Obserwacja plików na żywo: tail -f, less +F, watch

Poziom: podstawy/srednio
Dotyczy: wszystkie dystrybucje (sekcja journalctl — systemd) **Czas: ~12 min**

Jak patrzeć na log w trakcie awarii, zamiast otwierać go po fakcie. `tail -f`, `less +F`, `journalctl -f`, `watch` *— plus powody, dla których ekran potrafi zamrzeć, chociaż system pracuje.*

Skróty użyte na tej stronie (5)

Skrót	Co znaczy
eBPF	extended Berkeley Packet Filter: mechanizm jądra Linux pozwalający programowi obserwować zdarzenia systemowe (pliki, procesy, sieć) bez ładowania własnego modułu jądra
PID	Process ID: numer procesu w systemie
RHEL	Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa
SSH	Secure Shell: szyfrowane zdalne logowanie do serwera
WMS	Warehouse Management System: system zarządzania magazynem

Fiszka

TYLKO ODCZYT

Obserwacja rosnącego pliku odporna na rotację: tail -F

```
tail -n 200 -F /var/log/nginx/error.log
```

Śledzimy nazwę pliku, a nie deskryptor, więc przeżywamy rotację logu wykonywaną przez logrotate (mechanizm opisany na stronie o `head` i `tail`). `-n 200` daje zapas kontekstu sprzed uruchomienia — bez tego dostajemy tylko domyślne 10 linii, co przy szybko rosnącym logu jest za mało, żeby się zorientować w sytuacji.

tail = pokaż koniec pliku, z flagą -f śledzi plik na żywo

TYLKO ODCZYT

Obserwacja z możliwością cofnięcia się: less +F

```
less +F /var/log/nginx/error.log
```

Robi to samo co `tail -F`, ale w środku pełnoprawnego pagera: `Ctrl+C` przerywa podążanie i oddaje normalny `less` z wyszukiwaniem i przewijaniem, `F` wraca do trybu na żywo. Domyślnie śledzi deskryptor, tak jak `tail -f` — żeby śledzić nazwę i przetrwać rotację, dodajemy `--follow-name`:

```
less +F --follow-name /var/log/messages
```

less = przeglądarka tekstu z przewijaniem (nazwa to żart: less is more)

TYLKO ODCZYT

Cykliczne uruchamianie komendy: watch

```
watch -n 1 'ss -tunlp | grep NumerPortu'
```

Zamiast śledzić rosnący plik, `watch` uruchamia podaną komendę co N sekund (domyślnie co 2) i czyści ekran przed każdym nowym wynikiem. Przydaje się do rzeczy, które nie są plikami: stanu portów, zajętości dysku, liczby procesów. Cudzysłów wokół całej komendy jest obowiązkowy — patrz Częste błędy.

watch = uruchamiaj wskazane polecenie co kilka sekund i pokazuj wynik na żywo

TYLKO ODCZYT

Podświetlenie zmian: watch -d

```
watch -d free -m
```

Porównuje każdy nowy wynik z poprzednim i podświetla różnice. Pomaga, gdy szukamy, która z kilkunastu liczb w tabeli w ogóle się zmienia — bez tego trzeba by porównywać dwa zrzuty ekranu ręcznie.

Przykład:

```
Every 2,0s: free -m                                srv01: Thu Sep 17 10:44:03
2026
```

```
               total        used        free      shared  buff/cache   available
Mem:           15934        4021        6218         512        5695       11201
Swap:           2047           0        2047
```

Nagłówek `Every 2,0s: komenda` z nazwą hosta i aktualnym czasem pojawia się przy każdym uruchomieniu `watch`, niezależnie od tego, co obserwujemy. W realnym terminalu liczby, które zmieniły się względem poprzedniego cyklu, są dodatkowo wyróżnione (domyślnie trybem odwróconym) — czego w zwykłym tekście nie widać.

watch = uruchamiaj wskazane polecenie co kilka sekund i pokazuj wynik na żywo

TYLKO ODCZYT

Obserwacja dziennika systemd: journalctl -f

```
journalctl -f
```

Odpowiednik `tail -f` dla systemów, w których logi nie leżą w zwykłych plikach tekstowych, tylko w binarnym dzienniku `journald`. Wypisuje nowe wpisy w miarę ich powstawania, ze wszystkich jednostek naraz.

journalctl = journal control: czytanie dziennika systemd

TYLKO ODCZYT

Obserwacja jednej usługi: journalctl -u -f

```
journalctl -u NazwaUslugi -f
```

Filtruje dziennik do wpisów jednej jednostki systemd i dodatkowo podąża za nowymi. Flagi `-u` i `-f` można łączyć w dowolnej kolejności — `-fu NazwaUslugi` działa tak samo.

