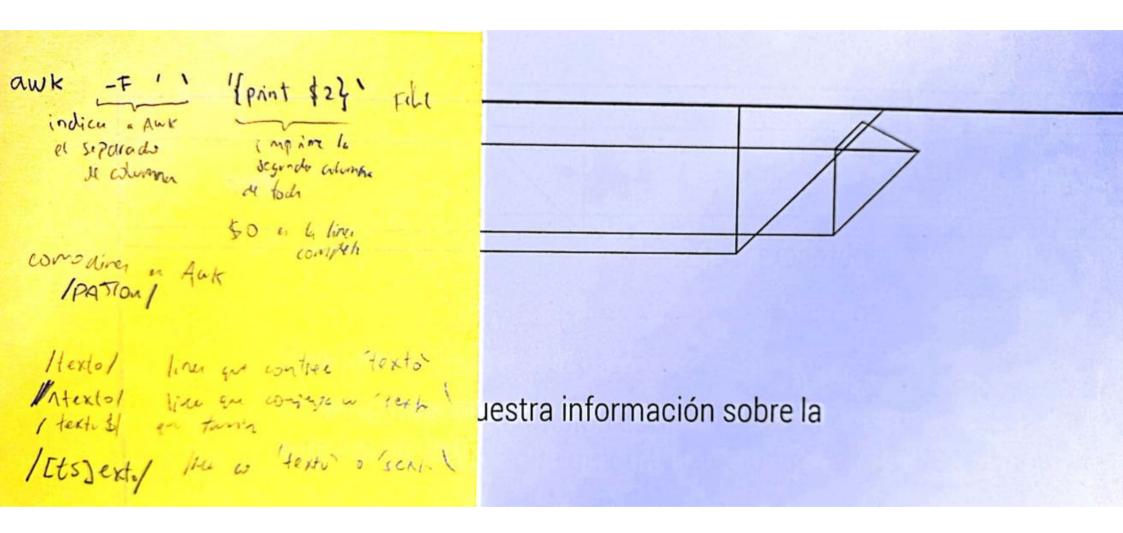
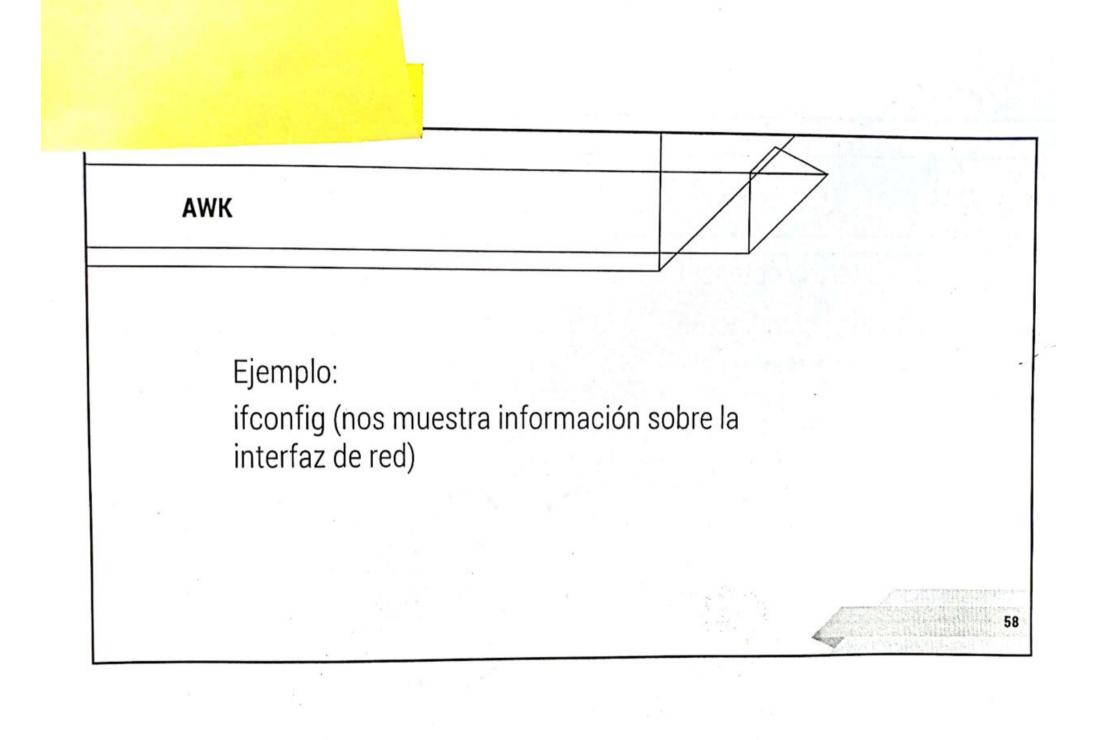
Amazornente usado para reemplazar & Sed 's/abc/fgh/gi' old>new global. de no ertor oqui; sed kemploga el primer patro \$ sed \s/ca-t]*/(R)/gi' old>new & representa el patron encontrado isi se deser que actue on linear o rangos específicos: Sed 13 st.... 1 A sed Start/, /stop/ s/#...
linea argue on the range can keyword start 7 stop Vango esos
de 11 hosts las ignora stra opción es !d
el hinal, pura que haga lo constrarso 4 Sed 1

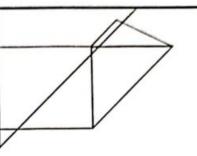
Awk -F' 'cordicion Sacriery Condition {acrier} ...

