Systemy operacyjne

Laboratorium 5

Awk – podstawy

Jarosław Rudy
Politechnika Wrocławska

28 lutego 2017

Temat obejmuje podstawowe pojęcia związane z komendą `awk` – uruchamienie, składnia, pola i rekordy i wykorzystaniem jej do przetwarzania tekstu. Informacje zawarte w tej instrukcji nie są wyczerpujące – zaleca się przejrzenie manuali i tutoriali dla komendy `awk`.

1 Wprowadzenie

Komenda `awk` (i pochodna, nieco zmieniona, komenda `gawk`) jest przykładem "dużej" skomplikowanej komendy, która może być uznana za osobny język programowania. Część jej składni przypomina bardziej język `C` niż `bash` (zwłaszcza, że znak dolara pełni dla `awk`a zupełnie inną funkcję niż dla `bash`a). Skupimy się teraz na podstawowym użyciu `awk`a, wliczając w to filozofię `awk`a i sposób działania.

Zaczniemy od celu istnienia `awk`a, a celem tym jest przetwarzanie tekstu zgodnie z zadanym programem (algorytmem). Z tego od razu wynika, że `awk` spodziewa się dwóch rzeczy na "wejściu": 1) tekstu do przetworzenia i 2) programu z instrukcjami. W praktyce często program umieszcza się w osobnym pliku, ale podczas laboratorium będziemy umieszczać program wprost w wywołaniu `awk`a:

```
awk 'tekst programu' nazwa_pliku_do_przetworzenia
```

lub

```
awk 'tekst programu' < nazwa_pliku_do_przetworzenia
```

Jeśli z kolei chcemy przekazać tekst dla `awk`a z użyciem potoku:

```
komenda | awk 'tekst programu'
```

Istotne jest, że w tekst programu ujmujemy w apostrofy (') a nie w cudzysłowy ("). W ten sposób zapewniamy sobie, że znaki w programie `awk`a nie będą interpretowane przez `bash`a. Rozwiązuje to między innymi problem ze znakiem dolara, ale nie do końca. Czasami bowiem chcemy w tekst programu wstawić wartość zmiennej `bash`a. Rozwiązanie jest jednak proste – nasz program będzie składał się z trzech "napisów" `bash`a sklejonych ze sobą:

```
awk 'program $ czesc 1'$zmiennabashowa'program $ czesc 2'
```

Pierwszym i trzeci dolar będą "`awk`owe", zaś drugi będzie "`bash`owy" (czyli oznaczający zmienną `bash`a). Pamiętamy przy tym, że `bash` skleja łańcuchy tekstowe wprost bez operatora. W tym miejscu warto wspomnieć, że w `awk`u sklejanie (konkatenacja) tekstu działa inaczej – operatorem łączenia tekstu w `awk` jest *spacja*. Tak więc poniższa instrukcja `bash`a:

```
echo "jakis" "tam" "tekst"
```

da wynik:

```
jakis tam tekst
```

zaś instrukcja `awk`a

```
print "jakis" "tam" "tekst"
```

da wynik:

```
jakistamtekst
```

Aby uzyskać spacje musimy wstawić je jawnie:

```
print "jakis " "tam" " tekst"
```

Alternatywą (przynajmniej domyślnie) jest użycie przecinka:

```
print "jakis", "tam", "tekst"
```

który zostanie w wyświetlonym tekście zamienione na spacje. Dlaczego tak się dzieje zostanie wyjaśnione w dalszej części instrukcji.

2 Rekordy i pola

Zwykle patrząc na pliki tekstowe dzielimy je na "linie", zaś linie na "wyrazy". Tak właśnie wygląda domyślne zachowanie `awk`a, który dzieli przetwarzane wejście podobnie. Jednakże siła `awk`a polega na tym, że można zmienić to zachowanie odwołując się do pojęcia rekordu (uogólnienie pojęcia linii) i pola (uogólnienie wyrazu). Pozwala to na pracę z tekstem o bardzo różnym formacie.

