sed: edytor strumieniowy

Poziom: średni **Dotyczy: wszystkie dystrybucje** **Czas: ~18 min**

`sed` *(stream editor) edytuje tekst linia po linii, bez otwierania pliku w edytorze. To narzędzie do masowych, powtarzalnych zmian — w jednym pliku i w tysiącu plików naraz.*

Skróty użyte na tej stronie (8)

Skrót	Co znaczy
BRE	Basic Regular Expression: podstawowa, starsza odmiana wyrażeń regularnych, wymaga ukośnika wstecznego przed nawiasami
BSD	Berkeley Software Distribution: rodzina Uniksów z uczelni Berkeley, inna gałąź niż Linux, stąd czasem inna składnia tych samych poleceń
CI	Continuous Integration: automatyczne budowanie i testowanie kodu po każdej zmianie
ERE	Extended Regular Expression: rozszerzona odmiana wyrażeń regularnych, bez ukośnika wstecznego przed nawiasami i plusem
IP	Internet Protocol: adres komputera w sieci; w elektryce: stopień ochrony obudowy (Ingress Protection)
IPv4	Internet Protocol wersja 4: adresy zapisane czterema liczbami z kropkami, na przykład 10.20.10.77
NTP	Network Time Protocol: synchronizacja zegara po sieci
POSIX	Portable Operating System Interface: wspólny zestaw zasad, dzięki któremu polecenie działa tak samo na różnych Uniksach

Fiszka

> **Pojęcie: edytor strumieniowy.** Program, który przetwarza tekst w locie, linia po linii, zamiast otwierać cały plik na ekranie do ręcznej edycji jak `vim` czy `nano`. Dane płyną przez niego jak przez rurę: wpada jedna linia, wychodzi przetworzona, wpada kolejna. Edytor strumieniowy nie pyta o potwierdzenie i nie pokazuje podglądu — dostaje gotowy skrypt poleceń i wykonuje go bezobsługowo na całym wejściu.

> **Pojęcie: przestrzeń robocza (pattern space).** Bufor, do którego `sed` wczytuje jedną linię wejścia, zanim wykona na niej cokolwiek. Wszystkie polecenia skryptu — `s///`, `d`, `p` i reszta — działają na zawartości tego bufora, nigdy bezpośrednio na pliku na dysku. Po przejściu skryptu zawartość przestrzeni roboczej trafia na standardowe wyjście (chyba że wyłączono to `-n`), bufor się czyści, i wczytywana jest kolejna linia.

Dane testowe używane w tej Fiszce — plik konfiguracyjny jednego serwera:

```
cat > serwer.conf <<'EOF'
# --- START bloku app01 ---
host=app01
ip=10.20.10.41
port=8080
docroot=/var/www/html
# backup_ip=10.20.10.42
timeout=30

max_connections=100
tags=prod,prod,web
retries=3
log_level=error
# --- END bloku app01 ---
status=aktywny
EOF
```

TYLKO ODCZYT

Podstawienie: `s///`

```
sed 's/app01/docker01/' serwer.conf
```

Podstawowa składnia: dla każdej linii szukamy `wzorca` i zamieniamy go na `tekst`. Domyślnie zamieniamy tylko **pierwsze** dopasowanie w każdej linii.

Przykład:

```
# --- START bloku docker01 ---
host=docker01
ip=10.20.10.41
port=8080
```

```
docroot=/var/www/html
# backup_ip=10.20.10.42
timeout=30

max_connections=100
tags=prod,prod,web
retries=3
log_level=error
# --- END bloku docker01 ---
status=aktywny
```

sed = stream editor: edytor strumienia tekstu

TYLKO ODCZYT

Wszystkie wystąpienia w linii: `g`

```
sed 's/prod/produkcja/' serwer.conf
```

Przykład:

```
tags=produkcja,prod,web
```

Bez `g` zmieniamy tylko pierwsze „prod" w linii `tags=prod,prod,web`. Z flagą `g` zmieniamy wszystkie wystąpienia w tej samej linii:

```
sed 's/prod/produkcja/g' serwer.conf
```

Przykład:

```
tags=produkcja,produkcja,web
```

sed = stream editor: edytor strumienia tekstu

TYLKO ODCZYT

Pomijanie wielkości liter: `i`

```
sed 's/PROD/produkcja/gi' serwer.conf
```

Przykład:

```
tags=produkcja,produkcja,web
```

Wzorzec `PROD` pisany wielkimi literami dopasował mimo to małe „prod" w pliku — flaga `i` (rozszerzenie GNU) każe ignorować wielkość liter przy dopasowaniu. Bez niej ten sam skrypt nie zmieniłby nic, bo `PROD` i `prod` to dla `sed` domyślnie dwa różne ciągi znaków.

sed = stream editor: edytor strumienia tekstu

TYLKO ODCZYT

Wybór konkretnego wystąpienia w linii

```
sed 's/prod/PROD/2' serwer.conf
```

Przykład:

```
tags=prod,PROD,web
```

Liczba po ukośniku wybiera, KTÓRE z kolei wystąpienie w linii ma zostać zmienione — tu drugie, nie pierwsze. Da się to łączyć z `g`: `2g` zmienia od drugiego wystąpienia w dół, do końca linii.

sed = stream editor: edytor strumienia tekstu

TYLKO ODCZYT

Wypisanie zmienionej linii: `p`

```
sed -n 's/error/BŁĄD/p' serwer.conf
```

Przykład:

```
log_level=BŁĄD
```

`p` doklejone do `s///` wypisuje linię, w której podstawienie faktycznie zaszło — ale tylko razem z `-n`, inaczej ta sama linia wyszłaby dwa razy: raz przez `p`, raz przez automatyczny druk.

