Edytor strumieniowy sed

Edytor strumieniowy sed działa poprzez poddawanie edycji danych wejściowych w sposób jaki zażąda tego użytkownik. Komendy i akcje w tym edytorze podejmowane są dla każdego wiersza po kolei. Rezultaty jego pracy są kierowane na standardowe wyjście (stdout), nie modyfikując oryginalnych plików wejściowych.

Przykład 1

```
$ sed -e 'd' /zadanie/text.txt
```

Jeżeli wykonamy to polecenie, to na wyjściu nic nie otrzymamy. Ponieważ w tym przykładzie wywołaliśmy polecenie sed z jednym z poleceń edycyjnych – d. Czyli sed otworzy plik /zadanie/text.txt, wczyta wiersz do bufora, wykona wskazane operacje (u nas "d" - usuń wiersz) i wypisze zawartość bufora (w naszym przykładzie bufor będzie pusty!). Takie czynności będą powtarzane dla każdego kolejnego wiersza. Nic nie otrzymamy na wyjściu, ponieważ usunęliśmy każdy wiersz!

Uwaga:

- plik /zadanie/text.txt nie został zmodyfikowany podczas pracy, ponieważ sed tylko czyta z pliku używając go tylko jako wejścia danych i nie próbuje go modyfikować (modyfikuje to co wczyta a nie plik wejściowy).
- sed jest narzędziem pracującym na wierszach. Polecenie d nie oznaczało po prostu usunięcie wszystkiego naraz, tylko sed wczytywał linijkę po linijce do bufora (zwanego buforem wzorca), a następnie wykonywał polecenie d i wypisywał zawartość bufora.
- zamknięcie polecenia d w pojedynczych apostrofach jest dobrym zwyczajem i nie powoduje kolizji z poleceniami konsolowymi.

Przykład 2

```
$ sed -e '1d' /zadanie/text.txt
```

Polecenie to jest bardzo podobne do użytego wcześniej z d wyjątkiem jest to, że zawiera ono teraz 1. Tym razem użyliśmy polecenia d poprzedzonego adresem. Poprzez użycie adresowania możemy przeprowadzić edycję tylko wybranych linii.

Przykład 3

Jeśli chcemy usunąć linijki od 1 do 3 to wpiszemy:

```
$ sed -e '1,3d' /zadanie/text.txt
```

Sed traktuje jako zakres dwa adresy oddzielone przecinkiem. Wtedy też operacje zostaną przeprowadzone na danym zakresie. Zakres rozpoczyna się od pierwszego adresu, kończy na drugim. W przykładzie polecenie d zostało zastosowane dla wierszy od 1 do 3 włącznie. Inne wiersze zostały pominięte.

Przykład 4

Jeśli chcemy obejrzeć plik `/zadanie/text.txt`, ale nie jesteśmy zainteresowani oglądaniem komentarzy (komentarze w pliku to linie zaczynające się od znaku '#'). Aby

pominąć komentarze wystarczy usunąć każdą linijkę zaczynającą się od znaku '#'. (Wstaw do swojego pliku znak '#' na początku wybranych trzech linii.)

```
$ sed -e '/^#/d' /zadanie/text.txt
```

Aby zrozumieć składnię '/^#/d' podzielmy to polecenie na kawałki. Najpierw usuńmy 'd' - kasowanie wiersza. Zostaje nam '/^#/' - jest to rodzaj adresowania z użyciem wyrażeń regularnych. Takie wyrażenie jest zawsze zamknięte między '/'. Definiuje ono wzorzec, a do wszystkich pasujących linii zostanie zastosowane polecenie znajdujące się za nim (u nas 'd' - usunięcie).

