head i tail: okno na początek i koniec pliku

Poziom: podstawy **Dotyczy: wszystkie dystrybucje** **Czas: ~10 min**

`head` pokazuje początek pliku, `tail` koniec. Razem z `cat` tworzą komplet: całość, góra, dół. Przy plikach, które mają gigabajty, to jedyne dwie komendy z tej trójki, których wolno użyć.

Skróty użyte na tej stronie (5)

Skrót	Co znaczy
CSV	w Windowsie: Cluster Shared Volume, dysk widoczny naraz dla wszystkich węzłów klastra; w danych: Comma-Separated Values, plik z wartościami oddzielonymi przecinkami
GB	Gigabyte: miliard bajtów, jednostka pojemności pamięci i plików, tu najczęściej rozmiar modelu albo pamięci karty graficznej
NFS	Network File System: udostępnianie katalogów po sieci w świecie Linuksa
PID	Process ID: numer procesu w systemie
POSIX	Portable Operating System Interface: wspólny zestaw zasad, dzięki któremu polecenie działa tak samo na różnych Uniksach

Fiszka

TYLKO ODCZYT

Wypisanie pierwszych linii pliku: head

```
head plik.log
```

Wypisujemy pierwsze 10 linii pliku. Bez żadnych flag — tyle wynosi wartość domyślna.

Przykład:

```
2026-09-17 08:14:02 INFO  uruchomienie usługi
2026-09-17 08:14:03 INFO  nasłuch na porcie 8080
2026-09-17 08:14:03 INFO  połączenie z bazą danych OK
2026-09-17 08:14:04 INFO  załadowano 42 moduły
2026-09-17 08:14:04 INFO  worker 1 gotowy
2026-09-17 08:14:04 INFO  worker 2 gotowy
2026-09-17 08:14:05 INFO  worker 3 gotowy
2026-09-17 08:14:05 INFO  usługa gotowa do pracy
2026-09-17 08:15:11 INFO  żądanie GET /health
2026-09-17 08:15:11 INFO  odpowiedź 200 w 3ms
```

□□ **head** = pokaż początek pliku, domyślnie dziesięć pierwszych linii

□□ TYLKO ODCZYT

Wypisanie ostatnich linii pliku: tail

```
tail plik.log
```

Dokładne przeciwieństwo `head`: ostatnie 10 linii, ta sama domyślna liczba.

Przykład:

```
2026-09-17 10:41:02 INFO  żądanie GET /api/orders
2026-09-17 10:41:03 WARN  wolna odpowiedź bazy: 1840ms
2026-09-17 10:41:05 INFO  żądanie GET /api/orders
2026-09-17 10:41:07 ERROR upstream timed out (110: Connection timed out)
2026-09-17 10:41:07 ERROR żądanie zakończone błędem 504
2026-09-17 10:41:09 INFO  żądanie GET /health
2026-09-17 10:41:09 INFO  odpowiedź 200 w 4ms
2026-09-17 10:41:12 INFO  żądanie GET /api/orders
2026-09-17 10:41:14 WARN  wolna odpowiedź bazy: 2210ms
2026-09-17 10:41:15 ERROR upstream timed out (110: Connection timed out)
```

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

□□ TYLKO ODCZYT

Wybór dowolnej liczby linii: -n N

```
head -n 50 plik.log
tail -n 50 plik.log
```

Zmieniamy domyślną dziesiątkę na dowolną liczbę — niezależnie po stronie `head` i po stronie `tail`. W GNU coreutils działa też skrót bez `-n`: `head -50` albo `tail -50`.

head = pokaż początek pliku, domyślnie dziesięć pierwszych linii · **tail** = pokaż koniec pliku, z flagą -f śledzi plik na żywo

TYLKO ODCZYT

Wypisanie wszystkiego oprócz ostatnich N linii: head -n -N

```
head -n -5 plik.log
```

Znak minus odwraca sens polecenia: zamiast pierwszych pięciu linii dostajemy cały plik **bez** ostatnich pięciu. Przydaje się, gdy plik ma stały fragment na końcu — sumę kontrolną, stopkę, podsumowanie — który chcemy odciąć przed dalszym przetwarzaniem.

head = pokaż początek pliku, domyślnie dziesięć pierwszych linii

TYLKO ODCZYT

Wypisanie od linii N do końca: tail -n +N

```
tail -n +100 plik.log
```

Znak plus zmienia znaczenie liczby: nie „ostatnie 100 linii", tylko „od setnej linii do końca pliku". Różnica z `-n 100` jest całkowita, nie kosmetyczna — patrz Częste błędy.

