Wyrażenia regularne

Po co wyrażenia regularne?

Polecenie:

```
$ grep est tekst.txt
```

Zawartość `tekst.txt`

```
To jest plik tekstowy.
Testujemy narzędzie grep.
Trzecia linia.
A to czwarta linia.
```

Efekt wywołania polecenia

```
To jest plik tekstowy.
Testujemy narzędzie grep.
```

Znaki

Znaki specjalne

`. ^ $ * ? [ ] \`

Zbiór znaków

Definiowany za pomocą konstrukcji `[set]`. Np. wyrazy Ala, Ola, Ela odpowiadają wyrażeniu `[AOE]la`
Aby dopasować wyrażenie do wszystkich znaków oprócz znaków z pewnego zbioru, używa się wyrażenia `[^set]`
Zakres znaków tworzy się następująco: `[a-z]`

Znak .

Znak . pasuje dokładnie do jednego, dowolnego znaku (z wyjątkiem znaku końca linii).

Klasy znakowe

Klasa	Dopasowywane znaki
`[:alnum:]`	Znaki alfanumeryczne
`[:alpha:]`	Znaki alfabetyczne
`[:blank:]`	Znaki spacji i tabulacji
`[:cntrl:]`	Znaki sterujące
`[:digit:]`	Znaki numeryczne
`[:punct:]`	Znaki przestankowe

Powtórzenia

Symbol	Opis
*	poprzedzający znak może wystąpić 0 lub więcej razy
+	poprzedzający znak może wystąpić 1 lub więcej razy
?	poprzedzający znak może wystąpić 0 lub raz
{n}	poprzedzający znak musi wystąpić dokładnie n razy
{n,}	poprzedzający znak musi wystąpić co najmniej n razy
{,m}	poprzedzający znak musi wystąpić co najwyżej m razy
{n,m}	poprzedzający znak musi wystąpić od n do m razy

Przykłady powtórzeń

Wyrażenie

```
11*0
```

Pasuje do:

```
10
110
111110
11111111111111111110
```

Wyrażenie:

```
10\{2,4\}1
```

Pasuje do:

```
1001
10001
100001
```

Nie pasuje do:

```
101
```

Pozycjonowanie wzorców

Symbol	Opis
^	wzorzec musi znaleźć się na początku linii
$	wzorzec musi znaleźć się na końcu linii
\<	w tym miejscu zaczyna się nowe słowo
\>	w tym miejscu kończy się słowo
\b	w tym miejscu jest krawędź słowa (nie zaczyna ani nie kończy się żadne słowo)
\B	w tym miejscu nie znajduje się krawędź słowa

Przykład

```
^[^.]*\.[^.]*$
```

Pozycjonowanie wzorców – przykłady

Dla linii

```
Jola jest lojalna
```

Wyrażenia dopasowane

```
\<jest\>
\bjest\b
\Best\b
\Bes\B
```

Wyrażenia niedopasowane

```
\<est\>
\best\b
\Bes\b
\>jest\<
```

Priorytety i nawiasy

Znaki `( ... )`
Znaki `( ... )` służą do grupowania elementów.

Przykład

```
([a-z][a-z])*
```

Alternatywa

Do oznaczenia alternatywy służy znak `|`.

Przykład

```
[0-9]*|[a-z]*|[A-Z]*
(Ta|Do)[a-z]*
```

Polecenie `grep`

Podstawowe użycie

```
$ grep wzorzec plik1 plik2 plik3 ...
```

Parametry polecenie `grep`

Parametr	Opis
`-e`	alternatywny sposób podania wzorca
`-i`	ignoruje rozróżnianie wielkich liter
`-c`	zlicza wystąpienie wzorca
`-w`	dopasowuje wzorzec tylko do całych słów
`-x`	dopasowuje wzorzec tylko do całych linii
`-v`	wypisuje tylko linie, w których nie odnaleziono wzorca
`-q`	nic nie wypisuje i kończy działanie po pierwszym dopasowaniu

Polecenie `grep`

Parametry polecenie `grep` – cd.

