cat: wypisywanie i sklejanie plików

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

`cat` *wypisuje zawartość pliku i skleja pliki w jeden strumień. Najprostsza komenda w systemie i jednocześnie ta, którą najczęściej używa się źle.*

Skróty użyte na tej stronie (2)

Skrót	Co znaczy
GB	Gigabyte: miliard bajtów, jednostka pojemności pamięci i plików, tu najczęściej rozmiar modelu albo pamięci karty graficznej
POSIX	Portable Operating System Interface: wspólny zestaw zasad, dzięki któremu polecenie działa tak samo na różnych Uniksach

Fiszka

TYLKO ODCZYT

Wypisanie zawartości pliku

```
cat /etc/os-release
```

Cała zawartość pliku ląduje na ekranie, od pierwszego do ostatniego bajtu.

Przykład:

```
NAME="Rocky Linux"
VERSION="9.4 (Blue Onyx)"
ID="rocky"
ID_LIKE="rhel centos fedora"
PRETTY_NAME="Rocky Linux 9.4 (Blue Onyx)"
```

□□ **cat** = concatenate: sklej i wypisz pliki

□□ ZMIENIA SYSTEM

Sklejenie kilku plików

```
cat naglowek.txt tresc.txt stopka.txt > dokument.txt
```

Pliki są przepuszczane po kolei, w podanej kolejności, i trafiają do jednego strumienia. To jest oryginalne zadanie tej komendy — nazwa pochodzi od *concatenate*, nie od kota.

□□ **cat** = concatenate: sklej i wypisz pliki

□□ ZMIENIA SYSTEM

Dopisanie na koniec innego pliku

```
cat dodatek.txt >> raport.txt
```

Dwa znaki `>>` zamiast jednego `>` dopisują zamiast nadpisywać. Różnica jest nieodwracalna, więc warto ją mieć w palcach.

□□ **cat** = concatenate: sklej i wypisz pliki

□□ TYLKO ODCZYT

Numerowanie linii: -n

```
cat -n /etc/fstab
```

Numerujemy **wszystkie** linie, łącznie z pustymi. Przydaje się, gdy komunikat błędu mówi „syntax error in line 14" i musimy tę linię znaleźć.

Przykład:

```
     1□#
     2□# /etc/fstab
     3□# Created by anaconda on Mon Apr 14 09:22:11 2025
     4□#
     5□/dev/mapper/vgOS-lvOS   /       xfs   defaults   0 0
```

□□ **cat** = concatenate: sklej i wypisz pliki

□□ TYLKO ODCZYT

Numerowanie z pominięciem pustych linii: -b

```
cat -b /etc/fstab
```

To samo co `-n`, ale puste linie nie dostają numeru. Wygodniejsze przy plikach konfiguracyjnych rozdzielonych pustymi liniami.

□□ **cat** = concatenate: sklej i wypisz pliki

□□ TYLKO ODCZYT

Pokazanie znaków niewidocznych: -A

```
cat -A config.ini
```

Wyświetlamy tabulatory jako `^I`, koniec linii jako `$`, a znaki sterujące w notacji `^X`. Sięgamy po to, gdy plik „wygląda dobrze", a usługa go odrzuca.

Przykład:

```
[main]$
port=8080^M$
host=localhost^M$
```

`^M` to znak powrotu karetki — plik ma zakończenia linii w stylu Windows. Aplikacja widzi port o nazwie `8080` plus znak sterujący i nie potrafi go sparsować.

□□ **cat** = concatenate: sklej i wypisz pliki

□□ TYLKO ODCZYT

Zwinięcie pustych linii: -s

```
cat -s brzydki.txt
```

Ciąg kilku pustych linii zostaje zredukowany do jednej. Nie usuwa pustych linii całkowicie — do tego jest `sed -i '/^$/d'`.

□□ **cat** = concatenate: sklej i wypisz pliki

□□ ZMIENIA SYSTEM

Zapis z klawiatury

```
cat > notatka.txt
```

Wszystko, co wpiszemy, trafia do pliku. Kończymy przez `Ctrl+D` (znak końca pliku). `Ctrl+C` przerywa i zostawia to, co już zostało zapisane.

