

Narzędzia Wspierające Programowanie

AWK – język skryptowy

- ✂ awk to proste, ale funkcjonalne narzędzie do procesowania plików tekstowych.
 - Procesuje plik linia po linii
 - Sposób kodowania: jako one-liner lub we własnym skrypcie
 - Posiada zmienne, na których można też liczyć
 - Posiada pętle i warunki

W sieci jest sporo samouczków awk. M.in.:
cykl na stronie AGH [1, 2, 3], na stronie Baeldung [tutaj] i autorstwa Bruce’a Barnett’a [tutaj].

● **Struktura** kodu w awk.

Ogólny blok kodu wygląda tak: `warunek_logiczny { działanie }`
Kod można składać z wielu bloków.

Jeśli chcemy, przed procesowaniem pliku
można wykonać działania w Bloku BEGIN `BEGIN { działanie }`

Blok bez warunku oznacza, że działanie
będzie wykonywane na każdej linii pliku: `{ działanie }`

Blok z warunkiem, ale bez działania oznacza,
że po spełnieniu warunku wypisze się cała linia. `warunek_logiczny`

Na końcu możemy też wykonać Blok END `END { działanie }`

● **Sposób kodowania:** najpierw jako one-liner. Typowa składnia:

`awk 'blok blok ...' {plik_analizowany}` → wynik zvacany jest do stdout

● **Zmienne specjalne** – zaczniemy od nich, bo są nieodzowne w kodzie.

Każda badana linia ("record") zostanie podzielona na pola ("fields").
Domyślny separator – to dowolna kombinacja spacji i/lub tabulatora.
Wówczas w kodzie dostępne są:

NR	(no. of record) numer badanej linii w pliku
NF	(no. of fields) liczba pól w badanej linii
\$1, \$2, ...	zawartość kolejnych pól badanej linii
FS	separator między polami

- ▶ Przykład: wyciągnijmy z pliku `/etc/passwd` loginy i wypiszmy je (1. pole, separator to „:”)

`awk 'BEGIN {FS=":"} {print $1}' /etc/passwd`

 ↵ jak widzimy, komendą `print` wypisujemy tekst do `stdout`.

- ▶ Przykład: rozważmy plik `people2.dat`:

Name	Surname	Gender	Age	Var_1	Var_2	Var_3
Henry	Stevens	Male	19	35	67	21
Mary	Douglas	Female	21	22	56	42
Maddie	Spencer	Female	20	81	41	85
Lyndon	Brown	Male	29	19	20	88
...						

Odrzućmy nagłówek (`NR>2`). Wybierzmy tylko kobiety (`$3 == "Female"`) i wypiszmy dane:

```
awk 'NR>2 && $3 == "Female" ' people2.dat
```

Teraz wybierzmy 18-latków (`$4 == 18`) i wypiszmy imiona oraz wartości `Var_1`:

```
awk 'NR>2 && $4==18 {print "Name: "$1"\t Var_1= "$5}' people2.dat
```

 ↵ Tekst piszemy w `" "` , ale zmienne – już poza.

Powróćmy do pierwszego przykładu – i zapiszmy rezultat do pliku:

```
awk 'NR>2 && $3 == "Female" ' people2.dat > females.dat
```

● Zmienne zwykłe

- ▶ Awk oferuje zmienne o typie dynamicznym (zależnym od kontekstu).
- ▶ Inicjalizują się automatycznie do `" "` (a w kontekście liczby – do `0`).
- ▶ Odnosimy się do nich wprost (bez znaku np. `$`)
- ▶ Jeśli zawierają liczby, to podlegają arytmetyce z operatorami w stylu języka C, w tym logiczne (`&&` , `||` , `!`)
- ▶ Arytmetyka uwzględnia ułamki

Przykład: spośród osób – policzmy mężczyzn i wypiszmy ich liczbę:

```
awk 'NR>2 && $3=="Male" {Males++} END {print Males}' people2.dat
```

Jak widać, użyliśmy zmiennej `Males` od razu (inicjalizacja do `0`), a inkrementowaliśmy p/ `++` .

