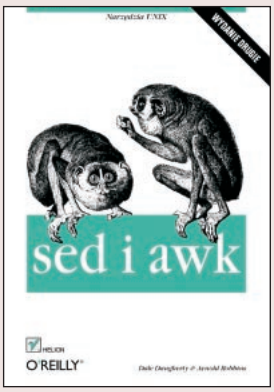
sed i awk



Autorzy: Dale Dougherty, Arnold Robbins
Tłumaczenie: Wojciech Derechowski
ISBN: 83-7197-540-6
Tytuł oryginału: sed & awk, Second Edition
Format: B5, stron: 384

Jest to książka o narzędziowych programach UNIX, nazywanych dziwnie sed i awk. Programy te mają ze sobą wiele wspólnego, zwłaszcza użycie wyrażeń regularnych dla dopasowywania wzorców. Ponieważ dopasowywanie wzorców jest tak ważne w stosowaniu obydwu programów, książka bardzo wyczerpująco wyjaśnia składnię wyrażeń regularnych UNIX. Skoro w uczeniu się naturalny jest postęp od grep, poprzez sed do awk, więc będziemy zajmować się wszystkimi trzema programami, choć skupimy się na sed i awk.

Sed i awk są narzędziami stosowanymi przez użytkowników, programistów i administratorów – przez każdego, kto pracuje z plikami tekstowymi. Sed, nazywany tak ze względu na to, że jest edytorem strumieniowym, doskonale nadaje się wprowadzania ciągu poprawek (ang. edits) do wielu plików. Awk, którego twórcami są Aho, Weinberger i Kernighan (skąd pochodzi nazwa) jest językiem programowania umożliwiającym łatwe działania na danych, które mają strukturę i tworzenie sformatowanych raportów. Książka kładzie nacisk na definicję POSIX dla awk. Prócz tego opisuje krótko pierwszą wersję awk przed omówieniem trzech wersji awk dostępnych bezpłatnie oraz dwóch będących w sprzedaży, z których wszystkie są implementacjami POSIX awk.

Książka skupia uwagę na pisaniu dla sed i awk skryptów, stanowiących szybkie rozwiązanie wybranych problemów użytkownika. Wiele ze skryptów tego rodzaju można określić jako „doraźne rozwiązanie”. Prócz tego opiszemy skrypty, które rozwiązują większe problemy, wymagają więc bardziej starannego projektowania i programowania.

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

2

Zrozumienie podstawowych działań

Zaczynając naukę sed i awk, warto zobaczyć, jak wiele wspólnych cech mają oba programy.

- Uruchamiane są przy użyciu podobnej składni.
- Oba są zorientowane strumieniowo, odczytują dane wejściowe z plików tekstowych po jednym wierszu na raz i kierują wynik na wyjście standardowe.
- Używają wyrażeń regularnych przy dopasowywaniu wzorców.
- Pozwalają użytkownikowi na dostarczanie instrukcji w postaci skryptu.

Jednym z powodów tylu podobieństw jest to, że źródeł obydwu programów należy szukać w tym samym edytorze wierszowym **ed**. Rozdział zaczynamy rzutem oka na **ed** i pokazujemy, że sed i awk zmierzały logicznie do stworzenia edytora programowalnego.

Tym, co różni sed i awk, jest rodzaj instrukcji, które kontrolują ich pracę. Nie łudźmy się — jest to wielka różnica i ma wpływ na rodzaje zadań, jakie najlepiej jest wykonywać za pomocą tych programów.

Obecny rozdział dotyczy składni wiersza poleceń sed i awk, oraz podstawowej struktury skryptów. Zawiera także ćwiczenie z użyciem listy adresowej, w którym będziemy mogli popróbować pisania skryptów. Warto przyjrzeć się skryptom sed i awk obok siebie, zanim skupimy się na którymś z tych programów.

Od ed poprzez grep i sed do awk

Rodowód awk można wywieść od sed i **grep**, i poprzez oba te programy — od **ed**, pierwszego edytora wierszowego UNIX.

Czy kiedykolwiek Czytelnik używał edytora wierszowego? Jeżeli tak, łatwiej zrozumie wierszową orientację sed i awk. Czytelnik, który używał **vi**, edytora ekranowego, zna liczne polecenia wywodzące się z leżącego u podłoża **vi** edytora wierszowego **ex** (który jest nadzbiorem własności należących do **ed**).

Przyjrzyjmy się pewnym podstawowym operacjom, używając edytora wierszowego **ed**. Bez obaw — jest to ćwiczenie, które ma nam pomóc w nauce sed i awk, a nie być próbą przekonania nas o urokach edytorów wierszowych. Polecenia **ed**, pokazane w tym ćwiczeniu, są identyczne z poleceniami **sed**, których Czytelnik nauczy się później. Można oczywiście eksperymentować z **ed** na własną rękę, aby przekonać się jak on działa. (Jeśli Czytelnik zna już **ed**, może pominąć poniższy punkt rozdziału.)

W edytorze wierszowym pracujemy nad jednym wierszem naraz. Ważne jest, by wiedzieć, przy którym wierszu w pliku jesteśmy. Kiedy otwieramy plik za pomocą **ed**, wyświetla on liczbę znaków pliku i ustawia się w ostatnim wierszu:

```
$ ed test
339
```

Nie ma zgłoszenia gotowości. Jeśli napiszemy polecenie, którego **ed** nie rozumie, drukowany jest znak zapytania jako komunikat o błędzie. Można napisać polecenie drukuj, **p**, by wyświetlić bieżący wiersz:

```
p
label on the first box.
```

Polecenie edytuje domyślnie tylko wiersz bieżący. Żeby zrobić poprawkę przechodzimy do wiersza, który chcemy edytować, a następnie stosujemy polecenie. Aby przejść do wiersza, podajemy jego *adres*. Adres może składać się z numeru wiersza, z symbolu wskazującego pewne miejsce w pliku bądź z wyrażenia regularnego. Do pierwszego wiersza przechodzimy, wpisując numer wiersza 1. Następnie możemy napisać polecenie usuń, aby ten wiersz usunąć:

```
1
You might think of a regular expression
d
```

Wpisanie "1" sprawia, że pierwszy wiersz staje się wierszem bieżącym i jest wyświetlany na ekranie. Poleceniem usuń w **ed** jest **d**, usuwa więc powyżej wiersz bieżący. Zamiast przechodzić do wiersza i następnie edytować ten wiersz, można zaopatrzyć polecenie edycji w przedrostek, adres, który wskazuje, jaki wiersz lub zakres wierszy jest przedmiotem polecenia. Jeżeli napiszemy "1d", zostanie usunięty pierwszy wiersz.