□□ **cat** = concatenate: sklej i wypisz pliki

□□ ZMIENIA SYSTEM

Zapis pliku ze skryptu: here-document

```
cat > /etc/motd <<'EOF'
Serwer produkcyjny.
Zmiany wyłącznie przez zgłoszenie w systemie ticketowym.
EOF
```

Wszystko między `<<'EOF'` a linią `EOF` trafia do pliku dosłownie. Apostrofy wokół `EOF` mają znaczenie: bez nich powłoka podstawi zmienne i wykona `$(komendy)` wewnątrz bloku. To najczystszy sposób generowania plików konfiguracyjnych w skryptach.

□□ **cat** = concatenate: sklej i wypisz pliki

□□ TYLKO ODCZYT

Sprawdzenie typu pliku przed wypisaniem

```
file /usr/bin/ls
```

Przykład:

```
/usr/bin/ls: ELF 64-bit LSB pie executable, x86-64, dynamically linked
```

`cat` na takim pliku rozsypie terminal. Odruch: przy nieznanym pliku najpierw `file`.

□□ **file** = rozpoznaj rzeczywisty typ pliku po jego zawartości, niezależnie od rozszerzenia

◫◫ TYLKO ODCZYT

Wszystkie flagi

Flaga	Co robi
`-n`	numeruje wszystkie linie
`-b`	numeruje tylko niepuste linie
`-A`	pokazuje znaki niedrukowalne (równoważne `-vET`)
`-E`	pokazuje koniec linii jako `$`
`-T`	pokazuje tabulatory jako `^I`
`-v`	pokazuje znaki sterujące
`-s`	zwija ciągi pustych linii do jednej

Uważaj

- `cat plik | grep wzorzec` to marnowanie procesu — `grep wzorzec plik` robi to samo.
- `cat` na pliku binarnym rozsypie terminal; ratunek to `reset` albo `stty sane`.
- `cat plik > plik` **kasuje plik**, bo powłoka czyści plik wyjściowy, zanim `cat` zdąży cokolwiek przeczytać.

Z życia wzięte

◫◫ Z ŻYCIA WZIĘTE

Wtorek, 13:40. Tomek chce dopisać nagłówek na początku pliku konfiguracyjnego na `app01` i odruchowo wpisuje `cat naglowek.txt plik.conf > plik.conf`. Plik `plik.conf` staje się pusty, zanim `cat` zdąży cokolwiek przeczytać, bo powłoka czyści plik wyjściowy jeszcze przed uruchomieniem komendy. Ratuje go kopia sprzed godziny, więc od tej pory zawsze pisze najpierw do pliku tymczasowego i dopiero potem robi `mv`.

Po co to istnieje

> **Pojęcie: standardowe wyjście.** Każdy uruchomiony program dostaje od systemu trzy gotowe kanały: jeden do czytania danych i dwa do pisania. Kanał, na który program wypisuje swój normalny wynik, nazywa się standardowym wyjściem. Domyślnie prowadzi on na ekran, ale powłoka może go przekierować do pliku albo do następnego programu — i program o tym nie wie, pisze zawsze tak samo.

Nazwa `cat` pochodzi od **concatenate** — „połącz". I to jest oryginalne zadanie tej komendy: wziąć kilka plików i wypuścić je jako jeden ciągły strumień danych.

W 1971 roku, kiedy `cat` trafiło do pierwszego wydania Uniksa, dane często były porozbijane na kawałki — plik nie mieścił się na taśmie, tekst przychodził w częściach. Potrzebne było narzędzie, które skleja je z powrotem. To, że przy okazji można podać jeden plik i zobaczyć go na ekranie, jest efektem ubocznym: „wypisz na standardowe wyjście" to po prostu domyślny cel sklejania.

Dziś używamy głównie tego efektu ubocznego. Ale warto pamiętać, do czego `cat` naprawdę służy, bo to tłumaczy, dlaczego niektóre rzeczy działają tak, a nie inaczej.

Analogia

Wyobraź sobie **podajnik dokumentów w kserokopiarce**.

Kładziesz na nim plik papierów i naciskasz start. Maszyna przepuszcza kartki jedna po drugiej, w kolejności w jakiej leżą, i wypuszcza je do tacki odbiorczej. Nie czyta ich. Nie sortuje. Nie zatrzymuje się, żeby zapytać, czy na pewno chcesz przepuścić osiemset stron.