sed = stream editor: edytor strumienia tekstu

TYLKO ODCZYT

Zapisanie zmienionych linii do pliku: `w`

```
sed -n 's/error/BŁĄD/w zmiany.txt' serwer.conf
cat zmiany.txt
```

Przykład:

```
log_level=BŁĄD
```

Zamiast wypisywać na ekran, zmieniona linia trafia do wskazanego pliku. Przydaje się do zbierania „dziennika zmian" przy masowej edycji wielu plików naraz.

sed = stream editor: edytor strumienia tekstu · **cat** = concatenate: sklej i wypisz pliki

TYLKO ODCZYT

Adres numeryczny

```
sed '6d' serwer.conf
```

Adres przed poleceniem mówi, KTÓRYCH linii ono dotyczy — tu wyłącznie linii 6, komentarza z zapasowym adresem IP.

Przykład:

```
# --- START bloku app01 ---
host=app01
ip=10.20.10.41
port=8080
docroot=/var/www/html
timeout=30
```

IP = Internet Protocol: adres komputera w sieci; w elektryce: stopień ochrony obudowy (Ingress Protection) · **sed** = stream editor: edytor strumienia tekstu

TYLKO ODCZYT

Zakres linii

```
sed '9,11d' serwer.conf
```

Zakres `9,11` obejmuje linie od 9 do 11 włącznie — tu `max_connections`, `tags` i `retries`.

Przykład:

```
# --- START bloku app01 ---
host=app01
ip=10.20.10.41
port=8080
docroot=/var/www/html
# backup_ip=10.20.10.42
timeout=30

log_level=error
# --- END bloku app01 ---
status=aktywny
```

sed = stream editor: edytor strumienia tekstu

TYLKO ODCZYT

Ostatnia linia: `$`

```
sed '$d' serwer.conf
```

`$` to adres specjalny: zawsze ostatnia linia pliku, niezależnie od tego, ile ich jest. Usuwamy `status=aktywny`, nie ruszając reszty.

Przykład:

```
# --- END bloku app01 ---
```

sed = stream editor: edytor strumienia tekstu

TYLKO ODCZYT

Adres wzorcowy: `/regex/`

```
sed -n '/^#/p' serwer.conf
```

Zamiast numeru linii, adresem może być wyrażenie regularne w ukośnikach — pasuje do każdej linii, która je zawiera.

Przykład:

```
# --- START bloku app01 ---
# backup_ip=10.20.10.42
# --- END bloku app01 ---
```

□□ **sed** = stream editor: edytor strumienia tekstu

□□ TYLKO ODCZYT

Zakres regexowy

```
sed -n '/^# --- START/,/^# --- END/p' serwer.conf
```

Zakres może zaczynać się i kończyć wzorcem zamiast numeru: włącza się na pierwszej linii pasującej do wzorca początkowego, wyłącza na pierwszej linii (po nim) pasującej do wzorca końcowego, WŁĄCZNIE z obiema.

Przykład:

```
# --- START bloku app01 ---
host=app01
ip=10.20.10.41
port=8080
docroot=/var/www/html
# backup_ip=10.20.10.42
timeout=30

max_connections=100
tags=prod,prod,web
retries=3
log_level=error
# --- END bloku app01 ---
```

□□ **sed** = stream editor: edytor strumienia tekstu

□□ TYLKO ODCZYT

Negacja adresu: `!`

```
sed -n '/^#/!p' serwer.conf
```

Wykrzyknik po adresie odwraca dopasowanie: wypisujemy wszystko, co NIE pasuje do `/^#/`, czyli wszystkie linie oprócz komentarzy.

Przykład:

```
host=app01
ip=10.20.10.41
port=8080
docroot=/var/www/html
timeout=30

max_connections=100
tags=prod,prod,web
retries=3
log_level=error
status=aktywny
```

sed = stream editor: edytor strumienia tekstu

TYLKO ODCZYT

Co N-ta linia: `0~3`

```
sed -n '0~3p' serwer.conf
```

Rozszerzenie GNU: `pierwsza~krok` wybiera co `krok`-tą linię, zaczynając od `pierwsza`. Tu: linie 3, 6, 9, 12.

Przykład:

```
ip=10.20.10.41
# backup_ip=10.20.10.42
max_connections=100
log_level=error
```

sed = stream editor: edytor strumienia tekstu

TYLKO ODCZYT

Usunięcie linii: `d`

```
sed '/^$/d' serwer.conf
```

`d` usuwa z przetwarzania linię pasującą do adresu — tu pustą linię między `timeout` a `max_connections` — i od razu przechodzi do kolejnego cyklu, bez wypisywania.

Przykład:

```
timeout=30
max_connections=100
```

sed = stream editor: edytor strumienia tekstu

TYLKO ODCZYT

Wypisanie zakresu linii: `-n` i `p`

```
sed -n '9,11p' serwer.conf
```

Ta sama para co przy adresie wzorcowym, tylko z zakresem numerycznym — wypisujemy wyłącznie linie 9-11, nic więcej.

Przykład:

```
max_connections=100
tags=prod,prod,web
retries=3
```

sed = stream editor: edytor strumienia tekstu

TYLKO ODCZYT

Dopisanie linii po: `a\`

```
sed '5a\
webroot_owner=www-data' serwer.conf
```

Za pomocą `a` dopisujemy podany tekst PO wskazanej linii — tu po linii 5 (`docroot=...`).

