Przekształcenia: tr, column, fmt, iconv, dos2unix

Poziom: podstawy **Dotyczy: wszystkie dystrybucje** **Czas: ~16 min**

`tr` podmienia pojedyncze znaki, `column` i `fmt`/`fold` formatują tekst do czytelnej postaci, `iconv` i `dos2unix` przepisują plik z jednego kodowania czy formatu końca linii na inny. Drobne narzędzia, bez których polskie dane tekstowe regularnie się psują.

Skróty użyte na tej stronie (6)

Skrót	Co znaczy
ASCII	American Standard Code for Information Interchange: podstawowy zestaw znaków (litery, cyfry) używany w komputerach
CSV	w Windowsie: Cluster Shared Volume, dysk widoczny naraz dla wszystkich węzłów klastra; w danych: Comma-Separated Values, plik z wartościami oddzielonymi przecinkami
DOS	Disk Operating System: bardzo prosty, jednozadaniowy system, na którym stały pierwsze wersje Windows przed Windows NT
JSON	JavaScript Object Notation: tekstowy format danych z klamrami i cudzysłowami
POSIX	Portable Operating System Interface: wspólny zestaw zasad, dzięki któremu polecenie działa tak samo na różnych Uniksach
RHEL	Red Hat Enterprise Linux: komercyjna dystrybucja Linuksa

Fiszka

TYLKO ODCZYT

Podmiana znak za znak: `tr`

```
echo "Hello World" | tr 'a-z' 'A-Z'
```

Buduje tabelę odwzorowań między dwoma zestawami znaków i zamienia każdy znak wejścia zgodnie z nią. To **nie** jest zamiana ciągów — działa znak po znaku, nie na całych słowach

(patrz *Częste błędy*).

Przykład:

```
HELLO WORLD
```

TYLKO ODCZYT

Zakresy znaków: `tr`

```
tr 'a-z' 'A-Z' < plik.txt
```

Zakres `a-z` to skrót zapisu wszystkich liter od `a` do `z`, zamiast wypisywania ich jednej po drugiej. Działa tak samo dla cyfr (`0-9`) i innych ciągłych przedziałów w tabeli znaków.

tr = translate: zamień albo usuń znaki w strumieniu tekstu

TYLKO ODCZYT

Klasy znaków POSIX: `tr '[:upper:]' '[:lower:]'`

```
echo "Hello World" | tr '[:upper:]' '[:lower:]'
```

Klasa znaków POSIX to nazwany zbiór znaków niezależny od konkretnego alfabetu czy locale — wygodny skrót zamiast ręcznego wypisywania zakresu.

Przykład:

```
hello world
```

POSIX = Portable Operating System Interface: wspólny zestaw zasad, dzięki któremu polecenie działa tak samo na różnych Uniksach

TYLKO ODCZYT

Usunięcie znaków: `tr -d`

```
tr -d '[:punct:]' < plik.txt
```

Usuwamy z wejścia każdy znak należący do podanego zbioru (tu: cała interpunkcja), bez podawania drugiego zestawu do podmiany.

tr = translate: zamień albo usuń znaki w strumieniu tekstu

TYLKO ODCZYT

Zwinięcie powtórzeń: `tr -s`

```
tr -s ' ' < plik.txt
```

Zwijamy **sąsiadujące powtórzenia** tego samego znaku do jednego wystąpienia — ciąg kilku spacji z rzędu zamieniamy w jedną. Przydaje się przed `cut -d' '`, gdy pola są rozdzielone nierówną liczbą spacji.

tr = translate: zamień albo usuń znaki w strumieniu tekstu

TYLKO ODCZYT

Dopełnienie zbioru znaków: `tr -c`

```
tr -cd '[:print:]\n' < plik.txt
```

`-c` odwraca podany zbiór na jego dopełnienie — „wszystkie znaki OPRÓCZ tych". W połączeniu z `-d` daje idiom „usuń wszystko, co NIE jest na tej liście": tu zostają tylko znaki drukowalne i nowa linia.