Przypomnij sobie wyrażenia regularne (tabele rozdane na zajęciach)
Kilka przykładów:

Wyrażenie regularne	**Opis**
`/./`	Oznacza wiersz, który zawiera co najmniej jeden znak
`/../`	Oznacza wiersz zawierający przynajmniej dwa znaki
`/^#/`	Oznacza wiersz zaczynający sie od '#'
`/^$/`	Oznacza wiersz pusty
`/}^/`	Oznacza wiersz, który kończy się '}' (bez spacji)
`/} *^/`	Oznacza wiersz kończący się '}' po której występuje zero lub więcej spacji
`/[abc]/`	Oznacza wiersz zawierający 'a', 'b' lub 'c'
`/^[abc]/`	Oznacza wiersz, który zaczyna się od 'a', 'b' lub 'c'

Przykład 5

```
$ sed -e '/Ala/d' /zadanie/text.txt
```

To polecenie spowoduje, że sed usunie każdy wiersz pasujący do wzorca (czyli usunie każdy wiersz zawierający słowo Ala). Jednak łatwiej było by wypisać pasujące wiersze, a te nie pasujące usunąć.

```
$ sed -n -e '/Ala/p' /zadanie/text.txt
```

Nowa opcja, '-n', nakazuje poleceniu sed nie wypisywać wzorca, dopóki nie zostanie wydane odpowiednie polecenie. Łatwo także zauważyć, że zastąpiliśmy d poleceniem p, które wymusza wypisanie wiersza pasującego do wzorca.

Przykład 6

Do tego miejsca potrafimy adresować wiersz, adresować zakres wierszy i wykorzystać wyrażenie regularne do adresowania, ale możliwości jest więcej. Można zdefiniować dwa wyrażenia regularne oddzielone przecinkiem, a sed edytuje wszystkie wiersze znajdujące się między pierwszym a drugim wyrażeniem włącznie. Na przykład następujące polecenie wyświetli blok tekstu zaczynający się od wiersza zawierającego "POCZATEK", a skończywszy na wierszu zawierającym "KONIEC":

```
$ sed -n -e '/POCZATEK/,/KONIEC/p' /zadanie/text.txt
```

Jeżeli "POCZATEK" nie zostanie odnaleziony nic nie zostanie wyświetlone. Natomiast jeżeli sed odnajdzie "POCZATEK", ale nie odnajdzie "KONIEC", otrzymamy wszystkie wiersze od

miejsca odnalezienia do końca. Dzieje się tak, ponieważ sed jest zorientowany strumieniowo, czyli nigdy nie wie kiedy nastąpi "KONIEC".

Przykład 7

```
$ sed -e 's/cos/cosinnego/' /zadanie/text.txt
```

Powyższe polecenie wypisze zawartość pliku /zadanie/text.txt na standardowe wyjście, zastępując w każdej linii pierwsze wystąpienie napisu "cos" (o ile w ogóle wystąpi) napisem "cosinnego". Zauważmy, że jest to pierwsze wystąpienie w każdej linii, pomimo iż zwykle nie o to nam chodzi. Najczęściej gdy wykonujemy podstawienie napisu, chcemy tego dokonać globalnie. Oznacza to, że chcemy zastąpić wszystkie wystąpienia napisu w każdej linii, w ten sposób:

```
$ sed -e 's/cos/cosinnego/g' /zadanie/text.txt
```

Dodatkowa opcja 'g' po ostatnim ukośniku wymusza, aby edytor sed wykonał globalne podstawienie.
Powinniśmy wiedzieć jeszcze kilka rzeczy o komendzie s///. Po pierwsze, możemy użyć polecenia s/// wraz z adresem, aby kontrolować, do których wierszy będzie ona zastosowana. Robimy to w następujący sposób:

```
$ sed -e '1,10s/zaczarowanie/zablokowanie/g' /zadanie/text.txt
```

Powyższy przykład sprawi, że wszystkie wystąpienia wyrazu "zaczarowanie" zostaną zastąpione wyrazem "zablokowanie", ale tylko w wierszach od pierwszego do dziesiątego włącznie.

```
$ sed -e '/^$/,/^END/s/wzgórza/góry/g' /zadanie/text.txt
```

