4 Edycja tekstu - sed

Edycja tekstu - sed

`sed` to edytor strumieniowy (*stream editor*) zawarty w systemach uniksowych. Podobnie jak `grep` pozwala znaleźć wystąpienia danego wzorca, ale dodatkowo daje możliwość edycji.

Przyjrzyjmy się prostemu wywołaniu `sed`.

```
echo "Sunday" | sed 's/day/night/'
⟶Sunnight
```

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

Komenda `s` (od *substitution*) jest podstawową komendą, wykorzystywaną do podmieniania wystąpienia wzorca na podany ciąg zastępujący.

Separator `/` nie zawsze jest najwygodniejszy, dlatego można wykorzystać dowolny inny znak. Znak separatora jest definiowany poprzez znak następujący bezpośrednio po `s`. Poniższe wywołania są równoważne:

```
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 ' (tak, spacja też działa)
```

Edytor strumieniowy `sed` przetwarza wejście linia po linii. Domyślnie (bez dodatkowych opcji) zamienia tylko pierwsze znalezione wystąpienie wzorca w linii na ciąg zastępujący.

Plik `tekst.in`:

```
one two three, one two three
four three two one
one hundred
```

```
sed 's/one/ONE/' < tekst.in
```

```
ONE two three, one two three
four three two ONE
ONE hundred
```

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

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

Należy zwrócić uwagę, że wyrażenie regularne w powyższych poleceniach jest dopasowywane do najwcześniejszego miejsca w tekście, a spośród dopasowań zaczynających się w tym samym miejscu wybierany jest najdłuższy tekst pasujący do wzorca. Dlatego w pierwszym przypadku `[0-9]*` dopasowuje się do ciągu pustego na samym początku linii.

Niestety `sed` nie potrafi obsługiwać wyrażeń perlowych tak jak `grep`. Za pomocą flagi `-r` lub `-E` dostępna jest jedynie tzw. "rozszerzona" wersja wyrażeń regularnych, uboższa od wyrażeń perlowych. Niemniej w zakresie jaki omówiliśmy na poprzednich zajęciach, większość jest obsługiwana przez ERE (*extended regular expression*).

Czy `sed` może się zapętlić zastępując wzorzec dwoma wystąpieniami wzorca, później czteroma, itd.? Nie. `sed` wyszukuje wzorzec w liniach tekstu na wejściu. Gdy już dokona podstawienia zmodyfikowana linia drukowana jest na wyjściu i nie jest dalej przetwarzana. Zatem `echo "a" | sed 's/a/aa/'` wydrukuje dwie literki `a`.

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

```
echo "kot dog" |sed -r 's/([a-z]+) ([a-z]+)/\2 \1/'  ⟶  "dog kot"
echo "ab 12 12 def" | sed -r 's/([123]*) \1/\1/'     ⟶  "ab12 12 def"
echo "ab 12 12 def" | sed -r 's/([123]+) \1/\1/'     ⟶  "ab 12 def"
```

Flagi seda

Po trzecim (ostatnim) separatorze można określić kilka dodatkowych flag które zmieniają działanie operacji *substitution*. Dla przykładu, 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'  ⟶  "(abc) (123)"
```

Flaga liczbowa k powoduje zastąpienie k-tego dopasowania wzorca ciągiem zastępującym.

```
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. Wówczas flagą `p` wymuszamy drukowanie linii zawierających dopasowanie do wzorca.

```
sed -n 's/[żź]/Z/gp'
```

Flaga `w` — zapisanie do podanego pliku linii zawierających dopasowanie do wzorca.

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

Flaga `I` — ignoruje wielkość liter przy dopasowywaniu.

```
echo "ABC" | sed 's/aBc/#/I'
```

Flagi można łączyć:

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

Łączenie komend

Każda linia wczytana przez `sed` może być przetwarzana przez więcej niż jedną komendę. Komendy można łączyć używając flagi `-e`. Poniższe dwa wywołania są równoważne:

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

Ciąg komend `sed`a można również zapisać jako skrypt (kolejne komendy możemy oddzielać od siebie za pomocą znaków nowej linii lub średników):

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

Zawężanie komend

Działanie każdej komendy można zawęzić do pojedynczej linii, przedziału linii, czy też linii pasujących do podanego wzorca. Parametry określające sposób zawężenia działania komendy umieszczamy na jej poczatku przed literą określającą rodzaj komendy. Działanie zawężania komend ilustrują poniższe przykłady.

