# Programación y Administración de Sistemas

## Práctica 3. Expresiones regulares para programación de la *shell*.

Pedro Antonio Gutiérrez

Asignatura "Programación y Administración de Sistemas"
2º Curso Grado en Ingeniería Informática
Escuela Politécnica Superior
(Universidad de Córdoba)
pagutierrez@uco.es

15 de abril de 2016

# ¿Qué son las expresiones regulares?

- Una expresión regular (*regex*) describe un conjunto de cadenas de texto.
- Se utilizan:
  - En entornos UNIX, con comandos como `grep`, `sed`, `awk`...
  - De manera intensiva, en lenguajes de programación como `perl`, `python`, `ruby`, `XML`...
  - En bases de datos.
- Ahorran mucho tiempo y hacen el código más robusto.

# ¿Qué son las expresiones regulares?

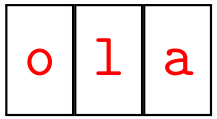
- La expresión regular más simple sería la que busca una secuencia fija de caracteres literales.
- La cadena cumple la expresión regular si contiene esa secuencia.

| o | l | a |
|---|---|---|

`Ella me dijo hola.` ⇒ Empareja.
`Ella me dijo mola.` ⇒ Empareja.
`Ella me dijo adiós.` ⇒ No empareja.

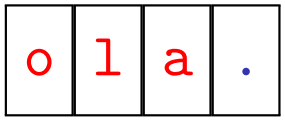
# ¿Qué son las expresiones regulares?

- Puede que la expresión regular empareje a la cadena en más de un punto:




`Lola me dijo hola.` ⇒ Empareja 2 veces.

- El carácter "." empareja cualquier cosa:




`Lola me dijo hola.` ⇒ Empareja 2 veces.

# ¿Por qué las necesito?

- ¿Para qué necesito aprender a utilizar las *regex*?
- Historia real[1]:
  - Direcciones de calles.
  - Quiero actualizar su formato, de "100 NORTH MAIN ROAD" a "100 NORTH MAIN RD.", sobre un conjunto de muchas carreteras.

```
1  pedroa@pagutierrezLaptop:~$ echo "100 NORTH MAIN ROAD" | sed -e 's/ROAD/RD\./'
2  100 NORTH MAIN RD.
3  pedroa@pagutierrezLaptop:~$ cat carreteras.txt
4  100 NORTH MAIN ROAD
5  45 ST JAMES ROAD
6  100 NORTH BROAD ROAD
7  pedroa@pagutierrezLaptop:~$ cat carreteras.txt | sed -e 's/ROAD/RD\./'
8  100 NORTH MAIN RD.
9  45 ST JAMES RD.
10 100 NORTH BRD. ROAD
```

[1] http://www.gulic.org/almacen/diveintopython-5.4-es/regular_expressions/street_addresses.html

# ¿Por qué las necesito?

- ¿Para qué necesito aprender a utilizar las *regex*?
  - A veces necesito hacer operaciones con cadenas con expresiones relativamente complejas.
  - P.Ej.: reemplazar "ROAD" por "RD." siempre que esté al final de la línea (carácter especial $).

```
1 pedroa@pagutierrezLaptop:~$ cat carreteras.txt | sed -e 's/ROAD$/RD\./'
2 100 NORTH MAIN RD.
3 45 ST JAMES RD.
4 100 NORTH BROAD RD.
```

# Caracteres especiales

- Las expresiones regulares se componen de caracteres normales (literales) y de caracteres especiales (o metacaracteres).
- "`[...]`": sirve para indicar un rango de caracteres:

| b | [iur] | e |
|---|---|---|

`Octubre me dijo bueno bien.` ⇒ Empareja 3 veces.

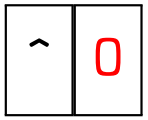
- "`[^...]`": sirve para *negar* la ocurrencia de un carácter:

| b | [^ur] | e |
|---|---|---|

`Octubre me dijo bueno bien.` ⇒ Empareja 1 vez.