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

TYLKO ODCZYT

Liczenie bajtów zamiast linii: -c

```
head -c 512 obraz.iso
tail -c 1 plik.txt
```

Obie komendy umieją liczyć bajty zamiast linii. `head -c 512` wyciąga pierwsze pół kilobajta — przydatne przy sprawdzaniu nagłówka pliku binarnego. `tail -c 1` wyciąga ostatni bajt pliku

— klasyczny sposób sprawdzenia, czy plik kończy się znakiem nowej linii.

Przykład:

```
$ tail -c 1 plik.txt | od -An -tx1
 0a
```

Bajt `0a` to znak nowej linii — plik kończy się poprawnie.

head = pokaż początek pliku, domyślnie dziesięć pierwszych linii · **tail** = pokaż koniec pliku, z flagą -f śledzi plik na żywo

TYLKO ODCZYT

Obserwacja pliku na żywo: -f

```
tail -f /var/log/messages
```

Po wypisaniu ostatnich linii `tail` nie kończy pracy: zostaje uruchomiony i wypisuje każdą nową linię w momencie, gdy trafia do pliku. Wyjście: `Ctrl+C`.

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

TYLKO ODCZYT

Obserwacja odporna na rotację: -F

```
tail -F /var/log/messages
```

To samo co `-f`, plus śledzenie **nazwy** pliku zamiast otwartego deskryptora, plus automatyczne ponawianie prób, jeśli plik na chwilę zniknie. Jedyny sensowny wybór na plikach obsługiwanych przez logrotate — czyli praktycznie każdym logu systemowym.

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

TYLKO ODCZYT

Obserwacja wielu plików naraz

```
tail -f /var/log/nginx/access.log /var/log/nginx/error.log
```

`tail` potrafi śledzić kilka plików jednocześnie. Przy każdej zmianie źródła wstawia nagłówek z nazwą pliku, więc od razu widać, skąd przyszła dana linia.

Przykład:

```
==> /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
```

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

□□ TYLKO ODCZYT

Wyłączenie nagłówków: -q

```
tail -q /var/log/nginx/access.log /var/log/nginx/error.log
```

Wymusza brak nagłówków `==> plik <==` nawet przy wielu podanych plikach — wynik wygląda tak, jakby wszystkie pliki były jednym strumieniem.

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

□□ TYLKO ODCZYT

Wymuszenie nagłówka: -v

```
tail -v plik.log
```

Odwrotność `-q`: nagłówek z nazwą pliku pojawia się nawet wtedy, gdy podamy tylko jeden plik. Przydaje się w skryptach, które mają zawsze pokazywać, z jakiego źródła pochodzi wynik.

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

□□ ZMIENIA SYSTEM

Zakończenie obserwacji po zakończeniu procesu: --pid

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

`tail -f` sam z siebie nigdy się nie kończy. Z `--pid=PID` sprawdza okresowo, czy proces o tym numerze jeszcze żyje, i wychodzi, kiedy on się zakończy. `$!` to PID ostatnio uruchomionego zadania w tle — dzięki temu `tail -f` wstawiony do skryptu nie zawiesi go na zawsze.

PID = Process ID: numer procesu w systemie · **tail** = pokaż koniec pliku, z flagą -f śledzi plik na żywo

TYLKO ODCZYT

Wycinanie zakresu ze środka pliku

```
tail -n +1000 plik.log | head -n 21
```

Żadna z tych dwóch komend osobno nie umie wyciąć zakresu ze środka pliku — złożone razem potrafią. Czyta się to jako „od linii 1000 do końca, a z tego pierwsze 21 linii", czyli linie 1000–1020. Alternatywa `sed -n '1000,1020p' plik.log` jest krótsza, ale mniej oczywista, jeśli nie znamy składni `sed`.

