Awk w przykładach, część druga

Spis treści: 1. Rekordy, pętle i tablice ▾

1. Rekordy, pętle i tablice

Rekordy wielowierszowe

Awk jest wspaniałym narzędziem do czytania oraz przetwarzania ustrukturyzowanyh danych, takich jak plik systemowy `/etc/passwd`. Zawiera on Uniksową bazę danych opisującą użytkowników systemu i podaje mnóstwo istotnych informacji, w tym wszystkie istniejące konta użytkowników i identyfikatory. W moim poprzednim artykule pokazałem jak awk może z łatwością przetwarzać ten plik. Ponieważ pola w tym pliku rozdzielane są dwukropkami, to jedynym, co trzeba było zrobić, to ustawić zmienną FS (separator pól) na ":".

Przed ustawienie w sposób prawidłowy zmiennej FS awk może zostać skonfigurowany do przeczytania prawie każdego rodzaju strukturalnych danych - o ile każdy rekord zawarty jest w jednej linii. W przeciwnym wypadku ustawianie FS nie da nam zbyt wiele. W takich sytuacjach konieczne jest również ustawienie zmiennej separatora rekordów RS. Informuje ona awk gdzie kończy się aktualny rekord a zaczyna następny.

Jako przykład spróbujemy poradzić sobie z przetwarzaniem listy adresów uczestników Federalnego Programu Ochrony Świadków:

Listing 1.1: Przykładowy wpis na liście uczestników Federalnego Programu Ochrony Świadków

```
Jimmy Łasica
Ulica Przyjemności 100
San Francisco, CA 12345

Duży Tony
Aleja Incognito 200
Przedmiasteczko, WA 67890
```

W idealnym przypadku chcielibyśmy, aby awk rozpoznawał każdy 3-wierszowy adres jako oddzielny rekord. Nasz kod stałby się prostszy jeżeli awk rozpoznawałby pierwszą linię adresu jako pierwsze pole ($1), ulicę jako pole drugie ($2), a miasto, stan oraz kod pocztowy jako trzecie ($3). Ten kod uczyni dokładnie to:

Listing 1.2: Robienie z adresu jednego rekordu

```
BEGIN {
    FS="\n"
    RS=""
}
```

Dzięki ustawieniu powyżej FS na "\n" awk będzie oczekiwał, że każde pole będzie znajdowało się w oddzielnym wierszu. Z kolei ustawienie RS na "" sprawia, że awk będzie traktowało puste wiersze jako separatory rekordów. Kiedy już awk wie, jak wejście jest sformatowane może wykonać za nas całe jego przetwarzanie i reszta skryptu staje się prosta. Spójrzmy na kompletny skrypt który wczyta listę adresów i wypisze każdy z nich w jednej linii, rozdzielając pola przecinkiem.

Listing 1.3: Pełen skrypt

```
BEGIN {
    FS="\n"
    RS=""
}
{ print $1 ", " $2 ", " $3 }
```

Jeśli ten skrypt zostanie zapisany jako `address.awk`, a dane adresowe znajdą się w pliku nazwanym `address.txt`, to można wykonać skrypt poleceniem `awk -f address.awk address.txt`. Wynik będzie nastepujący:

Listing 1.4: Wynik skryptu

```
Jimmy Łasica, Ulica Przyjemności 100, San Francisco, CA 12345
Duży Tony, Aleja Incognito 200, Przedmiasteczko, WA 67890
```

OFS oraz ORS

W zawartej w pliku `address.awk` intrukcji można zobaczyć, że awk łączy ze sobą natępujące po sobie łańcuchy. Wykorzystaliśmy tę możliwość do wstawienia przecinka oraz spacji (", ") pomiędzy trzy pola adresu w linii. Uzyskaliśmy w ten sposób oczekiwany efekt ale kod wygląda brzydko. Zamiast wstawiać literał ", " pomiędzy pola możemy sprawić, aby awk zrobił to za nas. Robi się to przez ustawienie zmiennej specjalnej OFS, na przykład:

Listing 1.5: Wycinek kodu

```
print "Hello", "there", "Jim!"
```

Przecinki w wierszu nie są literałami tylko separatorami informującymi awk że "Hello", "There" oraz "Jim!" to oddzielne pola i że pomiędzy nimi powinno się wydrukować zawartość zmiennej OFS. Domyślnie wyjście awk będzie takie:

Listing 1.6: Wyjście awk

```
Hello there Jim!
```

Widzimy, że domyślną wartością OFS jest " " - pojedyńcza spacja. Możemy jednak łatwo zmodyfikować OFS tak, aby awk wstawiał nasz ulubiony separator pól. Poniżej znajduje się nowa wersja `address.awk`, która wykorzystuje OFS aby rozdzielać pola łańcuchami ", ".

