sort i uniq: porządkowanie i zliczanie

Poziom: podstawy **Dotyczy: wszystkie dystrybucje** **Czas: ~15 min**

`sort` porządkuje linie tekstu, `uniq` usuwa z nich duplikaty — ale tylko sąsiednie. Razem tworzą jeden z najczęstszych idiomów powłoki: `sort | uniq -c | sort -rn`.

Skróty użyte na tej stronie (3)

Skrót	Co znaczy
ASCII	American Standard Code for Information Interchange: podstawowy zestaw znaków (litery, cyfry) używany w komputerach
IP	Internet Protocol: adres komputera w sieci; w elektryce: stopień ochrony obudowy (Ingress Protection)
UID	User ID: numer użytkownika w Linuksie

Fiszka

TYLKO ODCZYT

Sortowanie domyślne

```
sort plik.txt
```

Bez flag `sort` porównuje linie **leksykograficznie**, bajt po bajcie, jak ciągi znaków, a nie jak liczby czy daty.

Przykład:

```
10
2
9
```

Zwracamy uwagę: `10` wyszło przed `9`, bo pierwszy znak `'1'` jest „mniejszy" niż `'9'` — więcej o tym niżej.

sort = posortuj linie tekstu alfabetycznie albo liczbowo

TYLKO ODCZYT

Sortowanie numeryczne: `-n`

```
sort -n plik.txt
```

Interpretujemy każdą linię jako liczbę przed porównaniem, więc `2` wychodzi przed `9`, a `9` przed `10`.

Przykład:

```
2
9
10
```

sort = posortuj linie tekstu alfabetycznie albo liczbowo

TYLKO ODCZYT

Sortowanie rozmiarów „human readable": `-h`

```
du -sh /var/log/* | sort -h
```

Rozumie sufiksy `K`, `M`, `G`, `T` i sortuje po rzeczywistej wielkości, a nie leksykograficznie — bez `-h` `"2G"` wyszłoby przed `"500M"`, bo `'2'` < `'5'`.

Przykład:

```
128K    /var/log/anaconda
4,0M    /var/log/audit
1,2G    /var/log/journal
```

du = disk usage: ile miejsca zajmuje katalog

TYLKO ODCZYT

Sortowanie wersji: `-V`

```
printf 'v1.2\nv1.10\nv2.0\n' | sort -V
```

„Version sort" radzi sobie z numerami wersji, gdzie zwykłe sortowanie tekstowe albo `-n` dają błędny porządek.

Przykład:

```
v1.2
v1.10
v2.0
```

Zwykłe `sort` bez `-V` ustawiłoby `v1.10` przed `v1.2`, bo porównuje znak `1` z `2` na trzeciej pozycji tekstu.

printf = wypisz sformatowany tekst, ściślejsza odmiana echo

TYLKO ODCZYT

Odwrócenie kolejności: `-r`

```
sort -r plik.txt
```

Odwracamy dowolny porządek sortowania — łączy się z innymi flagami, np. `sort -rn` sortuje numerycznie i malejąco.

sort = posortuj linie tekstu alfabetycznie albo liczbowo

TYLKO ODCZYT

Ograniczenie klucza do jednego pola: `-k2,2n`

```
sort -k2,2n plik.txt
```

Sortujemy wyłącznie według pola 2 (domyślny separator to biały znak), numerycznie. Zapis `-k2,2` jest kluczowy: ogranicza klucz **wyłącznie do pola 2** — bez `,2` na końcu klucz rozciągnąłby się od pola 2 do końca linii (patrz *Częste błędy*).

sort = posortuj linie tekstu alfabetycznie albo liczbowo

□□ ZMIENIA SYSTEM

Ustawienie separatora pól: `-t`

```
sort -t: -k3,3n /etc/passwd
```

Flagą `-t:` ustawiamy separator pól na dwukropek (tak jak `-d` w `cut`), zamiast domyślnego dowolnego ciągu białych znaków. Tu sortujemy `/etc/passwd` po polu 3, czyli UID, numerycznie.