Jako adres można także podać wyrażenie regularne. Aby usunąć wiersz zawierający słowo "regular", wydajemy takie polecenie:

```
/regular/d
```

gdzie prawe ukośniki ograniczają wyrażenie regularne, a "regular" jest łańcuchem, który chcemy dopasowywać. Polecenie to usuwa pierwszy wiersz, który zawiera "regular", oraz sprawia, że wiersz następujący po nim staje się wierszem bieżącym.

Ważne jest, by rozumieć, że polecenie usuń usuwa wiersz w całości. Nie usuwa samego tylko słowa "regular" z tego wiersza.

Aby usunąć *wszystkie* wiersze zawierające wyrażenie regularne, polecenie zaopatruje się w przedrostek, literę **g** jak globalnie:

```
g/regular/d
```

Polecenie globalne sprawia, że wszystkie wiersze, które pasują do wyrażenia regularnego, stają się przedmiotem wyspecyfikowanych poleceń.

Oto, gdzie najdalej może nas doprowadzić usuwanie tekstu. Podstawianie tekstu (zastępowanie jednej cząstki tekstu przez inną) jest znacznie ciekawsze. Poleceniem podstaw, **s**, w **ed** jest:

```
[adres]s/wzorzec/zamiennik/znacznik_operacji
```

wzorzec jest wyrażeniem regularnym, które w bieżącym wierszu dopasowuje łańcuch, jaki należy zastąpić *zamiennikiem*. Na przykład, poniższe polecenie zastępuje pierwsze wystąpienie łańcucha "regular" przez "complex" w bieżącym wierszu:

```
s/regular/complex/
```

Adres nie jest wyspecyfikowany, więc to polecenie edytuje tylko pierwsze wystąpienie łańcucha w bieżącym wierszu. Jeśli "regular" nie daje się odszukać w bieżącym wierszu, jest to błąd. Aby wyszukiwać wielokrotne wystąpienia łańcucha w *tym samym* wierszu, należy podać **g** jako znacznik operacji (ang. *flag*):

```
s/regular/complex/g
```

Polecenie to zmienia wszystkie wystąpienia w bieżącym wierszu. Aby działanie tego polecenia skierować na więcej wierszy, niż tylko wiersz bieżący, należy wyspecyfikować adres. Następujące polecenie podstaw specyfikuje adres:

```
/regular/s/regular/complex/g
```

Polecenie to edytuje pierwszy wiersz, który w pliku pasuje do adresu. Należy pamiętać, że pierwszy łańcuch "regular" jest adresem, a drugi wzorcem do dopasowania przez polecenie podstaw. Aby zastosować je do wszystkich wierszy trzeba użyć polecenia globalnego, wstawiając **g** przed adresem.

```
g/regular/s/regular/complex/g
```

Teraz podstawienie wykonywane jest wszędzie — obejmuje wszystkie wystąpienia we wszystkich wierszach.

Zwróćmy uwagę na różne znaczenia "g". Na początku "g" jest poleceniem globalnym oznaczającym: zrób zmiany we wszystkich wierszach dopasowanych przez adres. Na końcu "g" jest znacznikiem operacji, który oznacza: zmień każde wystąpienie w wierszu, nie tylko pierwsze.

Adres i wzorzec nie muszą być identyczne:

```
g/regular expression/s/regular/complex/g
```

W każdym wierszu, który zawiera łańcuch “regular expression” zastąp “regular” przez “complex”. Jeżeli adres i wzorzec są identyczne, mówimy o tym **ed**, podając kolejno dwa ograniczniki (//).

```
g/regular/s//complex/g
```

W tym przykładzie “regular” specyfikuje się jako adres, a wzorzec, który ma być dopasowany przy podstawianiu, jest identyczny z adresem. Jeśli Czytelnikowi wydaje się, że omówiliśmy te polecenia pośpiesznie, a trzeba przyswoić sobie bardzo wiele, nie musi się martwić. Wrócimy do nich później.

Znane narzędzie systemu UNIX, **grep**, wywodzi się z następującego polecenia globalnego w **ed**:

```
g/re/p
```

na oznaczenie „global regular expression print”. **Grep** jest poleceniem edycji wierszowej, wydzielonym z **ed** i udostępnionym jako zewnętrzny program. Jest przeznaczone do wykonywania pojedynczego polecenia edycji. Jako argument przyjmuje w wierszu poleceń wyrażenie regularne i używa go jako adresu wierszy do wydrukowania. Oto przykład wyszukiwania wierszy pasujących do “box”:

```
$ grep 'box' test
You are given a series of boxes, the first one labeled "A",
label on the first box.
```

Drukuje on wszystkie wiersze pasujące do wyrażenia regularnego.

Jedną z ciekawszych cech **ed** jest to, że możemy napisać *skrypt* z poprawkami, umieścić je w osobnym pliku i skierować ten plik na wejście edytora wierszowego. Na przykład, jeżeli ciąg poleceń zostałby umieszczony w pliku o nazwie *ed-script*, następujące polecenie wykonywałoby ten skrypt.

```
ed test < ed-script
```

Własność ta sprawia, że **ed** jest edytorem programowalnym, to znaczy, że możemy napisać skrypt dowolnych działań, które moglibyśmy wykonać ręcznie.

