Jeden by wszystkimi rządzić - kontrola nad konsolą systemową.

Zygmunt Zawadzki

April 20, 2016

Contents

*Podstawy **awk***

awk jest prostym językiem skryptowym bardzo pomoagającym w pracy z logami (czyli plikami wyjściowymi z diagnostyką różnych programów, czy stanów urządzeń).

- `https://www.youtube.com/watch?v=u0wSncMHAyM` - 10 minutowe wprowadzenie do AWK (stworzony przez Unknowna z `uw-team.org`).

By wywołać **awk** należy wpisać w konsoli systemowej:

```
gawk '{print $0}' plik
```

plik określa plik wejściowy na którym ma pracować **awk** , natomiast pomiędzy ' ' ukryty jest jeden z najprostszych programów.

```
{print $0}

##
## Pinging www.google.pl [46.134.214.88] with 32 bytes of data:
## Reply from 46.134.214.88: bytes=32 time=27ms TTL=60
## Reply from 46.134.214.88: bytes=32 time=28ms TTL=60
```

```
## Reply from 46.134.214.88: bytes=32 time=28ms TTL=60
## Reply from 46.134.214.88: bytes=32 time=28ms TTL=60
##
## Ping statistics for 46.134.214.88:
##     Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
## Approximate round trip times in milli-seconds:
##     Minimum = 27ms, Maximum = 28ms, Average = 27ms
```

Wykonanie kodu tylko dla określonych linijek:

```
/Reply/ {print $0}

## Reply from 46.134.214.88: bytes=32 time=27ms TTL=60
## Reply from 46.134.214.88: bytes=32 time=28ms TTL=60
## Reply from 46.134.214.88: bytes=32 time=28ms TTL=60
## Reply from 46.134.214.88: bytes=32 time=28ms TTL=60
```

Znalezienie pierwszej linijki pasującej do wyrażenia:

```
/Reply/ {print $0; exit;}

## Reply from 46.134.214.88: bytes=32 time=27ms TTL=60
```

Podział pola:
Znalezienie pierwszej linijki pasującej do wyrażenia:

```
/Reply/ {print $3,$5}

## 46.134.214.88: time=27ms
## 46.134.214.88: time=28ms
## 46.134.214.88: time=28ms
## 46.134.214.88: time=28ms
```

Zamiana znaków z wykorzystaniem *gsub*[1].

[1] W R również występuje funkcja *gsub* o podobnym działaniu, jednak bardziej przyjazna w użyciu może być funkcja *stri_replace* z pakietu *stringi*.

```
/Reply/ {
  gsub(":", "");
  gsub("time=", "");
  gsub("ms", "");
  print $3,$5}

## 46.134.214.88 27
## 46.134.214.88 28
## 46.134.214.88 28
## 46.134.214.88 28
```

awk w R

system2 - konsola systemowa w R.

Do tej pory wywołania **awk** odbywały się z konsoli systemowej.
Jak jednak sprawić by wywołać **awk** z poziomu R? Bezpośrednie

wywołanie **awk** jest niemożliwe, gdyż R nawet nie wie, że taki program jest zainstalowany. Jednak R udostępnia możliwość wywołania dowolnej komendy systemowaj przy pomocy **system2**:

Potrzebna składnia system2:

- command - nazwa komendy którą będziemy wywoływać.
- args - dodatkowe argumenty
- stdout - określa miejsce do którego ma być skierowany wynik. Domyślnie wynik zostanie wypisany w konsoli R. Wartość TRUE oznacza, że R przechwyci go jako wektor, podanie nazwy pliku spowoduje zapisanie do niego wyniku działania komendy.

Poniżej proste wywołanie systemowej funkcji **wc**:

wc - to proste narzędzie zwracające dla zadanego pliku trzy liczby - ilość linii, słów i bajtów.

```
system2("wc", args = "ping.txt", stdout = TRUE)

## [1] " 11  63 472 ping.txt"
```

*Pierwsze wywołanie **awk** w R*

```
system2("awk",
        args = "-e '/Reply/ {print $0}' ping.txt",
        stdout = TRUE)

## [1] "Reply from 46.134.214.88: bytes=32 time=27ms TTL=60"
## [2] "Reply from 46.134.214.88: bytes=32 time=28ms TTL=60"
## [3] "Reply from 46.134.214.88: bytes=32 time=28ms TTL=60"
## [4] "Reply from 46.134.214.88: bytes=32 time=28ms TTL=60"
```