Przykład:

```
root:x:0:0:root:/root:/bin/bash
adm_user:x:1000:1000::/home/adm_user:/bin/bash
nobody:x:65534:65534:Kernel Overflow User:/:/sbin/nologin
```

□□ **UID** = User ID: numer użytkownika w Linuksie · **sort** = posortuj linie tekstu alfabetycznie albo liczbowo

□□ TYLKO ODCZYT

Sortowanie i usunięcie duplikatów: `-u`

```
sort -u plik.txt
```

Sortujemy i od razu usuwamy zduplikowane całe linie w jednym przebiegu — działa to jak `sort | uniq`, ale bez uruchamiania drugiego procesu. Cena: kolejność wynikowa jest zawsze alfabetyczna, oryginalna kolejność pierwszego wystąpienia ginie.

□□ **sort** = posortuj linie tekstu alfabetycznie albo liczbowo

□□ TYLKO ODCZYT

Sortowanie stabilne: `-s`

```
sort -k2,2n -s plik.txt
```

Przy remisie na kluczu sortowania (dwie linie mają tę samą wartość pola) `sort` domyślnie sam decyduje o ich wzajemnej kolejności, zwykle porównując dalej całą linię. Flagą `-s` wyłączamy to dodatkowe porównanie i zachowujemy kolejność linii-remisów taką, w jakiej wystąpiły w pliku wejściowym.

□□ **sort** = posortuj linie tekstu alfabetycznie albo liczbowo

□□ TYLKO ODCZYT

Bezpieczny zapis w miejscu: `-o`

```
sort plik.txt -o plik.txt
```

Zapisujemy posortowany wynik z powrotem do tego samego pliku wejściowego, bezpiecznie — w przeciwieństwie do `sort plik.txt > plik.txt`, które obcina plik do zera, zanim `sort` zdąży go przeczytać (mechanizm opisany niżej i na stronie o `cat`).

□□ **sort** = posortuj linie tekstu alfabetycznie albo liczbowo

□□ TYLKO ODCZYT

Rekordy rozdzielone NUL: `-z`

```
find . -print0 | sort -z | xargs -0 ls -l
```

Zamiast znaku nowej linii `\n` jako separatora rekordów używamy bajtu NUL (`\0`). Konieczne przy nazwach plików ze spacjami czy znakami nowej linii w nazwie, dokładnie tak samo jak `find -print0` i `xargs -0`.

□□ **find** = przeszukaj drzewo katalogów według nazwy, typu, daty albo rozmiaru

□□ TYLKO ODCZYT

Sortowanie niezależne od locale: `LC_ALL=C sort`

```
LC_ALL=C sort lista_klientow.txt
```

Wymuszamy czysto bajtowe porównanie znaków (locale `C`), niezależne od tego, jakie locale ma zalogowany użytkownik czy zadanie crona. Bez tego ten sam plik posortowany w różnych środowiskach może dać różny porządek — mechanizm opisany niżej. Więcej o zmiennych `LC_*` na stronie *Zmienne środowiskowe i PATH*.

□□ TYLKO ODCZYT

Usunięcie sąsiednich duplikatów: `uniq`

```
sort plik.txt | uniq
```

Usuwamy zduplikowane linie, ale **tylko sąsiednie** — dlatego niemal zawsze uruchamiamy to po `sort`, które gwarantuje, że identyczne linie znajdą się obok siebie.

sort = posortuj linie tekstu alfabetycznie albo liczbowo

TYLKO ODCZYT

Zliczanie wystąpień: `uniq -c`

```
sort plik_posortowany.txt | uniq -c
```

Liczymy, ile razy każda linia wystąpiła pod rząd. Wynik to liczba i linia, oddzielone spacjami.

