Sed w przykładach, część pierwsza

Spis treści: 1. Poznać porządny UNIX-owy edytor ▾

1. Poznać porządny UNIX-owy edytor

Wybierz edytor...

W świecie UNIX-ów mamy spory wybór jeżeli chodzi o edycję plików. Jest vi, emacs, jed i masa innych. Każdy ma swój ulubiony edytor. Z jego pomocą z łatwością poradzi sobie z zadaniami jakie czekają podczas programowania, czy administracji UNIX-em.

Edytory interaktywne są świetne, ale mają pewne ograniczenia. Ich główna cecha - oddziaływanie przez użytkownika podczas edycji - może okazać się ich słabością. Szczególnie w sytuacji, gdy potrzebujemy wprowadzić małe zmiany w pewnej grupie plików. Wypada wtedy za każdym razem odpalać nasz edytorek, wprowadzić żądane zmiany - wszystko ręcznie. Ale istnieje lepsze rozwiązanie ...

Przedstawienie Seda

Było by miło, gdyby można było zautomatyzować proces edycji istniejących plików przez polecenie w konsoli lub napisanie prostego skryptu. Na szczęście w takich sytuacjach istnieje lepsza droga niż ręczna edycja - sed.

sed jest małym edytorem strumieniowym, który można znaleźć we wszelkiej maści UNIX-ach (czyli także w Linuksie). Sed ma wiele przydatnych funkcji. Po pierwsze zajmuje naprawdę niewiele. Po drugie jest to edytor strumieniowy, dzięki czemu można przeprowadzać edycję na danych otrzymanych z stdin (takich jak pipe). Nie jest konieczne, aby te dane były zapisywane na dysku. Ponieważ dane można łatwo "wpuścić" przez strumień wejścia do seda, łatwe staje się używanie seda pod konsolą lub skryptach. Spróbuj zrobić to samo ze swoim ulubionym edytorem...

GNU sed

Dobrą wiadomością dla użytkowników Linuksa jest to, że jedna z lepszych wersji seda wydana jest jako GNU sed. Każda dystrybucja Linuksa zawiera seda (lub chociaż powinna). Na popularność GNU seda wpływa nie tylko jego otwarty charakter, ale również fakt, że zawiera wiele rozszerzeń które oszczędzają sporo czasu. GNU sed jest także pozbawiony wielu ograniczeń, które miała wcześniejsza i główna wersja (np. limit długości wiersza - GNU sed poradzi sobie z linią każdej długości. ;)

Najnowszy GNU sed

Podczas pisania tego artykułu zauważyłem, że niektóre serwisy internetowe odwołują się do wersji 3.02a GNU seda. Co dziwne - nie mogłem znaleźć jej na ftp://ftp.gnu.org (patrz materiały). Szukając w innych źródłach, znalazłem tę wersję na ftp://alpha.gnu.org w `/pub/sed`. Szczęśliwy ściągnąłem, skompilowałem i zainstalowałem tą wersję. Jak okazało się tylko po to, aby za parę minut okazało się, że najnowszą wersją seda jest 3.02.80. Można ją znaleźć tuż obok 3.02a na ftp://alpha.gnu.org. W końcu po zainstalowaniu wersji 3.02.80 byłem gotowy do pracy.

Praca z sedem na całego

W tej serii artykułów będziemy używać GNU sed 3.02.80. Niektóre z przykładów mogą nie działać poprawnie na starszych wersjach GNU sed. W przypadku używania nie-GNU seda można napotkać spore utrudnienia. Wystarczy poświęcić trochę czasu na zainstalowanie najnowszej wersji GNU seda. ;) Można zyskać nie tylko pewność, że zadziałają wszystkie przykłady, ale i dostać do użytku najlepszą wersję wspaniałego edytora tekstu.

Przykłady użycia seda

Sed działa poprzez poddawanie edycji danych wejściowych w sposób jaki zażąda tego użytkownik. Komendy i akcje w sedzie podejmowane są dla każdego wiersza po kolei. Wyrzuca on rezultat pracy na standardowe wyjście (stdout), nie modyfikując oryginalnych plików wejściowych.