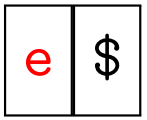
# Caracteres especiales

- "^": empareja con el principio de una línea:




Octubre me dijo bueno ⇒ Empareja 1 vez.

- "$": empareja con el final de una línea:




Bueno, me dijo octubre ⇒ Empareja 1 vez.

# Caracteres especiales

- "*": empareja con cero, una o más ocurrencias del carácter anterior:

| o | l | a | * | s |
|---|---|---|---|---|

Holaaaaaaas ⇒ Empareja 1 vez.

Hols ⇒ Empareja 1 vez.

- En caso de duda, el emparejamiento siempre es el de mayor longitud:

| a | . | * | e |
|---|---|---|---|

Olas emocionantes.

# Caracteres especiales

- Los paréntesis `()` (o `\(\)`) permiten agrupar caracteres a la hora de aplicar los metacaracteres:
  - `a*` empareja `a`, `aa`, `aaa`...
  - `abc*` empareja `ab`, `abc`, `abcc`, `abccc`...
  - `(abc)*` empareja `abc`, `abcabc`, `abcabcabc`...
- Dos tipos de expresiones regulares:
  - *Basic Regular Expressions* (BRE): propuesta inicial en el estándar POSIX.
  - *Extended Regular Expressions* (ERE): ampliación con nuevos metacaracteres.
- Cada aplicación utiliza una u otra.

# Caracteres especiales

| Carácter | BRE | ERE | Significado |
|---|---|---|---|
| `\` | ✓ | ✓ | Interpreta de forma literal el siguiente carácter |
| `.` | ✓ | ✓ | Selecciona un carácter cualquiera |
| `*` | ✓ | ✓ | Selecciona ninguna, una o varias veces lo anterior |
| `^` | ✓ | ✓ | Principio de línea |
| `$` | ✓ | ✓ | Final de línea |
| `[...]` | ✓ | ✓ | Cualquiera de los caracteres que hay entre corchetes |
| `\n` | ✓ | ✓ | Utilizar la `n`-ésima selección almacenada |
| `{n,m}` | X | ✓ | Selecciona lo anterior entre `n` y `m` veces |
| `+` | X | ✓ | Selecciona una o varias veces lo anterior |
| `?` | X | ✓ | Selecciona una o ninguna vez lo anterior |
| `\|` | X | ✓ | Selecciona lo anterior o lo posterior |
| `(...)` | X | ✓ | Selecciona la secuencia que hay entre paréntesis[2] |
| `\{n,m\}` | ✓ | X | Selecciona lo anterior entre `n` y `m` veces |
| `\(...\)` | ✓ | X | Selecciona la secuencia que hay entre paréntesis[2] |
| `\\|` | ✓ | X | Selecciona lo anterior o lo posterior |

[2] Se almacena la selección

# Rangos de caracteres

- `[aeiou]`: empareja con las letras `a`, `e`,`i`,`o` y `u`.
- `[1-9]` es equivalente a `[123456789]`.
- `[a-e]` es equivalente a `[abcde]`.
- `[1-9a-e]` es equivalente a `[123456789abcde]`.
- Los rangos típicos se pueden especificar de la siguiente forma:
  - `[[:alpha:]]`→ `[a-zA-Z]`.
  - `[[:alnum:]]`→ `[a-zA-Z0-9]`.
  - `[[:lower:]]`→ `[a-z]`.
  - `[[:upper:]]`→ `[A-Z]`.
  - `[R[:lower:]]`→ `[Ra-z]`.
  - Otros[3]: *digit*, *punct*, *cntrl*, *blank*...

[3] `man wctype`

# Comando grep

- `grep` proviene del editor `ed` (editor de texto Unix), y en concreto, de su comando de búsqueda de expresiones regulares "global regular expression print".
- Se utiliza cuando sabes que un fichero contiene una determinada expresión y quieres saber que fichero es.
- `grep` utiliza las BRE, `egrep` utiliza las ERE (no obstante, podemos usar `grep -E` para que considere ERE).
- Consejo: antes de incluirlas en el *script*, probar las expresiones regulares en la consola con `grep` (se resaltan los emparejamientos con `grep --colour`, que suele estar activo por defecto).