Przykład:

```
      2 srv01
      1 srv02
      2 srv03
```

sort = posortuj linie tekstu alfabetycznie albo liczbowo

TYLKO ODCZYT

Wypisanie tylko zduplikowanych linii: `uniq -d`

```
uniq -d plik_posortowany.txt
```

Pokazujemy wyłącznie te linie, które mają co najmniej jeden sąsiedni duplikat — reszta znika z wyniku.

uniq = unique: scala sąsiednie identyczne linie; z opcją -c liczy, ile razy każda się powtórzyła

TYLKO ODCZYT

Wypisanie tylko linii bez duplikatu: `uniq -u`

```
uniq -u plik_posortowany.txt
```

Odwrotność `-d`: pokazujemy wyłącznie linie, które nie mają sąsiedniego duplikatu — przydatne do znalezienia unikatów w danych, w których większość jest powielona.

uniq = unique: scala sąsiednie identyczne linie; z opcją -c liczy, ile razy każda się powtórzyła

TYLKO ODCZYT

Ignorowanie wielkości liter: `uniq -i`

```
uniq -i plik_posortowany.txt
```

Przy porównaniu sąsiednich linii traktujemy wielkie i małe litery jako identyczne — `"Alicja"` i `"alicja"` obok siebie zliczą się jako duplikat.

uniq = unique: scala sąsiednie identyczne linie; z opcją -c liczy, ile razy każda się powtórzyła

TYLKO ODCZYT

Pomijanie pól przy porównaniu: `uniq -f`

```
uniq -f1 plik_posortowany.txt
```

Pomijamy podaną liczbę pierwszych **pól** (oddzielonych białym znakiem) przy porównywaniu linii — `-f1` pomija pole 1, więc dwie linie różniące się tylko pierwszym polem zostaną uznane za duplikat.

uniq = unique: scala sąsiednie identyczne linie; z opcją -c liczy, ile razy każda się powtórzyła

TYLKO ODCZYT

Ograniczenie porównania do N znaków: `uniq -w`

```
uniq -w10 plik_posortowany.txt
```

Porównujemy tylko pierwsze 10 znaków każdej linii — reszta linii nie ma wpływu na to, czy linia zostanie uznana za duplikat.

uniq = unique: scala sąsiednie identyczne linie; z opcją -c liczy, ile razy każda się powtórzyła

TYLKO ODCZYT

Idiom: ranking najczęstszych wartości

```
awk '{print $1}' access.log | sort | uniq -c | sort -rn | head -10
```

Cztery kroki po kolei: wyciągamy kolumnę, ustawiamy identyczne wartości obok siebie, liczymy wystąpienia, sortujemy malejąco po liczbie i obcinamy do 10. Rozłożenie tego idiomu

na czynniki jest w *Krok po kroku*.

Przykład:

```
    842 203.0.113.17
    301 203.0.113.9
    156 203.0.113.46
```

□□ **awk** = od nazwisk autorów Aho, Weinberger, Kernighan: język do danych w kolumnach

□□ TYLKO ODCZYT

Deduplikacja bez sortowania: `awk '!seen[$0]++'`

```
awk '!seen[$0]++' plik.txt
```

Usuwamy duplikaty **z zachowaniem oryginalnej kolejności** — w przeciwieństwie do `sort -u`, które przy okazji przestawia linie alfabetycznie. Mechanizm: tablica asocjacyjna `seen` pamięta, co już widziano; `!seen[$0]++` jest prawdziwe tylko przy pierwszym wystąpieniu danej linii.

□□ **awk** = od nazwisk autorów Aho, Weinberger, Kernighan: język do danych w kolumnach

□□ TYLKO ODCZYT

Różnica i część wspólna dwóch plików: `comm`

```
comm plik1 plik2
```

Drukujemy trzy kolumny obok siebie: linie tylko w `plik1`, tylko w `plik2`, i wspólne dla obu. Oba pliki muszą być wcześniej posortowane. Flagi `-1`, `-2`, `-3` wyłączają odpowiednią kolumnę — `comm -12` (czyli `-1 -2` razem) zostawia tylko część wspólną, `comm -23` tylko unikaty pierwszego pliku, `comm -3` wszystko oprócz części wspólnej.