Rzućmy okiem na parę przykładów. W pierwszym przykładzie pokażę po prostu jak działa sed. Na początku ważne jest zrozumienie sposobu działania seda, później przyjdzie czas na jego efektywne wykorzystanie. Oto pierwszy przykład:

Listing 1.1: Przykład użycia seda

```
$ sed -e 'd' /etc/services
```

Jeżeli wykonamy to polecenie, nie otrzymamy nic na wyjściu. Dlaczego? W tym przykładzie wywołaliśmy seda z jednym z poleceń edycyjnych - d Sed otworzy plik `/etc/services`, 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. Nie otrzymamy nic na wyjściu, ponieważ za pomocą polecenia d usunęliśmy każdy wiersz!

Jest jeszcze parę spraw do omówienia w tym przykładzie. Po pierwsze - plik`/etc/services` nie został tak naprawdę zmodyfikowany. Stało się tak, ponieważ sed tylko czyta z pliku - używa go jako wejścia danych nie próbując go modyfikować. Drugą sprawą jest fakt, że sed jest zorientowany wierszowo. Polecenie d nie oznaczało po prostu usunięcie wszystkiego naraz - sed wczytywał linijkę po linijce do bufora (zwanego buforem wzorca), a następnie wykonywał polecenie d i wypisywał zawartość bufora (u nas nic). Później dowiemy się w jaki sposób używać adresowania zakresów aby móc kontrolować dla których wierszy zostanie wykonane polecenie d.

Trzecią rzeczą o której należy wspomnieć, jest zamknięcie polecenia d w pojedynczych apostrofach. Takie zamykanie poleceń w sedzie jest dobrym zwyczajem i nie powoduje kolizji z poleceniami konsolowymi.

Inny przykład zastosowania seda

Oto przykład użycia seda do usunięcia pierwszej linii z pliku`/etc/services`:

Listing 1.2: Inny przykład zastosowania seda

```
$ sed -e '1d' /etc/services | more
```

Jak łatwo zauważyć, polecenie to jest bardzo podobne do użytego wcześniejd z wyjątkiem tego, że zawiera ono teraz 1. Tym razem użyliśmy polecenia d poprzedzonego adresem. Poprzez użycie adresowania możemy przeprowadzić edycję tylko wybranych linii.

Adresowanie

Spróbujmy teraz przyjrzeć się w jaki sposób zaadresować pewien zakres. W tym przykładzie użyjemy seda do usunięcia linijek od 1 do 10 :

Listing 1.3: Adresowane zakresu

```
$ sed -e '1,10d' /etc/services | more
```

Kiedy oddzielamy dwa adresy przecinkiem sed traktuje to jako zakres na którym przeprowadzi żądane operacje. Zakres rozpoczyna się od pierwszego adresu, kończy na drugim. W przykładzie polecenie d zostało zastosowane dla wierszy od 1 do 10 włącznie. Inne wiersze zostały pominięte.

Adresowanie z pomocą wyrażeń regularnych

Nadszedł czas na bardziej zaawansowany przykład. Powiedzmy, że chcemy obejrzeć plik `/etc/services`, ale nie jesteśmy zainteresowani oglądaniem komentarzy. Jak wiadomo komentarze w pliku `/etc/services`, to linie zaczynające się od znaku '#'. Aby pominąć komentarze wystarczy usunąć każdą linijkę zaczynającą się od znaku '#'. Można to zrobić tak :

Listing 1.4: Usuwanie linii zaczynających się od #

```
$ sed -e '/^#/d' /etc/services | more
```

Odpalmy ten przykład i zobaczmy co otrzymamy. Zauważymy, że sed wykonał swoje zadanie poprawnie.

Aby zrozumieć składnię '/^#/d' podzielmy to polecenie na kawałki. Najpierw usuńmy 'd' -- tak jak było to powiedziane wcześniej, jest to kasowanie wiersza. Zostaje nam '/^#/' - jest to nowy 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)

Więc '/^#/' jest wyrażeniem regularnym. Ale co to robi? To chyba jest odpowiedni moment, aby opowiedzieć szerzej o wyrażeniach regularnych.

Odświezenie wiadomości o wyrażeniach regularnych

Można użyć wyrażeń regularnych, aby określić wzorzec, który chcemy znaleźć w tekście. Jeżeli ktoś używał znaku '*' w konsoli - używał czegoś co jest podobne do wyrażeń regularnych. Poniżej znajduje się lista specjalnych znaków, które mogą zostać użyte w wyrażeniach.

