Sed w przykładach, część druga

Spis treści: 1. Jak jeszcze lepiej wykorzystać edytor tekstowy systemu UNIX ▾

1. Jak jeszcze lepiej wykorzystać edytor tekstowy systemu UNIX

Podstawianie!

Przyjrzyjmy się jednej z najbardziej przydatnych komend edytora sed, podstawieniu. Przy jej pomocy możemy zastąpić określony ciąg znaków lub dopasowane wyrażenie regularne innym ciągiem znaków. Oto przykład najbardziej elementarnego zastosowania tej komendy:

Listing 1.1: Najbardziej elementarne zastosowanie komendy podstawienia

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

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

Listing 1.2: Zamiana wszystkich wystąpień w każdej linii

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

Dodatkowa opcja 'g' po ostatnim ukośniku mówi sedowi, aby wykonał globalne podstawienie.

Powinniśmy wiedzieć jeszcze kilka rzeczy o komendzie `s///`. Po pierwsze, mówiliśmy wyłącznie o komendzie; w powyższych przykładach nie podawaliśmy adresów. Oznacza to, że możemy użyć polecenia `s///` wraz z adresem, aby kontrolować, do których wierszy będzie ona zastosowana. Robimy to w następujący sposób:

Listing 1.3: Określanie wierszy, do których komenda zostanie zastosowana

```
$ sed -e '1,10s/zaczarowanie/zablokowanie/g' mojplik2.txt
```

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

Listing 1.4: Podawanie dalszych opcji

```
$ sed -e '/^$/,/^END/s/wzgórza/góry/g' mojplik3.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.

Kolejną miłą 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`:

Listing 1.5: Zamiana wszystkich wystąpień napisu innym napisem

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

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

Pułapki wyrażeń regularnych

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):

Listing 1.6: Kasowanie podanej frazy

```
$ sed -e 's/<.*>//g' mojplik.html
```

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. Nie było to problemem w poprzednim artykule o sedzie, 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:

Listing 1.7: Przykładowy kod HTML

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

W tę:

Listing 1.8: Niepożądany wynik

```
chodziło.
```

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

Listing 1.9: Pożądany wynik

```
Właśnie o to mi chodziło.
```

Na szczęście da się temu zaradzić. 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:

Listing 1.10:

```
$ sed -e 's/<[^>]*>//g' mojplik.html
```

W powyższym przykładzie wyrażenie "[^>]" oznacza "znak różny od '>'", a symbol "*" uzupełnia je o znaczenie "zero lub więcej znaków różnych od '>'". Warto wypróbować to polecenie na kilku przykładowych plikach html, przeglądając wyniki za pomocą programu more.

Dopasowanie większej ilości znaków

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:

Listing 1.11: Podawanie zakresu znaków

```
'[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 klasą znakową "[:space:]" do znajdywania odstępu. Poniżej znajduje się prawie kompletna lista 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).

Zaawansowane podstawianie

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: ":

Listing 1.12: Poprzedzanie każdej linii określonym napisem

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

Oto wynik jej działania:

Listing 1.13: Efekt działania powyższej komendy

```
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. Mówi on sedowi, aby wstawić w tym miejscu cały tekst znaleziony 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. Całkiem nieźle, ale sed potrafi jeszcze więcej.

Dyskretny urok nawiasów poprzedzonych odwrotnymi ukośnikami

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:

Listing 1.14: Przykładowy tekst

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

Załóżmy teraz, że chcemy napisać skrypt w sedzie, który 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:

Listing 1.15: Odpowiednie wyrażenie regularne

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

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

Listing 1.16: Definiowanie regionów

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

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 skrypt:

Listing 1.17: Ukończony skrypt

```
$ sed -e 's/\(.*\) \(.*\) \(.*\)/Victor \1-\2 Von \3/' mojplik.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:

Listing 1.18: Wynik działania powyższej komendy

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

W miarę jak poznajemy seda coraz bliżej, możliwe staje się dokonywanie skomplikowanych operacji na tekście przy minimum wysiłku. Warto zastanowić się nad rozwiązaniem powyższego problemu za pomocą dowolnego języka skryptowego -- czy udałoby nam się równie łatwo zmieścić rozwiązanie w jednej linijce?

Zamieszanie w komendach

Gdy zaczynamy tworzyć bardziej złożone skrypty, 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 każe sedowi wypisać numer linii, oraz komendy p, która specjalnie instruuje seda, aby wypisał linię (ponieważ pracujemy w trybie "-n"):

Listing 1.19: Pierwszy sposób, średniki

```
$ sed -n -e '=;p' mojplik.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:

Listing 1.20: Drugi sposób, wiele parametrów -e

```
$ sed -n -e '=' -e 'p' mojplik.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, a następnie wywołując go parametrem -f:

Listing 1.21: Trzeci sposób, zewnętrzny plik zawierający komendy

```
$ sed -n -f mojekomendy.sed mojplik.txt
```

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

Wiele komend wykonanych na jednym adresie

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:

Listing 1.22: Podawanie wielu komend dla jednego adresu

```
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:

Listing 1.23: Połączenie obu sposobów

```
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.

Dołączanie, wstawianie i zmienianie linii

Teraz gdy już piszemy skrypty seda 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:

Listing 1.24: Użycie komendy wstawiania wiersza

```
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:

Listing 1.25: Efekt działania powyższego polecenia

```
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:

Listing 1.26: Wstawianie wielu wierszy przed aktualną linią

```
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:

Listing 1.27: Dołączanie wiersza po aktualnej linii

```
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 sedowi za pomocą parametru "-f". Pozostałe metody wpisywania komend zaowocują problemami.

W następnym odcinku

W następnym artykule, który będzie już ostatnim na temat seda, ukazane zostaną znakomite przykłady z życia wzięte na zastosowanie seda do wielu różnych zadań. Nie tylko dowiemy się co skrypty robią, ale też w jaki sposób działają. Z pewnością zaowocuje to wieloma pomysłami na zastosowanie seda w rozmaitych projektach. Do zobaczenia w następnym odcinku!

2. Zasoby

Przydatne odnośniki

- Warto przeczytać pozostałe artykuły Daniela na temat seda, napisane dla develperWorks: Sed w przykładach, Część pierwsza i Część trzecia.

- Eric Pement napisał znakomity dokument FAQ na temat seda.
- Źródła seda można znaleźć pod adresem ftp://ftp.gnu.org/pub/gnu/sed.
- Najnowsza wersja dostępna jest na stronie ftp://alpha.gnu.org.
- Na stronie Erica Pementa można znaleźć również przydatny zestaw jednolinijkowych skryptów w sedzie, którymi powinien zainteresować się każdy, kto chciałby zostać autorytetem w dziedzinie tego języka.
- Dla tych, którzy preferują klasyczne książki znakomitym wyborem będzie "sed i awk" z wydawnictwa O'Reilly.
- Dzięki darmowemu samouczkowi napisanemu specjalnie dla developerWorks możemy podszkolić się w używaniu wyrażeń regularnych, aby łatwiej znajdywać i modyfikować wzorce w tekście.