Przykład:

```
□□srv01
srv02
□□srv03
```

◫◫ TYLKO ODCZYT

Wszystkie flagi

Flaga	Co robi
`-n`	sortowanie numeryczne
`-h`	sortowanie rozmiarów „human readable" (K, M, G, T)
`-V`	sortowanie numerów wersji
`-r`	odwraca kolejność
`-k2,2n`	klucz ograniczony do pola 2, numerycznie
`-t`	separator pól
`-u` (sort)	sortuj i usuń duplikaty całych linii
`-s`	sortowanie stabilne (bez przestawiania remisów)
`-o`	bezpieczny zapis w miejscu
`-z`	rekordy rozdzielone NUL
`-c` (uniq)	zlicza wystąpienia
`-d` (uniq)	pokazuje tylko duplikaty
`-u` (uniq)	pokazuje tylko linie bez duplikatu
`-i` (uniq)	ignoruje wielkość liter
`-f` (uniq)	pomija N pierwszych pól przy porównaniu
`-w` (uniq)	porównuje tylko pierwsze N znaków

Uważaj

- `-k2` bez `,2` to klucz od pola 2 **do końca linii**, nie samo pole 2 — przy kilku kluczach (`-k1 -k2n`) drugi klucz może nigdy nie zadziałać.
- `uniq` porównuje wyłącznie linię z linią **bezpośrednio przed nią**. Bez wcześniejszego `sort` duplikaty z odległych miejsc pliku przejdą niezauważone.
- `sort plik.txt > plik.txt` obcina plik do zera, dokładnie jak przy `cat` — użyj `sort -o plik.txt plik.txt`.

Z życia wzięte

> **Z ŻYCIA WZIĘTE**
>
> W ostatni dzień zamknięcia miesiąca, o 22:30, log transakcyjny na sql01 rośnie szybciej niż zwykle i pani Halina pyta, które zapytanie odpala się najczęściej. Jednym poleceniem sort, uniq -c i sort -rn z logu zapytań wyciągamy ranking i widzimy, że jeden raport uruchamia się co minutę zamiast raz dziennie. Po poprawieniu harmonogramu w cronie log wraca do normalnego tempa przyrostu.

Po co to istnieje

Masz plik z 50 000 adresów IP z logów i pytanie „które dziesięć adresów generuje najwięcej ruchu". Same `awk` czy `grep` nie odpowiedzą — potrzebujesz posortować dane, żeby identyczne wartości znalazły się obok siebie, policzyć je, i posortować wynik po liczności. To dokładnie robota dla pary `sort` i `uniq`: dwa małe narzędzia z coreutils, które w praktyce niemal zawsze występują razem, bo drugie bez pierwszego działa tylko połowicznie.

Analogia

Wyobraź sobie **sortownię przesyłek pocztowych z jednym stanowiskiem liczącym**.

Na sortowni najpierw paczki trafiają na taśmę **sortującą** — pracownik ustawia je według kodu pocztowego, jedna po drugiej, aż wszystkie paczki z tym samym kodem leżą obok siebie w jednym rzędzie. To jest `sort`: bierze cały stos i układa go od nowa według wybranego klucza, bez względu na to, w jakiej kolejności paczki przyjechały.

Zaraz za sortownią stoi pracownik **liczący** — ale ma bardzo ograniczoną pamięć. Patrzy tylko na paczkę, którą aktualnie trzyma w ręce, i porównuje ją z tą, którą trzymał chwilę wcześniej. Jeśli są identyczne (ten sam kod pocztowy), dolicza jeden do licznika i odkłada duplikat. Jeśli się różnią, zaczyna liczyć od nowa dla nowego kodu. To jest `uniq` — i to jest jego fundamentalne ograniczenie: **nie pamięta niczego sprzed dwóch kroków**. Jeśli na drugim końcu hali leży identyczna paczka, ale między nią a tą w ręce pracownika przewinęło się sto innych — nie ma szans, żeby je skojarzyć. Dlatego zanim paczki dotrą do liczącego, muszą przejść przez sortownię. Bez tego kroku pracownik zliczy tylko przypadkowe, sąsiadujące z przypadku duplikaty, a resztę przepuści jako unikaty.