Komenda `awk` dzieli swoje wyjście z użyciem tzw. *separatorów* (*ang.* delimiters, separators). Wejście posiada dwa separatory: separator rekordów (*ang.* record separator, RS) oraz pól (*ang.* field spearator, FS). Są to tak naprawde zmienne `awk` i jeśli ustawimy je w POPRAWNYM miejscu (zwykle oznacza

to akcję dla wzorca `BEGIN`, omówimy to później), to będziemy mogli wpływać na sposób podziału wejścia na rekordy i pola. Domyśnymi wartościami tych zmiennych są:

```
RS = "\n"
FS = " "
```

Gdy RS lub FS są pojedynczym znakiem (przypomnienie: \n jest jednym znakiem), to separatorem staje się ten znak. Wyjątkiem jest pojedyncza spacja (domyślna wartość FS), która oznacza "dowolną liczbę spacji, tabulatorów lub nawet znaków nowej linii". Jeśli wartością RS/FS jest więcej niż jeden znak, to wartość ta jest traktowana jako *wyrażenie regularne*, co jest tematem następnego laboratorium. Z tego powodu ustawianie separatorów na więcej niż jeden znak nie powinno być stosowane na laboratorium 5. Reasumując: domyślnie `awk` dzieli wejście na linie i wyrazy rozdzielone białymi znakami. Zauważmy też, że każdy rekord może mieć różną liczbę pól (tak jak linie mogą mieć różne liczby wyrazów).

Ważna uwaga: `awk` traktuje zmienne w sposób podobny do `C`, a nie do `bash`a, czyli używa się ich BEZ znaku dolara. Specjalne zmienne (jak RS i FS) złożone są z dużych liter, ale nie ma przeszkód do używania zmiennych składających się z małych liter. Zmienne w `awk` nie są nigdy tworzone czy deklarowane, po prostu się ich używa (akurat to zachowanie jest zbliżone do `bash`a). Ponadto, zawsze należy pamiętać o używaniu cudzysłowów wokół stałych tekstowych, bo bez tego zostaną one uznane za zmienną `awk`ową.

Oprócz separatorów wejściowych istnieją separatory wyjściowe, z dodatkową literą `O` (od output) – `ORS` i `OFS`. W przeciwieństwie do separatorów wejściowych są zawsze traktowane jako dosłowny tekst. Domyślna wartość ORS to znak nowej linii, a OFS to pojedyncza spacja. Separatory te są wykorzystywane m.in. przez (`awk`ową) instrukcję `print` – wartość OFS zostanie wypisana na standardowe wyjście `awk`a w miejsce każdego (nieujętego w cudzysłowy) przecinka w wywołaniu `print`, podczas gdy wartość ORS zostanie wypisana na wyjście na końcu instrukcji `print`.

Istnieją jeszcze inne zmienne specjalne. Przykładowo, zmienna `NR` (od Number of Recods) przechowuje liczbę przetworzonych do tej pory rekordów, a `NF` (od Number of Fields) przechowuje liczbę pól w aktualnym rekordzie.

Ostatnia ważna uwaga odnośnie rekordów i pól to... sposób odnoszenia się do nich (do ich wartości). Służy do tego znak dolara:

```
$0          // wartosc (tekst) calego aktualnego REKORDU
$1          // wartosc pierwszego POLA aktualnego rekordu
$2          // wartosc drugiego pola
zmienna=5
$zmienna    // wartosc PIĄTEGO pola
$NF         // wartosc ostatniego pola
$(NF-1)     // wartosc przedostatniego pola
```

Widzimy, że przy dolarze nie musi stać stała liczbowa – może to także być zmienna lub wyrażenie, którego wartością jest nieujemna liczba całkowita.

Zauważmy, że znalezienie wartości pól w rekordzie jest trywialne, ale znalezienie wartości separatorów pomiędzy poszczególnymi polami jest już trudniejsze (np. domyślnie nie wiemy czy wyrazy były oddzielone jedną spacją, kilkoma czy może tabulatorem). Wyjątkiem jest separator rekordów, który dla aktualnego rekordu przechowywane jest w zmiennej `RT` (od Rekord Terminator).

Do pól możemy też przypisywać wartości:

```
$0 = "nowy tekst"
$1 = zmienna
$2 = $3
```

Istotne jest, że rekord i jego pola zachowują się jak naczynia połączone. Zmiana wartości całego rekordu lub jego dowolnego pola spowoduje zmianę i *ponownie rozdzielenie rekordu na pola*. Tak więc po zmianie $0 mogą zmienić się wartości i LICZBA pól. Podobnie zmiana któregokolwiek pola wpływa na wartość rekordu.

