Expresiones regulares y expresiones regulares extendidas en Bash

**Alumno**: Guido Reboredo

**Legajo:** 159078-9

**Ciclo** **Lectivo**: 2021

**Curso**: K2054 – Jueves Noche

**GitHub:** guidoreboredo-frba

**Repositorio:** <https://github.com/guidoreboredo-frba/SintaxisYsemantica>

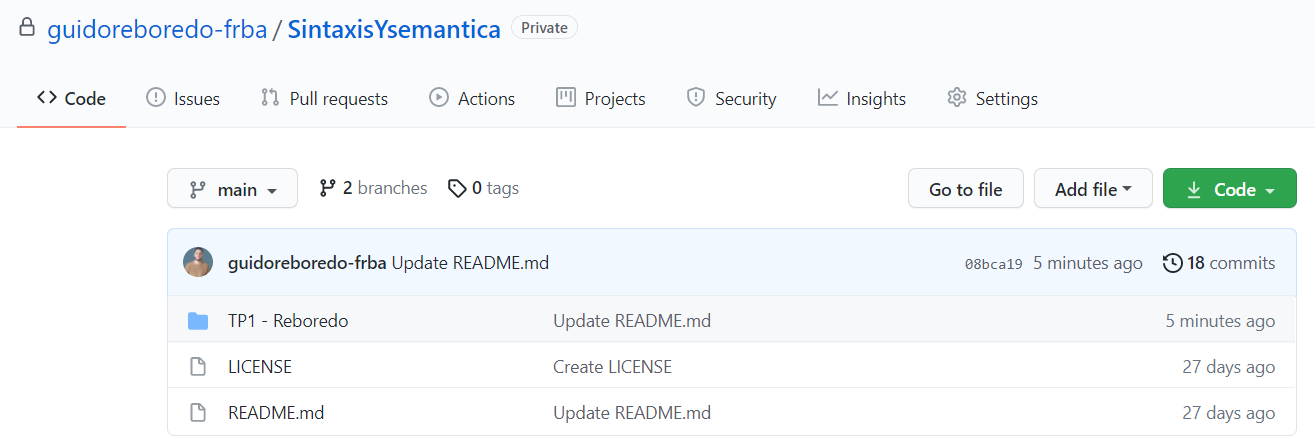
**Email:** guidoreboredo@frba.utn.edu.ar

**RESOLUCIONES:**

1. Crear un usuario en GitHub https://github.com/ con el correo institucional frba. Crear

un repositorio. Dentro del repositorio deberá subir todos los archivos que compongan la entrega de este trabajo dentro de una carpeta llamada “TP 1”. Para desarrollar este punto, ver el apéndice de este documento.

Repositorio creado: <https://github.com/guidoreboredo-frba/SintaxisYsemantica>



1. Debe entregar un único script que resuelva los siguientes puntos:
2. Reemplace cada punto del archivo “breve\_historia.txt” por punto y salto de línea generando un nuevo archivo.

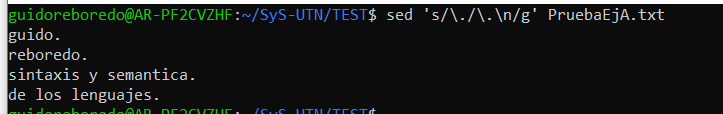
Para el ejercicio A se utilizó la siguiente sentencia:

**$ sed 's/\./\.\n/g' breve\_historia.txt > breve\_historia-PuntoA.txt**

Realizando la prueba con un archivo de ejemplo que contiene el siguiente texto:

**guido.reboredo.sintaxis y semantica.de los lenguajes.**

Se observa lo siguiente:



La salida de la sentencia será un nuevo archivo llamado **breve\_historia-PuntoA.txt** (Revisar la carpeta Anexos-Pruebas).

1. Borre todas las líneas en blanco.

Para borrar las líneas en blanco se utilizó la sentencia:

**$ sed -E '/^\s\*$/d' breve\_historia-PuntoA.txt > breve\_historia\_2.txt**

-E utiliza la sintaxis extendida,

**“^” inicio de línea**

**“\s” un espacio** **“\*”** **repetidos n veces**

**“$”** **final del archivo**

**“/d”** **Delete**

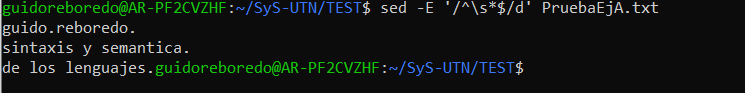
Si al mismo archivo de pruebas anterior le agregamos:

**guido.reboredo.**

**sintaxis y semantica.**

**de los lenguajes.**

El resultado es:



La respuesta de esta sentencia es un archivo llamado **“breve\_historia\_2.txt”** (Revisar la carpeta Anexos-Pruebas).