W pierwszym przykładzie ograniczamy się do linii nr 3 i tylko w niej zamieniamy *niepusty* ciąg cyfr przez znak pusty.

```
sed -r '3 s/[0-9]+//' < old > new
```

W drugim przykładzie ograniczamy się do linii zaczynających się od `#`

```
sed -r '/^#/ s/[0-9]+//'
```

Możemy ograniczyć się do przedziału numerów linii. W następnym przykładzie używamy komendy `s` do pierwszych stu linii. W kolejnym do lini o numerach od 101 do ostatniej `$`.

```
sed '1,100 s/A/a/'
sed '101,$ s/A/a/'
```

Oznaczenia końców przedziałów można mieszać. Możemy np. rozpatrywać linie od pierwszej do pierwszej takiej, która zawiera jakiś wzorzec (w poniższym przykładzie słowo `start`).

```
sed '1,/start/ s/#.*//'
```

W kolejnym przykładzie, poniżej, zawężenie `/start/,/stop/` działa w następujący sposób: (1) `sed` szuka pierwszej linii zawierającej słowo (ciąg liter) `start`; (2) począwszy od tej linii (włącznie) komenda jest wykonywana aż do linii zawierającej ciąg liter `stop` (włącznie). Należy zwrócic uwagę, że `sed` nie szuka wystąpienia `stop` w linii, w której wcześniej dopasował `start`, tylko dopiero w liniach następnych po niej. Analogicznie nie szuka już wystąpienia `start` w linii, w której wczesniej dopasował `stop`.

```
sed '/start/,/stop/ s/#.*//'
```

Inne komendy

Oprócz komendy `s` w `sed` dostępnych jest kilka innych komend. Komenda `d` kasuje wybrane linie:

`sed '11,$ d' <file` (usuwa linie od 11-tej do ostatniej)
`sed '/^$/ d'` (usuwa puste linie)
`sed 's/^#.*//'` (usuwa linie zaczynające się od `#`)
`sed 's/#.*//; /^$/ d'` (kasowanie zawartości linii po `#` i kasowanie pustych linii)

Komenda `p` — drukowanie, wymusza drukowanie przetwarzanych lini w przypadku uzycia opcji `-n` :

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

Symbol `!` umieszczony po parametrach określających zawężenie odwraca je:

`sed -n '/match/ !p'` (drukujemy linie w których NIE występuje wzorzec `/match/`)

Komenda `q` przerywa działanie programu. W przykładzie poniżej wtedy, gdy dociera do 11-ej linii.

```
sed '11 q'
```

Grupowanie

Grupowanie przy pomocy nawiasów klamrowych pozwala stosować zawężenie do sekwencji komend, a także zagnieżdżać zawężenia i pisać bardziej skomplikowane skrypty. Następny przykład to "krótki program", który pomiędzy liniami zawierającymi 'begin' oraz 'end', które są pomiędzy liniami zaiwerającymi 'start' i 'stop' usuwa wszystkie linie zaczynające się od `#`.
Tak jak wcześniej wspominaliśmy, całe polecenie seda działa dla każdej linii z osobna tzn. każda z komend w najbardziej wewnętrznym nawiasie klamrowym wykonana jest na tej samej linii. Najpierw zastępujemy całą linię zaczynającą się od `#` przez pusty ciąg znaków. Ponieważ `.` nie dopasowuje się do znaku końca linii `\n` musimy wykasować pustą linię. Dokonuje tego kolejna komenda. Z kolei komenda `p` potrzebna jest do drukowania ze względu na flagę `-n`.

```
#!/bin/bash
# This script removes #-type comments
# between 'begin' and 'end' words,
# located between 'start' and 'stop'.
sed -n '
  /start/,/stop/ {
    /begin/,/end/ {
         s/#.*//
         /^$/ d
         p
    }
  }'
```

W kolejnym przykładzie zamieniamy każde wystąpienie `old` na wystąpienie `new` pomiędzy liniami zawierającym `begin` oraz `end`, ale nie w tych liniach.

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