tr = translate: zamień albo usuń znaki w strumieniu tekstu

ZMIENIA SYSTEM

Usunięcie CR z pliku Windowsowego

```
tr -d '\r' < plik_dos.txt > plik_unix.txt
```

`\r` to znak powrotu karetki (CR, bajt `0x0D`) — na Windowsie każda linia kończy się parą `CR LF`, na Uniksie samym `LF`. Poleceniem `tr -d` usuwamy **wszystkie** wystąpienia CR w pliku, nie tylko na końcach linii.

tr = translate: zamień albo usuń znaki w strumieniu tekstu

TYLKO ODCZYT

Wygenerowanie losowego hasła

```
tr -dc 'A-Za-z0-9!@#%^&*' < /dev/urandom | head -c 20; echo
```

`/dev/urandom` produkuje strumień losowych bajtów, w tym mnóstwo, które nie odpowiadają żadnemu drukowalnemu znakowi ASCII. Filtrujemy ten strumień poleceniem `tr -dc` (usuń wszystko OPRÓCZ podanego zbioru), zostawiając tylko litery, cyfry i wybrane znaki specjalne, a `head -c 20` obcina do żądanej długości — bez niego `tr` czytałby z `/dev/urandom` bez końca.

Przykład:

```
Xk8$mQ2pL@vR9tYw!zC4
```

ASCII = American Standard Code for Information Interchange: podstawowy zestaw znaków (litery, cyfry) używany w komputerach · **tr** = translate: zamień albo usuń znaki w strumieniu tekstu

TYLKO ODCZYT

Wyrównanie w tabelę: `column -t`

```
column -t plik.txt
```

Dzielimy każdą linię na pola (domyślnie po dowolnym ciągu białych znaków) i wypisujemy je jako tabelę, w której każda kolumna ma szerokość najdłuższej wartości w niej plus odstęp.

TYLKO ODCZYT

Ustawienie separatora jawnie: `column -s`

```
column -t -s, dane.csv
```

Bez `-s,` `column` próbowałby dzielić po białych znakach, co przy CSV z przecinkami dałoby jedną, niepodzieloną kolumnę na wiersz.

Przykład:

```
id    imie      dzial
1001  Jan       IT
1002  Adam      Ksiegowosc
```

CSV = w Windowsie: Cluster Shared Volume, dysk widoczny naraz dla wszystkich węzłów klastra; w danych: Comma-Separated Values, plik z wartościami oddzielonymi przecinkami

🔍 TYLKO ODCZYT

Zawinięcie akapitów na granicach słów: `fmt`

```
fmt -w 60 opis_incydentu.txt
```

Zamieniamy jeden długi akapit (albo linie o nierównej długości) na tekst zawinięty do podanej szerokości, łamany zawsze między słowami — czytelny efekt, bez ucinania wyrazów w połowie.

🔍 TYLKO ODCZYT

Zawinięcie twarde po N znakach: `fold`

```
fold -w 72 zrzut_base64.txt
```

Łamiemy linię dokładnie po N-tym znaku, bez względu na to, czy trafi w środek słowa. Przydatne tam, gdzie w danych nie ma sensownych granic słów — ciąg base64, zrzut hex — a nie do zwykłego tekstu.

🔍 TYLKO ODCZYT

Zamiana tabulatorów na spacje: `expand`

```
expand -t 4 plik.txt
```

Zamieniamy każdy tabulator na podaną liczbę spacji (tu 4) — częsty krok przy ujednolicaniu stylu wcięć w repozytorium.

✏️ ZMIENIA SYSTEM

Zamiana spacji na tabulatory: `unexpand`

```
unexpand -a plik.py > plik_taby.py
```

Odwrotność `expand`. Flagą `-a` konwertujemy ciągi spacji **wszędzie** w linii, nie tylko na początku (domyślne zachowanie bez `-a` dotyczy tylko wcięcia).

□□ ZMIENIA SYSTEM

Zmiana kodowania znaków: `iconv -f -t`

```
iconv -f WINDOWS-1250 -t UTF-8 stary.txt > nowy.txt
```

Czytamy plik bajt po bajcie, interpretujemy go zgodnie z podanym kodowaniem źródłowym (`-f`) i zapisujemy wynik w kodowaniu docelowym (`-t`). `iconv` ufa `-f` bezkrytycznie — złe kodowanie źródłowe da bełkot, nie błąd (patrz *Częste błędy*).

