ma sensu tworzyć pustych archiwów). `create 0640 mojaapp adm` — po rotacji od razu tworzymy nowy, pusty plik logu z podanymi uprawnieniami i właścicielem. `dateext` — nazywamy rotowane pliki z datą w nazwie (`app.log-20260918`) zamiast numerem (`app.log.1`). `olddir` — przenosimy rotowane pliki do wskazanego katalogu zamiast zostawiać je obok oryginału. `sharedscripts` — przy dopasowaniu przez `*` do wielu plików uruchamiamy `postrotate` **raz** dla całej grupy, nie osobno dla każdego pliku. `postrotate`/`endscript` — blok komend uruchamiany po rotacji, tu: przeładowanie usługi, żeby zaczęła pisać do nowego pliku.

□□ **tee** = zapisz strumień wejścia do pliku i jednocześnie wypisz go na ekran ·**systemctl** = system control: sterowanie usługami systemd

□□ **TYLKO ODCZYT**

Rotacja po rozmiarze: `size` i `maxsize`

Przykład:

```
size 100M
```

Rotuje, gdy plik przekroczy podany rozmiar, **niezależnie** od harmonogramu `daily`/`weekly` — traktowane jako zamiennik częstotliwości czasowej, nie dodatek do niej. Odmienna dyrektywa `maxsize` **łączy się** z `daily`/`weekly`: rotacja następuje przy najbliższym uruchomieniu logrotate zgodnym z harmonogramem **albo** wcześniej, jeśli plik przekroczy podany rozmiar — który warunek nastąpi pierwszy.

Przykład:

```
daily
maxsize 500M
```

🔒 TYLKO ODCZYT

Wymuszenie użytkownika i grupy operacji: `su`

Przykład:

```
su mojaapp adm
```

Domyślnie `logrotate` (uruchamiany zwykle jako root przez cron albo timer) wykonuje operacje na plikach na prawach roota. Dyrektywa `su` każe mu przełączyć się na wskazanego użytkownika i grupę przy rotacji — istotne, gdy plik logu leży na zasobie sieciowym, gdzie root nie ma praw zapisu, a zwykły użytkownik aplikacji ma.

🔒 TYLKO ODCZYT

Uruchomienie suchego przebiegu: `logrotate -d`

```
sudo logrotate -d /etc/logrotate.d/mojaapp
```

Symulujemy cały proces i wypisujemy, co by zrobił — łącznie z tym, czy warunek rozmiaru albo czasu w ogóle jest spełniony — **bez** faktycznej rotacji, kompresji czy uruchomienia `postrotate`. Zawsze uruchamiamy to przed pierwszym wdrożeniem nowego pliku konfiguracji i po każdej zmianie.

Przykład:

```
reading config file /etc/logrotate.d/mojaapp
Reading state from file: /var/lib/logrotate/logrotate.status
Allocating hash table for state file, size 15360 B

Handling 1 logs

rotating pattern: /var/log/mojaapp/*.log  100M bytes (14 rotations)
empty log files are not rotated, old logs are removed
considering log /var/log/mojaapp/app.log
  log needs rotating
```

```
rotating log /var/log/mojaapp/app.log, log->rotateCount is 14
dateext suffix '-20260918'
glob pattern '-[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'
running postrotate script
```

TYLKO ODCZYT

Wymuszenie rotacji: `logrotate -f`

```
sudo logrotate -f /etc/logrotate.d/mojaapp
```

Rotuje **natychmiast**, nawet jeśli żaden warunek (czas, rozmiar) nie jest jeszcze spełniony. Używane do testowania całego łańcucha `postrotate` na żywym pliku albo po ręcznej zmianie konfiguracji, kiedy nie chcemy czekać do jutra.

TYLKO ODCZYT

Włączenie trybu gadatliwego: `logrotate -v`

```
sudo logrotate -v /etc/logrotate.d/mojaapp
```

Faktycznie wykonuje rotację (w przeciwieństwie do `-d`), ale opisuje krok po kroku, co robi — przydatne razem z `-f`, żeby zobaczyć realny przebieg zamiast zgadywać z pustego kodu wyjścia.