1. Cree un nuevo archivo: “breve\_historia\_2.txt” con el resultado de las operaciones a y b (redireccionamiento de la salida estándar).

Este archivo se creo en el paso anterior redireccionando la salida de la sentencia creada para ese punto.

1. Del archivo “breve\_historia.txt”, liste todas las oraciones que contengan la palabra “guerra” sin distinguir mayúsculas y minúsculas.

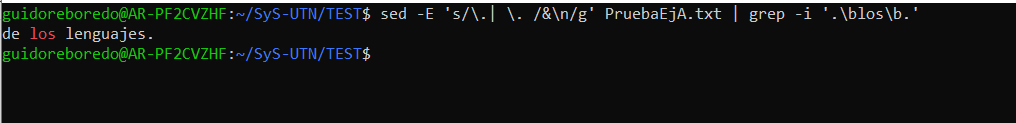
Se utilizó la sentencia:

**$ sed -E 's/\.| \. /&\n/g' breve\_historia.txt | grep -i '.\bguerra\b.'**

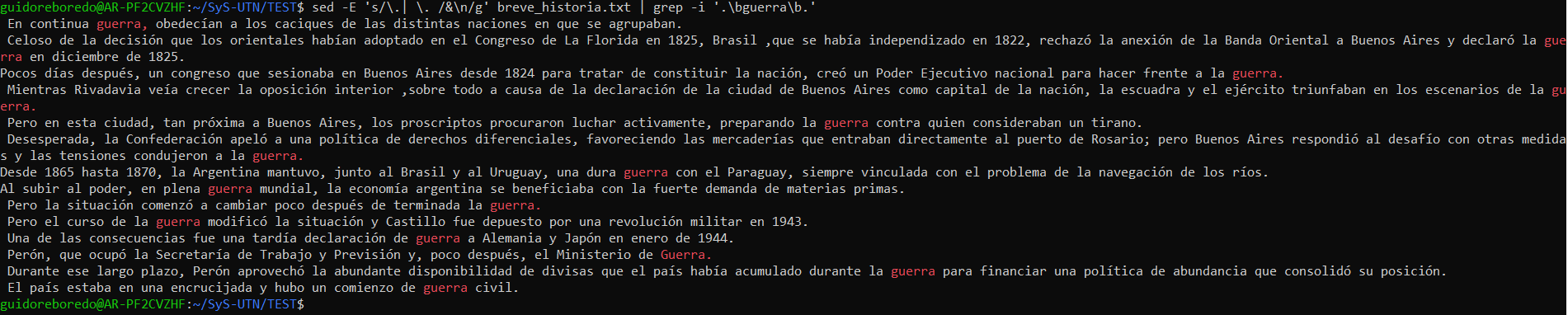
**La sentencia grep:** Busca la palabra guerra. **-i** lo vuelve NO Case Sensitive.

**El PIPE entre Grep y Sed** se hace para que solo liste las oraciones (es decir lo que sea que este entre dos puntos) que contienen la palabra guerra.

Realizando una prueba con el archivo anterior buscaremos la palabra “los”:



El resultado de esta sentencia sobre el archivo original es:

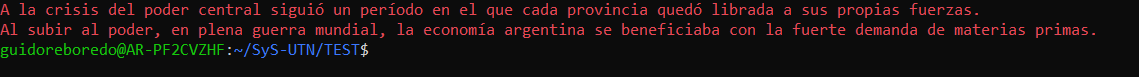


1. Muestre las líneas que empiecen con “A” y terminen con “s” o “s.” del archivo “breve\_historia.txt”.

En este caso se usa grep para buscar todas las palabras que comienzen con A luego lleven cualquier carácter n veces y concluyan con una letra “s” o “s.”.

**$ grep -E '^\A.\*(s|s\.)$' breve\_historia\_2.txt**

La prueba sobre el archivo devuelve:



1. Sobre el mismo archivo del punto anterior, Indique en cuántas oraciones aparece la palabra “peronismo”. Puede usar la opción -c para contar.