Jak to działa

Rozwiń: jak to działa pod spodem

> **Pojęcie: kolejność leksykograficzna i numeryczna.** Kolejność leksykograficzna porównuje dwa ciągi znak po znaku, tak jak w słowniku: patrzy na pierwszy znak obu ciągów, i jeśli się różnią, to on rozstrzyga porównanie, niezależnie od tego, co następuje dalej. Kolejność numeryczna najpierw interpretuje cały ciąg jako liczbę, a dopiero potem porównuje wartości. Dla `"9"` i `"10"` te dwa porządki dają sprzeczny wynik: leksykograficznie `"10"` wychodzi przed `"9"`, bo pierwszy znak `'1'` jest „mniejszy" niż `'9'`; numerycznie `9 < 10`, więc kolejność jest odwrotna. `sort` domyślnie sortuje leksykograficznie — `-n`, `-h` i `-V` to trzy warianty każące mu zamiast tego interpretować linię jako wartość (liczbę, rozmiar z jednostką, numer wersji) przed porównaniem.

To, jak dokładnie wygląda porównanie leksykograficzne, zależy od **locale**: zestawu ustawień językowo-regionalnych, które m.in. określają, w jakiej kolejności sortują się znaki. W locale `C` (czysto bajtowym, niezależnym od języka) wielkie litery sortują się przed małymi, bo w ASCII `A`-`Z` (kody 65-90) poprzedzają `a`-`z` (kody 97-122) — `"Banana"` wyjdzie przed `"apple"`. W typowym locale językowym (np. `pl_PL.UTF-8`, `en_US.UTF-8`) porównanie jest „inteligentniejsze" — zwykle ignoruje wielkość liter przy porządkowaniu i uwzględnia znaki narodowe we właściwym miejscu alfabetu — ale to oznacza też, że **ten sam plik posortowany na dwóch różnych systemach (albo w skrypcie i interaktywnie) może dać różny wynik**. `LC_ALL=C sort` wymusza czysto bajtowe porównanie, niezależne od tego, jakie locale ma akurat zalogowany użytkownik czy zadanie crona — więcej o mechanizmie zmiennych `LC_*` na stronie *Zmienne środowiskowe i PATH*.

Klucze pól (`-k`) mają składnię `-kPOCZĄTEK[,KONIEC][modyfikator]`. Jeśli pominiesz `KONIEC`, klucz rozciąga się **od podanego pola do końca linii** — to jest najczęstsza pułapka tej komendy, opisana szerzej w *Częste błędy*. `-t` ustawia separator pól (tak jak `-d` w `cut`), domyślnie dowolny ciąg białych znaków.

Zapis w miejscu przez `-o` jest bezpieczny, mimo że `sort -o plik plik` wygląda identycznie ryzykownie jak `sort plik > plik`. Różnica jest w tym, kto i kiedy otwiera plik wyjściowy: przy `>` to **powłoka** otwiera i obcina plik do zera, zanim `sort` w ogóle zacznie czytać — dokładnie ten sam mechanizm, który niszczy plik przy `cat plik > plik` (opisany na stronie *cat — wypisywanie i sklejanie plików*). Przy `-o` to **sam `sort`** zarządza plikiem: wczytuje całe wejście do pamięci, sortuje, i dopiero potem otwiera plik wyjściowy do zapisu — więc nawet jeśli plik wejściowy i wyjściowy to ta sama nazwa, dane są już bezpiecznie w pamięci procesu, zanim cokolwiek zostanie nadpisane.

`uniq` porównuje wyłącznie sąsiednie linie — to nie jest ograniczenie implementacyjne, tylko cała zasada działania tego narzędzia: czyta linię, porównuje z poprzednią zapamiętaną, i albo ją odrzuca (duplikat), albo zapamiętuje jako nową „poprzednią" i wypisuje. Stąd twarda zasada: `uniq` niemal zawsze występuje po `sort`, bo dopiero sortowanie gwarantuje, że

identyczne linie znajdą się obok siebie.