Parametr	Opis
`-E`	rozszerzona składnia wyrażeń regularnych
`-G`	podstawowa składnia wyrażeń regularnych
`-n`	wypisuje numer linii pliku, w której dopasowano wzorzec
`-l`	wypisuje tylko nazwę pliku, w którym znalazło się przynajmniej jedno dopasowanie
`-H`	dla każdego dopasowania wypisuje nazwę pliku
`--color=auto`	zaznacza kolorem dopasowany fragment

Przykłady użycia polecenia grep

Przykład 1

```
if grep -qw TODO opis_prac.txt; then
  echo "Zostało jeszcze coś do zrobienia"
fi
```

Przykład 2

```
TMPFILE=/tmp/xyzabcd
cp plik $TMPFILE
grep -xv wzorzec $TMPFILE >plik
rm -f $TMPFILE
```

Dopasowywanie wyrażeń

Od wersji 3.0 powłoki `bash` można dopasowywać wyrażenia

Składnia:

```
[[ napis =~ wyrażenie ]]
```

```
if [[ abcfoobarbletch =~ 'foo(bar)bl(.*)' ]]
 then
     echo "Dopasowanie udało się\!"
     echo $BASH_REMATCH           # wypisuje: foobarbletch
     echo ${BASH_REMATCH[1]}      # wypisuje: bar
     echo ${BASH_REMATCH[2]}      # wypisuje: etch
 fi
```

Polecenie `expr`

Składnia

```
expr łańcuch : wzorzec
```

Przykład

Wyświetlenie rozszerzenia pliku znajdującego się w zmiennej plik

```
expr "$plik" : ".*\.\([^.]*\)"
```

Zmiana rozszerzenia `.tar.gz` na `tgz` pliku zmiennej `plik`

```
if expr "$plik" : ".*\.tar\.gz$"; then
  mv $plik $(expr "$plik" : "\(.*\.\)tar\.gz")tgz
fi
```

Edytor strumieniowy `sed`

Składnia użycia

```
$ sed 'instrukcja' plik
```

Opcje wiersza poleceń

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

Podstawianie

Składnia z użyciem `sed`

```
$ sed '[pozycja]s/znajdz/zmien/flaga' plik > plik.mod
```

Objaśnienie

- `pozycja` — wskazuje linię do edytowania (opcjonalnie)
- `s` — wskazuje `sed` dokonywanie instrukcji podmiany
- `znajdz` — ciąg, który ma być znaleziony i który będzie zmieniany
- `zmien` — ciąg, na który zmieniony zostanie `znajdz`
- `flaga` — kontroluje zachowanie, np. `g` zmienia wszystkie wystąpienia w linii, `n` zmienia n-te wystąpienie w linii, `p` wypisuje dopasowany ciąg `znajdz`

Podstawianie – cd.

znaki specjalne w `zmien`

- `&` — zastępowany łańcuchem dopasowanym przez wyrażenie regularne
- `\n` — dopasowuje się do n-tego podłańcucha (n jest pojedynczą cyfrą) wcześniej określonego za pomocą nawiasów okrągłych
- `\` — cofa specjalne znaczenie znaków

Przykład

Polecenie

```
s/Patrz punkt [1-9][0-9]*\.[1-9][0-9]*/(&)/
```

zamieni `Patrz punkt 1.4` na `(Patrz punkt 1.4)`

Podstawianie – cd.