`cat` to ten podajnik. Podajesz mu pliki, on przepuszcza ich zawartość bajt po bajcie na standardowe wyjście. Jeśli podasz trzy pliki, potraktuje je jak trzy pliki papierów położone na podajniku jeden po drugim — na wyjściu dostaniesz jeden ciągły stos, bez żadnego znacznika, gdzie kończył się pierwszy plik, a zaczynał drugi.

I dokładnie stąd bierze się najczęstsze zaskoczenie: jeśli pierwszy plik nie kończy się znakiem nowej linii, ostatnia linia pierwszego pliku sklei się z pierwszą linią drugiego. Podajnik nie dokłada pustej kartki między plikami, bo nie wie, że były to osobne pliki.

Jak to działa

`cat` wykonuje w pętli trzy czynności:

1. Otwiera plik podany jako argument — albo czyta ze **standardowego wejścia**, jeśli argumentów nie ma.
2. Czyta blok bajtów do bufora.

3. Zapisuje ten bufor na standardowe wyjście.

I tyle. Nie interpretuje zawartości, nie zna pojęcia „linia" (poza flagami `-n` i `-b`, które linie liczą), nie ładuje całego pliku do pamięci. Dlatego `cat` na pliku 40 GB nie wysadzi serwera — wysadzi tylko Twoją cierpliwość i bufor terminala.

Brak argumentów jest istotny:

```
cat
```

Terminal wygląda, jakby się zawiesił. Nie zawiesił się — czeka na wejście z klawiatury. Każda linia, którą wpiszesz, zostanie od razu przepisana na wyjście. `Ctrl+D` wysyła znak końca pliku i kończy zabawę.

To zachowanie jest fundamentem tego, że `cat` da się wstawić w środek potoku.

Krok po kroku

1. Sprawdź, czy plik konfiguracyjny nie ma ukrytych znaków.

Plik przeklejony z Windowsa ma zakończenia linii CRLF. Na ekranie wygląda normalnie:

```
cat /etc/myapp/config.ini
```

```
[main]
port=8080
host=localhost
```

2. Zajrzyj pod spód.

```
cat -A /etc/myapp/config.ini
```

```
[main]$
port=8080^M$
host=localhost^M$
```

Teraz widać `^M` na końcu każdej linii.

3. Napraw.

```
sed -i 's/\r$//' /etc/myapp/config.ini
```

Albo `dos2unix config.ini`, jeśli pakiet jest zainstalowany. Ponowne `cat -A` powinno pokazać same `$`.

Częste błędy

Useless Use of Cat.

```
cat /var/log/messages | grep "error"
```

Działa, ale uruchamia dwa procesy zamiast jednego i zmusza jądro do przepchnięcia całego pliku przez potok. `grep` umie sam otworzyć plik:

```
grep "error" /var/log/messages
```

To nie jest czepialstwo o styl. Przy pliku na 5 GB różnica jest mierzalna, a w skrypcie uruchamianym co minutę z crona — zauważalna. Ta pomyłka ma własną nazwę w kulturze uniksowej: *UUOC*, a w grupie `comp.unix.shell` przez lata przyznawano za nią żartobliwą „nagrodę".

Jest jeden przypadek, w którym `cat` w potoku jest uzasadniony: gdy plików jest wiele i chcesz przetworzyć je jako jeden strumień, np. `cat log-2026-*.txt | sort | uniq -c`.

Zniszczenie pliku przez przekierowanie do samego siebie.

```
cat plik.txt > plik.txt
```

Plik jest teraz pusty. Powłoka przetwarza przekierowanie **przed** uruchomieniem komendy: otwiera `plik.txt` do zapisu, co obcina go do zera bajtów, i dopiero potem startuje `cat`, który dostaje pusty plik. Nie ma tu żadnego ostrzeżenia.

To samo dotyczy `sort plik > plik` i każdej innej komendy. Do modyfikacji w miejscu służy `sed -i`, `sponge` z pakietu `moreutils` albo plik tymczasowy.