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

TYLKO ODCZYT

Wszystkie flagi

Flaga	`head`	`tail`
`-n N`	pierwsze N linii	ostatnie N linii
`-n -N`	wszystko oprócz ostatnich N	—
`-n +N`	—	od linii N do końca
`-c N`	pierwsze N bajtów	ostatnie N bajtów
`-q`	bez nagłówków z nazwami plików	to samo
`-v`	zawsze z nagłówkiem	to samo
`-f`	—	śledź nowe dane
`-F`	—	śledź, przetrwaj rotację (`= -f --retry --follow=name`)

Flaga	`head`	`tail`
`--pid=PID`	—	zakończ `-f`, gdy proces PID umrze

Uważaj

- `tail -f` przestaje pokazywać cokolwiek po rotacji logu — używaj `tail -F` na plikach, które obsługuje logrotate.
- `tail -n 100` i `tail -n +100` to dwie zupełnie różne komendy. Minus/plus decyduje.
- Na logach systemd (`journald`) `tail -f` nie zadziała na pliku — tam jest `journalctl -f`.

PID = Process ID: numer procesu w systemie

Z życia wzięte

Z ŻYCIA WZIĘTE

Środa, 0:05, tuż po północy, gdy `logrotate` obraca logi. Tomek trzyma otwarte okno z `tail -f /var/log/aplikacja.log` na `app01` i po rotacji ekran po prostu przestaje się ruszać, choć aplikacja dalej działa i pisze do nowego pliku. Ola tłumaczy, że `-f` trzyma stary deskryptor, a nie nazwę, i każe mu zamienić nawyk na `tail -F` na wszystkich logach obsługiwanych przez logrotate.

Po co to istnieje

Plik `/var/log/messages` na serwerze, który chodzi od pół roku, ma 2,3 GB. Usługa przestała odpowiadać dwie minuty temu. Interesuje Cię ostatnie 40 linii.

`cat` wypisze wszystko od początku. Terminal będzie przewijał się przez kilka minut, bufor przewijania (domyślnie kilka tysięcy linii) i tak nie pomieści całości, a zanim skończy, incydent się rozwinie.

`tail -n 40 /var/log/messages` odpowie natychmiast.

I to jest cała idea: **plik może być większy niż to, co chcesz zobaczyć**, i prawie zawsze interesuje Cię jeden z dwóch końców. Początek — bo tam są nagłówki, deklaracje, pierwsze błędy

przy starcie usługi. Koniec — bo tam jest teraźniejszość.

Analogia

Pomyśl o **gazecie i dzienniku pokładowym**.

`head` to zerknięcie na pierwszą stronę gazety. Nie czytasz całej — chcesz wiedzieć, o czym jest numer, jaki ma nagłówek, z którego jest dnia. W pliku CSV pierwsza linia to nazwy kolumn. W pliku konfiguracyjnym — sekcja główna. W logu — moment startu usługi.

`tail` to dziennik pokładowy statku. Otwierasz go zawsze na ostatniej zapisanej stronie, bo interesuje Cię to, co wydarzyło się ostatnio. Nikt nie czyta dziennika od początku rejsu, żeby dowiedzieć się, co się właśnie zepsuło.

A `tail -f` to **stanie nad ramieniem oficera, który właśnie pisze**. Nie czytasz zapisanych stron — patrzysz, jak powstają kolejne linijki, w czasie rzeczywistym. Wychodzisz, kiedy zobaczysz to, na co czekałeś.

Ta analogia niesie też ograniczenie, które warto od razu wyłapać: jeśli oficer skończy jeden dziennik i zacznie nowy tom, a Ty dalej gapisz się w stary, to przestaniesz cokolwiek widzieć. Dokładnie to robi rotacja logów — i dokładnie dlatego istnieje `tail -F`.

Jak to działa

Rozwiń: jak to działa pod spodem

`head` czyta plik od początku, liczy linie (albo bajty), i po osiągnięciu limitu po prostu przestaje czytać i kończy pracę. Nie dotyka reszty pliku. Dlatego `head` na pliku 40 GB jest natychmiastowe.

