sed — uniwersalny edytor strumieniowy

`sed` to edytor strumieniowy zawarty w systemach uniksowych, służący do przetwarzania plików tekstowych. Jego funkcjonalność można docenić już po kilku pierwszych minutach użytkowania.

Podstawowa komenda pracująca w co najmniej 90% zastosowań to `s` (*for substitution*).

```
sed 's/day/night/' <oldfile >newfile
echo "Sunday" | sed 's/day/night/'
⟶Sunnight
```

Można wyróżnić cztery składniki powyższego użycia komendy `s`: nazwa komendy (`s`), separatory (`/`) , regularne wyrażenie dla wzorca (`day`), ciąg zastępujący (`night`).

Separator `/` nie zawsze jest najwygodniejszy:

```
sed 's/\/usr\/local\/bin/\/common\/bin/'
sed 's_/usr/local/bin_/common/bin_'
sed 's:/usr/local/bin:/common/bin:'
sed 's|/usr/local/bin|/common/bin|'
```

Użycie `&` w ciągu zastępującym odpowiada dopasowanemu ciągowi do wzorca:

```
echo "abc 123" | sed 's/[0-9]*/& &/'
⟶" abc 123"
echo "abc 123" | sed 's/[0-9][0-9]*/& &/'
⟶"abc 123 123"
echo "abc 123" | sed -r 's/[0-9]+/& &/'
⟶"abc 123 123"
```

Użycie `\1` we wzorcu lub w ciągu zastępującym odpowiada dopasowanemu ciągowi w bloku wzorca wyznaczonym przez `\(...\)` (lub `(...)` przy opcji `-r`):

```
echo "rower gruszka" |sed -r 's/([a-z]+) ([a-z]+)/\2 \1/'
⟶"gruszka rower"
echo "abc 123 123 def" | sed 's/\([a-z]*\) \1/\1/'
⟶"abc 123def"
```

Flaga `g` powoduje zastąpienie wszystkich wystąpień wzorca w linii ciągiem zastępującym:

```
echo "abc 123" | sed -r 's/[^ ]+/(&)/'
⟶"(abc) 123"
echo "abc 123" | sed -r 's/[^ ]+/(&)/g' <old >new
⟶"(abc) (123)"
```

Flaga liczbowa — zastąpienie kolejnego dopasowania wzorca wyrażonego podaną liczbą.

```
echo "abcdef" | sed 's/./&:/4'
⟶"abcd:ef"
```

Domyślnie `sed` drukuje każdą linię. Jeśli wykonywane jest zastąpienie, nowa linia jest drukowana zamiast starej. Gdy użyjemy opcji `-n` domyślne drukowanie będzie wyłączone.

Flaga `p` — drukowanie dopasowanych linii; każda linia dopasowana do wzorca zostanie wydrukowana.

```
sed -n 's/pattern/&/p'
```

Flaga `w` — zapisanie do podanego pliku.

```
sed -n 's/^[0-9]*[02468] /&/w even' <file
```

Flagi można łączyć:

```
sed -n 's/a/A/2pw /tmp/file' <old >new
```

Opcja `-e`, czyli wiele komend w jednym sedzie:

```
sed 's/BEGIN/begin/' <old | sed 's/END/end/' >new
sed -e 's/BEGIN/begin/' -e 's/END/end/' <old >new
```

Dłuższe polecenia nad którymi pracujemy warto zapisywać jako skrypty:

```
#!/bin/bash
sed '
s/a/A/g
s/e/E/g
s/i/I/g
s/o/O/g
s/u/U/g'
```

Zawężenie polecenia do pojedynczej linii, czy przedziału linii, czy też linii pasujących do podanego wzorca:

```
sed '3 s/[0-9][0-9]*//' <old >new
sed '/^#/ s/[0-9][0-9]*//'
sed '/^g/s/g/s/g'
sed '/^g/ s_g_s_g'
sed '1,100 s/A/a/'
sed '101,$ s/A/a/'
sed '/start/,/stop/ s/#.*//'
sed '1,/start/ s/#.*//'
```