Sed został stworzony jako wyspecjalizowany edytor, wyłącznie do wykonywania skryptów; w przeciwieństwie do **ed** nie może być używany interaktywnie. Sed różni się od **ed** tym, że jest zorientowany strumieniowo. Domyślnie wszelkie dane wejścia przepływają przez sed i są kierowane na wyjście standardowe. Sam plik wejściowy nie jest poddawany zmianom. Jeżeli rzeczywiście chcemy zmodyfikować plik wejściowy, stosujemy na ogół mechanizm shella do przekierowania wyjścia i gdy stwierdzimy, że poprawki są zadowalające, zastępujemy pierwotny plik wersją zmodyfikowaną.

ed nie jest zorientowany strumieniowo i zmiany przeprowadzane są w samym pliku. Skrypt **ed** musi zawierać polecenia zapisania pliku i zakończenia pracy edytora. Nie wyprowadza na ekran nic poza tym, co może być generowane przez któreś z poleceń.

Strumieniowa orientacja sed ma znaczny wpływ na sposób, w jaki stosowane jest adresowanie. W **ed** polecenie bez adresu edytuje tylko bieżący wiersz. Sed przechodzi poprzez plik wiersz za wierszem, tak że każdy z wierszy staje się wierszem bieżącym i do niego stosowane są polecenia. W rezultacie sed stosuje polecenie bez adresu do *każdego* wiersza w pliku.

Spójrzmy na następujące polecenie podstaw:

```
s/regular/complex/
```

Jeżeli wpiszemy to polecenie w trybie interaktywnym **ed**, podstawimy "complex" za pierwsze wystąpienie "regular" w bieżącym wierszu. W przypadku skryptu **ed**, jeśli byłoby to pierwsze polecenie skryptu, zostałoby zastosowane tylko do ostatniego wiersza pliku (domyślnego wiersza bieżącego **ed**). W skrypcie sed natomiast to samo polecenie stosuje się do wszystkich wierszy. Oznacza to, że polecenia sed są domyślnie globalne. W sed ostatni przykład daje taki sam wynik, jak następujące globalne polecenie w **ed**:

```
g/regular/s//complex
```

Zrozumienie różnicy pomiędzy adresowaniem bieżącego wiersza w **ed** i adresowaniem wierszy globalnie w sed jest bardzo ważne. W **ed** używa się adresowania, aby *powiększyć* liczbę wierszy, które są przedmiotem polecenia — w sed adresowania używa się, by *ograniczyć* liczbę wierszy edytowanych przez polecenie.

Sed został ponadto wyposażony w liczne, dodatkowe polecenia, które wspomagają pisanie skryptów. Wieloma z tych poleceń zajmiemy się w rozdziale 6., *Zaawansowane polecenia sed*.

Awk został opracowany jako programowalny edytor, który podobnie jak sed, jest zorientowany strumieniowo i interpretuje skrypt poleceń edycji. To, w czym awk odchodzi od sed, polega na zaniedbaniu zbioru poleceń edytora wierszowego. Zamiast tego awk oferuje język programowania, wzorowany na języku C. Instrukcja **print** zastępuje na przykład polecenie **p**. Pojęcie adresowania przenoszone jest tak, że:

```
/regular/ { print }
```

drukuje wiersze pasujące do "regular". Nawiasy klamrowe ({}) okalają ciąg jednej lub więcej instrukcji, które są stosowane do jednego adresu.

Zaletą użycia w skryptach języka programowania jest to, że oferuje on znacznie więcej sposobów kontrolowania, co może robić programowalny edytor. Awk oferuje wyrażenia, instrukcje warunkowe, pętle i inne konstrukcje programowe.

Jedną z najbardziej znamiennych cech awk jest to, iż analizuje on syntaktycznie (ang. *parses*), albo inaczej mówiąc rozdziela, każdy wiersz wejścia i umożliwia przetwarzanie za pomocą skryptu poszczególnych słów. (Edytor taki jak **vi** również rozpoznaje słowa, pozwalając poruszać się od słowa do słowa bądź zrobić słowo przedmiotem działania, lecz te własności mogą być wykorzystywane tylko interaktywnie).

Chociaż awk był projektowany jako programowalny edytor, użytkownicy przekonali się, że skrypty awk równie dobrze mogły spełnić szeroki zakres innych zadań. Autorzy awk nigdy nie sądzili, że będzie się go stosować do pisania dużych programów. Lecz wobec faktu, że awk był używany w taki sposób, autorzy dokonali rewizji języka, tworząc **nawk**, który zapewniał lepsze wsparcie przy pisaniu większych programów i rozwiązywaniu zadań programowania ogólnego. Ta nowa wersja z drobnymi poprawkami jest skodyfikowana przez standard POSIX.

Składnia wiersza poleceń

Sed i awk uruchamia się w bardzo podobny sposób. Oto składnia wiersza poleceń:

```
polecenie [opcje] skrypt nazwa_pliku
```

Jak niemal wszystkie programy UNIX, sed i awk mogą czytać z wejścia standardowego i wysyłać dane wyjścia na wyjście standardowe. Jeżeli wyspecyfikowana jest *nazwa pliku*, wejście pobierane jest z tego pliku. Wyjście zawiera przetworzoną informację. Wyjściem standardowym jest ekran wyświetlacza i na ogół tam kierowane jest wyjście programów. Można je wysłać również do pliku za pomocą przekierowania wejścia-wyjścia w shellu, lecz nie wolno kierować go do tego samego pliku, który dostarcza programowi danych wejścia.