La sentencia para resolver este punto es:

**$ grep -E '.\bperonismo\b.' breve\_historia.txt -c**

**“\b” indica el comienzo y fin de una palabra.**

El resultado:



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

Para este punto se utilizó la siguiente sintaxis:

**$ grep -E '(Sarmiento.\*Rosas)|(Rosas.\*Sarmiento)' breve\_historia\_2.txt -c )**

De esta forma, se busca en las oraciones las palabras sarmiento y rosas en cualquier orden posible. El resultado de esta ejecución es simplemente el número de ocurrencias debido al **“-c”** añadido al final.

Nota: Se utilizó breve\_historia\_2.txt ya que ya se encuentra separado por oraciones (1 por línea).

Al ejecutarlo se obtuvo la siguiente respuesta:



1. Muestre las oraciones que tengan fechas referidas al siglo XIX.

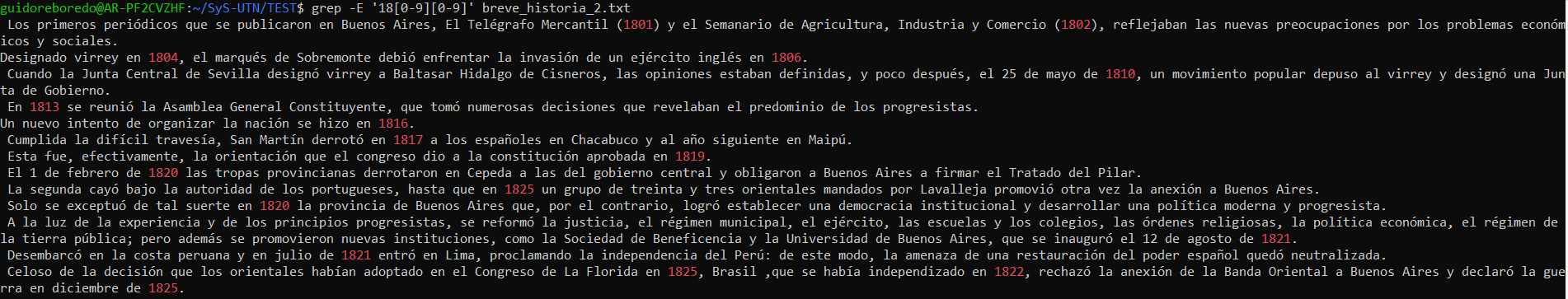
Usando la sentencia:

**$ grep -E '18[0-9][0-9]' breve\_historia\_2.txt**

Se buscan números que comiencen con 18 y continúen con dos números del 0 al nueve, es decir desde 1800 hasta 1899.

Nota: Se utilizó breve\_historia\_2.txt ya que ya se encuentra separado por oraciones (1 por línea).

Un extracto de la respuesta es el siguiente:



1. Borre la primera palabra de cada línea. Utilice substitución con sed. La sintaxis para sustituir la primera palabra de cada línea por “nada” sería: $sed “s/^[[a-zA-Z]]\*\b//g”nombre\_archivo (La “s” indica sustitución; entre los dos primeros /.../ está la expresión regular que queremos reemplazar, en este caso “/^[[a-zA-Z]]\*\b”; entre el segundo y el tercer “//” se indica la expresión por la cual será reemplazada, en este caso por la palabra vacía. Finalmente la “g” indica que el cambio será en todo el archivo.

La sentencia requerida es:

**$ sed 's/^[a-zA-Z]\*\b//g' breve\_historia.txt > breve\_historia-PuntoI.txt**

Su funcionamiento es explicado en el enunciado y su resultado es el siguiente:

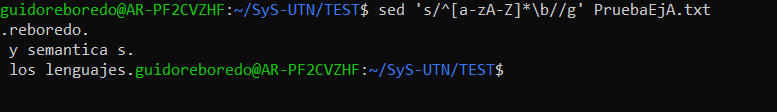
Si tomamos el documento de pruebas utilizado en el punto A:

guido.reboredo

sintaxis.y semántica s.

De los lenguajes.