Komenda `d` kasuje linię odpowiadającą zawężeniu:

```
sed '11,$ d' <file
sed '/^#/ d'
sed 's/^#.*//'
sed -e 's/#.*//' -e 's/[ ^I]*$//' -e '/^$/ d'
```

Komenda `p` — drukowanie:

```
sed -n '1,10 p'
sed -n '/match/ p'
```

Symbol `!` odwraca podane zawężenie:

```
sed -n '/match/ !p'
```

Komenda `q` przerywa działanie programu:

```
sed '11 q'
```

Grupowanie przy pomocy `{` i `}` pozwala zagnieżdżać zawężenia:

```
#!/bin/bash
# This script removes #-type comments
# between 'begin' and 'end' words.
sed -n '
1,100 {
/begin/,/end/ {
     s/#.*//
     s/[ ^I]*$//
     /^$/ d
     p
}
}'
```

```
#!/bin/bash
sed '
    /begin/,/end/ {
      /begin/ !{ /end/ !{
        s/old/new/
      }}
    }'
```

Komendy `a`, `i` oraz `c`:

```
#!/bin/bash
sed '
/WORD/ {
i\Add this line before
a\Add this line after
c\Change the line to this one
}'
```

```
#!/bin/bash
sed '
/WORD/ a\
Add this line\
This line\
And this line'
```

Komenda `=` drukuje numer bieżącej linii:

```
sed -n '/PATTERN/ =' <file
sed -n '$=' <file
sed -n '/begin/,/end/ ='
```

Pattern buffer to bufor w którym `sed` trzyma tekst, w którym wyszukuje wzorca. Domyślny przepływ danych to powtórzenia następującego cyklu. `sed` rozpoczyna wczytując pierwszą linię z wejścia do pattern buffer. Wtedy:

(i) Aplikacja kolejnych komend do bieżącej zawartości pattern buffer; Uwaga: komendy zazwyczaj zmieniają pattern buffer (np. komenda `s`);
(ii) Po aplikacji ostatniej komendy drukowanie bieżącej zawartości pattern buffer na wyjście (drukowanie można wyłączyć opcją `-n`);
(iii) Opróżnienie pattern buffer i wczytanie kolejnej linii z wejścia do pattern buffer.

Komendy `n`, `N`, `d`, `D`, `p` i `P`:

Komenda `n` drukuje pattern buffer na wyjściu (chyba, że opcja `-n` użyta), opróżnia pattern buffer i wczytuje do pattern buffer nową linię z wejścia. Komenda `N` nie drukuje pattern buffer, nie opróżnia pattern space. Czyta ona kolejną linię z wejścia i dokleja ją do aktualnego pattern buffer poprzedzając ją znakiem `\n`.

Komenda `d` opróżnia pattern buffer, czyta nową linię z wejścia, wkłada tę linię do pattern buffer i przerywa wykonywanie obecnej sekwencji komend zaczynając obrabianie nowej linii od pierwszej w kolejności komendy. Taki przeskok nazywamy rozpoczęciem nowego cyklu. Komenda `D` usuwa z pattern buffer pierwszą linię (to jest ciąg znaków do pierwszego `\n`), nie wczytuje nowej linii do pattern buffer i rozpoczyna nowy cykl.

Komenda `p` drukuje cały pattern buffer. Komenda `P` drukuje pierwszą linię pattern buffer. Te komendy nie zmieniają zawartości pattern buffer.