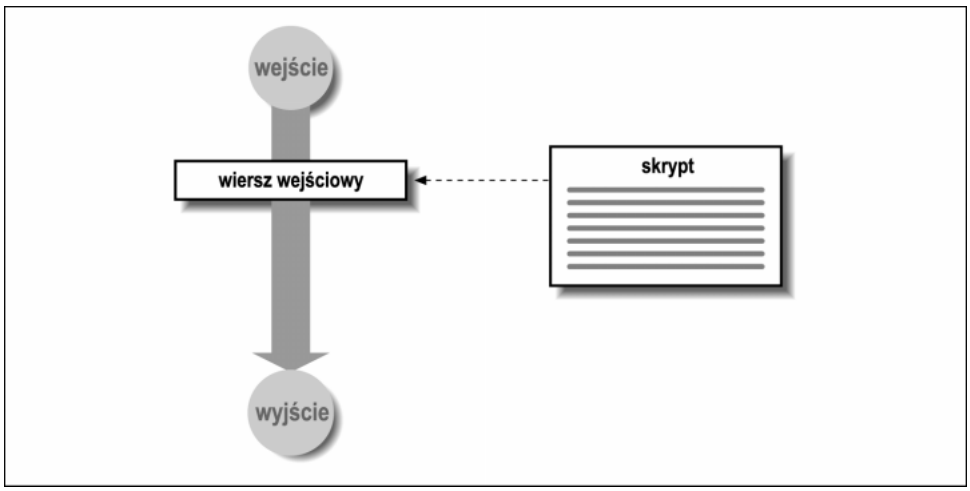
Dla każdego z poleceń *opcje* są różne. Wiele z nich przedstawimy w kolejnych punktach rozdziału. (Pełna lista opcji wiersza poleceń sed znajduje się w dodatku A, *Przewodnik sed*; pełna lista opcji awk jest w dodatku B, *Przewodnik awk*.)

Instrukcje do wykonania specyfikuje *skrypt*. Jeżeli skrypt zapisywany jest w wierszu poleceń, musi być otoczony pojedynczymi cudzysłowami, o ile zawiera spację lub jakikolwiek znak, który mógłby być interpretowany przez shell (na przykład $ lub *).

Opcją wspólną sed i awk jest opcja *-f*, która pozwala wyspecyfikować nazwę pliku ze skryptem. Kiedy skrypt się rozrasta, dogodnie jest umieścić go w pliku. Tak więc sed można uruchamiać w następujący sposób:

```
sed -f plik_skryptu plik_wejściowy
```

Rysunek 2.1 przedstawia podstawowe działanie sed i awk. Każdy z programów czyta z pliku wejściowego jeden wiersz wejścia naraz, robi kopię tego wiersza i wykonuje instrukcje wyspecyfikowane w skrypcie na tej kopii. Tak więc zmiany dokonane w wierszu wejścia faktycznie nie dotyczą pliku wejściowego.



Rysunek 2.1. Jak działają sed i awk

Pisanie skryptów

Skrypt jest miejscem, w którym mówimy programowi, co ma robić. Niezbędny jest przynajmniej jeden wiersz zawierający instrukcję. Krótkie skrypty mogą być zapisane w wierszu poleceń; dłuższe skrypty umieszczane są zwykle w pliku, w którym można je łatwo poprawiać i testować. Pisząc skrypt, należy pamiętać o kolejności, w której instrukcje będą wykonywane i o tym, jak każda instrukcja zmienia wiersz wejścia.

W sed i awk każda instrukcja ma dwie części: *wzorzec* i *procedurę*. Wzorzec jest wyrażeniem regularnym ograniczonym prawymi ukośnikami (/). Procedura specyfikuje jedno lub więcej działań, które mają być wykonane.

Kiedy czytany jest każdy wiersz wejścia, program czyta pierwszą instrukcję w skrypcie i sprawdza *wzorzec* w odniesieniu do bieżącego wiersza. Jeśli wzorzec nie pasuje, *procedura* jest pomijana i odczytywana jest następna instrukcja. Jeśli wzorzec pasuje, przeprowadzane jest działanie lub działania wyspecyfikowane przez *procedurę*. Czytane są wszystkie instrukcje, nie tylko pierwsza, która dopasowała wiersz wejścia.

Gdy wszystkie odpowiednie instrukcje są zinterpretowane i zastosowane do pojedynczego wiersza, sed wyprowadza ten wiersz na wyjście i powtarza cykl dla każdego wiersza wejścia. Awk, przeciwnie, nie wyprowadza tego wiersza *automatycznie*; instrukcje w skrypcie kontrolują to, co ostatecznie się z nim robi.

Treść procedury jest bardzo różna w sed i w awk. W sed procedura składa się z poleceń edycji, takich jak w edytorze wierszowym. Większość poleceń składa się z pojedynczej litery.

W awk procedura składa się z instrukcji języka i funkcji. Procedura musi być ujęta w nawiasy klamrowe.

W kolejnych punktach rozdziału przyjrzymy się kilku skryptom, które przetwarzają prostą listę adresową.

Próbka listy adresowej

Przykłady w poniższych podpunktach rozdziału wykorzystują plik o nazwie *list*. Zawiera on listę nazwisk i adresów, pokazaną poniżej:

```
$ cat list
John Daggett, 341 King Road, Plymouth MA
Alice Ford, 22 East Broadway, Richmond VA
Orville Thomas, 11345 Oak Bridge Road, Tulsa OK
Terry Kalkas, 402 Lans Road, Beaver Falls PA
Eric Adams, 20 Post Road, Sudbury MA
Hubert Sims, 328A Brook Road, Roanoke VA
Amy Wilde, 334 Bayshore Pkwy, Mountain View CA
Sal Carpenter, 73 6th Street, Boston MA
```

Czytelnik, jeśli chce, może utworzyć powyższy plik w swoim systemie albo użyć podobnego pliku własnego pomysłu. Ponieważ liczne przykłady w tym rozdziale są krótkie i interaktywne, można wprowadzać je z klawiatury i sprawdzać wyniki.

Użycie sed

Są dwa sposoby wywołania sed: specyfikujemy instrukcje edycji w wierszu poleceń lub umieszczamy je w pliku i dostarczamy nazwę pliku.

Specyfikacja prostych instrukcji

Proste polecenia edycji możemy specyfikować w wierszu poleceń:

```
sed [-e] 'instrukcja' plik
```

Opcja *-e* jest niezbędna tylko wtedy, gdy w wierszu poleceń specyfikuje się więcej niż jedną instrukcję edycji. Opcja ta mówi sed, że następny argument powinien być interpretowany jako instrukcja. Przy pojedynczej instrukcji sed jest w stanie ustalić to samodzielnie. Przyjrzyjmy się kilku przykładom.