Listing 1.7: Zmiana OFS

```
BEGIN {
    FS="\n"
    RS=""
    OFS=", "
}
{ print $1, $2, $3 }
```

W awk istnieje też zmienna specjalna ORS, "Wyjściowy separator rekordów" (ang. Output Record Separator). Przez modyfikację jego wartości z domyślnego znaku nowej linii ("\n") możemy zmienić znak, który jest automatycznie drukowany na końcu wyniku każdego polecenia print. Na przykład aby uzyskać wyjście rozdzielane pustymi wiersza należy ustawić ORS na "\n\n". Aby wyjście było rozdzielane pojedyńczą spacją (bez znaku nowego wiersza) ustawiamy ORS na " ".

Konwersja z rekordów wielowierszowych na rozdzielane tabulacją

Powiedzmy, że napisaliśmy skrypt który w celu importu do arkusza kalkulacyjnego konwertuje naszą listę adresów do postaci z jednym rekordem na wiersz, z polami rozdzielanymi znakiem tabulacji. Po użyciu lekko zmodyfikowanej wersji `address.awk` okazałoby się, że nasz program radzi sobie wyłącznie z trójliniowymi adresami. Jeżeli awk napotkałoby adres taki jak poniższy, to czwarty wiersz zostałby po cichu odrzucony:

Listing 1.8: Przykładowy wpis

```
Kuzyn Winnie
Sklep Samochodowy Winniego
Aleja Miejska 300
Sosueme, OR 76543
```

Aby radzić sobie z takimi sytuacjami warto by było, aby nasz kod brał pod uwagę liczbę pól w rekordzie i drukował je po kolei. W tej chwili drukujemy tylko pierwsze trzy pola adresu. Poniżej znajduje się poprawiony kod:

Listing 1.9: Poprawiony kod

```
BEGIN {
    FS="\n"
    RS=""
    ORS=""
}

{
    x=1
    while ( x<NF ) {
        print $x "\t"
        x++
    }
```

```
    print $NF "\n"
}
```

Najpierw ustalamy separator pól na "\n" oraz separator rekordów na "", tak, aby awk prawidłowo odczytywał adresy wielowierszowe. Następnie ustawiamy wyjściowy separator pół ORS na "", dzięki czemu intrukcja print przestanie za każdym razem wypisywać znaki nowej linii. Oznacza to, że kiedy będziemy chcieli aby jakiś tekst zaczął się od nowego wiersza, to musimy wypisać znak "\n" wprost.

W bloku głównym tworzymy zmienną x, która zawiera numer pola, który aktualnie przetwarzamy. Wstępnie ustawiamy ją na 1. Następnie wykorzystyujemy pętlę while (składnia while w awk jest identyczna jak ta w języku C) do iterowania przez wszystkie pola z wyjątkiem ostatniego. W końcu drukujemy ostatnie pole oraz znak nowego wiesza; print nie zrobi tego za nas ponieważ ORS jest ustawione na "". Wyjście programu wygląda teraz tak, jak powinno, czyli:

Listing 1.10: Nasze oczekiwane wyjście. Nieładne, ale dobre do importu do arkusza kalkulacyjnego

```
Jimmy Łasica, Ulica Przyjemności 100, San Francisco, CA 12345
Duży Tony, Aleja Incognito 200, Przedmiasteczko, WA 67890
Kuzyn Winnie Sklep Samochodowy Winniego Aleja Miejska 300 Sosueme, OR 76543
```

Pętle

Widzieliśmy już, jak w awk robi się pętle while, identyczne jak ich odpowiedinki w C. W awk dostępne są też pętle "do...while" które sprawdzają swój warunek na końcu bloku, a nie na początku. Jest podobna do dostępnej w niektórych innych językach pętli "repeat...until". Na przykład:

Listing 1.11: Przykład do...

```
{
    count=1
    do {
        print "Zostanę wydrukowana co najmniej raz niezależnie od wszystkiego"
    } while ( count != 1 )
}
```

Ponieważ warunek jest sprawdzany po wykonaniu bloku pętla "do...while" zawsze zostanie wykonana co najmniej raz. W przeciwieństwie do tego zwykła pętla while nigdy nie wykona swojego kodu jeżeli na samym początku warunek nie jest spełniony.