# Comando grep

- Como muchos de los caracteres especiales de las *regex* son también especiales en `bash`, es una buena costumbre rodear la *regex* con comillas simples (`' '`) cuando estemos escribiendo un *script* $\rightarrow$ Siempre que la *regex* no contenga variables.
- `-i`: hace que considere igual mayúsculas y minúsculas.
- `-o`: en lugar de imprimir las líneas completas que cumplen el patrón, solo muestra el emparejamiento del patrón.
- `-v`: mostrar las líneas que `no` cumplen el patrón.

# Comando grep

```
1  pedroa@pagutierrezLaptop:~$ cat ejemplo.txt
2  Este es otro ejemplo de expresiones regulares
3  La segunda parte ya la veremos
4  ,,,,adios,hola
5  pedroa@pagutierrezLaptop:~$ cat ejemplo.txt | grep '^E'
6  Este es otro ejemplo de expresiones regulares
7  pedroa@pagutierrezLaptop:~$ cat ejemplo.txt | grep -E '^(E|L)'
8  Este es otro ejemplo de expresiones regulares
9  La segunda parte ya la veremos
10 pedroa@pagutierrezLaptop:~$ cat ejemplo.txt | grep -E ',*'
11 Este es otro ejemplo de expresiones regulares
12 La segunda parte ya la veremos
13 ,,,,adios,hola
14 pedroa@pagutierrezLaptop:~$ cat ejemplo.txt | grep -E ',+'
15 ,,,,adios,hola
16 pedroa@pagutierrezLaptop:~$ cat ejemplo.txt | grep -E ',+' -o
17 ,,,,
18 ,
19 pedroa@pagutierrezLaptop:~$ cat ejemplo.txt | grep -E 'L(..).*\1'
20 La segunda parte ya la veremos
```

# Comando grep

- Encontrar todos los números con signo (con posibilidad o no de decimales):
  `[-+][0-9]+(\.[0-9]+)?`

```
1 pedroa@pagutierrezLaptop:~$ grep -E '[-+][0-9]+(\.[0-9]+)?' $(find -name "*.c")
2 ./svorex/loadfile.c: strcat (buf, pstr+4) ;
3 ./gpor/lgam1.c:              -0.0002109075,0.0742379071,0.0815782188,
```

- 5 números decimales o más (sin signo):
  `[0-9]+\.[0-9]{5,}`

```
1 pedroa@pagutierrezLaptop:~$ grep -E '[0-9]+\.[0-9]{5,}' $(find -name "*.c")
2 ./gpor/lgam1.c:              -0.0002109075,0.0742379071,0.0815782188,
```

# Comando sed

- Es parecido a `grep` pero permite cambiar las líneas que encuentra (en lugar de solo mostrarlas).
- En realidad, es un editor de textos no interactivo, que recibe sus comandos como si fuesen un *script*.
- Los comandos que utiliza son los mismos que los de `ed`.
- Solo vamos a estudiar algunos de los comandos posibles.
- Por defecto, todas las líneas se imprimen tras aplicar el comando.

# Comando sed

- `sed [-r] [-n] -e 'comando' archivo`:
  - `-r`: uso de EREs en lugar de BREs.
  - `-n`: modo silencioso → para imprimir una línea tienes que indicarlo explícitamente mediante el comando `p` (*print*).
  - `-e 'comando'`: ejecutar el comando o comandos especificados.
  - Sintaxis de comandos:
    `[direccionInicio[, direccionFin]][!]comando [argumentos]`:
    - Si la dirección es adecuada, entonces se ejecutan los comandos (con sus argumentos).
    - Las direcciones pueden ser expresiones regulares (`/regex/`) o números de línea (`1`).
    - Si no hay `direccionFin` solo se aplica sobre `direccionInicio`.
    - `!` emparejaría todas las direcciones distintas que la indicada.