Następujący przykład stosuje polecenie **s** przy podstawieniu, w którym "MA" jest zastępowane przez "Massachussets" w przykładowym pliku wejściowym *list*:

```
$ sed 's/MA/Massachusetts/' list
John Daggett, 341 King Road, Plymouth Massachusetts
Alice Ford, 22 East Broadway, Richmond VA
Orville Thomas, 11345 Oak Bridge Road, Tulsa OK
Terry Kalkas, 402 Lans Road, Beaver Falls PA
Eric Adams, 20 Post Road, Sudbury Massachusetts
Hubert Sims, 328A Brook Road, Roanoke VA
Amy Wilde, 334 Bayshore Pkwy, Mountain View CA
Sal Carpenter, 73 6th Street, Boston Massachusetts
```

Trzy wiersze są edytowane przez instrukcję, lecz wszystkie wiersze zostają wyświetlone.

Ujmowanie instrukcji w pojedyncze cudzysłowy nie we wszystkich przypadkach jest obowiązkowe, powinniśmy jednak przyzwyczaić się do tego. Pojedyncze cudzysłowy okalające powstrzymują shell od interpretowania znaków specjalnych lub spacji znajdujących się w instrukcji edycji. (Shell używa spacji do rozróżniania poszczególnych argumentów przekazywanych do programu; znaki, które są specjalne z punktu widzenia shella, rozwijane są przed wywołaniem polecenia.)

Tak więc pierwszy przykład mógł być uruchomiony bez nich. Lecz w następnym przykładzie są obowiązkowe, gdyż polecenie podstaw zawiera spację.

```
$ sed 's/MA/, Massachusetts/' list
John Daggett, 341 King Road, Plymouth, Massachusetts
Alice Ford, 22 East Broadway, Richmond VA
Orville Thomas, 11345 Oak Bridge Road, Tulsa OK
Terry Kalkas, 402 Lans Road, Beaver Falls PA
Eric Adams, 20 Post Road, Sudbury, Massachusetts
Hubert Sims, 328A Brook Road, Roanoke VA
Amy Wilde, 334 Bayshore Pkwy, Mountain View CA
Sal Carpenter, 73 6th Street, Boston, Massachusetts
```

W celu umieszczenia przecinka pomiędzy miastem i stanem instrukcja zastąpiła spację przed dwuliterowym skrótem przez przecinek i spację.

Są trzy sposoby, by wyspecyfikować wiele instrukcji w wierszu poleceń:

1. Rozdzielić instrukcje średnikiem:
```
sed 's/ MA/, Massachusetts/; s/ PA/, Pennsylvania/' list
```
2. Poprzedzić każdą z instrukcji przez *-e*:
```
sed -e 's/ MA/, Massachusetts/' -e 's/ PA/, Pennsylvania/' list
```
3. Skorzystać ze zwielokrotnienia wiersza poleceń w shellu Bourne'a[1]. Naciskamy RETURN po wpisaniu pojedynczego cudzysłowu i pojawia się wtedy pomocnicze zgłoszenie gotowości (>) wejścia dla wielu wierszy.
```
$ sed '
> s/ MA/, Massachusetts/
> s/ PA/, Pennsylvania/
> s/ CA/, California/
John Daggett, 341 King Road, Plymouth, Massachusetts
Alice Ford, 22 East Broadway, Richmond VA
Orville Thomas, 11345 Oak Bridge Road, Tulsa OK
Terry Kalkas, 402 Lans Road, Beaver Falls, Pennsylvania
Eric Adams, 20 Post Road, Sudbury, Massachusetts
Hubert Sims, 328A Brook Road, Roanoke VA
Amy Wilde, 334 Bayshore Pkwy, Mountain View, California
Sal Carpenter, 73 6th Street, Boston, Massachusetts
```

Technika ta nie będzie działać w shellu C. Zamiast niej należy użyć średników na końcu każdej z instrukcji, a polecenia wpisuje się w wielu wierszach, kończąc każdy nich lewym ukośnikiem. (Można też tymczasowo przejść do shellu Bourne'a wpisując **sh** a potem polecenie.)

W powyższym przykładzie zmiany dokonały się w pięciu wierszach i oczywiście wszystkie wiersze zostały wyświetlone. Pamiętajmy, że nic nie zmieniło się w pliku wejściowym.

Command garbled

Składnia polecenia sed może być nieugięta i łatwo jest zrobić błąd albo pominąć obowiązkowy element. Popatrzmy, co się stanie, gdy użyjemy niekompletnej składni:

```
$ sed -s 's/MA/Massachusetts' list
sed: command garbled: s/MA/Massachusetts
```

Sed wyświetla na ogół każdy wiersz, którego nie może wykonać, nie mówi jednak, co jest nie w porządku z samym poleceniem[2]. Tutaj na końcu polecenia podstaw brakuje prawego ukośnika, który oznacza w poleceniu części wyszukiwania i zastępowania.

Bardziej pomaga nam GNU sed:

```
$ gsed -e 's/MA/Massachusetts' list
gsed: Unterminated 's' command
```

[1] Obecnie istnieje wiele interpreterów, kóre są kompatybilne z shellem Bourne'a i działają w opisany tutaj sposób: **ksh**, **bash**, **pdksh** i **zsh**, by wymienić tylko kilka z nich.

[2] Pewni producenci starają się to najwyraźniej poprawić. Na przykład w SunOS4.1.x sed komunikuje "sed: Ending delimiter missing on substitution: s/MA/Massachussetts".