Przykład:

```
-- Logs begin at Mon 2026-09-14 06:02:11 CEST. --
wrz 17 10:41:07 srv01 nginx[1842]: 2026/09/17 10:41:07 [error] 1842#0: *91 upstream
timed out
wrz 17 10:41:09 srv01 nginx[1842]: 2026/09/17 10:41:09 [notice] 1842#0: worker restarted
wrz 17 10:44:00 srv01 systemd[1]: Reloading nginx - high performance web server...
wrz 17 10:44:00 srv01 systemd[1]: Reloaded nginx - high performance web server.
```

journalctl = journal control: czytanie dziennika systemd

TYLKO ODCZYT

Filtrowanie na żywo: --line-buffered

```
tail -F /var/log/messages | grep --line-buffered -iE 'error|fail|denied'
```

Wymusza na `grep` wypychanie każdej dopasowanej linii natychmiast, zamiast czekać na uzbieranie pełnego bloku (4 kB) do buforowania. Bez tej flagi przy rzadko występujących błędach można patrzeć na pusty ekran godzinami, mimo że dane cały czas płyną.

tail = pokaż koniec pliku, z flagą -f śledzi plik na żywo

TYLKO ODCZYT

Wymuszenie buforowania liniowego dla dowolnego programu: stdbuf -oL

```
tail -F plik.log | stdbuf -oL awk '{print $1, $2, $3, $NF}'
```

`--line-buffered` działa tylko w `grep`. Dla programów bez własnej flagi buforowania (`awk`, `cut`, własne skrypty korzystające ze standardowej biblioteki C) `stdbuf -oL` wymusza buforowanie liniowe z zewnątrz, bez modyfikowania programu.

tail = pokaż koniec pliku, z flagą -f śledzi plik na żywo

TYLKO ODCZYT

Kolorowanie bez filtrowania

```
tail -F /var/log/messages | grep --color=always -E 'error|timeout|$'
```

Trik: alternatywa `$` w wyrażeniu regularnym dopasowuje koniec każdej linii, czyli w praktyce każdą linię — więc `grep` przepuszcza wszystko, a jedynie podświetla `error` i `timeout` tam, gdzie występują. Efekt: widzimy cały kontekst, nie tylko trafienia, a szukane słowa rzucają się w oczy. `--color=always` jest konieczne, bo w potoku domyślne `auto` wyłącza kolory.

tail = pokaż koniec pliku, z flagą -f śledzi plik na żywo

◫◫ TYLKO ODCZYT

Wszystkie narzędzia

Narzędzie	Podąża za	Cofnięcie się w historię	Uwaga
`tail -f`	deskryptor pliku	nie	zamiera po rotacji logu
`tail -F`	nazwa pliku	nie	domyślny wybór dla logów systemowych
`less +F`	deskryptor	**tak** (`Ctrl+C`)	z `--follow-name` zachowuje się jak `-F`
`journalctl -f`	dziennik systemd	nie	jedyna opcja na logach bez pliku
`watch`	wynik komendy	nie	nie widzi tego, co zdarzyło się między odświeżeniami
`multitail` / `lnav`	wiele plików	tak	wymagają doinstalowania

Uważaj

- `less -F` i `less +F` to dwie różne rzeczy: `-F` kończy pager, jeśli treść mieści się na jednym ekranie, `+F` włącza podążanie.
- Potok bez `--line-buffered` albo `stdbuf -oL` wygląda identycznie jak brak zdarzeń — pusty ekran przez godziny.
- `watch -n 1 komenda | grep wzorzec` filtruje wyjście `watch`, nie komendy. Potok musi być w cudzysłowie.

Z życia wzięte

◫◫ Z ŻYCIA WZIĘTE

Wtorek, 6:05, chwilę po starcie zmiany w Swarzędzu. Marek dzwoni, że WMS na `ora01` muli, a Tomek zamiast szukać pliku logu, odpala `journalctl -u wms.service -f` i widzi błędy na żywo, zamiast czekać, aż coś się przewinie po ekranie. Po chwili widzi powtarzający się błąd połączenia z bazą i przekazuje go dalej razem z dokładnym czasem, bo dziennik systemd ma go zapisany co do sekundy.