TYLKO ODCZYT

Pamięć logrotate: `/var/lib/logrotate/logrotate.status`

```
cat /var/lib/logrotate/logrotate.status
```

Plik, w którym `logrotate` zapisuje datę ostatniej rotacji dla każdego wzorca — stąd wie, że „daily" już dziś wykonano, i nie rotuje po raz drugi przy kolejnym uruchomieniu tego samego dnia.

Przykład:

```
logrotate state -- version 2
"/var/log/mojaapp/*.log" 2026-9-18-3:0:0
```

```
"/var/log/nginx/*.log" 2026-9-15-3:0:0
```

cat = concatenate: sklej i wypisz pliki

TYLKO ODCZYT

Sprawdzenie, kto uruchamia logrotate: `systemctl status logrotate.timer`

```
systemctl status logrotate.timer
```

Na większości nowoczesnych dystrybucji RPM-owych `logrotate` jest uruchamiany przez timer systemd, nie przez wpis w `/etc/cron.daily/`. Warto to sprawdzić, zanim zaczniemy szukać wpisu w crontabie, którego tam po prostu nie ma.

Przykład:

```
● logrotate.timer - Daily rotation of log files
     Active: active (waiting)
    Trigger: Fri 2026-09-19 00:00:00 CEST; 14h left
```

systemctl = system control: sterowanie usługami systemd

TYLKO ODCZYT

Przegląd logów z ostatnich uruchomień: `journalctl -u logrotate`

```
journalctl -u logrotate --since -7d
```

Kiedy `logrotate` działa przez jednostkę systemd (`logrotate.service` uruchamiany przez `logrotate.timer`), jego wyjście trafia do journala tak samo jak każdej innej usługi — bez tego trzeba by szukać śladów w `/var/log/cron` albo liczyć na kod wyjścia z crontaba.

Przykład:

```
wrz 18 00:00:01 app01 systemd[1]: Started Rotate log files.
wrz 18 00:00:02 app01 logrotate[9821]: reloading nginx.service after rotation
wrz 18 00:00:02 app01 systemd[1]: logrotate.service: Deactivated successfully.
```

journalctl = journal control: czytanie dziennika systemd

◫ TYLKO ODCZYT

Wszystkie dyrektywy — ściąga

Dyrektywa	Co robi
`daily` / `weekly` / `monthly`	częstotliwość sprawdzania i rotacji
`rotate N`	ile starych kopii zachować
`size` / `maxsize`	rotacja po przekroczeniu rozmiaru
`compress`	kompresja rotowanych plików (gzip)
`delaycompress`	opóźnia kompresję o jeden cykl rotacji
`missingok`	brak pliku logu nie jest błędem
`notifempty`	nie rotuj pustego pliku
`create MODE USER GROUP`	utwórz nowy pusty plik po rotacji
`copytruncate`	skopiuj i wyzeruj plik w miejscu, bez `postrotate`
`su USER GROUP`	wykonuj operacje na prawach wskazanego użytkownika
`sharedscripts`	jeden `postrotate` dla całej grupy dopasowanych plików
`postrotate` / `endscript`	blok komend po rotacji
`dateext`	data w nazwie rotowanego pliku zamiast numeru
`olddir KATALOG`	przenoś rotowane pliki do innego katalogu

Uważaj

- `logrotate -d` **nie zmienia** stanu w `logrotate.status` — możesz uruchamiać suchy przebieg dowolną liczbę razy bez skutków ubocznych.
- Konfiguracja bez `create` ani `copytruncate` zostawia aplikację piszącą do starego, przeniesionego pliku, aż aplikacja sama go nie zamknie i otworzy nowy — patrz sekcja niżej.
- `rotate 0` usuwa stare logi bez trzymania żadnej kopii — łatwo się pomylić z „nie rotuj wcale".

Z życia wzięte

Z ŻYCIA WZIĘTE

Środa, 9:00: na app01 katalog `/var/log/mojaapp/` pokazuje `app.log` o zerze bajtów, a aplikacja Oli na pewno cały czas coś loguje, bo widać ruch w metrykach. Plik konfiguracji logrotate ma `create`, ale ktoś przy kopiowaniu z innej aplikacji zapomniał dopisać blok `postrotate` z przeładowaniem usługi. Proces dalej pisze do przeniesionego, niewidocznego już pod starą nazwą pliku, dopóki Ola ręcznie nie zrestartuje aplikacji.