Przykład: do każdej linii kończącej się znakiem `#` dokleja następną linię (ale do tej kolejnej nic nie jest doklejane):

```
#!/bin/bash
sed '
# look for a "#" at the end of the line
/#$/ {
# Found one - now read in the next line
N
# delete the "#" and the new line character,
s/#\n//
}'
```

Przykład: kasuje wszystko pomiędzy `ONE` i `TWO` pojawiających się odpowiednio po sobie w tej samej lub sąsiadujących liniach. Pytanie na zrozumienie: czy semantyka skryptu zmieni się jeśli zamiast komendy `D` napiszemy `d`?

```
#!/bin/bash
sed '
/ONE/ {
# append the next line
N
# look for "ONE" followed by "TWO"
/ONE.*TWO/ {
# delete everything between
s/ONE.*TWO/ONE TWO/
# print
```

```
P
# then delete the first line
D
}
}'
```

Przykład: doklejenie do początku każdej linii jej numeru:

```
 #!/bin/bash
sed '=' file | \
sed '{
N
s/\n/ /
}'
```

Poza pattern space `sed` ma jeszcze jedno miejsce w którym możemy zapamiętać i edytować linie przeczytane z wejścia, mianowicie *hold buffer*. Jest pięć komend obsługujących hold buffer: `x`, `h`, `H`, `g` oraz `G`.

Komenda `x` zamienia wartości pattern buffer i hold buffer. Jaki jest zatem efekt polecenia `sed 'x'`? Komenda `h` kopiuje pattern buffer do hold buffer. Komenda `H` dokleja pattern buffer do końca hold buffer poprzedzając go znakiem `\n`. Komenda `g` kopiuje hold buffer do pattern buffer. Komenda `G` dokleja hold buffer do końca pattern buffer poprzedzając go znakiem `\n`.

A teraz czas na bardziej rozbudowany i sensowny przykład. Oto implementacja grepa, który poza linią z dopasowaniem drukuje również jej konekst, czyli linię poprzednią i linię następną. Niestety, jeśli ta implementacja przeocza niektóre wystąpienia ...

```
#!/bin/bash
# if there is only one argument, exit

case $# in
1);;
*) echo "Usage: $0 pattern";exit;;
esac;

# I hope the argument doesn't contain a /

sed -n '
'/$1/' !{
# put the non-matching line in the hold buffer
h
}
'/$1/' {
# found a line that matches
# add the next line to the pattern space
```

```
N
# exchange the previous line with the
# 2 in pattern space
x
# now add the two lines back
G
# and print it.
p
# add the three hyphens as a marker
a\
---
# remove first 2 lines
s/.*\n.*\n\(.*\)$/\1/
# and place in the hold buffer for next time
h
}'
```

I na koniec dwie komendy sterujące `sed`. Komenda `b` przeskakuje do miejsca wskazanego przez podaną etykietę. Jeśli po komendzie `b` nie ma żadnej etykiety przeskakujemy do końca cyklu. Komenda `t` przeskakuje do podanej etykiety jeśli ostatnio wykonane podstawienie zmodyfikowało pattern buffer.

Przykład: jeśli paragraf zawiera szukaną frazę to drukujemy cały paragraf. Paragrafy oddzielone są pustymi liniami. Pierwszy skrypt bez użycia komendy `b`, drugi z użyciem `b`. Dlaczego drugi skrypt jest lepszy?

```
#!/bin/bash
sed -n '
/^$/ !{
  H
}
/^$/ {
  x
  /'$1'/ p
}
$ {
  x
  /'$1'/ p
}'
```

```
#!/bin/bash
sed -n '
# if an empty line, check the paragraph
/^$/ b para
# else add it to the hold buffer
```

```
H
# at end of file, check paragraph
$ b para
# now branch to end of script
b
# this is where a paragraph is checked for the pattern
:para
# return the entire paragraph
# into the pattern space
x
# look for the pattern, if there - print
/'$1'/ p
'
```

Przykład użycie komendy `t`: z podanego tekstu usuń nawiasy `(` i `)` występujące w podanej kolejności i przedzielone co najwyżej białymi znakami.

```
sed 's/([ ^I]*)//g'
```

A co jeśli chcemy wykonywać to czyszczenie tak długo dopóki nie będzie takich nawiasów (czyli dla ciągu `(())` wynikowy ciąg jest pusty)?

```
#!/bin/bash
sed '
:again
s/([ ^I]*)//
t again
'
```

Zadanie 1. Napisz polecenie wstawiające pustą linię po każdej linii z wejścia.