AWK es un lenguaje para procesar texto Supongamos que tenemos un texto con datos, con cierto formato

Duk - F' 'I net / Sprint \$25'







~\$ ifconfig

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500

inet 10.100.90.88 netmask 255.255.255.0 broadcast 10.100.90.255

inet6 fe80::9d91:497:85b5:fc46 prefixlen 64 scopeid 0x0<global>
ether 54:bf:64:49:0b:55 (Ethernet)

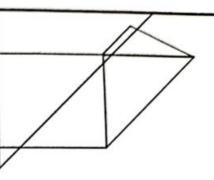
RX packets 0 bytes 0 (0.0 B)

RX errors 0 dropped 0 overruns 0 frame 0

TX packets 0 bytes 0 (0.0 B)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 1500
inet 127.0.0.1 netmask 255.0.0.0
inet6 ::1 prefixlen 128 scopeid 0x0<global>
loop (Local Loopback)
RX packets 0 bytes 0 (0.0 B)
RX errors 0 dropped 0 overruns 0 frame 0
TX packets 0 bytes 0 (0.0 B)
TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0



Nos interesa obtener las IPs:

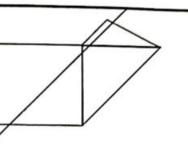
ifconfig | grep "inet "

inet 10.100.90.88 netmask 255.255.255.0 broadcast 10.100.90.255

inet 127.0.0.1 netmask 255.0.0.0

Sabemos que la ip es la segunda columna, al lado de "inet "





Nos interesa obtener las IPs:

ifconfig | grep "inet "

inet 10.100.90.88 netmask 255.255.255.0 broadcast 10.100.90.255
inet 127.0.0.1 netmask 255.0.0.0

Sabemos que la ip es la segunda columna, al lado de "inet "

Las columnas están separadas por espacios

awk -F ' ' '{print \$2}'
-F, le indica a AWK el separador de columnas
'{print \$2}', imprime la segunda columna
\$1..... \$n, variables que guardan la columna
correspondiente
\$0, es la línea completa

```
~$ ifconfig |grep 'inet ' | awk -F' ' '{print $2}' 10.100.90.88 127.0.0.1
```

```
¿Podemos evitar grep?

La forma general del programa en AWK es

'condición1 {acción1} condición2 {acción2} ...'

'{print $2}' es una acción para cualquier
condición
```

condiciones en AWK



var!= "texto" textos distintos

var == num numero en var es igual a num

var != num números distintos

var <, >, <=, >= num otras comparaciones

(var es, por ejemplo, \$1)

condiciones en AWK

o también podemos comparar texto con patrones encerrados en / /

/texto/ la línea contiene 'texto'

/^texto/ la línea comienza con 'texto'

/texto\$/ la línea termina con 'texto'

/[ts]exto/ le línea contiene 'texto' o 'sexto'

comodines? regex?

condiciones en AWK

/[ts]?exto/ -> coincide texto, sexto, exto
/[ts]*exto/ -> coincide con exto, texto, sexto, tsexto, ttexto,
ssexto, tttttsexto, sststtsssttsstststststststststststo
/[ts]+exto/ -> lo mismo que arriba, menos "exto"
?, *, y + se pueden usar después de cualquier caracter
comodines? regex?

ojo

El lenguaje de los patrones encerrados en // se llama "regexp" o "regex" (expresiones regulares) también se pueden usar con grep invocando grep como 'egrep' o usando grep -E

Volviendo a AWK:

```
~$ ifconfig |grep 'inet ' | awk -F' '
 '{print $2}'
10.100.90.88
127.0.0.1
~$ ifconfig | awk -F' ' '/inet / {print $2}'
10.100.90.88
127.0.0.1
                            aulyn caracter ( la xl)
~$ ifconfig | awk -F' ' '/^[a-zA-Z0-9]+:/ {iface=$1}
/inet / {print iface "\t"$2}'
                                                  agai & guardo
eth0:
      10.100.90.88
                                                   la primera columna
                                                   everte cas etho
        127.0.0.1
lo:
                                                      4 Io
```

si la biraccontre inet se imprine itale 141\$2
el primer y la segunda
columna (ip)
en corma de tento

Variables especiales en AWK:

NR: número de línea

NF: cantidad de "campos" (columnas)

length: número de caracteres en la línea

\$0: la línea completa

Bloques especiales:

BEGIN { acción } (una vez, al comienzo)

END { acción } (una vez, al final)

¿para qué podrían servir?