Po co to istnieje

> **Pojęcie: rotacja logu.** Proces przenoszenia bieżącego pliku logu w bok (pod nową nazwą, często z numerem albo datą w nazwie), zaczynania nowego, pustego pliku logu pod oryginalną nazwą i — po przekroczeniu limitu trzymanych kopii — trwałego usuwania najstarszych z nich. Bez rotacji plik logu rośnie bez końca: aplikacja gadatliwa przy błędach albo logująca każde żądanie potrafi w ciągu tygodni zapełnić cały wolumin, a `du` na jednym rosnącym pliku nie daje żadnego naturalnego sygnału ostrzegawczego, dopóki dysk się nie zapełni.

`logrotate` sam z siebie nie jest demonem ani usługą działającą w tle — to zwykły program uruchamiany cyklicznie przez `cron` (klasycznie: skrypt w `/etc/cron.daily/logrotate`) albo przez `logrotate.timer` na nowszych dystrybucjach. Przy każdym uruchomieniu czyta wszystkie pliki konfiguracji z `/etc/logrotate.conf` i `/etc/logrotate.d/`, sprawdza dla każdego wzorca, czy warunek rotacji (czas albo rozmiar) jest spełniony, i jeśli tak — wykonuje cały łańcuch operacji opisany dyrektywami.

Analogia

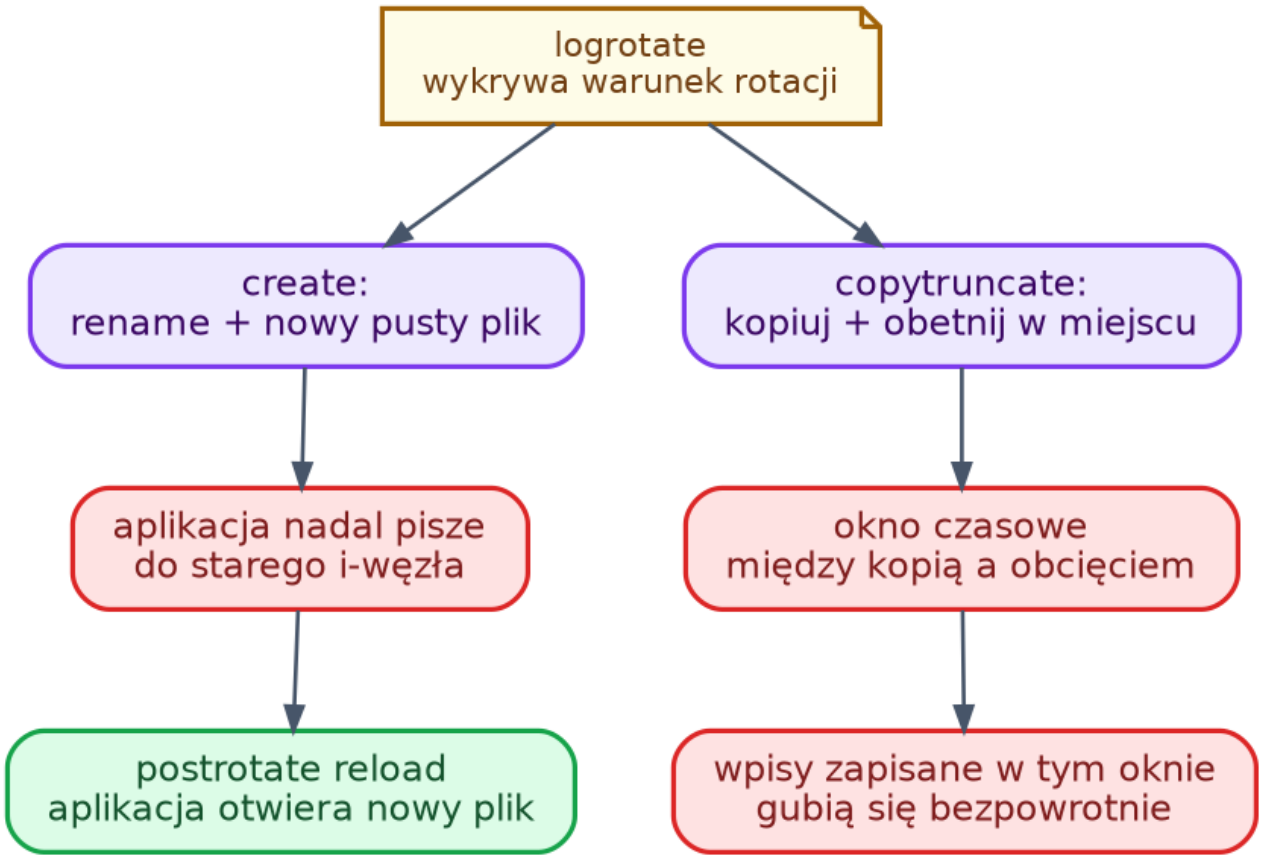
`logrotate` to **archiwista w kancelarii papierowej**, który co tydzień przychodzi, zabiera pełny segregator z biurka, wkłada na jego miejsce nowy, pusty, a najstarszy segregator z szafy archiwalnej wyrzuca, żeby zrobić miejsce.