Przykłady

```
$ echo abc12abcd | sed 's/\([a-z]*\)[0-9]*\([a-z]*\)/\1/'
```

wynik działania: abc

```
$ echo abc12abcd | sed 's/\([a-z]*\)[0-9]*\([a-z]*\)/\2/'
```

wynik działania: abcd

```
$ echo abc12 | sed 's/\([a-z]*\)[0-9]*\([a-z]*\)/\2/'
```

wynik działania:

Przykłady wywołań `sed`

Przykład

```
$ echo 'abc 123' | sed 's/abc/def/'
def 123

$ echo 'Hej!! Hej!!!! Tutaj!' | sed 's/!\+/!/g'
Hej! Hej! Tutaj!

$ echo 'Hej!! Hej!!!! Tutaj!' | sed 's/!\+/!/'
Hej! Hej!!!! Tutaj!
```

Usuwanie linii

Składnia

Składnia dla usuwania linii

```
sed '/wzor/d' plik > plik.mod
```

gdzie:

- `wzor` – wyrażenie regularne
- `d` – polecenie usuwania

Usunięcie każdej linii rozpoczynającej się znakiem #

```
sed '/^#/d' file > file.mod
```

Usuwanie linii a usuwanie słów

Różnica między nimi

Usuwanie wszystkich wystąpień słowa abc w pliku

```
sed 's/abc//g' plik > plik.mod
```

Usuwanie każdej linii zawierającej słowo abc

```
sed '/abc/d' plik > plik.mod
```

Polecenie transformuj

Składnia

Składnia dla przekształcania znaków

```
sed 'y/abc/xyz/' plik > plik.mod
```

gdzie:

- `abc` – znaki zastępowane
- `xyz` – znaki zastępujące

Każdy znak zastępujący odpowiada pojedynczemu znakowi zastępowanemu.

```
'y/abcdefghijklmnopqrstuvwxyz/ABCDEFGHIJKLMNOPQRSTUVWXYZ/'
```

Zastępuje małe litery ich dużymi odpowiednikami.

Separatory

Zamiana `/var/www/` ciągiem `/var/local/www/` wymaga anulowania specjalnego znaczenia znaku `\`

```
sed 's/\/var\/www\//\/var\/local\/www\//g' file > file.mod
```

Można użyć innego separatora w celu skrócenia zapisu

```
sed 's:/var/www/:/var/local/www/:g' file > file.mod
```

Pozycjonowanie

Przykłady

Usuń słowo abc w liniach rozpoczynających się od słowa xyz

```
sed '/^xyz/s/abc//g' file > file.mod
```

Usuń słowo abc w liniach od 1 do 50

```
sed '1,50s/abc//g' file > file.mod
```

Usuń słowo abc we wszystkich liniach oprócz linii od 1 do 50

```
sed '1,50!s/abc//g' file > file.mod
```

Pozycjonowanie

Przykłady

Usuń słowo abc w liniach, które zawierają się pomiędzy liniami zawierającymi aaa oraz ccc

```
sed '/aaa/,/ccc/s/abc//g' plik > plik.mod
```

Usuń pierwsze 50 linii pliku

```
sed '1,50d' plik > plik.mod
```

Pozostaw pierwsze 50 linii pliku i usuń wszystkie pozostałe

```
sed '1,50!d' plik > plik.mod
```

Praktyczny przykład

Usuwanie tagów HTML

Przykładowa linia zawierająca znaczniki HTML

```
<b>Zdanie</b> zawierające <i>tagi</i> HTML.
```

To polecenie dopasowuje tekst znajdujący się pomiędzy pierwszym znakiem <, a ostatnim znakiem >

```
sed 's/<.*>//g' plik.html
```

i daje w rezultacie

```
HTML.
```

Poprawniejsza składnia (usunięcie tylko tagów)

```
sed 's/<[^>]*>//g' plik.html
```

daje oczekiwany rezultat

```
Zdanie zawierające tagi HTML.
```

Język awk - podstawy

Podstawy składni

```
$ cat plik
abc sef
aaab fedg
abc abc
aa
a b c

$ awk '{print}' plik
abc sef
aaab fedg
abc abc
aa
a b c
```

Język awk - przykłady

Operacje na wzorcach

```
$ awk '/abc/ {print}' plik
abc sef
abc abc

$ awk '/abc/ {print $2}' plik
sef
abc

$ awk 'BEGIN {print "Kolumna"}; /abc/ {print $2}' plik
Kolumna
sef
abc
```