Ma to praktyczny skutek uboczny w potokach. Gdy `head` zamknie swoje wejście, proces po lewej stronie potoku dostanie sygnał `SIGPIPE` i zostanie zabity:

```
yes | head -n 3
```

```
y
y
y
```

Komenda `yes` wypisywałaby `y` w nieskończoność, ale `head` ją ubija. To jest cecha, nie błąd — tak się bezpiecznie ucina nieskończone strumienie.

`tail` ma trudniejsze zadanie, bo żeby pokazać ostatnie 10 linii, musiałby teoretycznie przeczytać cały plik i liczyć. Robi to sprytniej: jeśli plik jest zwykłym plikiem na dysku (a nie strumieniem), `tail` używa `lseek()`, żeby skoczyć na koniec, czyta blok wstecz, liczy w nim znaki nowej linii, i jeśli ich nie starczy — cofa się o kolejny blok. Dlatego `tail -n 10` na pliku 2 GB jest równie szybkie jak na pliku 2 kB.

Jeśli jednak dane przychodzą z potoku, tej sztuczki nie da się wykonać — nie ma po czym się cofać. Wtedy `tail` musi przeczytać cały strumień, trzymając ostatnie N linii w buforze. To wciąż działa, ale trwa:

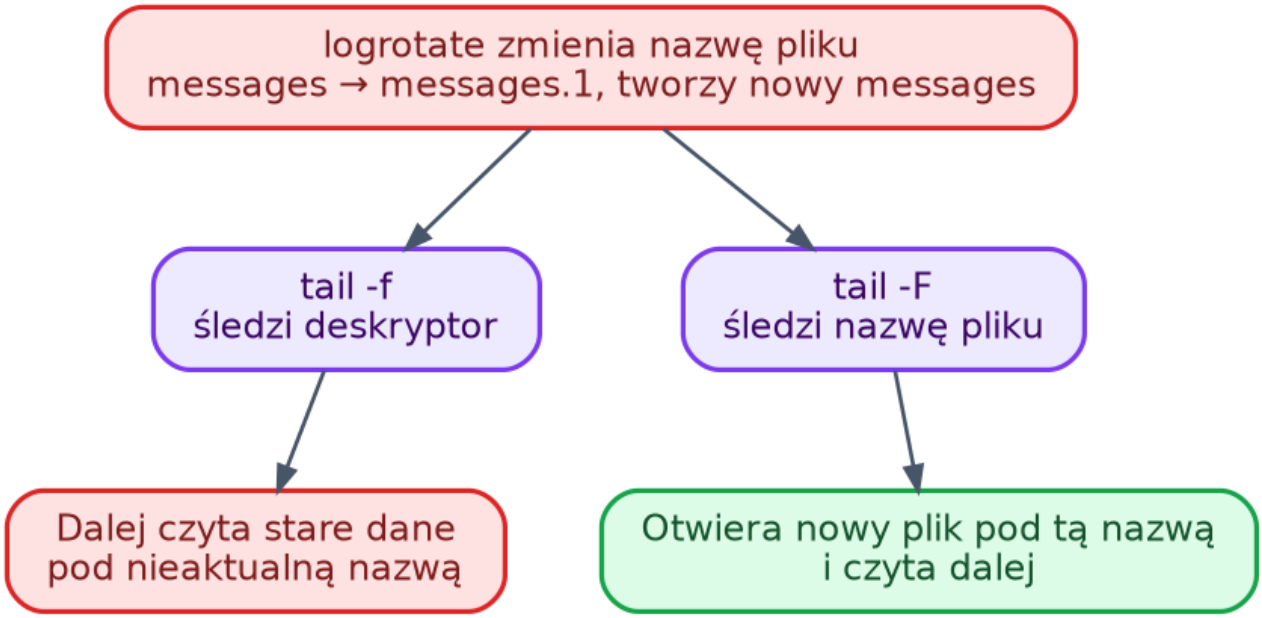
```
cat ogromny.log | tail -n 10     # wolno — cały plik przez potok
tail -n 10 ogromny.log           # natychmiast — lseek na koniec
```

Kolejny powód, żeby nie wrzucać `cat` przed `tail`.