□□ **iconv** = przekoduj plik tekstowy między kodowaniami znaków, na przykład z Windows 1250 na UTF 8

□□ TYLKO ODCZYT

Wyświetlenie listy wspieranych kodowań: `iconv -l`

```
iconv -l | grep -iE '1250|8859-2'
```

Bez argumentu wejściowego, tylko lista — pokazujemy wszystkie warianty i aliasy kodowań rozpoznawane na danym systemie.

Przykład:

```
CP1250//
ISO-8859-2//
WINDOWS-1250//
```

□□ **iconv** = przekoduj plik tekstowy między kodowaniami znaków, na przykład z Windows 1250 na UTF 8

□□ TYLKO ODCZYT

Usunięcie CR z końców linii: `dos2unix`

```
dos2unix plik.txt
```

Zamieniamy zakończenia linii CRLF na LF, w miejscu (nadpisuje to plik wejściowy). Odwrotność to `unix2dos`.

□□ ZMIENIA SYSTEM

Odpowiednik `dos2unix` przez `sed`

```
sed -i 's/\r$//' plik.txt
```

Robimy dokładnie to, co `dos2unix`, bez instalowania dodatkowego pakietu — usuwamy znak CR wyłącznie na końcu każdej linii (w przeciwieństwie do `tr -d '\r'`, które usunęłoby CR wszędzie w pliku). Odwrotność (`unix2dos`) to `sed -i 's/$/\r/' plik.txt`.

sed = stream editor: edytor strumienia tekstu

TYLKO ODCZYT

Wszystkie flagi

Narzędzie	Co robi
`tr`	podmienia/usuwa/zwija pojedyncze znaki
`tr -d`	usuwa znaki z podanego zbioru
`tr -s`	zwija sąsiadujące powtórzenia znaku
`tr -c`	odwraca zbiór na dopełnienie
`column -t`	wyrównuje kolumny w czytelną tabelę
`column -s`	ustawia separator pól
`fmt`	zawija akapit na granicach słów
`fold`	zawija tekst twardo, po N znakach
`expand`	tabulatory na spacje
`unexpand`	spacje na tabulatory
`iconv -f -t`	konwertuje kodowanie znaków
`iconv -l`	listuje wspierane kodowania
`dos2unix`	usuwa CR z końców linii

Uważaj

- `tr 'foo' 'bar'` **nie** zamienia ciągu `"foo"` na `"bar"` — mapuje znak po znaku, i przy powtórzonym znaku w pierwszym zbiorze liczy się **ostatnie** przypisanie (patrz *Częste błędy*).
- `file -i` zgaduje kodowanie ze statystyki bajtów — na polskich znakach w starszych kodowaniach regularnie zgaduje źle albo nie zgaduje wcale (`unknown-8bit`

).

- `iconv` bez znajomości oryginalnego kodowania to strzał w ciemno — spróbuj `iconv -f WINDOWS-1250` i `-f ISO-8859-2` osobno i porównaj, który dał sensowny tekst.

Z życia wzięte

Z ŻYCIA WZIĘTE

W czwartek o 14:20 Kasia przysyła z terenu plik CSV z zamówieniami, zapisany w Excelu na jej starym laptopie z Windows. Import na my01 zgłasza błędne znaki końca linii, a polskie litery w nazwach klientów wychodzą jako krzaki. Poleceniami dos2unix i iconv naprawiamy plik w dwie minuty, zanim zamówienia trafią do systemu.

Po co to istnieje

Dostajesz plik CSV od kogoś, kto pracuje na Windowsie. Otwierasz go w `cat` i widzi się normalnie, ale skrypt, który go przetwarza, zgłasza błędy w dziwnych miejscach — albo polskie znaki wychodzą jako krzaki, albo dopiero `grep` po numerze linii pokazuje, że każda linia kończy się niewidzialnym `^M`. To dwa osobne, klasyczne problemy: kodowanie znaków i format końca linii. `iconv` i `dos2unix` rozwiązują je osobno, bo to dwie niezależne warstwy tego samego pliku.