Krok po kroku

1. Zobacz różnicę między sortowaniem tekstowym a numerycznym.

```
printf '9\n10\n2\n' | sort
printf '9\n10\n2\n' | sort -n
```

```
10
2
9
```

```
2
9
10
```

Pierwsze porównuje `"9"`, `"10"`, `"2"` jako ciągi znaków — pierwszy znak decyduje, więc `1` < `2` < `9`. Drugie parsuje każdą linię jako liczbę przed porównaniem.

2. Posortuj `/etc/passwd` po UID.

```
sort -t: -k3,3n /etc/passwd
```

Separator `:` (`-t:`), klucz to dokładnie pole 3 (`-k3,3`), sortowanie numeryczne (`n`). Bez `,3` na końcu klucz objąłby też pola 4-7, co przy remisie UID (rzadkie, ale możliwe) dawałoby nieprzewidywalny efekt uboczny.

3. Sortuj po dwóch kluczach naraz — i zobacz, jak brak `,N` psuje drugi klucz.

```
cd /tmp
cat > multi.txt <<'EOF'
b 3
a 10
a 2
b 1
EOF
sort -k1,1 -k2,2n multi.txt
```

```
a 2
a 10
b 1
b 3
```

Poprawnie: najpierw grupy `a`/`b` alfabetycznie, wewnątrz każdej grupy `2` przed `10` numerycznie. Teraz to samo bez ograniczenia pierwszego klucza:

```
sort -k1 -k2n multi.txt
```

```
a 10
a 2
b 1
b 3
```

`a 10` wyszło przed `a 2` — bo `-k1` (bez `,1`) rozciąga klucz od pola 1 do końca linii, czyli w praktyce porównuje całe linie jako tekst. Skoro cała linia `"a 10"` różni się od `"a 2"` już na tym jednym, nieograniczonym kluczu, `sort` w ogóle nie dochodzi do sytuacji remisu, w której drugi klucz (`-k2n`) miałby cokolwiek do powiedzenia. Rozwinięcie tego mechanizmu jest w *Częste błędy*.

4. Rozmiary plików po ludzku.

```
du -sh /var/log/* | sort -h
```

`-h` rozumie sufiksy `K`, `M`, `G`, `T` i sortuje po rzeczywistej wielkości, a nie leksykograficznie (gdzie `"2G"` wyszłoby przed `"500M"`, bo `'2'` < `'5'`).

5. Zapisz posortowany wynik z powrotem do tego samego pliku.

```
sort -k1,1 dane.csv -o dane.csv
```

Bezpieczne — w przeciwieństwie do `sort -k1,1 dane.csv > dane.csv`, które obetnie plik do zera, zanim `sort` zdąży go przeczytać. Mechanizm opisany w *Jak to działa* i w *Częste błędy* na stronie o `cat`.

6. Policz i posortuj według liczności — klasyczny idiom rozłożony na czynniki.

```
awk '{print $1}' /var/log/nginx/access.log | sort | uniq -c | sort -rn | head -10
```

```
    842 203.0.113.17
    301 203.0.113.9
    156 203.0.113.46
```

Cztery kroki po kolei:

1. `awk '{print $1}'` — zostaw tylko adres IP z każdej linii logu.
2. `sort` — ustaw identyczne adresy obok siebie (wymóg dla `uniq`).
3. `uniq -c` — policz, ile razy każdy adres wystąpił pod rząd; wynik to `"  LICZBA adres"`.
4. `sort -rn` — posortuj te pary numerycznie (`-n`, bo teraz pierwszym „polem" jest liczba z `uniq -c`) i odwrotnie (`-r`, żeby najczęstsze były na górze), a `head -10` obcina do 10 najlepszych.

7. Usuń duplikaty z zachowaniem oryginalnej kolejności.

```
awk '!seen[$0]++' historia_komend.txt
```

`sort -u` też usuwa duplikaty, ale przy okazji **przestawia kolejność linii** na alfabetyczną — jeśli oryginalna kolejność ma znaczenie (np. historia poleceń, log zdarzeń), ten jednolinowiec z `awk` jest właściwym narzędziem, bo nie sortuje, tylko pamięta, co już widział.