Przykład:

```
docroot=/var/www/html
webroot_owner=www-data
# backup_ip=10.20.10.42
```

sed = stream editor: edytor strumienia tekstu

TYLKO ODCZYT

Wstawienie linii przed: `i\`

```
sed '2i\
# wygenerowano automatycznie' serwer.conf
```

Poleceniem `i` robimy to samo co `a`, ale wstawiamy tekst PRZED wskazaną linią.

Przykład:

```
# --- START bloku app01 ---
# wygenerowano automatycznie
host=app01
```

sed = stream editor: edytor strumienia tekstu

TYLKO ODCZYT

Zamiana linii lub zakresu: `c\`

```
sed '9,11c\
# sekcja limitów usunięta' serwer.conf
```

Poleceniem `c` zastępujemy CAŁY wskazany zakres jednym podanym tekstem — niezależnie od tego, ile linii ten zakres obejmował, w wyniku zostaje dokładnie jedna.

Przykład:

```
docroot=/var/www/html
# backup_ip=10.20.10.42
timeout=30
```

```
# sekcja limitów usunięta
log_level=error
```

sed = stream editor: edytor strumienia tekstu

TYLKO ODCZYT

Transliteracja znak po znaku: `y///`

```
sed 'y/,/;/' serwer.conf
```

`y` zamienia znaki jeden na jeden, jak `tr` — pierwszy znak pierwszego zbioru na pierwszy znak drugiego, drugi na drugi, i tak dalej. W przeciwieństwie do `s///` nie rozumie wzorców, tylko pojedyncze znaki.

Przykład:

```
tags=prod;prod;web
```

sed = stream editor: edytor strumienia tekstu

TYLKO ODCZYT

Zakończenie po N liniach: `q`

```
sed '5q' serwer.conf
```

`q` przerywa całe przetwarzanie po wskazanej linii — działa jak `head -n 5`, tylko wewnątrz `sed`, więc można to połączyć z innymi poleceniami w tym samym skrypcie.

Przykład:

```
# --- START bloku app01 ---
host=app01
ip=10.20.10.41
port=8080
docroot=/var/www/html
```

sed = stream editor: edytor strumienia tekstu

◫◫ ZMIENIA SYSTEM

Wczytanie zawartości pliku: `r`

```
printf '# wygenerowano automatycznie, nie edytuj recznie\n' > stopka.txt
sed '$r stopka.txt' serwer.conf
```

Poleceniem `r` wstawiamy całą zawartość WSKAZANEGO PLIKU po danej linii — tu po ostatniej (`$`), czyli jako stopkę.

Przykład:

```
# --- END bloku app01 ---
status=aktywny
# wygenerowano automatycznie, nie edytuj recznie
```

◫◫ **printf** = wypisz sformatowany tekst, ściślejsza odmiana echo · **sed** = stream editor: edytor strumienia tekstu

◫◫ TYLKO ODCZYT

Zapisanie pasujących linii do pliku: `w`

```
sed -n '/^#/w komentarze.txt' serwer.conf
cat komentarze.txt
```

Jako samodzielne polecenie (nie flaga `s///`) zapisujemy nim do pliku każdą linię pasującą do adresu — tu wszystkie komentarze.

Przykład:

```
# --- START bloku app01 ---
# backup_ip=10.20.10.42
# --- END bloku app01 ---
```

◫◫ **sed** = stream editor: edytor strumienia tekstu · **cat** = concatenate: sklej i wypisz pliki

◫◫ TYLKO ODCZYT

Inny separator niż `/`

```
sed 's#/var/www/html#/srv/www#' serwer.conf
```

Separator `s///` nie musi być ukośnikiem — może być dowolnym znakiem nieużywanym we wzorcu. Przy ścieżkach plikowych `#` oszczędza ucieczek w stylu `\/var\/www\/html`.

Przykład:

```
docroot=/srv/www
```

sed = stream editor: edytor strumienia tekstu

TYLKO ODCZYT

Przechwytywanie grup: `\1`

```
echo "Kowalski Jan" | sed -E 's/(\w+) (\w+)/\2 \1/'
```

Przykład:

```
Jan Kowalski
```

Nawiasy grupujące zapamiętują dopasowany fragment, żeby użyć go w części zamieniającej jako `\1`, `\2` itd. — tu zamiana miejscami nazwiska i imienia. W trybie domyślnym BRE nawiasy trzeba escapować: `\(...\)`.

BRE = Basic Regular Expression: podstawowa, starsza odmiana wyrażeń regularnych, wymaga ukośnika wstecznego przed nawiasami

TYLKO ODCZYT

Odwołanie do całego dopasowania: `&`

```
sed 's/error/[&]/' serwer.conf
```

Przykład:

```
log_level=[error]
```

`&` w części zamieniającej wstawia dokładnie to, co dopasował wzorzec — przydatne, gdy chcemy coś dopasowanego owinąć, bez przepisywania go w osobnej grupie.

sed = stream editor: edytor strumienia tekstu

TYLKO ODCZYT

Włączenie rozszerzonych wyrażeń regularnych: `-E`

```
sed -E 's/([0-9]{1,3}\.){3}[0-9]{1,3}/10.20.10.50/' serwer.conf
```

Przykład:

```
ip=10.20.10.50
```

`-E` włącza ERE (Extended Regular Expressions): `{ }` i `( )` działają jako metaznaki bez ucieczki, tak jak w większości innych narzędzi. Bez `-E` ten sam wzorzec trzeba by zapisać z `\{` i `\(`.