Pliki skryptów

Zapisywanie dłuższych skryptów w wierszu poleceń jest niepraktyczne. Dlatego najlepiej jest stworzyć plik skryptu zawierający instrukcje edycji. Skrypt edycji jest po prostu listą poleceń sed, które są wykonywane w porządku występowania. Forma taka, z użyciem opcji *-f*, wymaga, abyśmy wyspecyfikowali w wierszu poleceń nazwę pliku ze skryptem:

```
sed -f plik_skryptu plik
```

Wszystkie polecenia edycji, jakie chcemy wykonać umieszczone są w pliku. Przestrzegamy konwencji tworzenia pod nazwą *sedscr* tymczasowych plików ze skryptami.

```
$ cat sedscr
s/ MA/, Massachusetts/
s/ PA/, Pennsylvania/
s/ CA/, California/
s/ VA/, Virginia/
s/ OK/, Oklahoma/
```

Poniższe polecenie odczytuje wszystkie polecenia podstaw w *sedscr* i stosuje je do każdego wiersza pliku wejściowego *list*:

```
$ sed -f sedscr list
John Daggett, 341 King Road, Plymouth, Massachusetts
Alice Ford, 22 East Broadway, Richmond, Virginia
Orville Thomas, 11345 Oak Bridge Road, Tulsa, Oklahoma
Terry Kalkas, 402 Lans Road, Beaver Falls, Pennsylvania
Eric Adams, 20 Post Road, Sudbury, Massachusetts
Hubert Sims, 328A Brook Road, Roanoke, Virginia
Amy Wilde, 334 Bayshore Pkwy, Mountain View, California
Sal Carpenter, 73 6th Street, Boston, Massachusetts
```

Raz jeszcze wynik jest ulotny, wyświetlany na ekranie. W pliku wejściowym nie robi się żadnych zmian.

Jeżeli skrypt z sed nadaje się do użytku, powinniśmy zapisać go pod nową nazwą. Skrypty o sprawdzonej wartości można przechowywać w osobistej lub ogólnie dostępnej bibliotece.

Zapis wyjścia

Jeśli nie stosujemy przekierowania wyjścia sed do innego programu, będziemy chcieli przechwycić dane wyjścia do pliku. Robi się to przez wyspecyfikowanie nazwy pliku poprzedzonej jednym z symboli shella, oznaczających przekierowanie wejścia-wyjścia:

```
$ sed -f sedscr list > newlist
```

Nie powinniśmy kierować wyjścia do pliku, który jest edytowany pod groźbą uszkodzenia tego pliku. (Operator przekierowania ">" obcina plik, zanim shell zrobi cokolwiek innego.) Jeżeli chcemy, by plik wyjściowy zastąpił plik wejściowy możemy to uzyskać w osobnym kroku, za pomocą polecenia **mv**. Przedtem musimy jednak dobrze sprawdzić, czy skrypt edycji działa poprawnie!

W rozdziale 4., *Pisanie skryptów sed*, przyjrzymy się skryptowi shell o nazwie **runsed**, który automatyzuje proces tworzenia pliku tymczasowego oraz użycia **mv** w celu zatarcia (ang. *overwrite*) pierwotnego pliku.

Powstrzymanie automatycznego wyświetlania wierszy wejścia

Sed domyślne wyprowadza na wyjście każdy wiersz wejścia. Opcja *-n* powstrzymuje automatyczne wyprowadzanie. Kiedy specyfikujemy tę opcję, każda instrukcja, która powinna wyprowadzać wynik, musi zawierać polecenie drukuj, **p**. Spójrzmy na następujący przykład:

```
$ sed -n -e 's/MA/Massechusetts/p' list
John Daggett, 341 King Road, Plymouth Massachusetts
Eric Adams, 20 Post Road, Sudbury Massachusetts
Sal Carpenter, 73 6th Street, Boston Massachusetts
```

Porównajmy to wyjście z pierwszym przykładem w tym punkcie rozdziału. Teraz drukowane są tylko wiersze, które były edytowane przez polecenie.

Mieszanie opcji (POSIX)

Możemy zbudować skrypt, łącząc użycie opcji *-e* i *-f*. Skrypt stanowi kombinację wszystkich poleceń w podanym porządku. Funkcjonują tak wersje UNIX sed, ale własność ta nie jest jasno udokumentowana na stronie podręcznikowej. Standard POSIX wyraźnie uprawomocnia to postępowanie.

Podsumowanie opcji

Tabela 2.1 podsumowuje opcje wiersza poleceń sed.

Tabela 2.1. Opcje wiersza poleceń sed

Opcja	Opis
-e	Poprzedza instrukcję edycji.
-f	Poprzedza nazwę pliku ze skryptem.
-n	Powstrzymaj automatyczne wyprowadzanie wierszy wejścia.

Użycie awk

Awk, podobnie jak sed, wykonuje zbiór instrukcji dla każdego wiersza wejścia. Możemy specyfikować instrukcje w wierszu poleceń bądź stworzyć plik ze skryptem.

Uruchamianie awk

Dla wierszy poleceń składnia jest następująca:

```
awk 'instrukcje' pliki
```

Dane wejścia czytane są po jednym wierszu naraz, z jednego lub więcej *plików* lub z wejścia standardowego. Dla ochrony przed shellem *instrukcje* muszą być ujęte w pojedyncze cudzysłowy. (Instrukcje niemal zawsze zawierają nawiasy klamrowe i(lub) znaki dolara, które interpretowane są przez shell jako znaki specjalne.) Można wpisywać wiele wierszy poleceń w taki sam sposób, jak dla sed, rozdzielając polecenia średnikami lub stosując zwielokrotnienie wiersza poleceń w shellu Bourne'a.

Programy awk umieszcza się zwykle w pliku, gdzie można je testować i zmieniać. Składnia dla wywołania awk z plikiem skryptu jest następująca:

```
awk -f skrypt pliki
```

Opcja *-f* działa tak samo jak w sed.

Mimo że instrukcje awk mają tę samą budowę jak w sed, składając się z części *wzorca* i części *procedury*, procedury są zupełnie inne. Awk wygląda w nich mniej jak edytor, a bardziej jak język programowania. Występują tu instrukcje i funkcje zamiast jedno- lub dwuliterowych sekwencji poleceń. Używamy na przykład instrukcji **print**, aby wydrukować wartość wyrażenia albo zawartość bieżącego wiersza wejścia.