Sekretarka (aplikacja logująca) nigdy nie przestaje pisać — po prostu w pewnym momencie zaczyna pisać do nowego segregatora zamiast do starego. Kluczowe pytanie brzmi: skąd sekretarka wie, że ma sięgnąć po nowy segregator, a nie dalej pisać do tego, który archiwista już zabrał ze stołu? To jest dokładnie różnica między `create` i `copytruncate`, opisana niżej — i jest to najczęstsze źródło problemów z logrotate w praktyce.

Jak to działa

Rozwiń: jak to działa pod spodem

> **Pojęcie: przerwanie deskryptora przy rotacji.** Kiedy `logrotate` przenosi (`rename`) plik logu pod nową nazwę, aplikacja, która ten plik ma otwarty do zapisu, **nie dowiaduje się o tym automatycznie**. Proces trzyma otwarty deskryptor pliku wskazujący na konkretny i-węzeł na dysku, nie na nazwę pliku — a zmiana nazwy nie unieważnia deskryptora. Aplikacja więc dalej pisze do tego samego i-węzła, tyle że pod nową, przeniesioną nazwą, podczas gdy nazwa oryginalna nie istnieje albo wskazuje na zupełnie nowy, pusty plik. Ten mechanizm tłumaczy, dlaczego samo `mv` pliku logu nigdy nie wystarcza — trzeba jeszcze skłonić aplikację, żeby otworzyła plik ponownie pod starą nazwą.



Create przenosi plik i tworzy nowy, wymagając przeładowania aplikacji; copytruncate obcina plik w miejscu, ryzykując utratę wpisów zapisanych w oknie między kopią a obcięciem (kliknięcie otwiera pełny rozmiar)

`create` **kontra** `copytruncate` **— kluczowa różnica.** `create` mówi `logrotate`, żeby po przeniesieniu starego pliku od razu utworzył nowy, pusty plik pod oryginalną nazwą, z podanymi uprawnieniami. To rozwiązuje problem nazwy, ale nie problem deskryptora — aplikacja wciąż pisze do starego, przeniesionego pliku, dopóki sama go nie zamknie i nie otworzy na nowo pod oryginalną nazwą. Dlatego `create` niemal zawsze towarzyszy blokowi `postrotate`/`endscript` z komendą przeładowującą usługę (`systemctl reload nginx`, `kill -HUP $(cat /var/run/app.pid)`) — to sygnał dla aplikacji „zamknij stary plik, otwórz nowy". Bez tego przeładowania logi nadal lądują w przeniesionym pliku, mimo że pod oryginalną nazwą widnieje pusty plik zero bajtów — klasyczna pułapka, w której administrator patrzy na pusty `app.log` i myśli, że aplikacja przestała logować.

`copytruncate` rozwiązuje to inaczej: zamiast przenosić plik, **kopiuje** jego bieżącą zawartość do pliku archiwalnego, a potem **obcina oryginalny plik do zera bajtów w miejscu**, nie zmieniając jego nazwy ani i-węzła. Aplikacja nie musi niczego wiedzieć — pisze dalej do tego samego deskryptora, który teraz po prostu jest pusty. Wygląda na wygodniejsze, ale ma poważną wadę: **między momentem skopiowania a momentem obcięcia pliku aplikacja może zdążyć dopisać kolejne linie** — trafiają w lukę: nie ma ich w już skopiowanym archiwum, a operacja obcięcia je kasuje. Im więcej aplikacja pisze w tym krótkim oknie, tym więcej wpisów ginie. Dlatego `copytruncate` to **ostateczność, zarezerwowana dla aplikacji, których nie da się przeładować ani zrestartować** bez przerwania działania.

