UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Ingeniería en Sistemas de Información

Sintaxis y Semántica de los Lenguajes

Curso K2054

TRABAJO PRÁCTICO INDIVIDUAL N° 1

Expresiones regulares y expresiones regulares extendidas en Bash

Legajo: 163698-4
Alumno: Larramendy, Juan Camilo
Mail: jlarramendy@frba.utn.edu.ar

Repositorio:
https://github.com/larramendyjuan/ssl_k2054_larramendy

Usuario Github: larramendyjuan

Docente: Ing. Pablo Damián Mendez

2 - Casos de prueba

A- Reemplazar cada punto del archivo por punto y salto de línea generando un nuevo archivo.

Dado un archivo con el siguiente texto:

Esta lucha aún no ha cesado. No podría decirse que la Argentina es un país estabilizado. Sus problemas son profundos y complejos. Es difícil estabilizar una sociedad muy diversificada.

La historia de la Argentina es la de una vasta aventura. En esa historia se esconde el secreto de lo que hoy es la Argentina.

Se obtiene otro archivo con lo siguiente:

Esta lucha aún no ha cesado.
No podría decirse que la Argentina es un país estabilizado.
Sus problemas son profundos y complejos.
Es difícil estabilizar una sociedad muy diversificada.

La historia de la Argentina es la de una vasta aventura.
En esa historia se esconde el secreto de lo que hoy es la Argentina.

B- Borrar las líneas en blanco.

Dado un archivo

```
1  Línea con palabras.
2
3  Línea no en blanco.
4
5
6  Última línea.
7
```

Se obtiene:

```
user@DESKTOP-9OG2KCE:/mnt/c/users/user/desktop$ sed '/^\s*$/d' test.txt
Línea con palabras.
Línea no en blanco.
Última línea.
```

C- Cree un nuevo archivo con el resultado de las operaciones a y b (redireccionamiento de la salida estándar).

Dado un archivo:

```
1  Línea con palabras. Línea no en blanco. Última línea.
2
3  Última línea. Línea con palabras. Línea no en blanco.
4
```

Se genera:

```
1  Línea con palabras.
2  Línea no en blanco.
3  Última línea.
4  Última línea.
5  Línea con palabras.
6  Línea no en blanco.
```

D- Del archivo, liste todas las oraciones que contengan la palabra “guerra” sin distinguir mayúsculas y minúsculas

Dado un archivo:

```
1  Hubo una guerra.
2
3  Hoy es domingo.
4
5  El Ministerio de Guerra de la Nación.
```

Se genera:

```
user@DESKTOP-9OG2KCE:/mnt/c/users/user/desktop$ grep -i 'guerra' test.txt
Hubo una guerra.
El Ministerio de Guerra de la Nación.
```

E- Mostrar las líneas que empiecen con “A” y terminen con “s” o “s.”.

Dado un archivo:

```
1  Ahora mismo es lunes.
2
3  Aquí comienzan los días
4
5  Una oración distinta
6
7  Un lunes.
```

Se genera:

```
user@DESKTOP-9OG2KCE:/mnt/c/users/user/desktop$ grep -E '^A.*(s|s\.)$' test.txt
Ahora mismo es lunes.
Aquí comienzan los días
```

F- Indique en cuántas oraciones aparece la palabra “peronismo”

Dado un archivo:

```
1  El peronismo sin Perón.
2
3  Una línea de separación.
4
5  En Argentina, existe un movimiento político denominado peronismo.
```

Resulta:

```
user@DESKTOP-9OG2KCE:/mnt/c/users/user/desktop$ grep -c 'peronismo' test.txt
2
```

G- Muestre la cantidad de oraciones que tienen la palabra “Sarmiento” y “Rosas”.

Dado:

```
1  Sarmiento fue docente.
2
3  En 1793 nació Rosas.
4
5  Rosas y Sarmiento fueron determinantes para la historia de Argentina.
6
7  Sarmiento y Rosas sabían leer.
8
9  Última línea.
```

Resulta:

```
user@DESKTOP-9OG2KCE:/mnt/c/users/user/desktop$ grep -c -E '(Sarmiento.*Rosas|Rosas.*Sarmiento)' test.txt
2
```

H- Muestre las oraciones que tengan fechas referidas al siglo XIX.

Dado:

```
1   Corría el año 1901.
2
3   El año anterior fue 1900.
4
5   El descubrimiento de América fue en 1492.
6
7   En 1816 se declaró la independencia argentina.
8
9   1800 no es siglo XIX.
10
11  1801 sí lo es.
```

Resulta:

```
user@DESKTOP-9OG2KCE:/mnt/c/users/user/desktop$ grep -E '1900|18[0-9][1-9]' test.txt
El año anterior fue 1900.
En 1816 se declaró la independencia argentina.
1801 sí lo es.
```

I- Borre la primera palabra de cada línea

Dado:

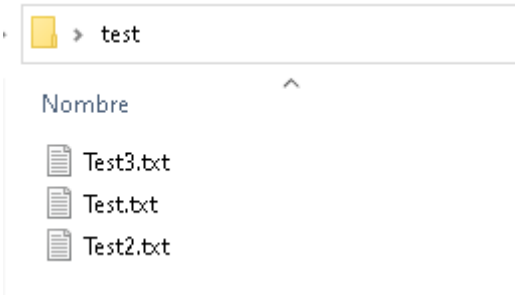
```
1   Corría el año 1901.
2
3   El año anterior fue 1900.
4
5   El descubrimiento de América fue en 1492.
6
7   En 1816 se declaró la independencia argentina.
8
9   1800 no es siglo XIX.
10
11  1801 sí lo es.
```