# Comando sed

- d: borrar líneas direccionadas.
- p: imprimir líneas direccionadas.
- s: sustituir una expresión por otra sobre las líneas seleccionadas. Sintaxis: s/patron/reemplazo/[banderas]
  - patron: expresión regular BRE.
  - reemplazo: cadena con qué reemplazarla.
  - Bandera n: reemplazar sólo la ocurrencia n-ésima.
  - Bandera g: reemplazar todas las ocurrencias.
  - Bandera p: forzar a imprimir la línea (solo tiene sentido si hemos utilizado -n).

# Comando sed

```
1  i02gupep@NEWTS:~/pas/1415/p2$ cat ejemplo.txt
2  Este es otro ejemplo de expresiones regulares
3  La segunda parte ya la veremos
4  ,,,,adios,hola
5  i02gupep@NEWTS:~/pas/1415/p2$ cat ejemplo.txt | sed -e '3p'
6  Este es otro ejemplo de expresiones regulares
7  La segunda parte ya la veremos
8  ,,,,adios,hola
9  ,,,,adios,hola
10 i02gupep@NEWTS:~/pas/1415/p2$ cat ejemplo.txt | sed -n -e '3p'
11 ,,,,adios,hola
12 i02gupep@NEWTS:~/pas/1415/p2$ cat ejemplo.txt | sed -n -e '1,2p'
13 Este es otro ejemplo de expresiones regulares
14 La segunda parte ya la veremos
15 i02gupep@NEWTS:~/pas/1415/p2$ cat ejemplo.txt | sed -n -e '1,2!p'
16 ,,,,adios,hola
17 i02gupep@NEWTS:~/pas/1415/p2$ cat ejemplo.txt | sed -e '/^L/d'
18 Este es otro ejemplo de expresiones regulares
19 ,,,,adios,hola
20 i02gupep@NEWTS:~/pas/1415/p2$ cat ejemplo.txt | sed -e '2,$d'
21 Este es otro ejemplo de expresiones regulares
22 i02gupep@NEWTS:~/pas/1415/p2$ cat ejemplo.txt | sed  -e '1,/s$/d'
23 ,,,,adios,hola
```

# Comando sed

```
1  i02gupep@NEWTS:~/pas/1415/p2$ cat ejemplo.txt
2  Este es otro ejemplo de expresiones regulares
3  La segunda parte ya la veremos
4  ,,,,adios,hola
5  i02gupep@NEWTS:~/pas/1415/p2$ cat ejemplo.txt | sed  -r -e 's/La/El/'
6  Este es otro ejemplo de expresiones regulares
7  El segunda parte ya la veremos
8  ,,,,adios,hola
9  i02gupep@NEWTS:~/pas/1415/p2$ cat ejemplo.txt | sed  -r -e 's/[Ll]a/El/'
10 Este es otro ejemplo de expresiones reguElres
11 El segunda parte ya la veremos
12 ,,,,adios,hoEl
13 i02gupep@NEWTS:~/pas/1415/p2$ cat ejemplo.txt | sed  -r -e 's/([Ll])a/era\1/'
14 Este es otro ejemplo de expresiones regueralres
15 eraL segunda parte ya la veremos
16 ,,,,adios,hoeral
17 i02gupep@NEWTS:~/pas/1415/p2$ cat ejemplo.txt | sed -r -n -e 's/(d[ea])/"\1"/p'
18 Este es otro ejemplo "de" expresiones regulares
19 La segun"da" parte ya la veremos
20 i02gupep@NEWTS:~/pas/1415/p2$ cat ejemplo2.txt
21 Grado:Informatica
22 Informatica2:Grado2
23 i02gupep@NEWTS:~/pas/1415/p2$ cat ejemplo2.txt | sed -r -n -e 's/(.*):(.*)
       /\2:\1/p'
24 Informatica:Grado
25 Grado2:Informatica2
```