W zwykłym przypadku awk interpretuje każdy wiersz wejścia jako rekord i każde słowo w tym wierszu, ograniczane znakami spacji lub tabulacji, jako pole. (To zachowanie domyślne można zmienić.) Jeden lub więcej znaków spacji lub tabulacji, występujących kolejno, liczy się jako jeden ogranicznik. Awk pozwala odwoływać się do tych pól we wzorcach bądź procedurach. $0 oznacza cały wiersz wejścia. $1, $2... odnoszą się do poszczególnych pól w wierszu wejścia. Awk rozdziela rekord wejścia przed zastosowaniem skryptu. Przyjrzyjmy się kilku przykładom z użyciem prostego pliku wejściowego *list*.

Pierwszy przykład zawiera pojedynczą instrukcję, która drukuje pierwsze pole każdego wiersza pliku wejściowego:

```
$ awk '{ print $1 }' list
John
Alice
Orville
Terry
Eric
Hubert
Amy
Sal
```

"$1" odnosi się do wartości pierwszego pola w każdym wierszu wejścia. Ponieważ nie specyfikuje się żadnego wzorca, instrukcja drukowania jest stosowana do wszystkich wierszy. W następnym przykładzie wyspecyfikowany jest wzorzec "/MA/", ale brakuje procedury. Działanie domyślne to drukowanie każdego wiersza, który pasuje do wzorca.

```
$ awk '/MA/' list
John Daggett, 341 King Road, Plymouth MA
Eric Adams, 20 Post Road, Sudbury MA
Sal Carpenter, 73 6th Street, Boston MA
```

Drukowane są trzy wiersze. Jak wspomnieliśmy w pierwszym rozdziale, program awk może być używany trochę jak język zapytań, wyciągający użyteczne informacje z pliku. Można powiedzieć,

że wzorzec nakłada warunek na wybór rekordów, które ma zawierać raport, a mianowicie, że powinny zawierać łańcuch "MA". Teraz możemy również wyspecyfikować, jaką część rekordu ma zawierać raport. Następny przykład stosuje instrukcję **print**, by ograniczyć dane wyjścia do pierwszego pola każdego rekordu:

```
$ awk '/MA/ { print $1 }' list
John
Eric
Sal
```

Powyższe polecenie łatwiej zrozumieć, jeśli przeczyta się je na głos: *drukuj pierwsze słowo każdego wiersza zawierającego łańcuch "MA"*. Możemy powiedzieć „słowo", ponieważ domyślnie awk rozdziela wejście na pola, używając znaków spacji lub tabulacji jako ograniczników pola.

W następnym przykładzie stosujemy opcję *-F*, by zamienić ogranicznik pola na przecinek. Pozwala to nam wyszukać każde z trzech pól: nazwisko, ulicę lub miasto ze stanem:

```
$ awk -F, '/MA/ { print $1 }' list
John Daggett
Eric Adams
Sal Carpenter
```

Nie należy mylić opcji *-F*, która zmienia ogranicznik pola, z opcją *-f*, która specyfikuje nazwę pliku ze skryptem.

W następnym przykładzie drukujemy każde pole w osobnym wierszu. Polecenia są rozdzielane średnikiem:

```
$ awk -F, '{ print $1; print $2; print $3 }' list
John Daggett
 341 King Road
 Plymouth MA
Alice Ford
 22 East Broadway
 Richmond VA
Orville Thomas
 11345 Oak Bridge Road
 Tulsa OK.
Terry Kalkas
 402 Lans Road
 Beaver Falls PA
Eric Adams
 20 Post Road
 Sadbury MA
Hubert Sims
 328A Brook Road
 Roanoke VA
Amy Vilde
 334 Bayshore Pkwy
 Mountain View CA
Sal Carpenter
 73 6th Street
 Boston MA
```

Przykłady z użyciem sed zmieniały treść przychodzących danych. Przykłady z użyciem awk zmieniają ich uporządkowanie. Zauważmy, że w powyższym przykładzie początkowy odstęp uważany jest teraz za część drugiego i trzeciego pola.

Komunikaty o błędzie

Każda z implementacji awk podaje inne komunikaty o błędzie, gdy napotka problemy w napisanym przez nas programie. Nie będziemy tutaj cytować komunikatów żadnej wersji; będzie oczywiste, że mamy problem. Komunikaty mogą być wywołane przez dowolną z następujących przyczyn:

- procedury nie są umieszczone pomiędzy nawiasami klamrowymi ({});
- instrukcje nie są objęte pojedynczymi cudzysłowami (");
- wyrażenia regularne nie są umieszczone pomiędzy prawymi ukośnikami (//).

Podsumowanie opcji

Tabela 2.2 podsumowuje opcje wiersza poleceń awk.

Tabela 2.2. Opcje wiersza poleceń awk

Opcja	Opis
-f	Po niej nazwa pliku ze skryptem.
-F	Zmień separator pola.
-v	Po niej *zmienna=wartość*

Opcja *-v*, określająca parametry w wierszu poleceń, jest omówiona w rozdziale 7., *Pisanie skryptów dla awk*.

Użycie sed wraz z awk

W systemie UNIX używa się potoku (ang. *pipe*), by przekazać wyjście z jednego programu jako wejście do następnego. Przyjrzyjmy się kilku przykładom, które łączą użycie sed i awk przy tworzeniu raportu. Skrypt sed, który zastępował pocztowy skrót stanu jego pełną nazwą, jest na tyle ogólny, że może być ponownie użyty jako plik skryptu pod nazwą **nameState**:

```
$ cat nameState
s/ CA/, California/
s/ MA/, Massachusetts/
s/ OK/, Oklahoma/
s/ PA/, Pennsylvania/
s/ VA/, Virginia/
```

Oczywiście, warto byłoby uwzględnić wszystkie stany, nie tylko pięć, a uruchamiając ten skrypt z dokumentami innymi niż listy adresowe, upewnić się, że nie wykonuje niechcianych podstawień.