Wyświetlenie pliku binarnego.

```
cat /usr/bin/ls
```

Terminal wypisuje śmieci, a po chwili przestaje reagować sensownie — w danych binarnych trafiły się sekwencje sterujące, które przełączyły go w inny zestaw znaków. Ratunek:

```
reset
```

Jeśli terminal nie pokazuje tego, co piszesz, wpisz `reset` na ślepo i naciśnij Enter. Alternatywa: `stty sane` zatwierdzone przez `Ctrl+J` zamiast Entera.

Sklejenie linii przy łączeniu plików.

Jeśli `czesc1.txt` nie kończy się znakiem nowej linii, `cat czesc1.txt czesc2.txt` zlepi ostatnią linię pierwszego pliku z pierwszą linią drugiego. Sprawdzenie:

```
tail -c 1 czesc1.txt | od -An -tx1
```

```
 6b
```

Ostatni bajt to `6b` (litera `k`), a nie `0a` (znak nowej linii) — plik nie kończy się poprawnie.

□□ Ćwiczenia

> Wszystko poniżej wykonujesz na własnej maszynie, w katalogu `/tmp`. Nic nie wymaga uprawnień administratora i nic nie zostawia śladu poza plikami, które sam tworzysz.

Ćwiczenie 1 — brakująca nowa linia

```
cd /tmp
printf 'pierwsza\ndruga' > a.txt
printf 'trzecia\nczwarta\n' > b.txt
cat a.txt b.txt
```

Ile linii zobaczysz i dlaczego?

Odpowiedź

Trzy, nie cztery:

```
pierwsza
drugatrzecia
czwarta
```

`a.txt` nie kończy się znakiem nowej linii, więc `druga` i `trzecia` znalazły się w jednej linii. Potwierdzisz to numeracją:

```
cat -n a.txt b.txt
```

Naprawa i ponowne sprawdzenie:

```
printf '\n' >> a.txt
cat -n a.txt b.txt
```

Ćwiczenie 2 — ukryte znaki CRLF

```
cd /tmp
printf 'port=8080\r\nhost=localhost\r\n' > config.ini
cat config.ini
cat -A config.ini
```

Zwróć uwagę, że pierwsza komenda nie pokazuje niczego podejrzanego. Napraw plik i sprawdź efekt.

Odpowiedź

```
sed -i 's/\r$//' config.ini
cat -A config.ini
```

Teraz każda linia kończy się samym `$`, bez `^M`.

Pytania kontrolne

1. Dlaczego `cat plik | wc -l` jest gorsze niż `wc -l < plik`?

Odpowiedź

`cat plik | wc -l` uruchamia dwa procesy i przepycha dane przez potok. `wc -l < plik` uruchamia jeden proces, a plik podaje mu powłoka jako standardowe wejście — bez dodatkowego procesu i bez kopiowania przez bufor potoku. Wersja `wc -l plik` też jest dobra, z tą różnicą, że `wc` wypisze wtedy nazwę pliku obok liczby.

2. Co zrobi `cat` uruchomione bez żadnego argumentu i bez przekierowania?

Odpowiedź

Będzie czytać ze standardowego wejścia, czyli z klawiatury, i przepisywać każdą linię na ekran. Kończysz przez `Ctrl+D` (koniec pliku) albo `Ctrl+C` (przerwanie).

3. Masz plik `dane.csv` i chcesz usunąć z niego puste linie, zapisując wynik do tego samego pliku. Dlaczego `cat -s dane.csv > dane.csv` nie zadziała?

Odpowiedź

Z dwóch niezależnych powodów. Po pierwsze powłoka obetnie `dane.csv` do zera bajtów przy otwieraniu przekierowania, zanim `cat` go przeczyta — plik będzie pusty. Po drugie `-s` nie usuwa pustych linii, tylko zwija ich ciągi do jednej. Do usunięcia wszystkich pustych linii w miejscu: `sed -i '/^$/d' dane.csv`.

Powiązane: head i tail: okno na początek i koniec pliku · Trzy strumienie: stdin, stdout, stderr · Przekierowania: >, >>, 2>, &>, <, tee · Pliki, które nie są tekstem: file, strings, hexdump, xxd

Źródła: `man 1 cat` · `info coreutils 'cat invocation'` · POSIX.1-2024

Revision #22
Created 2025-11-23 13:58:08 UTC by Tomasz Konieczny
Updated 2026-10-03 19:07:36 UTC by Tomasz Konieczny