8. Porównaj dwie posortowane listy.

```
comm -12 uzytkownicy_ad.txt uzytkownicy_ldap.txt
```

```
adm_user
jkowalski
```

`comm` (bez flag) drukuje trzy kolumny: linie tylko w pierwszym pliku, tylko w drugim, i wspólne dla obu. Flagi `-1`, `-2`, `-3` **wyłączają** odpowiednią kolumnę — `-12` (czyli `-1 -2` razem) zostawia tylko trzecią, czyli część wspólną. Oba pliki muszą być posortowane, dokładnie jak przy `join`.

Częste błędy

`-k2` zamiast `-k2,2` — drugi klucz „nie działa".

```
sort -k1 -k2n plik.txt
```

Piszący ten kod chciał: „posortuj po polu 1, a w obrębie tej samej wartości pola 1 — po polu 2 numerycznie". Napisał dokładnie to, tylko `-k1` bez `,1` oznacza „klucz od pola 1 do końca linii" — co w praktyce czyni klucz drugi bezużytecznym, bo pierwszy, nieograniczony klucz niemal zawsze już rozstrzyga porządek, zanim dojdzie do sytuacji remisu. Wynik wygląda na losowy albo „ignorujący" drugą flagę — a to `sort` robi dokładnie to, o co go poproszono, tylko nie to, co autor miał na myśli. Poprawnie:

```
sort -k1,1 -k2,2n plik.txt
```

`uniq` bez wcześniejszego `sort` — duplikaty przechodzą niezauważone.

```
uniq dane.txt
```

```
alicja
bartek
alicja
celina
```

Jeśli `alicja` występuje w pliku dwukrotnie, ale nie w sąsiadujących liniach, `uniq` obu wystąpień nie połączy — bo porównuje tylko z bezpośrednio poprzednią linią, a między dwoma `alicja` był `bartek`. Wynik wygląda, jakby deduplikacja się nie powiodła, choć narzędzie zadziałało dokładnie zgodnie ze swoją definicją. Poprawnie zawsze przez `sort` najpierw:

```
sort dane.txt | uniq
```

`sort plik > plik` — plik zamienia się w pustkę.

```
sort raport.csv > raport.csv
```

Ten sam mechanizm co przy `cat plik > plik`: powłoka otwiera `raport.csv` do zapisu i od razu go obcina, zanim `sort` zdąży cokolwiek przeczytać z (już pustego) pliku wejściowego. Wynik: `raport.csv` ma zero bajtów, a dane przepadły bez ostrzeżenia. Bezpieczna forma to wbudowana opcja `sort`, zaprojektowana właśnie do tego celu:

```
sort raport.csv -o raport.csv
```

Poleganie na porządku sortowania, który zmienia się z locale.

```
sort lista_klientow.txt | head -1
```

Skrypt działał poprawnie na serwerze produkcyjnym (locale `C`, `"Zabrze"` przed `"agencja"` — wielkie litery pierwsze), a po migracji na nowy serwer z domyślnym `pl_PL.UTF-8` wynik się zmienił, bo w tym locale sortowanie traktuje wielkość liter inaczej i uwzględnia polskie znaki diakrytyczne we właściwym miejscu alfabetu, nie na końcu tabeli ASCII. Skrypt nie zgłasza błędu — po prostu inny wiersz ląduje na górze. Jeśli zależy Ci na przewidywalnym, przenośnym porządku niezależnie od tego, gdzie skrypt się uruchomi:

```
LC_ALL=C sort lista_klientow.txt | head -1
```

□□ Ćwiczenia

>

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

Ćwiczenie 1 — sortowanie, klucze, deduplikacja

Wygeneruj dane testowe w `/tmp` i przećwicz sortowanie, klucze i deduplikację.

```
cd /tmp
cat > serwery.txt <<'EOF'
srv01 10 aktywny
srv03 2 aktywny
srv01 10 aktywny
srv02 30 nieaktywny
srv03 2 aktywny
storage01.ad.zielona.example 5 aktywny
EOF
```

Zadania:

1. Wypisz unikatowe linie, zachowując oryginalną kolejność pierwszego wystąpienia.
2. Posortuj plik po polu 2 (liczba) numerycznie, z zachowaniem tego, że dla równych wartości kolejność ma pozostać taka jak w pliku wejściowym.
3. Policz, ile razy występuje każda unikatowa nazwa serwera (pole 1), i wypisz od najczęstszej.