Wyjście tego programu dla wejściowego pliku *list* już widzieliśmy. W następnym przykładzie dane wyjścia produkowane przez **nameState** przekazywane są przez potok do programu awk, który wyciąga nazwę stanu z każdego rekordu:

```
$ sed -f nameState list | awk -F, '{ print $4 }'
 Massachusetts
 Virginia
 Oklahoma
 Pennsylvania
 Massachusetts
 Virginia
 California
 Massachusetts
```

Program awk przetwarza dane wyjścia wyprodukowane przez skrypt sed. Pamiętajmy, że skrypt sed zastępuje skrót przez przecinek i pełną nazwę stanu. Tym samym rozdziela trzecie pole, które zawiera miasto i stan, na dwa pola. "$4" jest odniesieniem do czwartego pola.

Wszystko, co tu robimy można by wykonać całkowicie w sed, lecz prawdopodobnie z dużo mniejszą ogólnością i ze znacznie większym trudem. Ponadto, skoro awk pozwala nam zastąpić dopasowany łańcuch znaków, wynik ten moglibyśmy osiągnąć całkowicie za pomocą skryptu awk.

Chociaż wynik tego programu nie jest szczególnie przydatny, można by go przekazać do **sort | uniq -c**, który posortowałby stany w postaci alfabetycznej listy z liczbą wystąpień każdego stanu.

Teraz zrobimy coś bardziej interesującego. Chcemy stworzyć raport, który sortuje nazwiska ze względu na stan i wymienia nazwę każdego stanu, a po niej nazwisko każdej z osób zamieszkałych w tym stanie. Poniższy przykład przedstawia program **byState**:

```
#! /bin/sh
awk -F, '{
        print $4 ", " $0
        }' $* |
sort |
awk -F, '
$1 == LastState {
        print "\t" $2
}
$1 != LastState {
        LastState = $1
        print $1
        print "\t" $2
}'
```

Ten skrypt shell ma trzy części. Wywołuje awk, aby wytworzyć wejście dla programu **sort**, a następnie wywołuje awk ponownie, aby przetestować posortowane wejście i stwierdzić, czy w bieżącym rekordzie nazwa stanu jest identyczna, jak w poprzednim. Obejrzyjmy ten skrypt w akcji:

```
$ sed -f nameState list | bySate
 California
         Amy Wilde
 Massachusetts
         Eric Adams
         John Dagett
         Sal Carpenter
 Oklahoma
         Orville Thomas
 Pennsylvania
         Terry Kalkas
 Virginia
         Alice Ford
         Hubert Sims
```

Nazwiska są posortowane ze względu na stan. Jest to typowy przykład użycia awk do tworzenia raportu z danych, które mają strukturę.

Aby zobaczyć, jak działa program **byState**, przyjrzyjmy się każdej części oddzielnie. Program jest przygotowany do czytania wejścia z programu **nameState** i spodziewa się, że "$4" będzie nazwą stanu. Popatrzmy na dane wyjścia produkowane przez pierwszy wiersz programu:

```
$ sed -f nameState list | awk -F, '{ print $4 ", " $0 }'
Massachusetts, John Daggett, 341 King Road, Plymouth, Massachusetts
Virginia, Alice Ford, 22 East Broadway, Richmond, Virginia
Oklahoma, Orville Thomas, 11345 Oak Bridge Road, Tulsa, Oklahoma
Pennsylvania, Terry Kalkas, 402 Lans Road, Beaver Falls, Pennsylvania
Massachusetts, Eric Adams, 20 Post Road, Sudbury, Massachusetts
Virginia, Hubert Sims, 328A Brook Road, Roanoke, Virginia
California, Amy Wilde, 334 Bayshore Pkwy, Mountain View, California
Massachusetts, Sal Carpenter, 73 6th Street, Boston, Massachusetts
```

Program **sort** domyślnie sortuje wiersze w porządku alfabetycznym, rozpatrując znaki od lewej do prawej. W celu posortowania rekordów ze względu na stan, a nie nazwiska, wstawiamy stan na początku rekordu jako klucz sortowania. Teraz program **sort** może wykonać swoje zadanie. (Zauważmy, że użycie narzędzia **sort** oszczędza nam pisania w awk programów sortujących.)

Gdy za drugim razem wywoływany jest awk, wykonujemy zadanie programistyczne. Skrypt sprawdza pierwsze pole każdego rekordu (stan), aby stwierdzić, czy jest ono identyczne, jak w poprzednim rekordzie. Jeśli nie jest identyczne, drukowana jest nazwa stanu, a po niej nazwisko osoby. Jeśli jest identyczne, drukowane jest tylko nazwisko.

```
$1 == LastState {
        print "\t" $2
}
$1 != LastState {
        LastState = $1
        print $1
        print "\t" $2
}
```

Jest tu kilka ważnych rzeczy, jak przypisanie do zmiennej, badanie pierwszego pola każdego z wierszy wejścia, żeby zobaczyć, czy zawiera zmienny łańcuch i drukowanie znaków tabulacji dla wyrównania wydruku danych. Zauważmy, że przypisanie przed użyciem zmiennej nie jest konieczne (gdyż zmienne awk są inicjowane łańcuchem pustym). To jest niewielki skrypt, ale zobaczymy program podobnego typu, stosowany do porównywania haseł indeksu w znacznie większym programie indeksującym z rozdziału 12., *Pełne aplikacje*. Jednak na razie nie przejmujmy się zbytnio tym, jak należy rozumieć każdą z instrukcji. Teraz chodzi nam raczej o to, by pokazać w ogólnych zarysach, do czego mogą służyć sed i awk.

W tym rozdziale omówiliśmy podstawowe działania sed i awk. Zrobiliśmy przegląd ważnych opcji wiersza poleceń i zapoznaliśmy Czytelnika z pisaniem skryptów. W następnym rozdziale przyjrzymy się wyrażeniom regularnym, czyli temu, czego używają obydwa programy, by dopasowywać wzorce w strumieniu wejściowym.