ERE = Extended Regular Expression: rozszerzona odmiana wyrażeń regularnych, bez ukośnika wstecznego przed nawiasami i plusem · **sed** = stream editor: edytor strumienia tekstu

ZMIENIA SYSTEM

Edycja w miejscu: `-i`

```
sed -i 's/app01/docker01/g' serwer.conf
cat serwer.conf
```

`-i` zapisuje wynik z powrotem do pliku wejściowego, zamiast wypisywać na ekran — bez tego trzeba by przekierować do pliku tymczasowego i podmienić ręcznie.

Przykład:

```
# --- START bloku docker01 ---
host=docker01
ip=10.20.10.41
port=8080
docroot=/var/www/html
# backup_ip=10.20.10.42
timeout=30

max_connections=100
tags=prod,prod,web
retries=3
log_level=error
```

```
# --- END bloku docker01 ---
status=aktywny
```

sed = stream editor: edytor strumienia tekstu · **cat** = concatenate: sklej i wypisz pliki

ZMIENIA SYSTEM

Kopia zapasowa przy edycji w miejscu: `-i.bak`

```
sed -i.bak 's/8080/8443/' serwer.conf
ls serwer.conf*
```

Dopisany sufiks każe zachować oryginał pod tą nazwą, zanim plik zostanie nadpisany — jedyna siatka bezpieczeństwa przy edycji w miejscu.

Przykład:

```
serwer.conf
serwer.conf.bak
```

sed = stream editor: edytor strumienia tekstu · **ls** = list: wypisz zawartość katalogu

TYLKO ODCZYT

Kilka poleceń w jednym wywołaniu: `-e`

```
sed -e 's/docker01/ci01/g' -e 's/8443/9443/' serwer.conf
```

Każde `-e` dodaje kolejne polecenie do tego samego skryptu — wykonywane po kolei, na tej samej przestrzeni roboczej. Równoważne jednemu `-e` ze średnikami: `sed 's/docker01/ci01/g; s/8443/9443/' serwer.conf`.

Przykład:

```
host=ci01
port=9443
```

sed = stream editor: edytor strumienia tekstu

TYLKO ODCZYT

Hold space — pamięć między liniami

```
sed -n '1!G;h;$p' serwer.conf
```

Klasyczny jednolinowiec odwracający kolejność linii, bez `tac`. Rozbicie: dla każdej linii poza pierwszą (`1!`) dokleja zawartość hold space PRZED bieżącą linią (`G`), zaraz potem `h` kopiuje ten rosnący stos z powrotem do hold space na następny obieg, a na końcu (`$`) `p` wypisuje cały nagromadzony stos.

Przykład:

```
status=aktywny
# --- END bloku app01 ---
log_level=error
retries=3
tags=prod,prod,web
max_connections=100

timeout=30
# backup_ip=10.20.10.42
docroot=/var/www/html
port=8080
ip=10.20.10.41
host=app01
# --- START bloku app01 ---
```

sed = stream editor: edytor strumienia tekstu

TYLKO ODCZYT

Wszystkie flagi i polecenia

Element	Co robi	Kiedy używasz
`s/wz/tekst/g`	zamienia wszystkie wystąpienia w linii	domyślnie zamienia tylko pierwsze
`s///i`	dopasowanie bez rozróżniania wielkości liter	dane o niespójnej pisowni
`s///N`	zamienia N-te wystąpienie w linii	precyzyjna zamiana, gdy wzorzec powtarza się w linii

Element	Co robi	Kiedy używasz
`s///p`	wypisuje linię, w której zaszło podstawienie	zawsze razem z `-n`
`s///w plik`	zapisuje zmienione linie do pliku	zbieranie dziennika zmian
`-n` + `p`	wypisuje TYLKO to, co jawnie zażądasz	wycinanie fragmentu, jak `grep`
`d`	usuwa całą linię	filtrowanie, usuwanie komentarzy/pustych linii
`a\`	dopisuje tekst po linii	wstrzykiwanie nowych wpisów
`i\`	wstawia tekst przed linią	nagłówki, komentarze
`c\`	zastępuje zakres jednym tekstem	wycięcie sekcji i podmiana
`y///`	transliteracja znak po znaku	zamiana pojedynczych znaków
`q`	kończy przetwarzanie po N liniach	odpowiednik `head` wewnątrz skryptu
`r plik`	wstawia zawartość pliku po linii	doklejanie szablonów, stopek
`w plik`	zapisuje pasujące linie do pliku	filtrowanie do osobnego pliku
`!`	neguje adres	„wszystko oprócz"
`0~3`	co N-ta linia (GNU)	próbkowanie, linie parzyste/nieparzyste
`-i`	edytuje plik w miejscu	zmiana configów bez `> plik`
`-i.bak`	edycja w miejscu z kopią zapasową	bezpieczniejsza wersja `-i`
`-E`	włącza rozszerzone regexy (ERE)	nawiasy grup bez `\(` `\)`
`-e`	dodaje kolejne polecenie do skryptu	kilka niezależnych zmian naraz
`\1`, `\2`, `&`	odwołanie do grupy / całego dopasowania	przestawianie fragmentów tekstu
`h H g G x`	operacje na hold space	porównania między liniami, odwracanie

Uważaj

- `sed -i 's/x/y/' plik` na BSD/macOS bez pustego argumentu po `-i` zepsuje polecenie po cichu — patrz *Częste błędy*.
- `p` bez `-n` wypisze linię **dwa razy**: raz przez `p`, raz przez automatyczny druk na końcu cyklu.