> **Pojęcie: kompresja opóźniona.** `compress` samo w sobie kompresuje rotowany plik natychmiast przy rotacji. Problem: jeśli aplikacja (przez brak przeładowania albo z opóźnieniem) nadal dopisuje coś do świeżo rotowanego pliku przez chwilę po rotacji, kompresja w locie może zablokować albo skomplikować ten zapis. Kompresja opóźniona, włączana dyrektywą `delaycompress`, odsuwa spakowanie o jeden pełny cykl — najświeższa rotowana kopia zostaje nieskompresowana do następnej rotacji, dając aplikacji margines czasu na dokończenie zapisów do niej, zanim zostanie spakowana.

`logrotate -d` **przed każdą zmianą.** Błąd składni w pliku `/etc/logrotate.d/` (brakujący nawias klamrowy, zła ścieżka we wzorcu) potrafi po cichu wyłączyć rotację dla całej reszty pliku, bez czytelnego komunikatu w typowym uruchomieniu przez cron/timer. `logrotate -d /etc/logrotate.d/plik` pokazuje dokładnie, co program zrozumiał ze wzorca i co planuje zrobić, zanim to się faktycznie stanie — jedyny pewny sposób na sprawdzenie nowej konfiguracji bez ryzyka utraty logów przy błędnym `postrotate`.

Pułapka `tail -f` **po rotacji.** Kto obserwuje log na żywo przez `tail -f` w momencie rotacji, zobaczy, że strumień nagle „zamiera" — `tail -f` śledzi deskryptor otwartego pliku, dokładnie tak samo jak aplikacja piszącą do niego, więc po `mv` wykonanym przez `logrotate` obserwuje

dalej stary, przeniesiony plik, do którego (po przeładowaniu aplikacji) nic już nie przybywa. Pełne omówienie tej pułapki i sposobu obejścia (`tail -F`, `less +F`) — strona *Obserwacja plików na żywo: tail -f, less +F, watch*.

Krok po kroku

1. Przygotuj testową konfigurację i plik logu.

```
mkdir -p /tmp/logtest
echo "linia startowa" > /tmp/logtest/app.log
sudo tee /tmp/logtest/logrotate-test.conf <<'EOF'
/tmp/logtest/app.log {
    rotate 3
    compress
    missingok
    notifempty
    copytruncate
}
EOF
```

2. Suchy przebieg — sprawdź, zanim cokolwiek ruszysz.

```
sudo logrotate -d /tmp/logtest/logrotate-test.conf
```

```
considering log /tmp/logtest/app.log
  log does not need rotating (log has already been rotated)
```

Bez pliku stanu logrotate może uznać, że rotacja nie jest jeszcze potrzebna — wymuszamy ją w kroku 3.

3. Wymuś rotację i obejrzyj wynik.

```
sudo logrotate -vf /tmp/logtest/logrotate-test.conf
ls -la /tmp/logtest/
```

```
app.log
app.log.1
```

`copytruncate` nie kompresuje przy pierwszej rotacji w tym przykładzie tak od razu widocznie jak `create` — plik `app.log` istnieje nadal (obcięty), a `app.log.1` to kopia sprzed obcięcia.

4. Sprawdź plik stanu.

```
grep logtest /var/lib/logrotate/logrotate.status
```

```
"/tmp/logtest/app.log" 2026-9-18-9:45:12
```

Częste błędy

Aplikacja pisze do „ducha” — przeniesionego pliku, którego już nie widać po nazwie.

```
ls -la /var/log/mojaapp/app.log
```

```
-rw-r----- 1 mojaapp adm 0 wrz 18 03:00 app.log
```

Plik `app.log` istnieje i ma zero bajtów, ale aplikacja produkuje logi bez przerwy — trafiają do usuniętego (albo przeniesionego) pliku, do którego proces wciąż trzyma otwarty deskryptor. Klasyczny skutek `create` bez działającego `postrotate`. Naprawa: dodać (albo naprawić) blok `postrotate` z komendą przeładowującą usługę; sprawdzenie na gorąco: `lsof | grep deleted` pokaże procesy trzymające otwarte, usunięte pliki.