Osobny problem to samo **formatowanie** — surowe dane w kolumnach, które trzeba wyrównać do czytelnej tabeli (`column`), albo długi akapit tekstu, który trzeba zawinąć do sensownej szerokości (`fmt`, `fold`). A `tr` to najprostszy z całej grupy: podmienia, usuwa albo zwija pojedyncze znaki — narzędzie z 1973 roku, które wciąż jest najkrótszym sposobem na wycięcie znaków sterujących z pliku czy wygenerowanie losowego hasła z `/dev/urandom`.

Analogia

Pomyśl o **warsztacie introligatorskim**, który dostaje różne zlecenia na starych dokumentach.

Pierwsze stanowisko to **wymiana pojedynczych liter** w składzie — jeśli w całym tekście trzeba zamienić każde duże `A` na `Ą`, robi się to czcionka po czcionce, mechanicznie, bez czytania sensu zdania. To jest `tr`: nie rozumie słów, tylko podmienia znaki jeden do jednego, według z góry ustalonej tabeli.

Drugie stanowisko **wyrównuje kolumny w tabeli** przed drukiem — surowe dane napływają z nierówną szerokością wpisów, a introligator dokłada spacje, żeby wszystko ładnie się zestawiło w rządki. To `column`. Trzecie zawija długie akapity do szerokości strony, łamiąc linię tam, gdzie kończy się słowo — to `fmt` (albo, przy twardym łamaniu bez względu na słowa, `fold`).

A na końcu warsztatu stoi **przepisywacz z jednego alfabetu na drugi** — dostaje stronę zapisaną w jednym systemie znaków (starym, regionalnym) i przepisuje ją na współczesny, uniwersalny alfabet, litera po literze, uważając, żeby żadna nie została błędnie odczytana. To `iconv`. Ograniczenie tej analogii jest realne: przepisywacz musi **wiedzieć, w jakim alfabecie napisano oryginał**, zanim zacznie — jeśli błędnie założy, że strona jest po polsku w `iso-8859-2`, a naprawdę jest w `cp1250`, przepisze ją na coś, co wygląda jak tekst, ale w niektórych miejscach będzie miało złe litery. Nikt mu tego nie powie automatycznie — musi zgadywać albo zapytać.

Jak to działa

Rozwiń: jak to działa pod spodem

`tr` czyta ze standardowego wejścia i buduje tabelę odwzorowań między dwoma zestawami znaków podanymi jako argumenty: `SET1` i `SET2`. Każdy znak z `SET1` na pozycji *i* mapuje się na znak z `SET2` na tej samej pozycji *i*. To jest **transliteracja znak po znaku**, nie wyszukiwanie i zamiana podciągu — `tr` nigdy nie patrzy na sąsiednie znaki, żeby rozpoznać całe słowo.

> **Pojęcie: klasa znaków POSIX.** Nazwany, przenośny zbiór znaków, zapisywany w nawiasach podwójnie kwadratowych, np. `[:upper:]` (wielkie litery), `[:lower:]` (małe litery), `[:digit:]` (cyfry), `[:punct:]` (interpunkcja), `[:space:]` (biały znak), `[:print:]` (znaki drukowalne). Klasa istnieje po to, żeby nie zależeć od konkretnego alfabetu ani locale — `[:upper:]` działa tak samo w skrypcie uruchamianym w dowolnym środowisku, podczas gdy ręcznie wypisany zakres `A-Z` obejmuje tylko litery łacińskie bez akcentów. Zakresy (`a-z`, `0-9`) i klasy POSIX to tylko dwa różne sposoby zapisu zestawu znaków dla `tr` — nie zmieniają zasady jego działania.

Flaga `-d` usuwa znaki z `SET1` (bez drugiego zestawu), `-s` zwija **sąsiadujące powtórzenia** tego samego znaku z `SET1` do jednego wystąpienia, a `-c` odwraca `SET1` na jego dopełnienie — „wszystkie znaki OPRÓCZ tych" — co w połączeniu z `-d` daje bardzo użyteczny idiom: „usuń wszystko, co NIE jest na tej liście" (`tr -cd 'A-Za-z0-9'`).

