

# Introducción a Perl (III): *Expresiones regulares*

Alex Sánchez

Departament d'Estadística-UB

Unitat d'Estadística I Bioinformàtica-  
IRHUVH



# Esquema

---

- ***Analisis (“parsing”) de textos***
- Que son las expresiones regulares
- Sintaxis de patrones
- Análisis de coincidencias en patrones (“pattern matching”)
- Paréntesis, sustituciones y otros ...



# Análisis de texto (“Parsing”)

---

- En muchos contextos, especialmente en bioinformática o en internet se dispone de datos en formato estructurado
  - Una parte varia libremente de un caso a otro
  - Pero existe cierta sintaxis (formalmente una gramática) que indica que secuencias son informativas y cuales no.
- HTML, FastA, BLAST o el listado de resultados de un análisis estadístico



# Archivos en formato FASTA

---

```
>gi|31794748|ref|NP_857241.1| hemoglobine-like protein [Mycobacterium bovis AF2122/97]
MTEAIGDEPLGDHVLELQIAEVVDETDEARSLVFAVPDGSDDPEIPPRRLRYAPGQFLTLRVPSERTGSV
ARCYSLCSPPYTDDALAVTVKRTADGYASNWLCDHAQVGMRIHVLAPSGNFVPTTLDADFLLLAAGSGIT
PIMSICKSALAEGGGQVTLTYANRDDRVSIFGDALRELAAYKYPDRLTVLHWLESQGLPSASALAKLVAP
YTDRPVFICGPGPFMQAARDALAALKVPAQQVHIEVFKSLESDFFAAVKVDDSGDEAPATAVVELDGQTH
TVSWPRTAKLLDVLLAAGLDAPFSCREGHCGACACTLRAGKVNMGVNDVLEQQDLDEGLILACQSRPESD
SVEVTYDE
```

```
>gi|61680281|pdb|1WXR|A Chain A, Crystal Structure Of Heme Binding Protein, An
Autotransporter Hemoglobine Protease From Pathogenic Escherichia Coli
GTVNNELGYQLFRDFAENKGMFRPGATNIAIYNKQGEFVGTLDKAAMPDFSAVDSEIGVATLINPQYIAS
VKHNGGYTNVSFGDGENRYNIVDRNNAPSLDFHAPRLDKLVTEVAPTAVTAQGAVAGAYLDKERYPVFYR
LGGSTQYIKDSNGQLTKMGGAYSWLTGGTVGSLSSYQNGEMISTSSGLVFDYKLNGAMPIYGEAGDSGSP
```

FastA: La primera linea es una descripción

El resto contienen la secuencia en tamaño variable



# Análisis (parsing) de archivos

---

- A menudo solo nos interesa una parte (o múltiples partes separadas) de un archivo. Por ejemplo
  - Sólo queremos las secuencias (o sólo los descriptores) de un archivo (multi)FastA.
  - Queremos obtener los nombres de los *datasets* cuyos p-valores tras un test eran menores que 0.01 de un listado con los resultados de 100 tests sobre 100 datasets.
- El proceso de extracción selectiva de la información deseada recibe el nombre de análisis de archivo de texto o *text file parsing*.



# Formas de análisis (parsing)

---

- Escribir un programa que analice todo el archivo (“parser”)
  - Pesado pero tan sólo hay que hacerlo una vez
  - Razonable si hay que repetir el análisis a menudo
- Extraer y reformatear el texto desde la consola de linux mediante
  - *grep* para extraer líneas
  - *sort* para ordenarlas según algún criterio
  - *Cut, join, uniq...* para otras opciones de formato
- Utilizar *expresiones regulares* para seleccionar las secuencias que nos interesan.



# Esquema

---

- Analisis (“parsing”) de textos
- ***Que son las expresiones regulares***
- Sintaxis de patrones
- Análisis de coincidencias en patrones (“pattern matching”)
- Paréntesis, sustituciones y otros ...



# Expresiones regulares

---

- Una expresión regular o patrón es una cadena (*string*) que describe un conjunto de cadenas mediante una sintaxis determinada,  
... aunque también puede definirse como...
- Una plantilla que coincide o no con una cadena:  
dado una expresión regular todas las cadenas existentes van a coincidir o no coincidir con ella.
  - Enlaza con la primera definición diciendo que serán o no descritas por la regexp





# Sintaxis de expresiones regulares

---

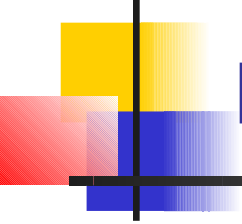
- Existen multitud de sintaxis que se parecen, pero no son lo mismo
  - Formato tradicional unix
  - Formato unix extendido (POSIX)
  - Formato Perl
  - Formato PROSITE
  - Para busca en internet (google)
- El formato perl es el mas extendido (por ejemplo R entiende dicho formato).