Język awk – “Witaj świecie”

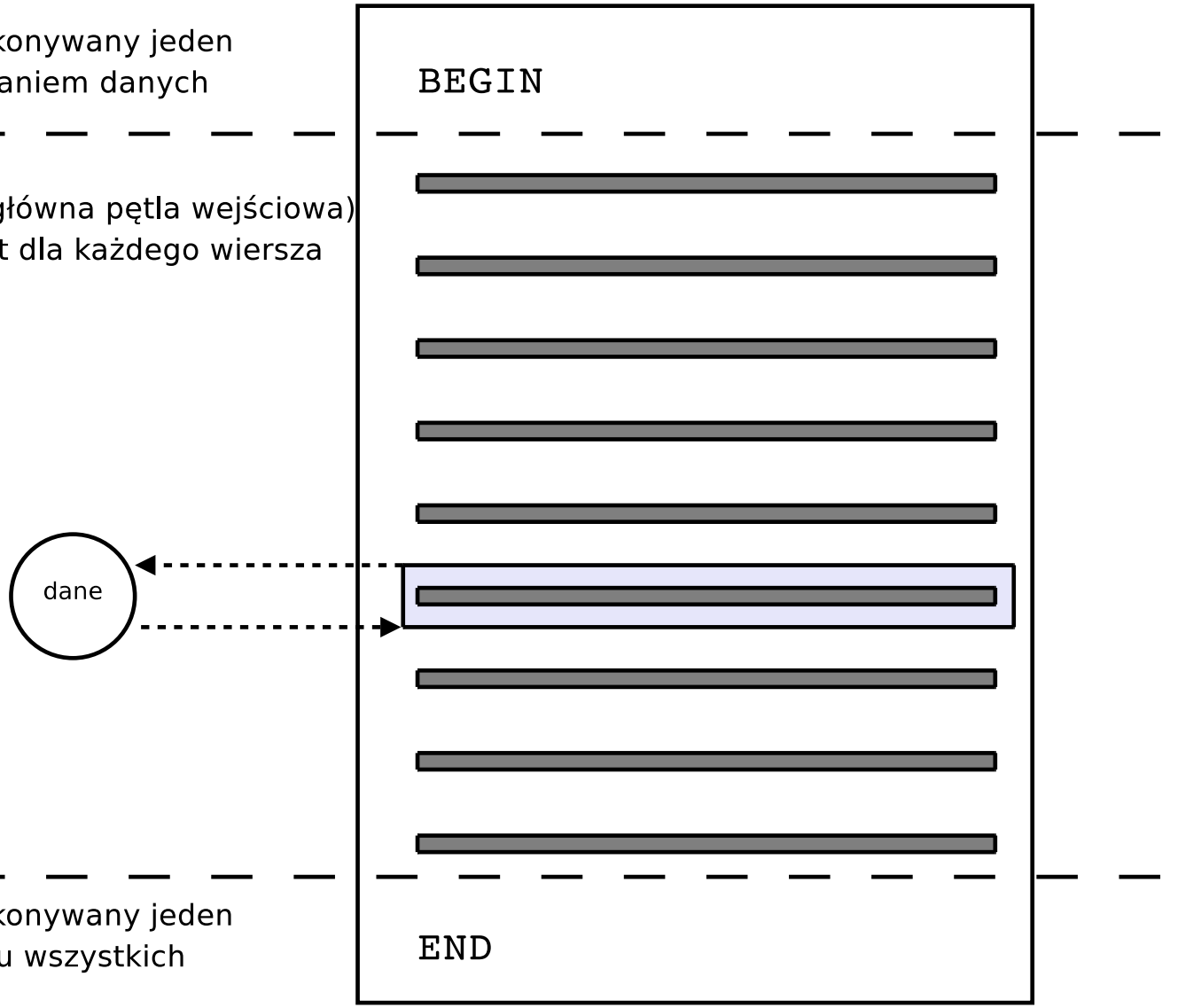
```
$ awk ’{print "Hello World"}’ plik
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World

$ awk ’BEGIN {print "Hello World"}’
Hello World
```