Istnieją także inne zmienne, zwykle nieużywane podczas laboratorium. Zmienna `FIELDWIDTHS` powinna zawierać listę liczb np. "12 10 6", co sprawi, że `awk` przyjmie, że pierwsze 12 znaków w rekordzie to pole pierwsze, 10 kolejnych to pole drugie itp. Zmienna `FPAT` działa "odwrotnie" niż FS – opisuje za pomocą wyrażenia regularnego co jest POLEM, a nie co jest separatorem. Ustawienie którejkolwiek z tych trzech zmiennych (FS, FPAT, FIELDWIDTHS) anuluje poprzednie przypisania.

3 Wzorce i akcje

Programy `awk`a mają bardzo specyficzny format. Każdy program składa się z par typu wzorzec-akcja w następującej postaci:

```
wzorzec { akcja }
```

Zasada działania jest następująca: dla każdego rekordu wejścia rozpatrywane są po kolei wszystkie wzorce (z wyjątkiem wzorców `BEGIN` oraz `END`, o których później). Jeśli wzorzec jest prawdziwy (spełniony) dla aktualnego rekordu, to akcja skojarzano w parze z tym wzorcem zostaje wykonana. Należy jednak zwracać uwagę na znaki nowej linii, ponieważ zapis:

```
wzorzec
{ akcja }
```

oznacza zupełnie co innego niż zapis w jednej linii. W tym przypadku mamy dwie pary. W pierwszej parze występuje tylko wzorzec. W takim przypadku akcją jest akcja *domyślna*, czyli { `print` } (co poprzez argument domyślny jest równoważne { `print $0` }, co oznacza drukowanie aktualnego rekordu). W drugiej linii występuje akcja bez wzorca. Taki przypadek traktowany jest tak jakby był tam wzorzec, który jest zawsze prawdziwy, w wyniku czego akcja zostanie wykonana dla *każdego* rekordu. Zauważmy też, że zarówno w `bash`u

jak i w awku "samotny" backlash na końcu linii ("wyescape'owanie końca linii") pozwala na kontynuuowanie komendy i/lub programu awka w kolejnej linii. Tak więc wywołania:

```
awk ' { print $0 } '
```

oraz

```
awk \
' { \
print \
$0 } \
'
```

są równoważne. Ta sztuczka jest czasami przydatna do "wizualnego" rozdzielenia wzorca i akcji na oddzielne linie, tak aby całość wciąż traktowana była jako jedna para.

Dwa wzorce – BEGIN oraz END – są specjalne. Akcje poprzedzone wzorcem BEGIN zostają wykonane tylko RAZ i zostają wykonane PRZED przetworzoniem jakichkolwiek rekordów wejściowych. Z tego powodu akcje ze wzorcem BEGIN są dobrym miejscem na inicjalizację potrzebnych zmiennych, w szczególności zmiennych RS, FS, ORS, OFS (jeśli ich pożądane wartości są inne niż domyślne). Ponieważ akcje z wzorcem BEGIN wykonują się przed analizą rekordów, to nie ma w nich sensu używanie zmiennych związanych z wartością/liczbą rekordów/pól, takich jak $0, $1, NF, NR itp.

Analogicznie, akcje ze wzorcem END zostaną wykonane RAZ i zostaną wykonane PO przetworzeniu wszystkich rekordów. Akcje te są dobrym miejscem do wypisania (jednorazowego) podsumowania. Ponadto, zmienna NR będzie wtedy wskazywała na całkowitą liczbę przetworzonych rekordów.

Można użyć więcej niż jednej akcji BEGIN (END) – zostaną one wtedy wykonane w kolejności wystąpienia w programie.

Przejdziemy teraz do podstawowego opisu wzorców i akcji. Jeśli chodzi o wzorce, to w praktyce bardzo często są nimi wyrażanie regularne, ale ten mechanizm stosowany będzie w kolejnym laboratorium. Na razie skupimy się na wzorcach wykorzystujących wyrażenia "porównawcze", które będziemy uzupełniać zmiennymi, wyrażeniami logicznymi (analogicznymi do języka C) oraz wywołaniami funkcji. Przykładowo:

```
( NF $>$ 3 || NR % 2 == 0 ) && var == 4 { action }
```

W tym przypadku akcja zostanie wykonana tylko wtedy, gdy zmienna var ma wartość 4 ORAZ aktualny rekord ma więcej niż 3 pola LUB liczba do tej pory przetworzonych rekordów jest parzysta. Więcej informacji dotyczących możliwych wzorców znajduje się w manualu do awk.