Obtenemos:



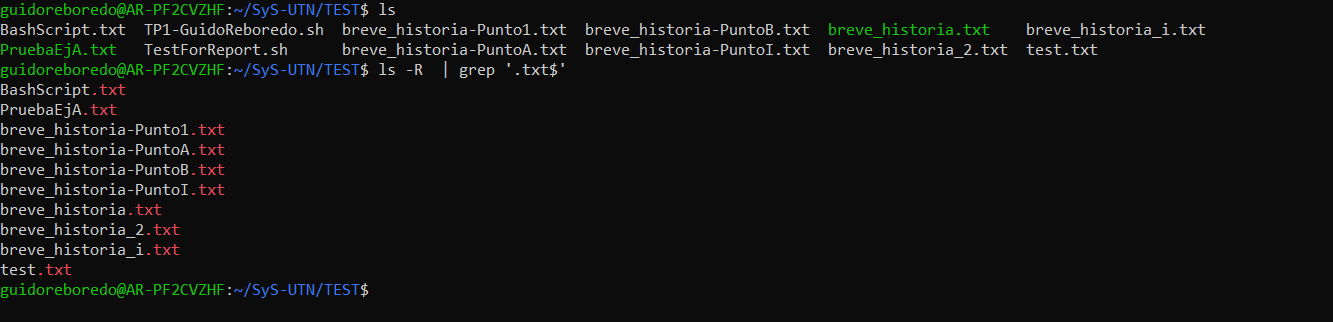
La salida se guarda en el archivo **breve\_historia-PuntoI.txt** (Revisar la carpeta Anexos-Pruebas).

1. Enumere todos los archivos de una carpeta que contengan extensión “.txt”. (tip: pipe con el comando ls).

Realizando un PIPE entre un grep y la función LS obtenemos la siguiente sentencia:

**$ ls -R | grep '.txt$'**

Cuyo resultado es el siguiente:



1. Investigue y explique, dando ejemplos cómo se utilizan los siguientes elementos en bash:

- Variables.

- Sentencias condicionales.

- Sentencias cíclicas.

- Subprogramas

Dé ejemplos de cada una.

**#Variables:**

Para utilizar variables en bash no es necesario introducir ningún caracter especial antes del nombre de la variable, pero si debe utilizarse el caracter “$” en caso de querer leer su contenido.

Ejemplo de utilización de variables:

variable ="Hello World"

echo $variable

Un Script sencillo podría ser

#!/bin/sh

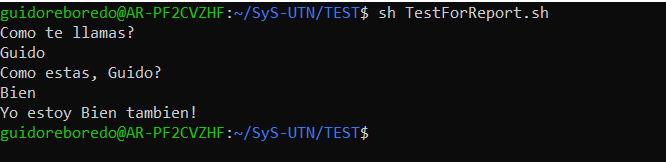
echo "Como te llamas?"

read name # lee lo ingresado en pantalla y lo guarda en la variable name

echo "Como estas, $name?" # “$name” esta leyendo el contenido de la variable name

read remark

echo "Yo estoy $remark tambien!"



**#Sentencias Condicionales:**

Comencemos por la sentencia **IF:**

Se debe comenzar nombrando la palabra reservada IF y terminar utilizando la palabra reservada FI. Luego de la palabra IF debe ir la [ Condición ] a ser evaluada encerrada por “[ ]” (Debe haber un espacio entre la condición y los corchetes de apertura y cierre). Luego de la condición se coloca Punto y Coma “;” y luego la palabra reservada “then” y las sentencias a ser ejecutadas. Ejemplo:

#!/bin/sh

Estudiante="Guido"

if [ $Estudiante=="Guido" ] ;

then

echo Estoy estudiando Bash.

fi



Añadiendo un else:

#!/bin/sh

Estudiante=27

if [ $Estudiante -eq 30 ] ;

then

echo Estoy estudiando Bash.

else

echo Estoy estudiando Comunicaciones.

fi



Y ahora podemos añadir “elif” para consultar por nuevas condiciones:

#!/bin/sh

echo ingresar dos numeros:

read num1

read num2

if [ $num1 -gt $num2 ]; then

echo el numero 2 es menor.

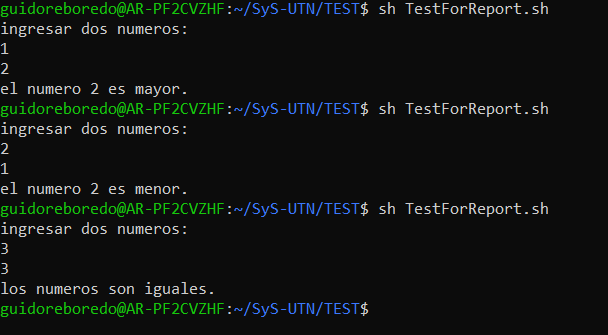
elif [ $num1 -lt $num2 ]; then

echo el numero 2 es mayor.

elif [ $num1 -eq $num2 ]; then

echo los numeros son iguales.

fi



Siguiendo con las sentencias condicionales podemos ver el **“CASE”:**

El case se puede entender como una simplificación del **“IF-ELIF”**. Para esto utilizamos la palabra reservada “case” seguido de la variable que será el primer operando (Recordando utilizar el signo $ para ver su valor) y luego la palabra reservada “in”. Seguido de “in” debe oprimirse la tecla enter y comenzaran a escribirse entre paréntesis los segundos operandos de la condición.