Zadanie 2. Napisz polecenie przeplatające wejście pustymi liniami w taki sposób, że pomiędzy dwiema kolejnymi niepustymi liniami jest dokładnie jedna pusta. Uwaga: plik wejściowy może mieć puste linie.

Zadanie 3. Wydrukuj tylko nieparzyste linie z wejścia.

Zadanie 4. Wstaw pustą linię co piątą linię z wejścia (po 5tej, 10tej, itd.).

Zadanie 5. Wstaw pustą linię powyżej każdej linii zawierającej słowo `krowa`.

Zadanie 6. Wstaw pustą linię powyżej i poniżej każdej linii zawierającej słowo `krowa`.

Zadanie 7. Poprzedź każdą linię jej numerem i spacją (doklej numer na początek linii).

Zadanie 8. Poprzedź każdą niepustą linię jej numerem i spacją.

Zadanie 9. Skasuj wiodące białe znaki (spacje, tabulatory) w każdej linii.

Zadanie 10. Wstaw 5 spacji na początek każdej linii.

Zadanie 11. Wydrukuj linie z wejścia w odwróconym porządku.

Zadanie 12. Dla każdej linii z wejścia wydrukuj ją z odwróconym porządkiem znaków wewnątrz linii.

Zadanie 13. Jeśli linia zakończona jest znakiem `#` doklej do jej końca kolejną linię i skasuj `#`.

Zadanie 14. Jeśli linia rozpoczyna się znakiem `#` doklej ją do końca poprzedniej linii i skasuj `#`.

Zadanie 15. Wydrukuj 10 pierwszych linii z wejścia.

Zadanie 16. Wydrukuj ostatnią linię z wejścia.

Zadanie 17. Wydrukuj 2 ostatnie linie z wejścia.

Zadanie 18. Wydrukuj 10 ostatnich linii z wejścia.

Zadanie 19. Wydrukuj przedostatnią linię z wejścia.

Zadanie 20. Wydrukuj tylko linie o co najwyzej 10 znakach.

Zadanie 21. Wydrukuj wszystkie linie od pierwszego wystąpienia słowa `krowa` (włącznie z tą linią) do końca pliku.

Zadanie 22. Wydrukuj wszystkie linie od pierwszego wystąpienia słowa `krowa` (bez tej linii) do końca pliku.

Zadanie 23. Wydrukuj linie z wejścia pomijając te linie, które są identyczne z linią poprzednią (to jest jeśli konkretna linia występuje trzy razy po sobie wydrukuj tylko pierwsze wystąpienie).

Zadanie 24. Wydrukuj linie z wejścia pomijając linie, które są identyczne z którąś z wcześniejszych linii.

Zadanie 25. Przyjmijmy, że komentarz w wejściowym pliku tekstowym jest zaznaczony liniami postaci `begin` i `end` (nic więcej nie może być w tych liniach). Wydrukuj linie z wejścia pomijając komentarze.

Zadanie 26. Przyjmijmy, że komentarz w wejściowym pliku tekstowym jest zaznaczony słowami `begin` i `end`. Wydrukuj linie z wejścia pomijając komentarze. Uwaga: w jednej linii może byc wiele słów `begin` oraz `end`.

Zadanie 27. Przepisz standardowe wejście w taki sposób, aby wszystkie słowa zaczynały się z dużej litery a wszystkie pozostałe litery były małe. Uwaga: w ciągu zastępującym moga się przydać niektóre modyfikatory spośród `\L`, `l`, `\U`, `\u` i `\E`.

Zadanie 28 (dodatkowe)**.** Napisz emulator maszyny Turinga w `sed` skrypcie. Proszę skonsultować się z prowadzącym w celu otrzymania specyfikacji wejścia i wyjścia.