Opakowanie w funkcję

```
awk = function(file, code, stdout = TRUE)
{
  args = sprintf("-e '%s' %s", code, file)

  system2("awk",
        args = args,
        stdout = stdout)
}
```

```
awk(code = '/Reply/ {print $0}',
    file = "ping.txt")

## [1] "Reply from 46.134.214.88: bytes=32 time=27ms TTL=60"
## [2] "Reply from 46.134.214.88: bytes=32 time=28ms TTL=60"
## [3] "Reply from 46.134.214.88: bytes=32 time=28ms TTL=60"
## [4] "Reply from 46.134.214.88: bytes=32 time=28ms TTL=60"
```

Funckja zwracająca funckję

W R funkcja może zwrócić funkcję.

```
make_sqr = function()
{
  function(x) x*x
}

sqr = make_sqr()
sqr(2)

## [1] 4
```

Najważniejsza cechą tego mechanizmu jest to, że funkcja będąca rezultatem zapamiętuje wszystkie zmienne powstałe w trakcie działania funkcji która ją stworzyła (również argumenty funkcji-matki). Z tego też powodu możliwe są konstrukcje takie jak poniżej[2]:

[2] Taka funkcja zwrócona przez funkcję jest nazywana **closure**.

```
make_pow = function(n)
{
  function(x) x^n
}

pow3 = make_pow(3)
pow3(2)

## [1] 8

pow3(5)

## [1] 125

make_pow(10)(2)

## [1] 1024
```

Co więcej - z poziomu wynikowej funckji możliwa jest modyfikacja parametrów closure:

```
counter = function()
{
  i = 0;
  function()
  {
    # by zmodyfikowac wartosc i
    # nalezy uzyc operatora
    # <<-
    i <<- i + 1
    i
  }
}

ct = counter()
ct()

## [1] 1

ct()

## [1] 2

ct()

## [1] 3
```

Napisz funkcję **iter** która zwraca iterator dla zadanego wektora, tak by poniższy kod wykonywał się poprawnie:

```
x = c("Ala", "Ela","Ola")
it = iter(x)
it()

## [1] "Ala"

it()

## [1] "Ela"

it()

## [1] "Ola"
```

Iteratorem w R będzie funkcja bez żadnych argumentów zwracająca kolejne elementy wektora. Koniecznie sprawdź pakiet **iterators** - bywa bardzo przydatny!

Wykorzystanie funkcji zwracającej funckję w interfejsie do awk

```
make_awk = function(code)
{
  function(file, stdout = TRUE)
  {
    args = sprintf("-e '%s' %s", code, file)

    system2("awk",
          args = args,
          stdout = stdout)
  }
}

first_column = make_awk('{print $1}')
first_column("ping.txt")

##  [1] ""            "Pinging"
##  [3] "Reply"       "Reply"
##  [5] "Reply"       "Reply"
##  [7] ""            "Ping"
##  [9] "Packets:"    "Approximate"
## [11] "Minimum"
```

Zadanie - pingowanie strony

```
ping = function(site,
                pingFile = tempfile(),
                awkFile = tempfile())
{
    parse_ping = make_awk(
  '/Reply/ {
    gsub(":", "");
    gsub("time=", "");
    gsub("ms", "");
    print $3,$5}
  ')

  system2("ping", args = site, stdout = pingFile)
  parse_ping(pingFile, stdout = awkFile)
  read.delim(awkFile, sep = " ", header = FALSE)
}
```

```
ping("www.google.pl")
```

```
##              V1 V2
## 1 216.58.213.35 55
## 2 216.58.213.35 56
## 3 216.58.213.35 57
## 4 216.58.213.35 56
```

Podsumowanie

By móc łatwo stworzyć interface do potrzebengo programu wystarczy, by był on możliwy do wykonania z poziomu konsoli systemowej. Wtedy wystarczy tylko za pomocą kodu w R zbudować odpowiednie wywołanie i wykorzystać interface **system2**. Oczywiście jeżeli wynikiem działania programu jest jakiś plik, wtedy potrzeba jeszcze przygotować odpowiedni kod wczytujący go do R. Ale to już inna historia...

Odpowiedzi

Iterator:

```
iter = function(x)
{
  i = 0;
  function()
  {
    i <<- i + 1
    x[[i]]
  }
}
```