# Usos de las expresiones regulares

---

- Validar una entrada
  - Si lo que debe ser un número contiene letras rechazarlo
- Extraer información del listado producido por un programa
- Reconocer cadenas (o patrones, o motivos...) dentro de una cadena.
- Modificar texto
  - Leer un listado, encontrar las coincidencias y escribir la salida en un formato dado (por ejemplo excel)



# Patrones, caracteres y metacaracteres

---

- Un patron es una secuencia de caracteres (=una cadena)
- Todo carácter es o bien un *literal* o un *metacaracter*.
- Un literal solo coincide con él mismo
  - \_ “a” coincide con “Barb**a**papa” en 4 posicines
- Un metacaracter es un carácter con un significado especial
  - \_ “.” coincide con cualquier carácter individual
  - \_ “a.” coincide con “Barb**a**papa” en 3 posiciones



# Metacaracteres como caracteres y viceversa...

---

En ocasiones es preciso usar un metacaracter como un literal

- Por ejemplo para encontrar el final de frase acabado en “.”
  - \_ Para hacerlo se precede el metacaracter con “\”
  - \_ “\.” representa un punto. Coincide con una posición con la cadena
    - “Hasta aquí hemos llegado.”
    - “.” representa cualquier caracter. Coincide con todas las posiciones de la cadena anterior (incluido el “.”)

Algunos metacaracteres se componen de contrabarra seguida de un

- carácter alfabético: \n, \t

El literal contrabarra se obtiene con “\\”

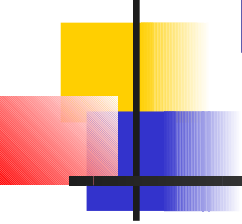
-



# Expresiones regulares en Perl

---

- Los operadores “=~” y “!~” sirven para ver si una cadena dada contienen un patron.
- “=~” devuelve
  - TRUE si el patron coincide con la cadena y
  - FALSE en caso contrario



# Ejemplo (1)

---

```
#!/usr/bin/perl
use strict; use warnings;
my $patron = "HUMAN";
my $ARCHIVO = "pruebaFasta.txt";
open FILE, $ARCHIVO or die "No puedo
    abrir el archivo $ARCHIVO $!";
while (my $line = <FILE>) {
    if ($line =~ /$patron/) {
        print $line."\n";
    }
}
close FILE;
exit;
```

```
>3BHS1_RAT
PGWSCLVTGAGGFVGQRIIRMLVQEK
>3BHS2_RAT
PGWSCLVTGAGGFVGQRIIRMLVQ
>3BHS4_HUMAN
PGWSCLVTGAGGFLGQRIIVQLLVQ
```

perl ejemplo\_3\_1.pl

```
>3BHS4_HUMAN
```



# Metacaracteres

---

- Los metacaracteres no se interpretan literalmente sino que tienen un significado especial dentro de la ER.

## ***Symbol    Meaning***

<code>\...</code>	Used to escape metacharacters (including itself) or to make the next character a metacharacter (like <code>\s</code> , <code>\w</code> , <code>\n</code> )
<code>... ...</code>	Alternation (match one or the other)
<code>(...)</code>	Grouping (treat as a unit)
<code>[...]</code>	Character class (match one character from a set)
<code>^</code>	True at the beginning of string (or sometimes after any newline)
<code>\$</code>	True at the end of the string (or sometimes before any newline)
<code>.</code>	Match any one character (except newline, normally)



# Ejemplos

---

Regex	Matches	Doesn't match
(dog)	dog, dogo, (dog)	dolar
\(dog\)	(dog)	dog
gat gos	gat, gos, gatigos, gatita	got
Mi[gqc][uh]el	Miquel, Miquel, Michel	Michael
^lone	lonely	abalone
me\$	give me, dame	love me tender
Mr. Sanchez	Mr. Sanchez, Mrs Sanchez	Mrs. Sanchez





# Cuantificadores

---

- Sirven para indicar cuantas veces puede o debe coincidir con la ER el grupo de caracteres que precede al cuantificador

## Quantifier Meaning

*	Match 0 or more times
+	Match 1 or more times
?	Match 0 or 1 time
{COUNT}	Match exactly COUNT times
{MIN, }	Match at least MIN times
{MIN, MAX}	Match at least MIN times but not more than MAX times



# Clases de caracteres

- Agrupaciones especiales de caracteres para los que es de esperar que posean cierta propiedad.

Symbol	Meaning	Character Class
<code>\d</code>	Digit	<code>[0-9]</code>
<code>\D</code>	Nondigit	<code>[^0-9]</code>
<code>\s</code>	Whitespace	<code>[ \t\n\r\f]</code>
<code>\S</code>	Nonwhitespace	<code>[^ \t\n\r\f]</code>
<code>\w</code>	Word character	<code>[a-zA-Z0-9_]</code>
<code>\W</code>	Non-(word character)	<code>[^a-zA-Z0-9_]</code>



# Cuantificadores y metacaracteres

---

- La combinación de metacaracteres y cuantificadores permite describir muchos tipos de cadenas
  - **m/.\*/** coincide con cualquier cadena de cualquier tipo
  - **m/ +/** coincide con 1 o mas espacios
  - **m/\s+/** coincide con 1 o mas espacios, tabuladores etc.