`tail -f` działa inaczej niż obie powyższe. Po wypisaniu ostatnich linii nie kończy pracy — zapamiętuje pozycję i cyklicznie sprawdza, czy plik urósł. Na nowoczesnym Linuksie używa do tego `inotify` (powiadomienia z jądra), więc nie marnuje procesora na odpytywanie. Jeśli `inotify` jest niedostępne (np. na NFS), spada do sprawdzania co sekundę.

Kluczowe: `tail -f` śledzi otwarty deskryptor pliku, nie jego nazwę.

> **Pojęcie: deskryptor kontra nazwa pliku.** Nazwa pliku (`/var/log/messages`) to tylko wpis w katalogu, wskazujący na dane leżące na dysku. Deskryptor to uchwyt, który proces dostaje od jądra w momencie otwarcia pliku — wskazuje bezpośrednio na te dane, niezależnie od tego, czy nazwa w międzyczasie zniknie albo zacznie wskazywać na coś innego. Kiedy `mv` albo `logrotate` zmienia nazwę pliku, proces trzymający stary deskryptor dalej widzi te same, stare dane — po prostu pod nazwą, która już nie jest aktualna. Nazwa i deskryptor się rozjeżdżają. Stąd bierze się najczęstsza pułapka na tej stronie.



Po rotacji logu tail -f zostaje przy starych danych, tail -F otwiera nowy plik pod tą samą nazwą (kliknięcie otwiera pełny rozmiar)

Krok po kroku

1. Sprawdź nagłówek pliku CSV, zanim go przetworzysz.

```
head -n 3 /tmp/eksport.csv
```

```
id;nazwa;data_utworzenia;status
1001;Pozycja A;2026-08-14;aktywny
1002;Pozycja B;2026-08-15;nieaktywny
```

Teraz wiesz, że separatorem jest średnik, a nie przecinek — co zmienia każdą kolejną komendę.

2. Zobacz koniec logu po awarii.

```
tail -n 50 /var/log/messages
```

3. Obserwuj log na żywo w trakcie restartu usługi.

Otwórz dwie sesje. W pierwszej:

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

W drugiej:

```
sudo systemctl restart nginx
```

Wracasz do pierwszej i widzisz, co się wydarzyło, linia po linii. Wyjście: `Ctrl+C`.

4. Filtruj strumień na żywo.

```
tail -F /var/log/messages | grep --line-buffered -i 'error\|fail'
```

Flaga `--line-buffered` jest tu obowiązkowa. Bez niej `grep` widzi, że jego wyjście to nie terminal tylko potok, więc przełącza się na buforowanie blokowe (4 kB) — i pokaże Ci coś dopiero, kiedy uzbiera pełny blok. Przy rzadkich błędach możesz czekać godzinami, patrząc na pusty ekran i będąc pewnym, że nic się nie dzieje.

5. Wytnij fragment ze środka pliku.

Kombinacja obu komend daje dowolny zakres linii. Linie 1000–1020:

```
tail -n +1000 plik.log | head -n 21
```

Czyta się to jako: „od linii 1000 do końca, a z tego pierwsze 21 linii". Alternatywa przez `sed` jest krótsza, ale mniej oczywista: `sed -n '1000,1020p' plik.log`.

6. Losowa próbka z zachowaniem nagłówka.

```
{ head -n 1 dane.csv; tail -n +2 dane.csv | shuf -n 20; } > probka.csv
```

Nawiasy klamrowe grupują komendy tak, że przekierowanie `>` obejmuje obie. Przydaje się, kiedy chcesz zobaczyć reprezentatywny wycinek pliku, którego nie da się otworzyć w arkuszu.

Częste błędy

`tail -f` przestaje pokazywać cokolwiek po rotacji logu.

To jest *ten* błąd. Zostawiasz `tail -f /var/log/messages` na noc. O północy logrotate zmienia nazwę pliku na `messages-20260917` i tworzy nowy, pusty `/var/log/messages`. Twój `tail` trzyma deskryptor do starego pliku — tego pod nową nazwą, do którego już nic nie trafia. Ekran zamiera. Ty myślisz, że system się uspokoił.