- **Sposób kodowania:** we własnym skrypcie.

Zakodujemy skrypt o nazwie np. `avg_age.awk` :

```
NR > 2 {
    Num_of_People++
    Sum_of_Ages += $4
}
END {
    Avg_Age = Sum_of_Ages / Num_of_People
    gsub ( ",", ".", Avg_Age )           ← zamieni separator ułamka z , na .
    print "Average age: " Avg_Age
}
```

Wywołanie go przez `awk` tak, by pracował na `people2.dat`, wygląda tak:

```
$ awk -f avg_age.awk people2.dat
```

Można też rozpocząć skrypt shebang'iem:
i ewentualnie nadać mu prawa wykonania:
Taki skrypt można wykonać wprost:

```
#!/usr/bin/awk -f
$ chmod 755 avg_age.awk
$ ./mystat.awk people2.dat
```

- **Blok warunkowy `if / else if / else`**

▸ Wygląda tak samo, jak dla C/C++ .

Np. dodajmy na koniec bloku `END`:

```
if (Avg_Age < 25) {
    print "Oj, to bardzo młodzi ludzie :) "
}
```

- ⊕ Z racji ograniczeń czasowych, tu kończymy naukę narzędzi `Awk`.
Jeżeli Cię ten język zaciekał, warto poczytać o:
 - tablicach (arrays)
 - pętlach `for`
 - funkcjach
 - funkcjach matematycznych
 - osadzaniu skryptu `awk` w skrypcie `Bash'a`

Natomiast zapraszamy na przykład użycia – na następną stronę.

- **Przykład:** szybki przegląd wyników z symulatora zderzeń jądrowych Smash

Przyjrzyjmy się strukturze danych, np. `run1/outrun/particle_lists.oscar`:

```
#!/OSCAR2013 particle_lists t x y z mass p0 px py pz pdg ID charge
# Units: fm fm fm fm GeV GeV GeV GeV GeV none none e
# SMASH-3.1
# event 0 out 68
20 2.38534 -5.02135 11.7509 0.938 1.48097481 0.0639428589 -0.0100318556 1.14422596 2112 0 0
20 1.94341 3.2962 11.843 0.938 1.447409 0.118372399 0.0817943681 1.09290735 2112 1 0
...
20 2.7284 -6.9614 -7.02452 0.138 0.375332315 0.0868583264 -0.169680046 -0.292394698 -211 143 -1
# event 0 end 0 impact 3.187 scattering_projectile_target yes
# event 1 out 62
...
# event 9 end 0 impact 4.989 scattering_projectile_target yes
```

Mamy więc taką strukturę:

- Nagłówek (3 linie z wieloma polami; każda zaczyna się od #)
 - Zestaw eventów (tu: 0 .. 9)
 - Nagłówek eventu (5 pól, czwarte = „out”, piąte = liczba cząstek w evencie (N))
 - Zestaw N cząstek
 - Linia cząstki: 12 pól. Pole 10 = identyfikator typu hadronu.
Np. proton = 2212, neutron = 2112, pi+ = 211, pi- = -211, pi0 = 111
 - Stopka eventu (9 pól, czwarte = “end”)
- Napiszmy prosty skrypt , czy liczba zadeklarowanych cząstek (`NpartDeclared`) zgadza się z liczbą linii o cząstkach (`NpartFound`). Nazwijmy skrypt: `checkNparticles.awk` .
I wykonajmy, przez: `./checkNparticles.awk Smash/run1/particle_lists.oscar`

```
#!/usr/bin/awk -f

$2 == "event" && $4 == "out" {
    NpartDeclared += $5
}
NF == 12 {
    NpartFound ++
}
END {
    print "No. of particles: Declared = " NpartDeclared " , Found = " NpartFound
    if (NpartDeclared == NpartFound) {
        print "All ok."
    } else
        print "Inconsistency!"
}
```