Po co to istnieje

Restartujesz usługę i chcesz wiedzieć, czy wstała. Klient zgłasza, że „czasem nie działa", a Ty nie wiesz kiedy — więc musisz siedzieć i patrzeć. Puszczasz import na czterdzieści minut i chcesz widzieć postęp, zamiast co chwilę wpisywać `tail -n 20`.

To ta sama potrzeba: **zobaczyć zdarzenie w momencie, w którym się dzieje**. Rekonstrukcja po godzinie jest zawsze gorsza — nie masz kontekstu innych usług, a część stanu (otwarte połączenia, zajęte porty, katalog tymczasowy) już nie istnieje.

Narzędzia z tej strony dzielą się na dwie rodziny. Jedna śledzi **rosnący plik** (`tail -f`, `less +F`, `multitail`). Druga cyklicznie uruchamia **komendę i pokazuje jej wynik** (`watch`). To nie jest to samo i mylenie ich kosztuje.

Mechanizm `tail -f` i różnicę `-f` vs `-F` opisuje strona head i tail — początek i koniec pliku. Tutaj zakładamy, że to już wiesz, i idziemy dalej.

Analogia

Pomyśl o **dyspozytorni z monitoringiem**. Na ścianie wisi ściana monitorów, każdy z innej kamery, na żywo. Siedzisz i patrzysz. To jest `tail -f`: strumień, który leci, i albo go widzisz, albo przegapiłeś.

Problem pojawia się, kiedy coś przemknie przez kadr i chcesz cofnąć się o dwadzieścia sekund. Na podglądzie na żywo się nie da — musisz przełączyć stanowisko w tryb odtwarzania nagrania, przewinąć, obejrzeć, a potem wrócić do obrazu bieżącego. To robi `less +F`: `Ctrl+C` przełącza w tryb archiwum, `F` wraca do obrazu na żywo. `tail -f` jest monitorem bez nagrywarki — jedyne, co możesz z nim zrobić, to go wyłączyć.

Kamera jest przy tym wycelowana w **konkretne miejsce**, nie w „to, co ciekawe". Jeśli ktoś przestawi biurko do innego pokoju, kamera dalej pokazuje pusty kąt. To jest różnica `-f` (patrzę w ten plik, który mi kiedyś otworzono) i `-F` (patrzę w miejsce o tej nazwie — jeśli je podmienią, przekieruję się na nowe).

A `watch` to nie kamera, tylko **wartownik obchodzący budynek co dwie minuty**. Dostajesz regularny raport stanu, ale jeśli ktoś przebiegnie przez korytarz między obchodami, w raporcie tego nie będzie — `watch -n 2 'ss -tn state established'` nie pokaże połączenia, które trwało pół sekundy, choćby ich były setki.

Jak to działa

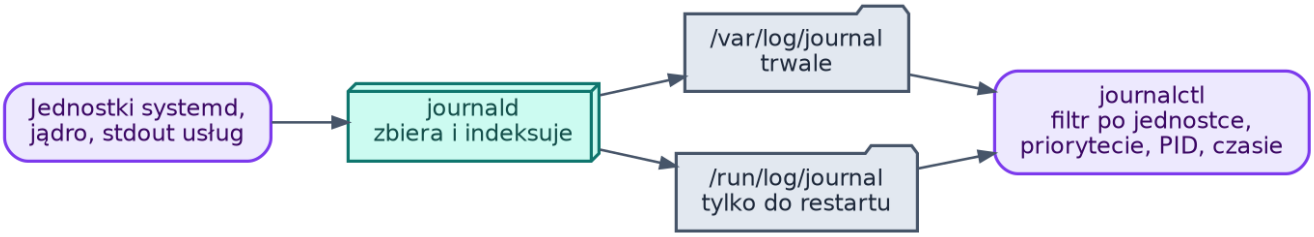
🔍 Rozwiń: jak to działa pod spodem

`tail -f` i `less +F` robią technicznie to samo: trzymają otwarty plik, zapamiętują pozycję odczytu i czekają na przyrost rozmiaru (`tail` przez `inotify`, `less` odpytując co pół sekundy). Różnica leży nie w mechanizmie śledzenia, tylko w tym, **co potrafi program wokół niego**.

`tail` to filtr: bierze dane i wypycha na standardowe wyjście. Kiedy linia przewinie się przez terminal, jest poza zasięgiem — trzyma ją tylko bufor przewijania emulatora, po którym żadna komenda nie wyszuka.