> **Pojęcie: rotacja logów i logrotate.** Logi rosłyby w nieskończoność, gdyby nikt ich nie porządkował. `logrotate` to program uruchamiany cyklicznie (zwykle raz dziennie, przez `systemd timer` albo `cron`), który zmienia nazwę bieżącego pliku logu — na przykład na `messages-20260917` albo z numerem, `messages.1` — tworzy w jego miejsce nowy, pusty plik o starej nazwie, a po kilku cyklach kompresuje albo kasuje najstarsze wersje. Dzięki temu dysk się nie zapełnia, ale każdy proces, który akurat czyta stary plik przez otwarty deskryptor, musi się jakoś zorientować, że nazwa teraz wskazuje na coś nowego — a `tail -f` się nie orientuje.

Rozwiązanie:

```
tail -F /var/log/messages
```

`-F` to skrót od `--follow=name --retry`: śledź **nazwę**, a jeśli plik zniknie, ponawiaj próby aż się pojawi. Przy rotacji `tail` zgłosi to wprost:

```
tail: '/var/log/messages' has been replaced;  following new file
```

Praktyczna zasada: na pliku, który obsługuje logrotate, zawsze `-F`. A skoro logrotate obsługuje praktycznie każdy log systemowy — używaj `-F` domyślnie. Kosztuje jedną wielką literę.

Pomylenie `-n N` z `-n +N`.

```
tail -n 100 plik     # ostatnie 100 linii
tail -n +100 plik    # od setnej linii do końca
```

Przy pliku na 10 000 linii pierwsza komenda pokaże 100 linii, druga 9901. Pomyłka jest cicha — nie ma błędu, są po prostu inne dane. Analogicznie po stronie `head`: `head -n -5` to „wszystko oprócz ostatnich pięciu".

Oczekiwanie, że `tail -f` zadziała na logach systemd.

Na systemach z `journald` większość logów nie jest zwykłymi plikami tekstowymi — leżą w binarnym dzienniku w `/var/log/journal/`. `tail -f` na nich nie ma sensu. Odpowiednik to:

```
journalctl -f
journalctl -u nginx -f          # tylko jedna usługa
journalctl -u nginx -n 50       # ostatnie 50 wpisów
journalctl -u nginx --since '10 min ago'
```

Pełne omówienie `journalctl` i pojęcia dziennika systemd jest na osobnej stronie o obserwacji logów na żywo.

Buforowanie w potoku.

Opisane wyżej przy `--line-buffered`. Dotyczy nie tylko `grep` — `awk`, `sed` i większość narzędzi zachowuje się tak samo. Uniwersalne obejście:

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

`stdbuf -oL` wymusza buforowanie liniowe na dowolnym programie.

`tail -f` na pliku, którego proces trzyma i nadpisuje w miejscu.

Niektóre aplikacje (zwłaszcza własne, pisane bez myśli o logrotate) nie dopisują do logu, tylko go nadpisują. Wtedy plik się nie wydłuża, tylko zmienia w środku, a `tail -f` nie zobaczy nic, bo czeka na przyrost rozmiaru. Nie ma tu dobrego obejścia poza naprawieniem aplikacji — albo `watch -n 1 'tail -n 20 plik.log'` jako proteza.

□□ Ćwiczenia

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

Ćwiczenie 1 — rotacja logu na własne oczy

```
cd /tmp
printf 'linia 1\n' > test.log

# sesja A:
tail -f test.log
```

W drugiej sesji terminala:

```
cd /tmp
echo 'linia 2' >> test.log        # sesja A to pokaże
mv test.log test.log.1            # symulacja rotacji
echo 'linia 3' > test.log         # sesja A tego NIE pokaże
```

Dlaczego sesja A nie pokaże trzeciej linii, mimo że plik `test.log` istnieje i ma zawartość?