Model programowania `awk`

1. Program jest wykonywany jeden raz przed odczytaniem danych

2. Drugi program (główna pętla wejściowa) Wykonywany jest dla każdego wiersza danych.



3. Program jest wykonywany jeden raz po odczytaniu wszystkich danych

Ogólna postać programu w języku `awk`

```
BEGIN                {<instrukcje inicjalizacji>}
<wzorzec 1>          {<operacje na wierszach>}
<wzorzec 2>          {<operacje na wierszach>}
...
END                  {<instrukcje zamykające>}
```

Dopasowywanie wzorca

Przykład skryptu

```
#!/usr/bin/awk -f
# zbadaj wprowadzony znak
/[0-9]+/ {print "Liczba"}
/[A-Za-z]+/ {print "Litera"}
/^$/ {print "Wiersz pusty"}
```

Wywołanie

```
./nazwa_skryptu
```

lub

```
awk -f nazwa_skryptu
```

Zmienne

Przykłady

```
$ awk 'BEGIN {a=1; b=2; print a+b}'
3
$ awk 'BEGIN {a=1.3; b=2.4; print a+b}'
3.7
$ awk 'BEGIN {a="abc"; b="cde"; print a+b}'
0
$ awk 'BEGIN {a="abc"; b="cde"; c=a b; print a b, c}'
abccde abccde
$ awk 'BEGIN {print sqrt(2)}'
1.41421
```

Zmienne specjalne

Separator pola

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

można go również określić w linii poleceń

```
awk -F: '{print $1}' /etc/passwd
```

Inne zmienne

- `NF` – liczba pól
- `RS` – separator rekordu
- `OFS` – separator pola dla wyjścia
- `ORS` – separator rekordu dla wyjścia
- `NR` – numer linii w pliku

Instrukcje sterujące

Instrukcja `if`

Składnia:

```
if ( warunek ) akcja_1  [else akcja_2]
```

Przykład:

```
$ awk -F: '{if ($3>=1000) print $1}' /etc/passwd
```

Instrukcje sterujące

Instrukcja `for`

Składnia

```
for (a_początkowa; warunek; a_koniec_iteracji ) akcja
```

Przykład:

```
$ awk 'BEGIN {for (i=0; i<6; i++) print i}'
```

Instrukcje sterujące

Instrukcja `while`

Składnia:

```
while (warunek) akcja
```

Przykład:

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

BEGIN {
        i = 0
        while (i<6) {
            print i
            i+=1
        }
      }
```

Wyszukiwanie linii

Dziesiąta linia

```
$ awk -F: 'NR==10 {print $1}' /etc/passwd
```

W trzeciej kolumnie znajdować musi się wartość większa od 100

```
$ awk -F: '$3>100 {print $1}' /etc/passwd
```

W pierwszej kolumnie znajdować musi się napis "root"

```
$ awk -F: '$1 ~ /root/ {print $3}' /etc/passwd
```

Domyślną powłoką użytkownika ma być `bash`

```
$ awk -F: '$NF ~ /\/bash$/ {print $1}' /etc/passwd
```

Tablice

Tablice zwykłe

```
tablica[1], tablica[2], tablica[3] ...
```

Tablice asocjacyjne

```
debts["Kim"], debts["Roberto"], debts["Vic"] ...
```

Przykład wykorzystania tablic asocjacyjnych

count_users0.awk

```
#!/bin/awk -f
{
    username[$3]++;
}
END {
    for (i in username) {
        print username[i], i;
    }
}
```

Wywołanie:

```
$ ls -l | ./count_users0.awk
```

W wykładzie wykorzystano materiały

- Środowisko programisty, http://mediawiki.ilab.pl/index.php/%C5%9Arodowisko_programisty
- Dale Dougherty, Arnold Robbins, *sed i awk*, Helion, 2002
- `http://www.grymoire.com/Unix/Awk.html`