- Ukośnik `/` w ścieżce trzeba escapować (`\/`) albo — prościej — zmienić separator na `#` czy `|`.

ERE = Extended Regular Expression: rozszerzona odmiana wyrażeń regularnych, bez ukośnika wstecznego przed nawiasami i plusem · **BSD** = Berkeley Software Distribution: rodzina Uniksów z uczelni Berkeley, inna gałąź niż Linux, stąd czasem inna składnia tych samych poleceń

Z życia wzięte

Z ŻYCIA WZIĘTE

O 7:50 Tomek dostaje zadanie: zmienić adres starego serwera NTP w plikach konfiguracyjnych dwudziestu kontenerów na docker01. Jedną komendą sed -i uruchamia podmianę we wszystkich plikach naraz, ale w pośpiechu nie dodaje flagi g tam, gdzie adres powtarzał się dwa razy w tej samej linii. Po dziesiątej Ola zgłasza, że część aplikacji nadal loguje błędy synchronizacji czasu, bo w ich configu stary adres został na drugiej pozycji.

Po co to istnieje

Masz 40 plików konfiguracyjnych, w każdym trzeba zmienić adres serwera NTP. Otwieranie każdego w edytorze i ręczna zmiana to piętnaście minut roboty i piętnaście okazji do literówki. `sed` robi to samo w jednej linii poleceń, w ułamku sekundy, dla jednego pliku albo dla wszystkich naraz:

```
sed -i 's/dc01\.ad\.zielona\.example/dc02.ad.zielona.example/g' /etc/chrony.conf
```

Drugi powód, dla którego `sed` istnieje: skrypty. Kod uruchamiany bez człowieka przy klawiaturze — w automatyzacji wdrożeń, w cronie, w pipeline CI — nie może otworzyć `vim`-a i czekać na decyzję. Potrzebuje narzędzia, które przyjmie instrukcję i wykona ją bezobsługowo. `sed` powstał do tego w 1974 roku i do dziś jest pierwszym wyborem tam, gdzie zmiana jest prosta i mechaniczna: podmień, usuń, wstaw.

Analogia

Wyobraź sobie **taśmę produkcyjną z jednym stanowiskiem kontrolnym**.

Przez stanowisko przejeżdża jeden element naraz. Pracownik przy stanowisku ma przed sobą kartkę z instrukcją — „jeśli na elemencie jest rysa, odrzuć go", „jeśli brakuje śrubki, dokręć" — i

widzi **tylko to, co akurat leży przed nim**. Nie widzi elementu, który przejechał minutę temu, ani tego, który dopiero jedzie. Skończy z jednym, taśma rusza, przyjeżdża następny.

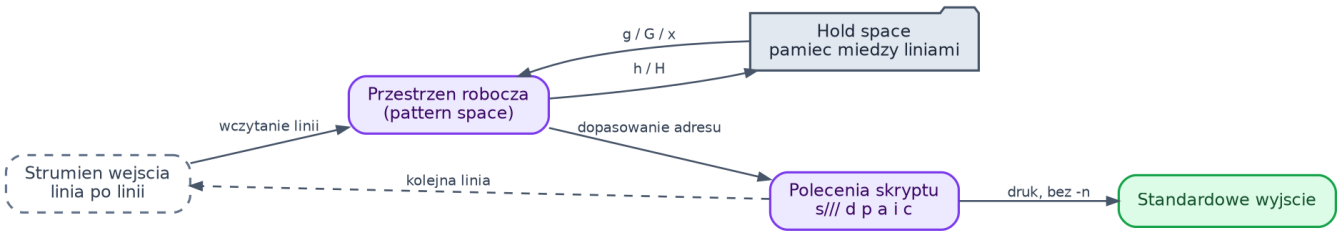
To jest dokładnie model pracy `sed`: wczytaj jedną linię do przestrzeni roboczej, wykonaj na niej skrypt, wypisz wynik, weź kolejną linię. `sed` nie ładuje całego pliku do pamięci i nie „wie", co jest trzy linie dalej — dlatego polecenie w rodzaju „usuń akapit, jeśli linia dwie niżej zawiera słowo X" nie jest tu trywialne. Trzeba by pracownikowi dać dodatkowe polecenie: *zanim odłożysz element, zapisz sobie na boku, jaki miał numer* — i to jest dokładnie to, do czego służy hold space: półka obok stanowiska, na której pracownik może odłożyć coś, do czego wróci przy kolejnym elemencie. Sam element z taśmy (bieżąca linia, pattern space) i to, co leży na półce (hold space, zapamiętane wcześniej) to dwa oddzielne miejsca — i większość zaawansowanych sztuczek w `sed` polega na przekładaniu zawartości między nimi.

Ograniczenie tej analogii jest jednocześnie ograniczeniem `sed`: pracownik nie cofnie taśmy. Jeśli trzeba porównać bieżący element z poprzednim, musisz to zaplanować z wyprzedzeniem (odłożyć poprzedni na półkę), bo `sed` nie da Ci wrócić do linii, która już przejechała.