`column -t` dzieli każdą linię wejścia na pola według separatora (domyślnie dowolny ciąg białych znaków, jawnie ustawiany przez `-s`) i wypisuje je jako tabelę, w której każda kolumna ma szerokość najdłuższej wartości w tej kolumnie plus odstęp. Nowsze wersje `column` (util-linux od wersji 2.34, oparte na bibliotece `libsmartcols` — tej samej, której używają `lsblk` i `findmnt`)

potrafią też nazwać kolumny jawnie (`-N`, `--table-columns`) i wypisać wynik jako JSON (`-J`, `--json`, wymaga podanych nazw kolumn przez `-N`). Na RHEL 9 (util-linux 2.37.4) i RHEL 10 te flagi są dostępne; na RHEL 7/8 (starsze util-linux) — nie, tam `column` ma tylko podstawowe `-t`, `-s`, `-o`.

`fmt` i `fold` rozwiązują ten sam problem — zbyt długie linie — dwiema różnymi metodami. `fmt` traktuje tekst jako ciąg słów oddzielonych spacjami i łamie linię wyłącznie na granicy słowa, nigdy w jego środku — to jest właściwe narzędzie do akapitów przeznaczonych do czytania. `fold` łamie linię dokładnie po N-tym znaku, bez względu na to, czy trafi w środek słowa — przydatne tam, gdzie w danych w ogóle nie ma sensownych granic słów, np. przy zawijaniu ciągu base64 czy zrzutu hex do stałej szerokości terminala.

> **Pojęcie: kodowanie znaków.** Tabela, według której komputer zamienia bajty na litery i odwrotnie. Ten sam bajt oznacza zupełnie inny znak w zależności od tego, jakiego kodowania używa program, który go odczytuje — dlatego plik może „wyglądać normalnie" u autora i zamienić się w krzaki u odbiorcy. Polskie teksty historycznie krążyły w co najmniej trzech kodowaniach 8-bitowych: `ISO-8859-2` (Latin-2, starszy standard), `WINDOWS-1250` (cp1250, wariant Microsoftu, bliski, ale nie identyczny z ISO-8859-2) i `IBM852` / `CP852` (DOS). Dziś wszystko powinno być `UTF-8` — kodowanie zmiennej długości, w którym znaki ASCII zajmują 1 bajt, a większość pozostałych znaków (w tym polskie litery z ogonkami) 2 bajty, i które jest jedynym z tej listy, obsługującym praktycznie każdy alfabet świata w jednym pliku. `iconv` jest narzędziem do migracji z tamtych trzech starszych kodowań na ten jeden współczesny.

`iconv -f KODOWANIE_ŹRÓDŁOWE -t KODOWANIE_DOCELOWE` czyta plik bajt po bajcie, interpretuje go zgodnie z podanym kodowaniem źródłowym (czyli mapuje bajty na punkty kodowe Unicode według tabeli tego kodowania) i zapisuje wynik w kodowaniu docelowym. Kluczowe jest to, że `iconv` **ufa** podanemu `-f` bezkrytycznie — jeśli podasz złe kodowanie źródłowe, `iconv` i tak wykona konwersję, tylko wynik będzie bełkotem, bo zinterpretuje bajty według niewłaściwej tabeli.

Krok po kroku

1. Usuń znaki CR z pliku Windowsowego.

```
tr -d '\r' < config_windows.txt > config_unix.txt
```

`\r` to znak powrotu karetki (CR, bajt `0x0D`) — na Windowsie każda linia kończy się parą `CR LF`, na Uniksie samym `LF`. `tr -d` usuwa **wszystkie** wystąpienia CR w pliku, nie tylko na końcach linii — dla typowego pliku tekstowego to bez znaczenia, bo CR nigdzie indziej nie występuje, ale przy danych binarnych byłoby to zbyt agresywne.