Znaki	Opis
^	Oznacza początek wiersza
$	Oznacza koniec wiersza
.	Oznacza pojedynczy znak
*	Oznacza zero lub więcej wystąpień poprzedniego znaku
[ ]	Wszystkie znaki pomiędzy [i]

Chyba najlepszym sposobem na sprawne wejście w świat wyrażeń regularnych jest zobaczenie kilku przykładów w akcji. Wszystkie będą zaakceptowane przez seda jako poprawny sposób adresowania. Oto 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'

Zachęcam do wypróbowania kilku przykładów na własnej skórze. Potrzeba trochę czasu na oswojenie się z wyrażeniami regularnymi. A oto sposób ich poprawnego użycia:

Listing 1.5: Poprawne użycie wyrażenia regularnego

```
$ sed -e '/regexp/d' /sciezka/do/mojego/testowego/pliku | more
```

To spowoduje, że sed usunie każdy wiersz pasujący do wzorca. Jednak łatwiej było by wypisać pasujące wiersze, a te nie pasujące usunąć. Można tego dokonać poleceniem:

Listing 1.6: Wypisywanie pasujących wierszy

```
$ sed -n -e '/regexp/p' /sciezka/do/mojego/pliku | more
```

Zwróćmy uwagę na nową opcję, '-n', która nakazuje sedowi nie wypisywać wzorca, dopóki nie zostanie wydane odpowiednie polecenie. Łatwo także zauważyć, że zastąpiliśmy `d` poleceniem `p`, które to nakazuje sedowi wypisanie wiersza pasującego do wzorca.

Więcej o adresowaniu

Na razie powiedzieliśmy parę słów o adresowaniu wierszy, adresowaniu zakresów i wykorzystaniu wyrażeń regularnych 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":

Listing 1.7: Wyświetlenie opisanego bloku tekstu

```
$ sed -n -e '/POCZATEK/,/KONIEC/p' /moj/plik | more
```

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 - więc nigdy nie wie kiedy nastąpi "KONIEC".

Przykład ze źródłami C

Jeżeli ktoś chciałby zobaczyć jedynie zawartość funkcji main() w pliku źródłowym C, może napisać:

Listing 1.8: Podgląd funkcji main() z pliku źródłowego C

```
$ sed -n -e '/main[[:space:]]*(/,/^}/p' zlodlo.c | more
```

Polecenie zawiera dwa wyrażenia regularne '/main[[:space:]]*(/' i '/^}/' oraz polecenie p. Pierwsze wyrażenie regularne oznacza, że sed będzie szukał ciągu znaków "main" po którym może wystąpić dowolna ilość spacji i tabulatorów oraz otwarty nawias "(". To powinno nam pozwolić na znalezienie deklaracji funkcji main() zgodnie z ANSI C.

W tym wyrażeniu regularnym wystąpiła klasa znaków '[[:space:]]'. Jest to po prostu specjalne słowo-klucz które mówi sedowi aby brał pod uwagę TAB lub spacje. Zamiast pisać '[[:space:]]' można napisać '[', CTRL+V, tabulator i ']', ale chyba bardziej czytelna jest klasa znaków '[[:space:]]'.

Przejdźmy teraz do drugiego wyrażenia :'/^}/'. Sed będzie szukał znaku '}' na początku wiersza. Jeżeli plik źródłowy jest poprawnie sformatowany, sed powinien znaleźć zamykający nawias '}' funkcji main(). Jeżeli nie znajdzie - będzie wypisywał wszystko aż do końca pliku - taki jego urok.

Polecenie p robi to co zawsze - nakazuje sedowi wypisać linijkę (od kiedy mamy tryb "cichy" -n). Polecam wypróbowanie tego przykładu na kilku plikach C - powinniśmy dostać zawartość funkcji main() z "main()" i "}" włącznie.

W następnym odcinku

Na razie przedstawiłem tylko podstawy seda. Temat ten rozwinę w dwóch następnych. Polecam również lekturę materiałów na temat seda i wyrażeń regularnych, których lista znajduje się poniżej.

2. Materiały

Linki:

- Warto przeczytać pozostałe artykuły Daniela na temat seda, napisane dla develperWorks: Sed w przykładach, Część druga i Część trzecia.
- Eric Pement napisał znakomity dokument FAQ na temat seda.
- Tu można znaleźć źródła seda: 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.
- Artykuł Davida Mertza na temat przetwarzania tekstu w Pythonie.
- How-to na temat wyrażeń regularnych z Python.org.
- Przydatny przegląd wyrażeń regularnych na Uniwersytecie w Kentucky.