Jak to działa

Rozwiń: jak to działa pod spodem

Cykl `sed` dla każdej linii wejścia wygląda tak:



Sed wczytuje linie do przestrzeni roboczej, wykonuje na niej polecenia skryptu i wypisuje wynik, a hold space trzyma dane między liniami (kliknięcie otwiera pełny rozmiar)

1. Wczytaj kolejną linię do przestrzeni roboczej, bez znaku końca linii.
2. Wykonaj po kolei polecenia ze skryptu, które pasują do bieżącego adresu (numer linii, zakres, wzorzec).
3. Jeśli nie użyto `-n` — wypisz zawartość przestrzeni roboczej na standardowe wyjście.
4. Wróć do kroku 1, dopóki są linie wejścia.

Kluczowa konsekwencja: `sed` **nigdy nie widzi całego pliku naraz**. Widzi jedną linię (dwie, jeśli użyjesz `N`, które dokleja kolejną linię do bieżącej przestrzeni roboczej). To dlatego `sed` radzi sobie świetnie z plikami wielogigabajtowymi bez zająknięcia — nie ma czego trzymać w pamięci poza bieżącą linią — i dlatego zadania typu „zamień coś, co rozciąga się na trzy linie" wymagają jawnego sklejenia tych linii (`N;N`) albo zabawy z hold space.

Adresy decydują, dla których linii polecenie w ogóle się wykona. Możesz podać:

- numer linii: `5`
- ostatnią linię: `$`
- zakres: `10,20`
- zakres do końca: `10,$`
- wzorzec: `/error/`
- zakres między wzorcami: `/START/,/END/` — włącza się na linii pasującej do `START`, wyłącza na pierwszej linii pasującej do `END` (włącznie z nią)
- co N-tą linię (rozszerzenie GNU): `pierwsza~krok`, np. `0~3` to linie 3, 6, 9…, a `1~2` to linie nieparzyste
- negację dowolnego z powyższych: dopisz `!` — `/^#/!d` znaczy „usuń wszystko, co NIE jest komentarzem"

Flaga `-n` wyłącza automatyczny druk z kroku 3. Jest obowiązkowa zawsze, gdy chcesz decydować sama/sam, co się wypisze — typowo w parze z `p`. Bez `-n` polecenie `p`, które jawnie wypiszesz, wyjdzie **drugi raz** przy automatycznym druku.

Wiele poleceń można podać na trzy sposoby: kilka flag `-e`, jedno polecenie ze średnikami, albo nawiasy klamrowe `{ }`, które grupują polecenia pod jednym adresem — przydatne, gdy dwie albo trzy operacje mają dotyczyć tych samych linii:

```
sed '/^#/{s/#//; s/$/  # aktywne/}' plik.conf
```

Krok po kroku

1. Podmień prostą wartość, ostrożnie z kropkami.

Kropka w regexie oznacza „dowolny znak", nie dosłowną kropkę — więc przy adresach IP trzeba ją escapować:

```
sed 's/10\.20\.10\.41/10.20.10.50/' serwer.conf
```

```
ip=10.20.10.50
```

Bez `\.` wyrażenie `10.20.10.41` dopasowałoby też np. `10a20a10a41` — bo każda kropka znaczy „cokolwiek". `g` na końcu ma znaczenie tylko wtedy, gdy w jednej linii adres mógłby się powtórzyć więcej niż raz.

2. Zmień separator, żeby nie walczyć ze ścieżkami.

Ten skrypt jest już pokazany w Fiszce, w sekcji „Inny separator niż `/`" — separator `s///` nie musi być ukośnikiem, przy ścieżkach `#` albo `|` oszczędza ucieczek w stylu `\/var\/www\/html`.

3. Zakomentuj i odkomentuj zakres linii.

```
sed '9,11s/^/#/' serwer.conf      # dodaj # na początku każdej z linii 9-11
sed '9,11s/^#//' serwer.conf      # usuń pierwszy # z początku linii 9-11
```

```
#max_connections=100
#tags=prod,prod,web
#retries=3
```

`^` to adres wewnątrz linii — początek. Wstawiasz tam `#` albo, w drugą stronę, dopasowujesz `#` na początku i zamieniasz na nic (`//` — pusty tekst po drugim separatorze).

4. Usuń puste linie.

Ten skrypt jest już pokazany w Fiszce, w sekcji „`d` — usuwanie linii": `/^$/` to wzorzec „początek linii natychmiast po którym jest koniec linii", czyli linia bez żadnego znaku.

5. Przestaw fragmenty tekstu grupami przechwytywania.

Skrypt i wynik są pokazane w Fiszce, w sekcji „Grupy przechwytywania `\1`". `\( ... \)` w trybie BRE (domyślnym dla `sed`) albo `( ... )` z `-E` zapamiętują dopasowany fragment do odwołania jako `\1`, `\2` — w ERE nawiasy grupujące nie wymagają `\`, za to dosłowny nawias trzeba wtedy escapować, dokładnie odwrotnie niż w BRE. Dokładniejsze wyjaśnienie BRE/ERE/PCRE i tego, kiedy który tryb działa, jest na stronie o wyrażeniach regularnych.

6. Zamień adres IP dowolnym wyrażeniem, nie tylko dosłownym.

Skrypt i wynik pokazane w Fiszce, w sekcji „`-E` — rozszerzone wyrażenia regularne". Rozbite na części: `[0-9]{1,3}` to od jednej do trzech cyfr, `\.` to kropka, całość w grupie powtórzona `{3}` razy (trzy oktety zakończone kropką), a na końcu jeszcze jeden blok cyfr bez kropki. Pasuje do każdego poprawnie zapisanego IPv4 (nie sprawdza, czy liczby mieszczą się w zakresie 0-255 — `999.999.999.999` też by przeszło).