Debajo de estos irán las sentencias a ejecutarse en caso que se cumplan, terminados con “;;”. Para cerrar se coloca el operando default (\*) con sus sentencias y al final la palabra reservada “esac”.

Un ejemplo poco práctico pero demostrativo sería:

#!/bin/sh

echo ingresar un nombre:

read name

case $name in

"pedro")

echo el alumno se llama Pedro

;;

"juan")

echo el alumno se llama juan

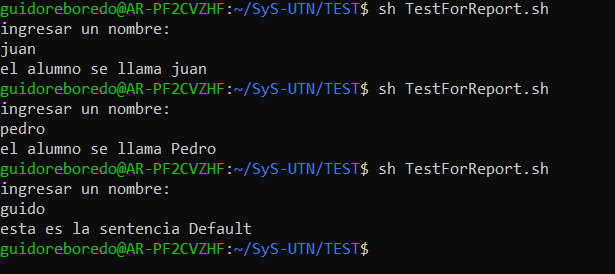
;;

\*)

echo esta es la sentencia Default

;;

esac



**#Secuencias Cíclicas**

El bucle **FOR**:

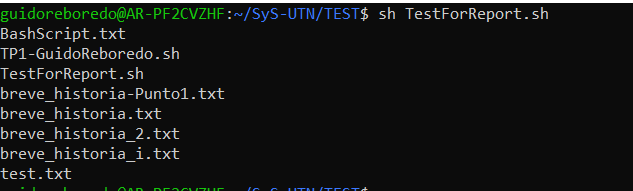
El bucle **for** permite iterar sobre una serie de elementos de una cadena, lista, range, etc.

en la segunda linea del codigo mostrado más abajo declaramos i como la variable que recibirá los diferentes valores contenidos en $(ls). La tercera linea puede contener todas las instancias que sean necesarias sobre la variable $i. Por último, en la cuarta línea se da por cerrado el ciclo for con done, después de done se cierra el ciclo for.

for i in $(ls); do

echo $i

done



El bucle **WHILE:**

El bucle for no es la única forma de hacer ciclos en scripts de bash. El ciclo **while** hace el mismo trabajo, pero verifica una condición antes de cada iteración.

El comando **while** tiene la siguiente estructura:

while [condition]; do

commands

done

Un ejemplo Sería:

#!/bin/sh

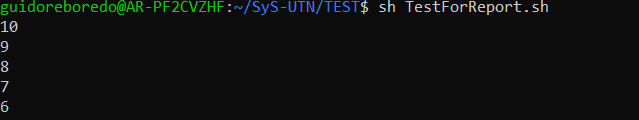
number=10

while [ $number -gt 5 ];do

echo $number

number=$(($number-1))

done



**#Subprogramas**

Las Funciones de bash son bloques de código que puedes reutilizar en cualquier parte de tu código. Siempre que necesites utilizar este bloque de código en tu script, solo escribe el nombre que le diste a la función.

Se puede definir una función de la siguiente forma:

functionName() {

#Sentencias de la funcion

}

Los paréntesis () son requeridos para definir la función.

El comando **return** retorna un entero desde la función.

#!/bin/bash

myfunc() {

read -p "Enter a value: " value

echo "adding value"

return $(($value + 10))

}

myfunc

echo "The new value is $?"

Tambien pueden pasarse parametros a una función

Puedes utilizar las variables de ambiente para procesar los parámetros pasados a la función. El nombre de la función es declarado como la variable $0, y los parámetros pasados son $1, $2, $3, etc.

Puedes obtener el número de parametros pasados a la función utilizando la variable ($#) variable.

Pasamos parámetros de la siguiente forma:

myfunc $val1 10 20

Un ejemplo del uso de funciones sería:

#!/bin/bash

sumar() {

if [ $# -gt 2 ]; then

# Si no se envian dos parametros

echo "Cantidad de parametros incorrecta"

else

# Caso contrario sumarlos

echo $(($1 + $2))

fi

}

echo -n "Sumando 10 y 15: "

value=$(sumar 10 15)

echo $value

echo -n "sumando tres numeros: "

value=$(sumar 10 15 20)

echo $value