Pętle for

Awk pozwala na tworzenie pętli for, które, podobnie jak pętle while, są identyczne jak ich odpowiedniki w C:

Listing 1.12: Przykładowa pętla

```
for ( wstępne przypisanie; warunek; modyfikacja ) {
    blok kodu
}
```

Prosty przykład:

Listing 1.13: Prosty przykład

```
for ( x = 1; x <= 4; x++ ) {
    print "iteracja",x
}
```

Ten kod da następujące wyjście:

Listing 1.14: Wyjście powyższego fragmentu

```
iteracja 1
iteracja 2
iteracja 3
iteracja 4
```

Break oraz continue

Podobnie jak C awk udostępnia instrukcje break oraz continue. Te polecenia usprawniają kontrolę nad pętlami. Na przykład poniżej znajduje się fragment kodu w którym naprawdę przydałaby się intrukcja break:

Listing 1.15: Fragment kodu wymagający instrukcji break

```
while (1) {
    print "I tak już zawsze..."
}
```

Ponieważ 1 zawsze jest prawdziwe ta pętla jest nieskończona. Poniżej znajduje się pętla która zostanie wykonana dziesięć razy:

Listing 1.16: Pętla, której kod wykonuje się tylko 10 raz

```
x=1
while(1) {
    print "iteracja",x
    if ( x == 10 ) {
        break
    }
    x++
}
```

Tutaj polecenie break służy do "wyrwania się" z njbardziej zagnieżdżonej pętli. "break" powoduje, że pętla natychmiast zostaje zakończona i wykonanie kodu jest wznawiane od linii następującej po kodzie pętli.

Polecenie continue uzupełnia break i działa tak:

Listing 1.17: Instrukcja continue uzupełniająca break

```
x=1
while (1) {
    if ( x == 4 ) {
        x++
        continue
    }
    print "iteracja",x
    if ( x > 20 ) {
        break
    }
    x++
}
```

Ten kod napisze "iteracja 1" - "iteracja 21", pomijając "iteracja 4". Jeżeli x jest równe 4, to x jest zwiększane po czym wykonywana jest instrukcja continue, co sprawia że awk natychmiast rozpoczyna kolejne wykonanie pętli - pomija więc resztę kodu. Instrukcja ta działa dla każdego rodzaju pętli w awk, tak samo jak break. Kiedy zostanie wykonana w pętli for zmienna kontrolna zostanie automatycznie zwiększona. Poniżej znajduje się przykładowa pętla for:

Listing 1.18: Przykładowa pętla for

```
for ( x=1; x<=21; x++ ) {
    if ( x == 4 ) {
        continue
    }
    print "iteracja",x
}
```

Zwiększenie x nie było konieczne, ponieważ pętla for automatycznie zwiększa x.

Tablice

Dobrą wiadomością jest z pewnością to, że w awk dostępne są tablice. W awk zwyczajem jest rozpoczynanie indeksu tablicy od 1 a nie od 0.

Listing 1.19: Przykładowe tablice w awk

```
myarray[1]="jim"
myarray[2]=456
```

Kiedy awk napotyka pierwsze przypisanie, tablica myarray jest tworzona i jej element myarray[1] jest ustawiany na wartość "jim". Po wykonaniu drugiego przypisaniu tablica ma już dwa elementy.

Po zdefiniowaniu tablicy awk dostarcza wygodnego mechanizmu do iterowaniu po wszystkich jej elementach, na przykład:

Listing 1.20: Iterowanie po zawartości tablicy

```
for ( x in myarray ) {
    print myarray[x]
}
```

Ten kod wypisze wszystkie elementy tablicy myarray. Przy wykonywanu tej formy pętli for awk będzie przypisywało każdy istniejący indeks myarray do x (zmiennej kontrolnej) po kolei, dla każdej wartości raz wykonując zawartość bloku. Jest to wprawdzie bardzo praktyczne, ale składnia ta ma pewną wadę -- porządek przeglądania indeksów tablicy nie jest w żaden sposób określony. To znaczy, że nie wiemy czy wynikiem powyższego kodu będzie:

Listing 1.21: Wynik powyższego kodu

```
jim
456
```

czy

Listing 1.22: Inny wynik powyższego kodu

```
456
jim
```

Parafrazując Forresta Gumpa, iterowanie po zawartości tablicy jest jak pudełko czekoladek - nigdy nie wiesz co z niego wyciągniesz. Ma to trochę wspólnego z "łańcuchowością" tablic awk.