7. Użyj `&`, żeby odwołać się do całego dopasowania.

Skrypt i wynik pokazane w Fiszce, w sekcji „`&` — cała dopasowana treść" — wstawia dokładnie to, co dopasował wzorzec, przydatne, gdy chcesz coś dopasowanego owinąć, bez przepisywania go w

grupie.

8. Wytnij fragment pliku, tak jak `grep`, ale z kontrolą nad wypisywaniem.

```
sed -n '/^# --- START/,/^# --- END/p' serwer.conf
```

Wynik identyczny jak w Fiszce, w sekcji „Zakres regexowy". Bez `-n` ta komenda wypisałaby cały plik (bo `p` dopisuje drugi raz to, co i tak wyszłoby automatycznie), a nie tylko wybrany zakres.

9. Edytuj plik w miejscu — i zrozum różnicę Linux/BSD.

```
sed -i 's/stary/nowy/g' serwer.conf             # GNU/Linux — bez kopii zapasowej
sed -i.bak 's/stary/nowy/g' serwer.conf         # GNU/Linux — z kopią serwer.conf.bak
sed -i '' 's/stary/nowy/g' serwer.conf          # BSD/macOS — pusty argument obowiązkowy
```

Pierwsze dwie formy są już pokazane w Fiszce. Trzecia — dla BSD/macOS — nie ma tam odpowiednika, bo reszta strony zakłada GNU sed z Linuksa. Więcej o tej pułapce w sekcji *Częste błędy* — to jeden z najczęściej powielanych błędów w skryptach testowanych tylko na jednym systemie.

10. Odwróć kolejność linii pliku, bez `tac`.

Skrypt i wynik pokazane w Fiszce, w sekcji „Hold space — pamięć między liniami".

Częste błędy

Zapomniany `-n` przy `p` — każda linia na ekranie dwa razy.

```
sed '/error/p' plik.log
```

```
2026-09-17 12:03:01 error: connection refused
2026-09-17 12:03:01 error: connection refused
2026-09-17 12:04:12 info: connected
```

Linie z `error` wyszły dwukrotnie: raz przez jawne `p`, raz przez automatyczny druk na końcu cyklu, którego nikt nie wyłączył. Linie bez `error` wyszły raz — bo do nich `p` się nie odnosi. Rozwiązanie: dopisz `-n`.

```
sed -n '/error/p' plik.log
```

`-i` na BSD/macOS bez argumentu — po cichu zjada pierwszy plik.

Skrypt przetestowany na Linuksie (`sed -i 's/stary/nowy/g' plik.conf`) na macOS (BSD sed) nie zgłosi zrozumiałego błędu składni — potraktuje `'s/stary/nowy/g'` jako **argument backupu** dla `-i`, a `plik.conf` jako skrypt do wykonania, co zwykle kończy się komunikatem w stylu `sed: 1: "plik.conf": unknown command`. Poprawnie na BSD/macOS trzeba podać pusty ciąg jako osobny argument:

```
sed -i '' 's/stary/nowy/g' plik.conf
```

Jeśli piszesz skrypt, który ma działać wszędzie, sprawdź wersję sed albo unikaj `-i` na rzecz jawnego pliku tymczasowego (`sed 's/x/y/' plik > plik.tmp && mv plik.tmp plik`), co działa identycznie na obu systemach.

Niezaescapowany ukośnik w ścieżce.

```
sed 's/\var\www\html/\srv\www/g' plik.conf
```

Wygląda niewinnie, ale `sed` zgłosi coś w rodzaju `sed: -e expression #1, char 6: unknown option to 's'` — bo tu `/` w ogóle nie pełni roli separatora pól, tylko literału w środku wzorca, i to bez ucieczki. Prawidłowo trzeba albo escapować każdy `/`:

```
sed 's/\/var\/www\/html/\/srv\/www/g' plik.conf
```

albo, znacznie czytelniej, zmienić separator, jak w Fiszce:

```
sed 's#/var/www/html#/srv/www#g' plik.conf
```

> **Pojęcie: inode.** Wewnętrzny numer, którym system plików identyfikuje konkretny plik na dysku — jego dane, uprawnienia, właściciela — niezależnie od nazwy, pod jaką jest widoczny w katalogu. Jedna nazwa pliku zawsze wskazuje na dokładnie jeden inode, ale ten sam inode może mieć wiele nazw naraz (twarde dowiązania), a program, który otworzył plik, trzyma odniesienie do inode, nie do nazwy.

`-i` zmienia inode pliku — łamie `tail -f` i twarde dowiązania.

Masz otwartą sesję z `tail -f /var/log/aplikacja.log`, a w drugim terminalu robisz na nim `sed -i` (np. maskowanie danych osobowych przed wysłaniem do audytu). `sed -i` nie modyfikuje pliku bajt po bajcie — tworzy nowy plik tymczasowy z wynikiem, a potem **podmienia go pod starą nazwą** (rename). To oznacza nowy inode. `tail -f` trzyma otwarty deskryptor do starego inode i nie zobaczy już nic, bo nowe dane trafiają pod tę samą nazwę, ale do zupełnie innego pliku na dysku. Rozwiązanie: `tail -F`, które śledzi nazwę, a nie deskryptor. Ten sam mechanizm łamie twarde dowiązania — drugie dowiązanie do tego samego inode dalej wskazuje na starą, nieedytowaną treść.