Odpowiedź

```
# 1. dedup z zachowaniem kolejności
awk '!seen[$0]++' serwery.txt

# 2. sortowanie po polu 2, stabilne (bez przestawiania remisów)
sort -k2,2n -s serwery.txt

# 3. zliczanie po nazwie serwera
awk '{print $1}' serwery.txt | sort | uniq -c | sort -rn
```

W zadaniu 2 flaga `-s` (stabilność) jest kluczowa: bez niej `sort` przy remisie na kluczu `-k2,2n` sam decyduje o kolejności, zwykle porównując dalej całą linię — co akurat tutaj mogłoby dać ten sam wynik, ale nie ma takiej gwarancji przy innych danych.

Pytania kontrolne

1. Dlaczego `sort` bez żadnych flag umieszcza `"10"` przed `"9"`?

Odpowiedź

Bo domyślnie `sort` porównuje linie jako ciągi znaków, znak po znaku, nie jako liczby. Pierwszy znak `"10"` to `'1'`, pierwszy znak `"9"` to `'9'` — a `'1'` jest „mniejsze" niż `'9'` w porządku znaków, więc `"10"` wygrywa porównanie, mimo że liczbowo jest mniejsze. Żeby porównywać wartości liczbowe, trzeba dodać `-n`.

2. Dlaczego `uniq` prawie zawsze widzisz w potoku razem z `sort`, a rzadko samodzielnie?

Odpowiedź

Bo `uniq` usuwa wyłącznie duplikaty **sąsiadujące** — porównuje bieżącą linię tylko z bezpośrednio poprzednią. Jeśli identyczne linie w pliku są rozrzucone i nie leżą obok siebie, `uniq` samodzielnie ich nie wykryje. `sort` przed `uniq` gwarantuje, że wszystkie identyczne linie trafią obok siebie, więc dopiero wtedy zliczanie czy usuwanie duplikatów ma sens.

3. Masz `sort -k2 plik` i wynik różni się od `sort -k2,2 plik`, mimo że oba „sortują po polu 2". Dlaczego?

Odpowiedź

`-k2` bez podanego pola końcowego oznacza klucz sortowania od pola 2 **do końca linii** — czyli w praktyce porównywana jest cała reszta linii, nie samo pole 2. `-k2,2` jawnie ogranicza klucz wyłącznie do pola 2. Różnica ujawnia się, gdy pola po polu 2 różnią się między liniami, które mają to samo pole 2 — `-k2` weźmie je pod uwagę przy porównaniu, `-k2,2` całkowicie je zignoruje.

4. Chcesz usunąć duplikaty z listy adresów e-mail, ale zachować kolejność, w jakiej pojawiły się pierwszy raz w pliku. Czy `sort -u` jest dobrym wyborem?

Odpowiedź

Nie do końca — `sort -u` usuwa duplikaty, ale przy okazji sortuje całą listę alfabetycznie, więc oryginalna kolejność ginie. Jeśli kolejność ma znaczenie, właściwym narzędziem jest `awk '!seen[$0]++' plik.txt`, które śledzi już zobaczone linie w tablicy asocjacyjnej i wypisuje tylko pierwsze wystąpienie każdej, bez zmiany kolejności.

Powiązane: cut, paste, join: praca na kolumnach · awk: język do danych w kolumnach · cat: wypisywanie i sklejanie plików · Zmienne środowiskowe i PATH

Źródła: `man 1 sort` · `man 1 uniq` · `man 1 comm` · `info coreutils 'sort invocation'` · GNU Coreutils Manual (`gnu.org/software/coreutils`)

Revision #19
Created 2026-09-17 15:07:05 UTC by Tomasz Konieczny
Updated 2026-10-03 19:08:03 UTC by Tomasz Konieczny