`copytruncate` gubi wpisy zapisane w oknie między kopiowaniem a obcięciem.

Po rotacji brakuje kilku linii z logu tuż przed momentem rotacji, mimo że aplikacja na pewno je zapisała (widać je np. w metrykach albo powiadomieniach). To nie błąd aplikacji — to natura `copytruncate`: wpisy zapisane dokładnie w oknie między skopiowaniem starej zawartości a obcięciem pliku do zera nigdy nie trafiają do archiwum ani nie zostają w oryginale. Naprawa: przejście na `create` z odpowiednim `postrotate`, jeśli aplikacja w ogóle wspiera przeładowanie; `copytruncate` zostaje tylko tam, gdzie przeładowanie jest fizycznie niemożliwe.

Błąd składni w jednym pliku `.d/` wywala rotację innych aplikacji.

```
error: /etc/logrotate.d/mojaapp:5: unexpected } (missing previous '{'?)
```

Zależnie od wersji `logrotate`, błąd składni w jednym pliku konfiguracyjnym może przerwać przetwarzanie całego przebiegu, zanim dojdzie do kolejnych plików w `/etc/logrotate.d/` — logi innych, poprawnie skonfigurowanych aplikacji też przestają się rotować, mimo że ich konfiguracja jest bez zarzutu. Naprawa: `logrotate -d` na każdym zmienianym pliku osobno, zanim trafi do produkcji.

`missingok` pominięty, a plik czasem nie istnieje.

```
error: stat of /var/log/mojaapp/debug.log failed: No such file or directory
```

Aplikacja tworzy `debug.log` tylko w trybie diagnostycznym, więc zwykle go nie ma. Bez `missingok` logrotate zgłasza to jako błąd przy każdym uruchomieniu — nieszkodliwy, ale zaśmiecający logi i powiadomienia o błędach cron/systemd. Naprawa: dopisać `missingok` do bloku.

Rotacja po rozmiarze nie działa, bo pomylono `size` z `maxsize`.

Administrator chce „rotuj codziennie, ale też wcześniej, jeśli plik urośnie do 500 MB", wpisuje `size 500M` bez `daily`, po czym dziwi się, że rotacja **nigdy** nie następuje o stałej porze — tylko po przekroczeniu rozmiaru, bo `size` samodzielnie **zastępuje** harmonogram czasowy, nie dodaje się do niego. Naprawa: `daily` razem z `maxsize 500M`, nie z `size`.

□□ Ćwiczenia

> Wszystko poniżej wykonujesz w katalogu `/tmp`, na własnych plikach testowych. `logrotate -d` nigdzie nic nie zmienia, więc możesz go uruchamiać bez obaw.

Ćwiczenie 1 — `create` bez `postrotate`, czyli pisanie do ducha

```
mkdir -p /tmp/ducholog
cd /tmp/ducholog
echo "start" > app.log
exec 3>> app.log   # symulacja procesu trzymającego plik otwarty do zapisu
tee create-test.conf <<'EOF'
/tmp/ducholog/app.log {
    rotate 2
    create 0644 $(whoami) $(whoami)
    missingok
}
EOF
logrotate -f create-test.conf
echo "wpis po rotacji" >&3
cat app.log
```

```
cat app.log.1
exec 3>&-
```

Do którego pliku trafił „wpis po rotacji" i dlaczego, skoro `app.log` istnieje jako nowy, pusty plik?

Odpowiedź

„wpis po rotacji" trafia do `app.log.1`, nie do nowego `app.log` — deskryptor `3` wciąż wskazuje na stary i-węzeł, przeniesiony pod nazwę `app.log.1`, mimo że nazwa `app.log` teraz odnosi się do zupełnie nowego pliku utworzonego przez `create`. To dokładnie mechanizm opisany w „Jak to działa": rotacja zmienia nazwy, nie deskryptory, więc bez przeładowania procesu (tu: zamknięcia i ponownego otwarcia deskryptora) proces dalej pisze do starego pliku pod nową nazwą.