W tym przykładzie wyraz "wzgórza" zostanie zastąpiony wyrazem "góry", ale tylko w tych blokach tekstu, które zaczynają się pustą linią, a kończą linią zaczynającą się od liter "END" włącznie.
Ważną rzeczą dotyczącą komendy s/// jest mnogość opcji dotyczących separatorów /. Jeśli chcemy dokonać podstawienia napisów i wyrażenie regularnie lub napis, który chcemy zmienić zawiera wiele ukośników, możemy zmienić separator, podając inny znak po literze "s". Zilustrujmy to przykładem, w którym zamienimy wszystkie wystąpienia /usr/local na /usr:

```
$ sed -e 's:/usr/local:/usr:g' /zadanie/text.txt
```

W powyższym przykładzie użyliśmy dwukropka jako separatora. Jeśli musielibyśmy użyć znaku separatora w wyrażeniu regularnym, należałoby poprzedzić go odwrotnym ukośnikiem.

Przykład 8

Do tej pory wykonywaliśmy jedynie podstawienia zwykłych napisów. Mimo iż często się to przydaje, to jednak możemy zrobić to samo dla wyrażeń regularnych. Poniższa komenda znajdzie frazę rozpoczynającą się znakiem "<" i kończącą się znakiem ">", posiadającą dowolną ilość znaków pomiędzy nimi. Fraza ta zostanie skasowana (zastąpiona pustym napisem):

```
$ sed -e 's/<.*>//g' /zadanie/text.txt
```

Jest to dobry początek skryptu, który usunie tagi języka HTML z pliku, jednakże nie zadziała dobrze ze względu na pewną właściwość wyrażeń regularnych. Mianowicie gdy sed szuka wyrażenia regularnego w linii, znajduje najdłuższe możliwe dopasowanie w tej linii. Do tej pory nie mieliśmy z tym problemów, ponieważ używaliśmy komend d i p, które tak czy inaczej skasują lub wypiszą całą linię. W przypadku komendy s/// nie odpowiada nam to, ponieważ cały dopasowany fragment tekstu zostanie zastąpiony przez docelowy napis, a w tym przypadku skasowany. Oznacza to, że powyższy przykład zamieni następującą linię:

```
<b>Właśnie</b> o to <b>mi</b> chodziło.
```

W tę:

```
chodziło.
```

Zamiast w poniższą, czyli w to, co chcieliśmy osiągnąć:

```
Właśnie o to mi chodziło.
```

Więc, zamiast wyrażenia regularnego, które znajdzie "dowolną ilość znaków, poprzedzoną znakiem '<' i zakończoną znakiem '>'" musimy użyć takiego, które odnajdzie "dowolną ilość znaków różnych od znaku '>', poprzedzoną znakiem '<' i zakończoną znakiem '>'". W ten sposób znajdziemy najkrótsze możliwe dopasowanie, zamiast najdłuższego. Nowe polecenie powinno wyglądać tak:

```
$ sed -e 's/<[^>]*>//g' /zadanie/text.txt
```

W powyższym przykładzie wyrażenie "[^>]" oznacza "znak różny od '>'", a symbol "*" uzupełnia je o znaczenie "zero lub więcej znaków różnych od '>'".

Przykład 9

Składnia wyrażenia regularnego "[ ]" oferuje jeszcze kilka możliwości. Możemy określić zakres znaków za pomocą symbolu "-", o ile nie znajduje się on na pierwszym lub ostatnim miejscu:

```
'[a-x]*'
```

W ten sposób wyszukamy zero lub więcej znaków, dopóki wszystkie należeć będą do zbioru "a","b","c"..."v","w","x". Oprócz tego dysponujemy listą dostępnych klas znakowych:

Klasa znakowa	**Opis**
`[:alnum:]`	Znaki alfanumeryczne [a-z A-Z 0-9]
`[:alpha:]`	Znaki alfabetyczne [a-z A-Z]
`[:blank:]`	Spacje lub tabulatory
`[:cntrl:]`	Dowolny znak kontrolny
`[:digit:]`	Cyfry [0-9]
`[:graph:]`	Znaki drukowalne (bez odstępów)
`[:lower:]`	Małe litery [a-z]
`[:print:]`	Znaki drukowalne z odstępami

[:punct:]	Znaki drukowalne za wyjątkiem odstępów, liter i cyfr
[:space:]	Wszystkie znaki odstępu
[:upper:]	Duże litery [A-Z]
[:xdigit:]	Cyfry w systemie szesnastkowym [0-9 a-f A-F]

Używanie klas znakowych wszędzie tam, gdzie to możliwe jest wysoce pożądane, ponieważ znakomicie dopasowują się one do innych niż angielskie zestawów znaków (uwzględniając znaki akcentowane, itd).