"Łańcuchowość" indeksów tablic

W poprzednim artykule pokazałem, że awk przechowuje zmienne liczbowe w postaci łańcuchów. Wprawdzie awk wykonuje konwersje konieczne, aby to działało, ale pozostawia możliwości tworzenia dziwnego kodu:

Listing 1.23: Dziwny kod

```
a="1"
b="2"
c=a+b+3
```

Po wykonaniu tego kodu c jest równe 6. Ponieważ awk jest "łańcuchowe" dodawanie łańcuchów "1" oraz "2" jest w zasadzie tym samym co dodawanie liczb 1 oraz 2. W obydwu przypadkach awk dokona prawidłowego dodawania. Łańcuchowa natura awk bywa dość intrygująca -- na przykład, co się stanie, gdy wykorzystamy łańcuch do indeksowania tablic? Na przykład, w poniższym kodzie:

Listing 1.24: Przykład

```
myarr["1"]="Mr. Whipple"
print myarr["1"]
```

Zgodnie z oczekiwaniami kod wypisze "Mr. Whipple". Co się stanie, gdy pominiemy cudzysłowy wokół jedynki w indeksie?

Listing 1.25: Pomijamy cudzysłowy

```
myarr["1"]="Mr. Whipple"
print myarr[1]
```

Odgadnięcie wyniku tego fragmentu może być nieco trudniejsze. Czy rozróżnia myarr["1"] od myarr[1]? Odpowiedzią jest nie, awk uważa, że jedna i druga forma odnosi się do tego samego elementu i awk napisze "Mr. Whipple", dokładnie tak samo jak w pierwszym fragmencie. Może się to wydawać dziwne, ale awk zawsze wykorzystuje łańcuchy do indeksowania tablic.

Po poznaniu tej dziwnej właściwości niektórzy z nas mogą skusić się na przetestowanie dziwacznego kodu, który wygląda tak:

Listing 1.26: Dziwaczny kod

```
myarr["name"]="Mr. Whipple"
print myarr["name"]
```

Nie dość, że ten kod nie powoduje błędów, to na dodatek jest funkcjonalnie identyczny z poprzednimi przykładami i tak jak one napisze "Mr. Whipple"! Jak widać w awk nie trzeba koniecznie korzystać z indeksów całkowitych; można też używać łańcuchów i nie powoduje to żadnych komplikacji. Za każdym razem, kiedy wykorzystujemy nie-liczbowe indeksy tablic, takie jak myarr["name"], wykorzystujemy tablice asocjacyjne. Technicznie rzecz biorąc awk nie robi jednak wtedy nic szczególnego (ponieważ awk traktuje indeksy "liczbowe" jak łańcuchy).

Narzędzia tablicowe

W przypadku tablic awk daje nam dość duże możliwości. Możemy wykorzystywać łańcuchy jako indeksy i nie jesteśmy zmuszeni do wykorzystywania ciągłej sekwencji liczb (na przykład można zdefiniować myarr[1] oraz myarr[1000] a pozostałe elementy pozostawić niezdefiniowanymi). Jest to wprawdzie przydatne, ale może czasem doprowadzić do bałaganu. Na szczęście awk dostarcza kilku narzędzi dzięki którym łatwiej jest zarządzać tablicami.

Po pierwsze, możemy kasować elementy tablicy. Na przykład, aby skasować element 1 tablicy fooarray:

Listing 1.27: Kasowanie elementów tablicy

```
delete fooarray[1]
```

Jeżeli chcemy sprawdzić czy konkretny element tablicy istnieje, to można wykorzystać specjalny operator "in":

Listing 1.28: Sprawdzanie czy dany element tablicy istnieje

```
if ( 1 in fooarray ) {
    print "Tak! Jest tutaj."
} else {
    print "Nie! Nie mogę go znaleźć."
}
```

W następnym odcinku

W tym artykule wykonaliśmy sporo pracy. W następnym odcinku podsumuję pokazaną wiedzę o awk przez pokazanie jak wykorzystywać zawarte w awk funkcje do obsługi łańcuchów oraz matematyki oraz jak tworzyć własne funkcje. Następnie przeprowadzę czytelnika przez program do obsługi finansów osobistych. Do tego czasu zachęcam do pisania własnych programów oraz przejrzenia następujących zasobów.

2. Zasoby

- Inne artykuły o Awk autorstwa Daniela z developerWorks Awk w przykładach, Część pierwsza orazCzęść trzecia.
- Książka O'Reilly's sed & awk, 2nd Edition jest doskonałym wyborem dla tych, którzy wolą papier.
- Konieczne należy odwierdzić FAQ comp.lang.awk. Zawiera ono również mnóstwo dodatkowych linków związanych z Awk.
- Samouczek AWK Patricka Hartigana zawiera mnóstwo przydatnych skryptów
- Thompson's TAWK Compiler kompiluje skrypty awk czyniąc z nich szybkie binarki. Jest on dostępny w wersjach dla Windows, OS/2, DOS i UNIX.
- Podręcznik użytkownika GNU Awk zawiera informator.