Resulta:

```
user@DESKTOP-9OG2KCE:/mnt/c/users/user/desktop$ sed 's/^[^ ]* //g' test.txt
el año 1901.

año anterior fue 1900.

descubrimiento de América fue en 1492.

1816 se declaró la independencia argentina.

no es siglo XIX.

sí lo es.
```

J- Enumere todos los archivos de una carpeta que contengan extensión “.txt”

Dada la carpeta test:



Resulta:

```
user@DESKTOP-9OG2KCE:/mnt/c/users/user/desktop$ ls test | grep -E '*\.txt'
Test.txt
Test2.txt
Test3.txt
```

3 - **Variables**:

Para declarar y asignar una variable en Bash se utiliza la siguiente sintaxis:

```
variable=valor
```

Para utilizar el valor de esa variable, se antepone el símbolo $. Por ejemplo, para imprimir el contenido de la variable, la sintaxis será:

```
echo $variable
```

A las variables se les puede asignar una expresión regular o el resultado de un comando. Por ejemplo:

```
exp='peronismo'
numero=$(grep -c $exp breve_historia.txt)
echo "La palabra peronismo se menciona $numero veces"
```

La variable declarada en un script tendrá alcance global, a menos que se declare dentro de una función y sea expresamente declarada como local. En ese caso, tendrá el alcance de la función.

En Bash, hay algunas variables especiales y que están definidas por defecto, y que se refieren al script, al que ha ejecutado el script, o al entorno en el que se ha ejecutado el script. Algunas de ellas son:

- $0 representa el nombre del script

- $1 – $9 los primeros nueve argumentos que se pasan a un script en Bash
- $# el número de argumentos que se pasan a un script
- $@ todos los argumentos que se han pasado al script
- $? la salida del último proceso que se ha ejecutado
- $$ el ID del proceso del script
- $USER el nombre del usuario que ha ejecutado el script
- $HOSTNAME se refiere al *hostname* de la máquina en la que se está ejecutando el script
- $SECONDS se refiere al tiempo transcurrido desde que se inició el script, contabilizado en segundos.
- $RANDOM devuelve un número aleatorio cada vez que se *lee* esta variable.
- $LINENO indica el número de líneas que tiene nuestro script.
- $SHELL que indica el *shell* que estás ejecutando

- $EDITOR donde está indicado el editor por defecto.
- $HOME la ruta del usuario.
- $USERNAME el nombre del usuario.
- $PATH la ruta por defecto donde encontrar binarios, etc.

Sentencias condicionales

Las sentencias condicionales en Bash son if then else y case.

Ejemplo:

```
if [[ $(($numero % 2)) == 0 ]] then
  echo Par
else
  echo Impar
fi
```

En Bash también se puede anidar tantos if then else como sean necesarios.

Otra forma de hacerlo es utilizando la instrucción elif que sirve para indicar “si la condición no se cumplió, verifica esta otra”.

Ejemplo:

```
if [[ $edad -le 18 ]] then
  echo "Tu edad es igual o menor a 18 años"
elif [[ $edad -gt 18 ]] && [[ $edad -le 60 ]] then
  echo "Tu edad es mayor a 18 y menor o igual a 60 años"
else
  echo "Tu edad es mayor a 60 años"
fi
```

Un ejemplo de la utilización de la sentencia `case` sería el siguiente:

```
case $color in
    naranja)
        echo "amigable, social, seguridad"
        ;;
    violeta)
        echo "creatividad, imaginativo, sabio"
        ;;
    verde)
        echo "paz, salud, crecimiento"
        ;;
    *)
        echo "Lo siento, no conozco ese color"
        ;;
esac
```

Sentencias cíclicas

Una de las sentencias cíclicas en Bash es `while`.

Ejemplo:

```
i=0
while [ $i -lt 1000 ]
do
    echo $i
    ((i++))
done
```

El siguiente tipo de sentencia cíclica es `until` y es muy parecido al anterior. La diferencia entre ambos está en que con `while` el bucle continuará mientras se cumpla la condición. Sin embargo, en el caso de `until` el bucle continuará hasta que se cumpla la condición.

Ejemplo:

```
i=10
until [ $i -lt 0 ]
do
    echo $i
    ((i--))
done
```

La instrucción `for` es otra de las sentencias cíclicas. Se puede utilizar declarando rangos, utilizando una variable de iteración o con el resultado de la ejecución de un comando.

Ejemplo:

```
for ((i=1;i<$1;i++))
do
    if ((i%2))
    then
        echo $i
    fi
done
```

Dos opciones para controlar el flujo del programa son `break` y `continue`. El primero te permite interrumpir el flujo y el segundo continuarlo.

Ejemplo:

```
i=-1
while :
do
    ((i++))
    if [ $i -eq 2 ];then
        continue
    fi
    echo $i
    if [ $i -gt 5 ];then
        break
    fi
done
echo "Finaliza con $i"
```

Subprogramas

En Bash, los subprogramas pueden ser expresados como funciones.

Para declarar una función, se utiliza la siguiente sintaxis:

```
function mi_primera_funcion(){
    echo Hola Mundo
}
```

Para invocarla, basta con utilizar su nombre.

Si se declara una variable dentro de una función, el alcance será global. La única forma de que la variable tenga alcance local, es que se lo exprese explícitamente.

```
var1='fuera'
var2='fuera'

funcion_ambito(){
var1='dentro'
local var2='dentro'
var3='dentro'
local var4='dentro'
echo $var1 $var2 $var3 $var4
}

echo $var1 $var2
funcion_ambito
echo $var1 $var2 $var3 $var4
```