Przykład 10

Do tej pory przyjrzeliśmy się prostym i w miarę złożonym podstawieniom, ale sed potrafi jeszcze więcej. Możemy odnosić się do dowolnej części lub całości tekstu odnalezionego przez wyrażenie regularne i użyć jej do stworzenia podstawianego napisu. Przykładowo, załóżmy że chcemy odpowiedzieć na wiadomość. Poniższa komenda poprzedzi każdą linię wyrażeniem "ralph powiedział: ":

```
$ sed -e 's/.*/ralph powiedział: &/' /zadanie/text.txt
```

Oto wynik:

```
ralph powiedział: Cześć Jim,
ralph powiedział:
ralph powiedział: Bardzo podoba mi się ta lekcja seda!
ralph powiedział:
```

W powyższym przykładzie wykorzystaliśmy znak "&" w podstawianym napisie. Nakazuje on wstawienie w tym miejscu całego tekstu znalezionego przez wyrażenie regularne. Tak więc cokolwiek zostało znalezione przez ".*" (największa grupa zera lub więcej znaków w linii lub cała linia) może zostać podstawione w dowolnym miejscu podstawianego tekstu nawet kilka razy.

Przykład 11

Komenda s/// pozwala nam na definiowanie regionów w wyrażeniu regularnym, do których możemy się następnie odwoływać w podstawianym tekście. W ramach przykładu przyjrzyjmy się następującemu tekstowi:

```
bla ble bli
ele mele dudki
ala ma kota
kot ma alę
```

Załóżmy teraz, że chcemy napisać polecenie, które zamieni "ele mele dudki" na "Victor ele-mele Von dudki" i tak dalej. Aby tego dokonać najpierw powinniśmy napisać wyrażenie regularne, które odnajdzie nasze trzy wyrazy oddzielone spacjami:

```
'.* .* .*'
```

Następnie zdefiniujmy regiony, umieszczając wokół każdego interesującego nas miejsca nawiasy poprzedzone wstecznym ukośnikiem:

```
'\(.*\) \(.*\) \(.*\)'
```

Powyższe wyrażenie regularne zadziała indentycznie jak pierwsze. Jedyną różnicą będzie fakt, iż definiuje ono trzy logiczne regiony, do których będziemy mogli odnieść się w naszym podstawianym napisie. Oto ostateczny wynik:

```
$ sed -e 's/\(.*\) \(.*\) \(.*\)/Victor \1-\2 Von \3/'
/zadanie/text.txt
```

Jak widać, odnosimy się do każdego obszaru ograniczonego nawiasami za pomocą wyrażenia "\x", gdzie x jest numerem regionu, zaczynając od jedynki. Efekt działania skryptu będzie następujący:

```
Victor bla-ble Von bli
Victor ele-mele Von dudki
Victor ala-ma Von kota
Victor kot-ma Von alę
```

Przykład 12

Gdy zaczynamy tworzyć bardziej złożone polecenia, potrzebujemy możliwości wpisywania więcej niż jednej komendy naraz. Można tego dokonać na kilka sposobów. Po pierwsze, możemy rozdzielić komendy średnikami. Na przykład ten zestaw poleceń używa komendy "=", która nakazuje wypisanie numer linii, oraz komendy p, która specjalnie instruuje program sed, aby wypisał linię (ponieważ pracujemy w trybie "-n"):

```
$ sed -n -e '=;p' /zadanie/text.txt
```