Odpowiedź

`tail -f` trzyma otwarty deskryptor do pliku, który teraz nazywa się `test.log.1` — do niego nic już nie trafia. Nowy `test.log` to zupełnie inny plik na dysku, którego `tail -f` nie widzi, bo śledzi deskryptor, nie nazwę. Zabij `tail` (`Ctrl+C`), uruchom `tail -F test.log` i powtórz eksperyment — tym razem trzecia linia się pojawi, bo `-F` sprawdza nazwę, a po jej podmianie otwiera plik na nowo.

Ćwiczenie 2 — wycięcie zakresu

```
seq 1 1000 > /tmp/liczby.txt
```

Wyciągnij z tego pliku wyłącznie linie 45–54, używając wyłącznie `head` i `tail`.

Odpowiedź

```
tail -n +45 /tmp/liczby.txt | head -n 10
```

Wynik: liczby od 45 do 54. Odwrotna kolejność też zadziała (`head -n 54 /tmp/liczby.txt | tail -n 10`), ale przy dużym pliku pierwsza wersja czyta mniej, bo `tail -n +N` na zwykłym pliku korzysta z szybkiego skoku na dysku.

Ćwiczenie 3 — koniec pliku bez nowej linii

```
cd /tmp
printf 'ostatnia linia bez konca' > bez_konca.txt
```

Sprawdź, czy `bez_konca.txt` kończy się znakiem nowej linii, używając `tail -c 1`.

Odpowiedź

```
tail -c 1 bez_konca.txt | od -An -tx1
```

```
 61
```

Ostatni bajt to `61` (litera `a`), a nie `0a` — plik nie kończy się nową linią. Krótsza wersja bez `od`:

```
[ -n "$(tail -c 1 bez_konca.txt)" ] && echo "BRAK nowej linii na końcu"
```

Pytania kontrolne

1. Dlaczego `tail -n 5 wielki.log` jest natychmiastowe, a `cat wielki.log | tail -n 5` trwa minutę?

Odpowiedź

W pierwszym przypadku `tail` dostaje zwykły plik i może użyć `lseek()`, żeby skoczyć na koniec i czytać wstecz. W drugim dane przychodzą potokiem — nie da się w nim cofnąć, więc `tail` musi przeczytać cały strumień od początku do końca, trzymając w buforze ostatnie 5 linii.

2. Kiedy `head` zabija proces po lewej stronie potoku i dlaczego to dobrze?

Odpowiedź

Gdy `head` osiągnie limit linii, kończy pracę i zamyka swoje wejście. Proces po lewej stronie przy próbie zapisu do zamkniętego potoku dostaje `SIGPIPE` i ginie. Dzięki temu `jakas_nieskonczona_komenda | head -n 20` kończy się od razu, zamiast mielić w nieskończoność.

3. `tail -F` czy `tail -f` na `/var/log/secure`?

Odpowiedź

`-F`. `/var/log/secure` jest obsługiwane przez logrotate, więc prędzej czy później zostanie podmienione, a `-f` po cichu przestanie pokazywać nowe wpisy.

4. Masz plik 8 GB i chcesz zobaczyć linie od 5 000 000 do 5 000 010. `head -n 5000010 plik | tail -n 11` czy `tail -n +5000000 plik | head -n 11`?

Odpowiedź

Obie działają i obie muszą przeczytać plik do linii 5 000 010 — `tail -n +N` nie umie skoczyć do konkretnej linii, bo nie wie, gdzie ona leży (linie mają zmienną długość). Różnica jest kosmetyczna. Jeśli zależy Ci na szybkości przy bardzo dużych plikach i znasz przybliżony offset bajtowy, praktyczniejsze bywa `tail -c +BAJT`, które skacze natychmiast.

Powiązane: cat: wypisywanie i sklejanie plików · less i more — pagery · Obserwacja logów na żywo: tail -f i less +F · Potoki i filozofia Unix · Logi systemd: journalctl

Źródła: `man 1 head` · `man 1 tail` · `man 8 logrotate` · `man 1 stdbuf` · POSIX.1-2024

Revision #20
Created 2026-09-17 11:56:28 UTC by Tomasz Konieczny
Updated 2026-10-03 19:07:38 UTC by Tomasz Konieczny