Podwójny cudzysłów zamiast pojedynczego wokół skryptu ze zmienną powłoki.

```
wzorzec="error"
sed "s/$wzorzec/OSTRZEŻENIE/g" plik.log
```

To akurat zadziała — ale tylko dlatego, że `$wzorzec` nie zawiera znaków specjalnych dla `sed` ani dla powłoki. Problem pojawia się, gdy wartość zmiennej zawiera ukośnik, ampersand albo cudzysłów — wtedy podwójny cudzysłów pozwala powłoce podstawić wartość *przed* uruchomieniem `sed`, i jeśli ta wartość zawiera np. `/`, rozjedzie Ci separator skryptu. Pełne wyjaśnienie różnicy między `'...'` a `"..."` i tego, kiedy powłoka wykonuje podstawienia, jest na stronie *Cudzysłowy, escapowanie i ekspansja*.

□□ Ćwiczenia

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

Ćwiczenie 1 — bezpieczna edycja konfiguracji

```
cd /tmp
cat > dane.conf <<'EOF'
# konfiguracja testowa
server_ip=10.20.10.41
port=8080
# backup_ip=10.20.10.42
timeout=30

max_connections=100
EOF
```

Zadania:

1. Wypisz (bez modyfikowania pliku) same linie, które NIE są komentarzami i nie są puste.
2. Zamień `10.20.10.41` na `10.20.10.50` — ale nie ruszaj zakomentowanej linii z `10.20.10.42`.
3. Usuń wszystkie puste linie, edytując plik w miejscu, z kopią zapasową `.bak`.

Odpowiedź

```
# 1. linie „żywe" — nie komentarz i nie pusta
sed -n '/^#/!{/^$/!p}' dane.conf

# 2. zamiana tylko aktywnej linii (adres /^server_ip/ wybiera konkretną linię)
sed '/^server_ip=/s/10\.20\.10\.41/10.20.10.50/' dane.conf

# 3. usuwanie pustych linii z kopią zapasową
sed -i.bak '/^$/d' dane.conf
diff dane.conf.bak dane.conf
```

Zadanie 1 łączy dwie negacje w zagnieżdżonym bloku: „jeśli linia NIE zaczyna się od `#`, to w środku: jeśli NIE jest pusta, wypisz". Bez `-n` zobaczyłbyś cały plik plus duplikaty wybranych linii.

Pytania kontrolne

1. Dlaczego `sed 's/error/BŁĄD/' plik.log` może nie zmienić linii, w której `error` występuje dwukrotnie, w oczekiwany sposób?

Odpowiedź

Bo bez flagi `g`, `s///` zamienia tylko **pierwsze** wystąpienie wzorca w każdej linii. Drugie `error` w tej samej linii zostanie nietknięte. Trzeba dopisać `g`: `sed 's/error/BŁĄD/g' plik.log`.

2. Masz skrypt z `sed -i 's/a/b/' plik`, który działał na serwerze produkcyjnym (Rocky Linux), a przestał działać na laptopie kolegi z macOS. Dlaczego i jak to naprawić przenośnie?

Odpowiedź

GNU sed (Linux) akceptuje `-i` bez argumentu jako „edytuj w miejscu bez kopii". BSD sed (macOS) wymaga po `-i` argumentu — nawet pustego: `-i ''`. Bez niego BSD sed potraktuje kolejny argument (samo polecenie `s/a/b/`) jako sufiks backupu, a nazwę pliku jako skrypt do wykonania, co się wysypie. Przenośne rozwiązanie: albo rozróżnić systemy w skrypcie (`sed -i.bak` vs `sed -i ''`), albo ominąć `-i` całkowicie i użyć pliku tymczasowego z `>` i `mv`.

3. Czym różni się `sed -n '5,10p' plik` od `sed '5,10d' plik`?

Odpowiedź

Pierwsze wypisuje WYŁĄCZNIE linie 5-10 (dzięki `-n`, które wyłącza automatyczny druk pozostałych linii). Drugie wypisuje CAŁY plik OPRÓCZ linii 5-10 (bo `d` usuwa te linie z przetwarzania, a reszta pliku wychodzi normalnie przez automatyczny druk). Efektywnie są to swoje dopełnienia.

4. Do czego służy hold space i dlaczego zwykły pattern space nie wystarcza do napisania odpowiednika `tac`?

Odpowiedź

Pattern space zawiera tylko bieżącą linię — jest nadpisywane przy każdym nowym cyklu, więc nic z poprzednich linii nie przetrwa do następnego obiegu. Żeby odwrócić kolejność linii, trzeba gdzieś trzymać już zobaczone linie między cyklami — do tego służy hold space, osobny bufor, który nie jest czyszczony automatycznie. Polecenia `h`/`H` kopiują/dopisują z pattern space do hold space, `g`/`G` w drugą stronę, a `x` je zamienia miejscami.

Powiązane: awk: język do danych w kolumnach · Wyrażenia regularne: BRE, ERE, PCRE · Cudzysłowy, escapowanie i ekspansja · Obserwacja plików na żywo: tail -f, less +F, watch · grep i rodzina: grep, egrep, fgrep, rekurencja

Źródła: `man 1 sed` · `info sed` · GNU sed manual (`sed.gnu.org`) · POSIX.1-2024, `sed`

Revision #19
Created 2026-09-17 15:06:59 UTC by Tomasz Konieczny
Updated 2026-10-03 19:07:58 UTC by Tomasz Konieczny