Ćwiczenie 2 — `copytruncate` i utracone wpisy

```
mkdir -p /tmp/copytest
cd /tmp/copytest
seq 1 1000 > app.log
tee copytruncate-test.conf <<'EOF'
/tmp/copytest/app.log {
    rotate 2
    copytruncate
}
EOF
(for i in $(seq 1001 1010); do echo "$i" >> app.log; sleep 0.05; done) &
logrotate -f copytruncate-test.conf
wait
wc -l app.log app.log.1
```

Zsumuj linie z obu plików. Czy suma zawsze wynosi dokładnie 1010? Co by się stało, gdyby proces dopisujący linie pracował znacznie szybciej niż `logrotate` wykonuje kopiowanie i obcinanie?

Odpowiedź

Przy tak małym i szybkim przykładzie suma zwykle wychodzi bliska 1010, ale nie ma na to gwarancji — w zależności od tego, w którym dokładnie momencie `logrotate` zdąży skopiować plik i go obciąć względem równoległego dopisywania, część linii z okna „między kopiowaniem a obcięciem" może się zdublować (trafić i do kopii, i przetrwać obcięcie, jeśli zapis nastąpił tuż przed truncate) albo, częściej w realnych warunkach z większym obciążeniem, zniknąć całkowicie. Im szybszy strumień zapisu względem czasu wykonania `copytruncate`, tym większe ryzyko i większa liczba utraconych wpisów — to jest dokładnie ryzyko opisane w sekcji „Jak to działa", tu odtworzone w mikroskali.

Pytania kontrolne

1. Dlaczego samo `mv stary.log nowy.log` nigdy nie wystarcza, żeby aplikacja zaczęła pisać do nowego pliku?

Odpowiedź

Bo proces trzyma otwarty deskryptor wskazujący na i-węzeł pliku, nie na jego nazwę. Zmiana nazwy nie unieważnia deskryptora — proces dalej pisze do tego samego i-węzła pod nową nazwą, dopóki sam nie zamknie pliku i nie otworzy go ponownie pod oryginalną nazwą (co robi np. w reakcji na sygnał HUP albo przy pełnym restarcie).

2. Kiedy `copytruncate` jest uzasadnionym wyborem, mimo ryzyka utraty wpisów?

Odpowiedź

Gdy aplikacji nie da się przeładować ani zrestartować bez przerwania jej działania i nie ma ona żadnego mechanizmu reopen pliku logu na sygnał — typowo starsze albo zamknięte oprogramowanie. To ostateczność, nie domyślny wybór.

3. Co konkretnie robi `delaycompress` i jaki problem rozwiązuje?

Odpowiedź

Opóźnia kompresję rotowanego pliku o jeden pełny cykl — najświeższa rotowana kopia zostaje nieskompresowana aż do następnej rotacji. Rozwiązuje sytuację, w której aplikacja jeszcze przez chwilę po rotacji dopisuje coś do świeżo przeniesionego pliku (np. z powodu opóźnionego przeładowania) — kompresja w locie mogłaby to zakłócić.

4. Uruchamiasz `tail -f` na pliku logu i w pewnym momencie strumień na ekranie zamiera, mimo że aplikacja na pewno cały czas pisze logi. Co się prawdopodobnie stało i gdzie szukać pełnego wyjaśnienia?

Odpowiedź

Najpewniej nastąpiła rotacja logu — `tail -f` śledzi deskryptor otwartego w danym momencie pliku, więc po przeniesieniu pliku przez `logrotate` obserwuje dalej stary, przeniesiony plik, do którego (po przeładowaniu aplikacji) nic już nie przybywa. Pełne wyjaśnienie i sposób obejścia: strona „Obserwacja plików na żywo: tail -f, less +F, watch".

Powiązane: cron i crontab: zadania cykliczne · systemd timers: nowocześniejsza alternatywa dla crona · Obserwacja plików na żywo: tail -f, less +F, watch · Dlaczego zadanie nie zadziałało: diagnostyka crona i timerów

Źródła: `man 8 logrotate` · `man 5 logrotate.conf` · `/usr/share/doc/logrotate*/`

Revision #15
Created 2026-09-18 06:57:09 UTC by Tomasz Konieczny
Updated 2026-10-03 19:09:47 UTC by Tomasz Konieczny