2. Zamień ciąg spacji na jedną, przed dalszym przetwarzaniem.

```
ps aux | tr -s ' ' | cut -d' ' -f2
```

Dokładnie ten problem opisany jest szerzej na stronie o `cut` — `tr -s ' '` naprawia go, zanim `cut` w ogóle zobaczy dane.

3. Wygeneruj losowe hasło z `/dev/urandom`.

```
tr -dc 'A-Za-z0-9!@#%^&*' < /dev/urandom | head -c 20; echo
```

```
Xk8$mQ2pL@vR9tYw!zC4
```

`/dev/urandom` produkuje strumień losowych bajtów — w tym mnóstwo, które nie odpowiadają żadnemu drukowalnemu znakowi ASCII. `tr -dc` (usuń wszystko OPRÓCZ podanego zbioru) filtruje strumień, zostawiając tylko litery, cyfry i wybrane znaki specjalne, a `head -c 20` obcina do żądanej długości. Bez `head -c` `tr` czytałby z `/dev/urandom` bez końca.

4. Wyrównaj CSV do czytelnej tabeli w terminalu.

```
column -t -s, dane.csv
```

```
id    imie      dzial
1001  Jan       IT
1002  Adam      Ksiegowosc
```

Bez `-s,` `column` próbowałby dzielić po białych znakach, co przy CSV z przecinkami dałoby jedną, niepodzieloną kolumnę na wiersz.

5. Zawijaj długi tekst do szerokości terminala.

```
fmt -w 60 opis_incydentu.txt
```

Zamienia jeden długi akapit (albo linie o nierównej długości) na tekst zawinięty do 60 znaków, łamany zawsze między słowami — czytelny efekt, bez ucinania wyrazów w połowie.

6. Napraw plik z Windowsa: kodowanie i końce linii razem.

```
iconv -f WINDOWS-1250 -t UTF-8 raport_stary.csv | tr -d '\r' > raport_utf8.csv
```

Kolejność ma znaczenie: najpierw konwersja kodowania (bo `\r` jako bajt `0x0D` jest identyczne w obu kodowaniach, więc kolejność `tr` względem `iconv` akurat tu nie zaszkodzi, ale w ogólnym przypadku bezpieczniej najpierw ujednolicić kodowanie, a dopiero potem operować na konkretnych bajtach).

7. Sprawdź, jakie kodowania obsługuje `iconv` na Twoim systemie.

```
iconv -l | grep -iE '1250|8859-2'
```

```
CP1250//
ISO-8859-2//
WINDOWS-1250//
```

Ta sama nazwa kodowania bywa dostępna pod kilkoma aliasami (`CP1250` i `WINDOWS-1250` to w praktyce to samo) — `iconv -l` (bez argumentu wejściowego, tylko lista) pokazuje wszystkie warianty rozpoznawane na danym systemie.

8. Zamień odstępy wiodące na taby, np. przed wklejeniem kodu do formatu, który tego wymaga.

```
unexpand -a plik.py > plik_taby.py
```

`-a` konwertuje ciągi spacji **wszędzie** w linii, nie tylko na początku (domyślne zachowanie bez `-a` dotyczy tylko wcięcia). Odwrotność, `expand -t 4`, zamienia taby na 4 spacje — częsty krok przy ujednolicaniu stylu wcięć w repozytorium.

Częste błędy

Oczekiwanie, że `tr` zamieni CAŁY CIĄG, a nie pojedyncze znaki.

```
echo "foo" | tr 'foo' 'bar'
```

```
brr
```

Wynik to `brr`, nie `bar`, choć intencja autora była jasna: „zamień słowo foo na słowo bar". `tr` zbudował tabelę pozycyjną: `f`→`b`, `o`→`a`, `o`→`r`. Ponieważ `o` występuje w `SET1` dwukrotnie (pozycja 2 i 3), **druga, późniejsza definicja nadpisuje pierwszą** — każde `o` w wejściu mapuje się finalnie na `r`, nie na `a`. Stąd `f`→`b`, a oba `o`→`r`, czyli `brr`. Do zamiany całych ciągów `tr` się nie nadaje — trzeba `sed`: `sed 's/foo/bar/'`.