# Ejemplos de combinación de Cuantificadores y metacaracteres

---

ACH1_BOMMO	Q03383	400	Antichymotrypsin-1 precursor (Antichymotrypsin I) (ACHY-I)
ACH1_CAEEL	P48180	498	Acetylcholine receptor protein subunit alpha-type acr-16 precursor
ACH1_ASHGO	Q754Q2	523	Acetyl-CoA hydrolase (EC 3.1.2.1)

{

\w+ \s+



# Ejemplos de combinación de Cuantificadores y metacaracteres

---


ACH1_BOMMO	Q03383	400	Antichymotrypsin-1 precursor (Antichymotrypsin I) (ACHY-I)
ACH1_CAEEL	P48180	498	Acetylcholine receptor protein subunit alpha-type acr-16 precursor
ACH1_ASHGO	Q754Q2	523	Acetyl-CoA hydrolase (EC 3.1.2.1)

$\underbrace{\hspace{1.5cm}}$        $\underbrace{\hspace{1.5cm}}$

$\backslash w+ \backslash s+ \backslash w+ \backslash s+$

# Ejemplos de combinación de Cuantificadores y metacaracteres

ACH1_BOMMO	Q03383	400	Antichymotrypsin-1 precursor (Antichymotrypsin I) (ACHY-I)
ACH1_CAEEL	P48180	498	Acetylcholine receptor protein subunit alpha-type acr-16 precursor
ACH1_ASHGO	Q754Q2	523	Acetyl-CoA hydrolase (EC 3.1.2.1)



`\w+ \s+ \w+ \s+ \d+ \s+ . *`



# Ejemplos de combinación de Cuantificadores y metacaracteres

---

ACH1_BOMMO	Q03383	400	Antichymotrypsin-1 precursor (Antichymotrypsin I) (ACHY-I)
ACH1_CAEEL	P48180	498	Acetylcholine receptor protein subunit alpha-type acr-16 precursor
ACH1_ASHGO	Q754Q2	523	Acetyl-CoA hydrolase (EC 3.1.2.1)

*(Note: In the original image, the third row is color-coded: ACH1\_ASHGO is red, Q754Q2 is blue, 523 is green, and Acetyl-CoA hydrolase (EC 3.1.2.1) is blue. Braces are placed under each column.)*

`\w+\s+\w+\s+\d+\s+.*\s+(EC 3\.1\.2\.1\)`

Este último patrón es tan especializado  
que coincide tan sólo con una de las cadenas



# Captura

---

- Consiste en extraer partes de las cadenas que coinciden con un patrón
- Asignando paréntesis sucesivos a las variables \$1, \$2, etc.

```
if ( $line =~ /(DNA|Protein) sequence is (.+)/) {  
    my $moleculeType = $1;  
    my $sequence = $2;}  
}
```

- Asignando el resultado a una lista

```
my ($moleculeType, $sequence) = $line =~ /(DNA|Protein) sequence is (.+)/;
```

- Asignando el resultado a un array

```
my @results = $line =~ /$pattern/;
```





# Sustituciones

---

- `$string =~ s/pattern/replacement/`  
*sustituye la primera coincidencia únicamente*
- `$string =~ s/pattern/replacement/g`  
*sustituye todas las coincidencias*
- `$string =~ s/pattern/replacement/ig`  
*sustituye todas las coincidencias e ignora las mayúsculas*
- Si se omite `$string` se aplica a la variable genérica `$_`



# Ejemplo de sustituciones

---

```
#!/usr/bin/perl
use strict;
use warnings;
my ($input, $output)=@ARGV;
    # get input and output from command-line
open IN, $ input or die "Cannot open $input : $!";
open OUT, ">". $output or die "Cannot create $output : $!";
while (my $line = <IN>) {
    $line =~ s/\r\n/\n/;
    # convert a Windows to a UNIX format
    print OUT $line;
    # write new line to output file
}
close(IN);
close(OUT)
```



# Operadores de regexps: MATCHING

---

- Coincidencia:

EXPR =~ m/PATTERN/

```
my $nMatch = $sequence =~ m/AUG/;
```

```
    # return 1 if match has succeeded
```

```
my @matchList = $sequence =~ m/AUG/;
```

```
    # return list of matches
```



# Operadores de regexps: SUSTITUCIÓN

---

- Sustitución:

`VAR =~ s/PATTERN/REPLACEMENT/`

```
$sequence =~ s/T/U/gi;
```

```
    # replace Thymine with Uracil
```

```
$replacements = $sequence =~ s/T/U/gi;
```

```
    # replace and count number of  
    replacements
```



# Operadores de regexps: TRANSLITERACIÓN

---

- Transliteración

```
VAR =~ tr/SEARCHLIST/REPLACEMENTLIST/
```

```
$sequence =~ tr/ATGC/TACG/;
```

```
    # reverse DNA complement
```

```
$count = $sequence =~ tr/ATGC/TACG/;
```

```
    # complement and return # of  
    replacements
```



# Aplicaciones en bioinformatica

---

- El capítulo 10 del libro *Beginning perl for Bioinformatics*

<http://oreilly.com/catalog/begperlbio/chapter/ch10.html>

contiene ejemplos de uso de perl para el análisis de archivos de Genbank