`less` to pager z własnym buforem i własną nawigacją. W trybie `F` wypisuje nowe linie tak samo, ale całość ma pod ręką. `Ctrl+C` przerywa pętlę podążania i oddaje Ci normalny pager — `/wzorzec`, `?wzorzec`, `g`, `G`, `n` działają. Wciśnięcie `F` uruchamia pętlę ponownie, od miejsca, do którego plik urósł; nic nie tracisz, bo dane są na dysku, a nie w terminalu. Widać to dopiero w trakcie incydentu: przez ekran przelatuje stacktrace, Ty widzisz tylko jego koniec. W `tail -f` masz przerąbane, w `less +F` naciskasz `Ctrl+C`, przewijasz w górę, czytasz i wracasz klawiszem `F`.

`journalctl -f` nie śledzi pliku, tylko dziennik systemd.



Źródła trafiające do journald i dwa miejsca przechowywania, z których czyta journalctl (kliknięcie otwiera pełny rozmiar)

> **Pojęcie: dziennik systemd (journald).** `journald` to usługa systemd zbierająca komunikaty ze wszystkich jednostek, z jądra i ze standardowych wyjść usług, i zapisująca je w jednym binarnym formacie — w `/var/log/journal/` (trwale) albo w `/run/log/journal/` (tylko do restartu). W przeciwieństwie do zwykłego pliku tekstowego dziennik ma indeksy: po jednostce, priorytecie, PID-zie i czasie. Dlatego `journalctl -u nginx.service -p err` filtruje błyskawicznie, korzystając z indeksu, zamiast czytać linia po linii cały plik tekstowy, tak jak musiałby to zrobić `grep`.

`watch` nie ma nic wspólnego z plikami: uruchamia komendę przez `sh -c`, czyści ekran, wypisuje wynik, śpi N sekund (domyślnie 2) i powtarza. Z `-d` podświetla różnice wobec poprzedniego cyklu, a z `--differences=permanent` podświetlenie się kumuluje — pomaga, kiedy szukasz, która z trzydziestu liczb w tabeli w ogóle drga. Domyślnie `watch` **wycina sekwencje kolorów**; jeśli ich chcesz, dodaj `-c`.

Krok po kroku

1. Podgląd z zapasem kontekstu, na jednym albo kilku plikach.

```
tail -n 200 -F /var/log/nginx/error.log
tail -F /var/log/nginx/access.log /var/log/nginx/error.log
```

Przy kilku plikach `tail` wstawia nagłówki:

```
==> /var/log/nginx/error.log <==
2026/09/17 10:41:07 [error] 1842#0: *91 upstream timed out (110: Connection timed out)

==> /var/log/nginx/access.log <==
10.20.20.15 - - [17/Sep/2026:10:41:07 +0200] "GET /api/orders HTTP/1.1" 504 167
```

Nagłówek pojawia się przy każdej zmianie źródła. Przy dwóch spokojnych logach to czytelne; przy pięciu ruchliwych nagłówków będzie więcej niż treści i to moment, w którym sięgasz po `multitail` albo `lnav`.

2. Filtrowanie i kolorowanie na żywo, złożone razem.

```
tail -F /var/log/messages | grep --line-buffered -iE 'error|fail|denied'
```

`--line-buffered` wymusza wypchnięcie każdej linii natychmiast. Dla programu bez takiej flagi jest `stdbuf -oL`, narzucające buforowanie liniowe każdemu programowi korzystającemu ze standardowej biblioteki C: `tail -F plik | stdbuf -oL cut -d' ' -f1-3,6-`.

Filtrowanie odcina kontekst — widzisz błąd, ale nie to, co działo się wokół. Trik z alternatywą `$` (pokazany w Fiszce jako „Kolorowanie bez filtrowania") pozwala pokazać wszystko, podświetlając tylko istotne słowa.

3. Automatyczne zakończenie po padnięciu procesu.

```
./migracja.sh > /tmp/migracja.log 2>&1 &
tail -f --pid=$! /tmp/migracja.log
```

`$!` to PID ostatnio uruchomionego zadania w tle. `tail` skończy się sam, kiedy skrypt się zakończy — wstawiony do skryptu, nie zawiesi go na zawsze.

4. `watch` do rzeczy, które nie są plikami.

```
watch -d 'df -h /var /srv'
watch -d -n 1 'cat /proc/mdstat'
```

Działa to tak samo, jak przykład `watch -d free -m` z Fiszki wyżej — inna komenda, ten sam mechanizm porównania z poprzednim cyklem i ten sam nagłówek `Every N,Ns: ...`.