# Comando sed

- Ejercicio: Utilizar expresiones regulares con sed, para transformar la salida del comando df al formato indicado abajo.

```
1  pedroa@pedroa-laptop ~ $ ./espacioLibre.sh
2  El fichero de bloques /dev/sda2, montado en /, tiene usados 18218120 bloques de
       un total de 49410864 (porcentaje de 39%).
3  El fichero de bloques udev, montado en /dev, tiene usados 0 bloques de un total
       de 10240 (porcentaje de 0%).
4  El fichero de bloques tmpfs, montado en /run, tiene usados 928 bloques de un
       total de 601488 (porcentaje de 1%).
5  El fichero de bloques tmpfs, montado en /run/lock, tiene usados 0 bloques de un
       total de 5120 (porcentaje de 0%).
6  El fichero de bloques tmpfs, montado en /run/shm, tiene usados 1560 bloques de
       un total de 2025480 (porcentaje de 1%).
7  El fichero de bloques /dev/sdb1, montado en /boot/efi, tiene usados 42932
       bloques de un total de 262144 (porcentaje de 17%).
8  El fichero de bloques /dev/sda3, montado en /home, tiene usados 50397976 bloques
        de un total de 65282844 (porcentaje de 82%).
9  El fichero de bloques /dev/sdb6, montado en /home2, tiene usados 282248360
       bloques de un total de 372531364 (porcentaje de 80%).
10 El fichero de bloques none, montado en /sys/fs/cgroup, tiene usados 0 bloques de
        un total de 4 (porcentaje de 0%).
```

# Inciso: problemas con espacios en blanco y arrays

- Cuando intentamos construir un *array* a partir de una cadena, `bash` utiliza determinados caracteres para separar cada uno de los elementos del *array*.
- Estos caracteres están en la variable de entorno IFS y por defecto son el espacio, el tabulador y el salto de línea.

```
1  pedroa@Laptop:~$ array=($(echo "1 2 3"))
2  pedroa@Laptop:~$ echo ${array[0]}
3  1
4  pedroa@Laptop:~$ echo ${array[1]}
5  2
6  pedroa@Laptop:~$ echo ${array[2]}
7  3
8  pedroa@Laptop:~$ array=($(echo -e "1\t2\n3"))
9  pedroa@Laptop:~$ echo ${array[0]}
10 1
11 pedroa@Laptop:~$ echo ${array[1]}
12 2
13 pedroa@Laptop:~$ echo ${array[2]}
14 3
```

# Inciso: problemas con espacios en blanco y arrays

- Esto nos puede producir problemas si estamos procesando elementos con espacios (por ejemplo, nombres de ficheros con espacios):

```
1  pedroa@Laptop:~$ array=($(echo -e "El uno\nEl dos\nEl tres"))
2  pedroa@Laptop:~$ echo ${array[0]}
3  El
4  pedroa@Laptop:~$ echo ${array[1]}
5  uno
```

- Solución: cambiar el IFS para que solo se utilice el `\n`:

```
1  pedroa@Laptop:~$ OLDIFS=$IFS
2  pedroa@Laptop:~$ IFS=$'\n'
3  pedroa@Laptop:~$ array=($(echo -e "El uno\nEl dos\nEl tres"))
4  pedroa@Laptop:~$ echo ${array[0]}
5  El uno
6  pedroa@Laptop:~$ echo ${array[1]}
7  El dos
8  pedroa@Laptop:~$ IFS=$OLDIFS
```

# Referencias

Kochan and Wood.
Unix shell programming
Sams Publishing. Tercera Edición. 2003.

# Programación y Administración de Sistemas

## Práctica 3. Expresiones regulares para programación de la *shell*.

Pedro Antonio Gutiérrez

Asignatura "Programación y Administración de Sistemas"
2º Curso Grado en Ingeniería Informática
Escuela Politécnica Superior
(Universidad de Córdoba)
pagutierrez@uco.es

15 de abril de 2016