Przechodzimy teraz do akcji. Akcja zawsze otoczona jest parą nawiasów klamrowych i jest traktowana jako całość (tzn. przypisana do poprzedzającego ją wzorca). Zawartość akcji przypomina język C. Średnik nie jest potrzebny w przypadku ostatniej instrukcji akcji:

```
{ print "Hello" }
```

ale jest potrzebny do każdej instrukcji poza ostatnią:

```
{ print "Hello "; print "world!" }
```

Akcje wykorzystują takie instrukcje jak `if-else`, `while` oraz `for` (tak jak w języku `C`). Pętla `for` posiada specjalną wersję do użycia wraz z tablicami (patrz dalej).

4 Tablice

Tablice w `awk`u, tak jak inne zmienne, są tworzone wraz z pierwszym użyciem. Jednak tablice te nie są indeksowane liczbami, lecz tekstem (kluczem). Są to więc tablice asocjacyjne (mapy, słowniki). Przykładowo:

```
tablica[ "tekst" ] = jakas_wartosc;
tablica[ zmienna ]  = jakas_wartosc;
tablica[ 10 ]  = jakas_wartosc;
```

W ostatnich dwóch przypadkach wartość zmiennej i liczba zostaną po prostu potraktowane jako tekst. Jak wspomniano wcześniej, jedna z postaci pętli `for` może zostać użyta do przejrzenia zawartości tablicy. Jest to często konieczne, zwłaszcza, gdy nie wiemy jakie klucze wykorzystywane są w danej tablicy. Wykorzystuje się do tego operator `in`, który sprawdza czy dany klucz występuje w tablicy:

```
if ( klucz in tablica )
 print tablica[ klucz ];
```

Pętla, która drukuje wszystkie elementy tablicy wygląda analogicznie:

```
for ( klucz in tablica )
 print tablica[ klucz ];
```

5 Funkcje

Komenda `awk` posiada wbudowany zestaw funkcji użytecznych przy przetwarzaniu tekstu. Szczegółowe informacje znajdują się w manualu. W tym laboratorium skupimy się na funkcjach niezwiązanych z wyrażeniami regularnymi. Zauważmy, że wiele funkcji może zostać wywołanych "bez" argumentu, ponieważ zostanie wtedy użyty argument domyślny, którym zwykle jest aktualny rekord (czyli `$0`). Wartości pól i rekordów są tekstem, wobec czego wywołania typu `length($1)` są poprawne i powszechne. Najważniejsze funkcje (część z nich lub niektóre ich opcje mogą być niedostępne dla niektórych wersji `(g)awk`a lub będą dostępne dopiero, gdy poda się odpowiednią opcję dla `awk`):

1. `index` – zwraca pozycję pod którą dany łańcuch znajduje się (zaczyna się) w innym łańcuchu. Pozycja zaczyna się od 1, wartość 0 oznacza, że łańcucha nie znaleziono.

2. `length` – zwraca długość podanego łańcucha.

3. `split` – dzieli łańcuch na elementy według danego separatora (podobne zasady jak dla separatorów RS/FS, w ogólności może to być wyrażenie regularne) i umieszcza otrzymane "kawałki" w podanej tablicy. Można użyć tej funkcji do uzyskania trzeciego poziomu podziału (po RS i FS). Ponadto, można podać nazwę drugiej tablicy, do której trafią separatory (tekst pomiędzy "kawałkami").

4. `substring` – tworzy podłańcuch z podanego łańcucha. NIE należy tego mylić z funkcją `sub` (która jest częścią rodziny funkcji `gsub`/`gensub`).

5. `tolower` – zwraca *kopię* podanego łańcucha, w której wszystkie litery zamienione są na małe. Oryginał pozostaje bez zmian.

6. `toupper` – analogicznie do `tolower`.

Komenda `awk` posiada również zestaw funkcji matematycznych.