5. Dwa widoki obok siebie w tmux.

```
tmux new -s incydent
# Ctrl+B "  podział poziomy · Ctrl+B %  podział pionowy · Ctrl+B o  przeskocz
```

W jednym panelu `journalctl -fu aplikacja`, w drugim `watch -d 'ss -s; free -m'`. Sesja przeżyje rozłączenie SSH — wracasz przez `tmux attach -t incydent`. Odpowiednik na starszych systemach: `screen -S incydent` (`Ctrl+A S`, `Ctrl+A Tab`, powrót `screen -r`).

6. Narzędzia dodatkowe: `multitail` i `lnav`.

Żadne nie jest instalowane domyślnie.

```
sudo dnf install epel-release && sudo dnf install multitail lnav   # RHEL/Rocky/Alma
sudo apt install multitail lnav                                    # Debian/Ubuntu
sudo zypper install multitail lnav                                 # openSUSE/SLES
```

`multitail` dzieli terminal na panele, z niezależnym przewijaniem w każdym:

```
multitail /var/log/nginx/access.log /var/log/nginx/error.log
multitail -l 'journalctl -fu nginx' -l 'journalctl -fu postgresql'
```

`lnav` (`lnav /var/log/nginx/*.log`) rozpoznaje popularne formaty logów, parsuje znaczniki czasu i **scala kilka plików w jedną oś czasu**. To ta funkcja, dla której warto go zainstalować: układa zdarzenia z kilku usług w kolejności, w jakiej naprawdę zaszły, zamiast przeplatać je nagłówkami `==>`.

Częste błędy

Potok bez `--line-buffered` — pusty ekran, który wygląda jak spokój.

```
tail -F /var/log/app.log | grep -i error
```

Uruchamiasz to w trakcie incydentu. Ekran pusty, więc wnioskujesz, że błędy przestały lecieć, i idziesz szukać gdzie indziej. Tymczasem `grep` ma w buforze czternaście linii i czeka, aż uzbiera 4 kB — przy jednym błędzie na minutę poczekasz kwadrans. Reguła: **każdy potok za `tail -f` wymaga wymuszenia buforowania liniowego** (`--line-buffered`, a dla reszty `stdbuf -oL`).

`watch -n 1 ss -tunlp | grep 8080` — komenda niby działa, ale nic nie widać.

Powłoka dzieli to na `watch -n 1 ss -tunlp` oraz `grep 8080`. `watch` widzi, że jego wyjście nie jest terminalem, a `grep` filtruje strumień pełen sekwencji sterujących. Efekt: pustka albo śmieci. Poprawnie: `watch -n 1 'ss -tunlp | grep 8080'`. Ta sama pułapka dotyczy przekierowań i `&&`.

`tail -f /var/log/messages` na świeżym RHEL 9/10 — plik nie istnieje.

```
tail: cannot open '/var/log/messages' for reading: No such file or directory
```

Od RHEL 8 `rsyslog` nie jest instalowany w minimalnych profilach. Logi idą wyłącznie do `journald` i żadnego pliku tekstowego nie ma. Nie instaluj `rsyslog` tylko po to, żeby móc użyć `tail` — użyj `journalctl -f`, `journalctl -fu nazwa.service`, `journalctl -f -p warning`. Jeśli potrzebujesz strumienia tekstowego do potoku: `journalctl -f -o cat | grep --line-buffered -i error`.

`tail -f` zostawiony na noc na pliku, który logrotate usunął — a `df` pokazuje pełny dysk.

`logrotate` po kilku cyklach kasuje najstarszy plik. Ale Twój `tail -f`, dalej wiszący w `screen`, trzyma do niego otwarty deskryptor, a jądro nie zwolni bloków, dopóki ostatni deskryptor się nie zamknie. `du -sh /var/log` pokazuje 400 MB, `df -h /var` pokazuje 100% zajętości i nic się nie zgadza.

```
sudo lsof +L1
```

```
COMMAND   PID USER FD TYPE DEVICE   SIZE/OFF NLINK    NODE NAME
tail    18422 root 3r  REG  253,2 3221225472     0 1049231 /var/log/app.log (deleted)
```

`NLINK` równe zero i dopisek `(deleted)` to potwierdzenie. Zabijasz proces, miejsce wraca natychmiast. Profilaktyka: `-F`, które po rotacji przeskoczy na nowy plik i zwolni stary.