`cut -c` (i myślenie, że działa na znakach) na polskim tekście.

```
echo "źdźbło" | cut -c1-3
```

```
d
```

To nie błąd `tr`, ale ściśle powiązany temat z tej samej rodziny narzędzi tekstowych: GNU coreutils od lat, mimo nazwy `-c` („characters"), liczy **bajty**. Polskie znaki diakrytyczne w UTF-8 zajmują 2 bajty, więc cięcie po pozycji bajtowej może rozciąć znak w połowie. Więcej o tym na stronie o `cut, paste, join`.

Ufanie `file -i` bez zastrzeżeń przy wykrywaniu kodowania.

```
file -i raport_od_klienta.csv
```

```
raport_od_klienta.csv: text/plain; charset=unknown-8bit
```

`file -i` zgaduje kodowanie na podstawie **statystycznej analizy rozkładu bajtów**, nie ma dostępu do żadnej wiarygodnej metadanej mówiącej, czym plik naprawdę jest. Dla czystego ASCII i UTF-8 zgadnięcie jest praktycznie pewne (rozpoznawalne wzorce bajtów wielobajtowych), ale dla starszych kodowań 8-bitowych (`cp1250`, `iso-8859-2`) `file` często odpowiada bezradnym `unknown-8bit` — bo oba te kodowania używają tej samej przestrzeni bajtów (0x80-0xFF), a bez słownika języka nie da się jednoznacznie rozstrzygnąć, które konkretne znaki reprezentują. Jedyny pewny sposób to sprawdzenie treści po konwersji próbnej: `iconv -f WINDOWS-1250 -t UTF-8 plik | head` i porównanie, czy polskie słowa wyglądają sensownie, a jeśli nie — próba z `ISO-8859-2`. Głębszy opis zawodności wykrywania typu pliku jest na stronie *Pliki, które nie są tekstem: file, strings, hexdump, xxd*.

Konwersja `iconv` z błędnym kodowaniem źródłowym — plik „działa", ale ma złe litery.

```
iconv -f ISO-8859-2 -t UTF-8 plik_cp1250.csv -o plik.csv
```

Polecenie **nie zwróci błędu** — `ISO-8859-2` i `WINDOWS-1250` mają zbliżony, ale nie identyczny układ znaków w górnej połowie tabeli (różnice m.in. w pozycjach niektórych znaków z ogonkiem i cudzysłowów typograficznych). Plik wyjściowy będzie wyglądał na poprawny UTF-8 i większość liter się zgodzi, ale pojedyncze znaki (często `ą`, `ę` albo cudzysłowy) wyjdą błędne — literówki, które trudno zauważyć bez uważnego czytania całego pliku. Jeśli nie masz pewności co do oryginalnego kodowania, przetestuj oba warianty na próbce i porównaj wizualnie:

```
head -5 plik.csv | iconv -f WINDOWS-1250 -t UTF-8
head -5 plik.csv | iconv -f ISO-8859-2 -t UTF-8
```

□□ Ćwiczenia

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

Ćwiczenie 1 — CRLF i kodowanie

Przygotuj dane w `/tmp` i przećwicz `tr`, `column` oraz `iconv`.

```
cd /tmp
printf 'id,imie,miasto\r\n1,Jan,Warszawa\r\n2,Adam,Krakow\r\n' > dane_windows.csv
iconv -f UTF-8 -t WINDOWS-1250 <<'EOF' > opis_cp1250.txt
Zażółć gęślą jaźń — dane testowe w starym kodowaniu.
EOF
```

Zadania:

1. Sprawdź, czy `dane_windows.csv` ma końce linii CRLF (przez `cat -A`), a potem usuń CR i zapisz jako `dane_unix.csv`.
2. Wyświetl `dane_unix.csv` jako czytelną tabelę w terminalu.
3. Sprawdź, co pokazuje `file -i` na `opis_cp1250.txt`, a potem przekonwertuj go do UTF-8 i porównaj wynik.