Gdy podajemy dwie lub więcej komend, każda z nich jest zastosowana (po kolei) do każdej linii pliku. W powyższym przykładzie na pierwszej linii zostanie użyta komenda "=", a następnie p. Sed wówczas przechodzi do drugiej linii i proces powtarza się. Pomimo iż średnik okazał się być przydatny, w niektórych sytuacjach nie zadziała. Alternatywą będzie użycie dwóch parametrów -e w celu podania dwóch osobnych komend:

```
$ sed -n -e '=' -e 'p' /zadanie/text.txt
```

Jednak gdy tylko dojdziemy do bardziej złożonych poleceń dodawania i wstawiania, nawet wiele parametrów "-e" nam nie pomoże. Wówczas najlepiej zrobimy zapisując wszystkie komendy w osobnym pliku, czyli każde polecenie w apostrofach wstawiamy w osobnej linii i bez apostrofu:

```
=
p
```

a następnie wywołujemy skrypt parametrem -f:

```
$ sed -n -f skrypt test.txt
```

Ten sposób, choć mniej wygodny, nigdy nas nie zawiedzie.

Przykład 13

Czasem zajdzie potrzeba wykonania kilku poleceń na jednym przedziale. Szczególnie jest to przydatne gdy chcemy wykonać wiele komend s/// w celu przekształcenia wyrazów lub składni w pliku źródłowym. Aby wykonać wiele komend na jednym adresie zapiszmy je w pliku i użyjmy znaków "{ }" do pogrupowania ich:

```
1,20{
        s/[Ll]inux/GNU\/Linux/g
        s/samba/Samba/g
        s/posix/POSIX/g
}
```

Powyższe polecenia zastosują trzy podstawienia w liniach od pierwszej do dwudziestej włącznie. Jako adresów moglibyśmy również użyć wyrażenia regularnego lub połączyć jedno z drugim:

```
1,/^END/{
        s/[Ll]inux/GNU\/Linux/g
        s/samba/Samba/g
        s/posix/POSIX/g
        p
}
```

W powyższym przykładzie wszystkie komendy między znakami "{ }" zostaną użyte na tekście od pierwszej linii aż po linię, która zaczyna się literami "END" lub, w przypadku braku takowej, aż po koniec pliku.

Przykład 14

Teraz gdy umiemy już pisać skrypty do edytora sed w osobnych plikach, możemy wykorzystać polecenia dołączania, wstawiania i zmiany linii. Polecenia te wstawią wiersz po aktualnej linii, przed aktualną linią lub zastąpią ją. Można ich także używać do wstawiania wielu wierszy jednocześnie. Komendy wstawiania używa się w następujący sposób:

```
i\
Ten wiersz zostanie wstawiony przed każdą linią
```

Jeśli nie podamy adresu dla tej komendy, zostanie ona zastosowana do każdej linii, a wynik jej działania będzie podobny do poniższego:

```
Ten wiersz zostanie wstawiony przed każdą linią
oto linia 1
Ten wiersz zostanie wstawiony przed każdą linią
oto linia 2
Ten wiersz zostanie wstawiony przed każdą linią
oto linia 3
Ten wiersz zostanie wstawiony przed każdą linią
oto linia 4
```

Jeśli zechcemy wstawić więcej niż jeden wiersz przed aktualną linią, wystarczy że dopiszmy je, dodając odwrotny ukośnik na końcu każdego poprzedniego wiersza:

```
i\
wstaw ten wiersz\
i ten\
i ten\
o, nie zapomnijmy tez o tym.
```

Komenda dołączania działa podobnie, tylko dołącza wiersz lub wiersze po aktualnej linii. Oto przykład:

```
a\
proszę wstawić ten wiersz po każdej linii. Z góry dziękuję! :)
```

Z kolei polecenie "zmień linię" podmieni ją na nową.
Ze względu na to, iż komendy dołączania, wstawiania i zamiany linii muszą być podawane w kilku wierszach, należy wpisywać je w plikach tekstowych i podawać edytorowi sed za pomocą parametru "-f". Pozostałe metody wpisywania tych komend zaowocują problemami.