□□ Ćwiczenia

> Wszystko poniżej wykonujesz na własnej maszynie, w katalogu `/tmp`. Nic nie wymaga uprawnień administratora.

Ćwiczenie 1 — `tail -f` kontra `less +F`

```
cd /tmp
cat > gen.sh <<'EOF'
#!/bin/bash
i=1
while true; do
  (( i % 7 )) && p=INFO || p=ERROR
  echo "$(date +%T) $p zadanie $i"
  i=$((i+1)); sleep 0.5
done
EOF
chmod +x gen.sh
./gen.sh > /tmp/demo.log &
```

Uruchom `tail -f /tmp/demo.log` i spróbuj znaleźć trzeci od końca `ERROR`.

Odpowiedź

Nie da się — `/wzorzec` nie istnieje w `tail -f`. `Ctrl+C`, żeby przerwać.

Teraz `less +F /tmp/demo.log`: `Ctrl+C`, `?ERROR`, `Enter`, potem `n` kilka razy — cofasz się po kolejnych błędach. `G` skacze na koniec, `F` wraca do podążania. Wyjście: `Ctrl+C`, potem `q`.

Ćwiczenie 2 — podświetlanie i buforowanie

```
tail -F /tmp/demo.log | grep --line-buffered ERROR
tail -F /tmp/demo.log | grep --color=always -E 'ERROR|$'
```

Pierwsza pokazuje tylko błędy, bez kontekstu; druga wszystko, z błędami na czerwono. Usuń `--line-buffered` z pierwszej i policz, ile sekund czekasz na pierwszą linię.

Czego się spodziewać

Z `--line-buffered` błędy pojawiają się co ~3,5 sekundy (co siódma linia, pół sekundy na linię). Bez tej flagi pierwsza linia wyjdzie dopiero po uzbieraniu przez `grep` 4 kB — przy tym generatorze to kilka minut ciszy, mimo że błędy lecą cały czas.

Sprzątanie: `pkill -f gen.sh; rm -f /tmp/gen.sh /tmp/demo.log`.

Pytania kontrolne

1. `tail -F /var/log/app.log | grep -i timeout` nie pokazuje nic od dwudziestu minut. Wymień dwie przyczyny, które nie oznaczają „nie ma timeoutów".

Odpowiedź

Po pierwsze: buforowanie blokowe w `grep` — brak `--line-buffered`, więc dopasowane linie siedzą w buforze, dopóki nie uzbiera się 4 kB. Po drugie: aplikacja może pisać teraz gdzie indziej (np. do `journald`). Sprawdź `ls -l --time-style=full-iso /var/log/app.log` — jeśli mtime nie rośnie, nic tam nie trafia.

2. Dlaczego `watch -n 2 'ss -tn state established | wc -l'` to zły sposób na wykrywanie krótkich skoków liczby połączeń?

Odpowiedź

`watch` robi migawkę stanu raz na dwie sekundy. Połączenie, które nawiązało się i zamknęło pomiędzy migawkami, nie pojawi się w żadnej z nich. Do zliczania zdarzeń, a nie stanu, potrzebujesz obserwacji ciągłej: logu aplikacji, licznika z `/proc/net/snmp` albo narzędzia opartego o eBPF.

3. Kiedy `less +F` NIE zastąpi `tail -F` i co z tym zrobić?

Odpowiedź

W dwóch przypadkach. Pierwszy: przy rotacji logu — `less +F` domyślnie śledzi deskryptor, więc po podmianie pliku zamiera tak samo jak `tail -f`; ratunek to `--follow-name`. Drugi: kiedy

chcesz przepuścić strumień przez potok albo do skryptu — `less` jest interaktywny i do potoku się nie nadaje. Tam zostaje `tail -F`.

Powiązane: head i tail — początek i koniec pliku · Pagery: less i more · cat: wypisywanie i sklejanie plików · Logi systemd: journalctl · logrotate — rotacja logów · tmux — sesje, które przeżyją rozłączenie

Źródła: `man 1 tail` · `man 1 less` · `man 1 watch` · `man 1 journalctl` · `man 1 stdbuf` · `man 8 lsof` · `man 1 multitail` · lnav.org

---

Revision #20
Created 2026-09-17 11:56:29 UTC by Tomasz Konieczny
Updated 2026-10-03 19:07:40 UTC by Tomasz Konieczny