Odpowiedź

```
# 1. sprawdzenie i usunięcie CR
cat -A dane_windows.csv | head -2
tr -d '\r' < dane_windows.csv > dane_unix.csv

# 2. tabela
column -t -s, dane_unix.csv

# 3. wykrywanie i konwersja kodowania
file -i opis_cp1250.txt
iconv -f WINDOWS-1250 -t UTF-8 opis_cp1250.txt
```

W zadaniu 3 `file -i` najpewniej zwróci `charset=unknown-8bit` — plik jest poprawnym `cp1250`, ale dla `file` to tylko „jakieś bajty z górnej połowy tabeli", nieodróżnialne bez znajomości języka od `iso-8859-2` czy innego kodowania 8-bitowego.

Pytania kontrolne

1. `tr 'ab' 'xy'` na wejściu `"aabb"` daje zgodnie z intuicją `"xxyy"`. Dlaczego `tr 'aab' 'xyz'` na tym samym wejściu `"aabb"` już NIE daje `"xxyy"`?

Odpowiedź

W pierwszym przypadku żaden znak nie powtarza się w `SET1` (`ab`), więc mapowanie jest jednoznaczne: `a`→`x`, `b`→`y`. W drugim `SET1` to `aab` — znak `a` występuje dwukrotnie, na pozycji 1 i 2, mapowany kolejno na `x` i na `y`. Ponieważ późniejsza definicja nadpisuje wcześniejszą, finalnie każde `a` mapuje się na `y` (drugie wystąpienie), a `b` (trzecia pozycja w `SET1`, odpowiadająca `z` w `SET2`) na `z`. Wynik dla `"aabb"` to `"yyzz"`, nie `"xxyy"`.

2. Masz plik, o którym wiesz tylko, że pochodzi z systemu księgowego sprzed lat i zawiera polskie znaki. `file -i` zwraca `unknown-8bit`. Jak dalej ustalić kodowanie?

Odpowiedź

Trzeba wypróbować konwersję z najbardziej prawdopodobnymi kandydatami (dla polskich danych: `WINDOWS-1250`, potem `ISO-8859-2`, ewentualnie `CP852` dla bardzo starych danych z DOS-a) i wizualnie ocenić, który wariant daje sensowny tekst — `iconv -f KODOWANIE -t UTF-8 plik | head`. Automatyczne, pewne wykrycie nie istnieje dla kodowań 8-bitowych, bo różne kodowania używają tej samej przestrzeni bajtów do reprezentowania różnych znaków.

3. Czym różni się `fmt -w 70` od `fold -w 70` przy tym samym pliku wejściowym?

Odpowiedź

`fmt` łamie linię wyłącznie na granicach słów (spacjach), więc wynik jest czytelnym akapitem bez ucinania wyrazów, ale poszczególne linie mogą mieć nieco mniej niż 70 znaków. `fold` łamie dokładnie po 70. znaku, bez względu na to, czy trafi w środek słowa — przydatne dla danych bez naturalnych granic słów (np. ciąg base64), ale nieczytelne dla zwykłego tekstu.

4. Dlaczego `column -t -J` bez `-N` kończy się błędem albo bezużytecznym wynikiem?

Odpowiedź

Format JSON wymaga nazwanych kluczy dla każdego obiektu w tablicy wynikowej, a `column` nie ma skąd wziąć nazw kolumn, jeśli nie zostaną podane jawnie przez `-N`/`--table-columns`. Dokumentacja wprost zaznacza, że `--table-columns` jest wymagane przy `--json` — bez tego `column` nie ma z czego zbudować kluczy `"nazwa": "wartość"` w wyjściowym obiekcie.

Powiązane: cut, paste, join: praca na kolumnach · sed: edytor strumieniowy · Pliki, które nie są tekstem: file, strings, hexdump, xxd · Zmienne środowiskowe i PATH

Źródła: `man 1 tr` · `man 1 column` · `man 1 fmt` · `man 1 fold` · `man 1 iconv` · `man 1 dos2unix` · `column(1)` (man7.org, util-linux) · POSIX.1-2024

Revision #18
Created 2026-09-17 15:07:06 UTC by Tomasz Konieczny
Updated 2026-10-03 19:08:05 UTC by Tomasz Konieczny