6. Estructuras Condicionales

Las estructuras condicionales le permiten decidir si se realiza una acción o no; esta decisión se toma evaluando una expresión.

6.1 Pura teoría

Los condicionales tienen muchas formas. La más básica es: **if** *expresión* **then** *sentencia* donde 'sentencia' sólo se ejecuta si 'expresión' se evalúa como verdadera. '2<1' es una expresión que se evalúa falsa, mientras que '2>1' se evalúa verdadera.

Los condicionales tienen otras formas, como: **if** *expresión* **then** *sentencia1* **else** *sentencia2*. Aquí 'sentencia1' se ejecuta si 'expresión' es verdadera. De otra manera se ejecuta 'sentencia2'.

Otra forma más de condicional es: if expresión1 then sentencia1 else if expresión2 then sentencia2 else sentencia3. En esta forma sólo se añade "ELSE IF 'expresión2' THEN 'sentencia2'", que hace que sentencia2 se ejecute si expresión2 se evalúa verdadera. El resto es como puede imaginarse (véanse las formas anteriores).

Unas palabras sobre la sintaxis:

La base de las construcciones 'if' es ésta:

if [expresión];

then

código si 'expresión' es verdadera.

fi

6.2 Ejemplo: Ejemplo básico de condicional if .. then

```
#!/bin/bash
if [ "petete" = "petete" ]; then
    echo expresión evaluada como verdadera
fi
```

El código que se ejecutará si la expresión entre corchetes es verdadera se encuentra entre la palabra 'then' y la palabra 'fi', que indica el final del código ejecutado condicionalmente.

6.3 Ejemplo: Ejemplo básico de condicional if .. then ... else

```
#!/bin/bash if [ "petete" = "petete" ]; then
    echo expresión evaluada como verdadera
else
    echo expresión evaluada como falsa
fi
```

6.4 Ejemplo: Condicionales con variables

```
#!/bin/bash
T1="petete"
T2="peteto"
if [ "$T1" = "$T2" ]; then
    echo expresión evaluada como verdadera
else
    echo expresión evaluada como falsa
fi
```

6.5 Ejemplo: comprobando si existe un fichero

un agradecimiento más a mike

```
#!/bin/bash
FILE=~/.basrc
if [ -f $FILE ]; then
    echo el fichero $FILE existe
else
    echo fichero no encontrado
fi
if [ 'test -f $FILE']
```

7. Los bucles for, while y until

En esta sección se encontrará con los bucles for, while y until.

El bucle **for** es distinto a los de otros lenguajes de programación. Básicamente, le permite iterar sobre una serie de `palabras' contenidas dentro de una cadena.

El bucle **while** ejecuta un trozo de códico si la expresión de control es verdadera, y sólo se para cuando es falsa (o se encuentra una interrupción explícita dentro del código en ejecución).

El bucle **until** es casi idéntico al bucle loop, excepto en que el código se ejecuta mientras la expresión de control se evalúe como falsa.

Si sospecha que while y until son demasiado parecidos, está en lo cierto.

7.1 Por ejemplo

```
#!/bin/bash
for i in $( ls ); do
echo item: $i
done
```

En la segunda línea declaramos i como la variable que recibirá los diferentes valores contenidos en \$(ls).

La tercera línea podría ser más larga o podría haber más líneas antes del done (4).

`done' (4) indica que el código que ha utilizado el valor de \$i ha acabado e \$i puede tomar el nuevo valor.

Este script no tiene mucho sentido, pero una manera más útil de usar el bucle for sería hacer que concordasen sólo ciertos ficheros en el ejemplo anterior.

7.2 for tipo-C

Fiesh sugirió añadir esta forma de bucle. Es un bucle for más parecido al for de C/perl...

```
#!/bin/bash
for i in `seq 1 10`;
do
echo $i
done
```

7.3 Ejemplo de while

```
#!/bin/bash
CONTADOR=0
while [ $CONTADOR -lt 10 ]; do
```

echo El contador es \$CONTADOR let CONTADOR=CONTADOR+1 done

Este script 'emula' la conocida (C, Pascal, perl, etc) estructura `for'.

7.4 Ejemplo de until

#!/bin/bash
CONTADOR=20
until [\$CONTADOR -lt 10]; do
echo CONTADOR \$CONTADOR
let CONTADOR-=1
done

10. Miscelánea

10.1 Leyendo información del usuario

En muchas ocasiones, puede querer solicitar al usuario alguna información, y existen varias maneras para hacer esto. Ésta es una de ellas:

#!/bin/bash echo Por favor, introduzca su nombre read NOMBRE echo "¡Hola \$NOMBRE!"

Como variante, se pueden obtener múltiples valores con read. Este ejemplo debería clarificarlo.

#!/bin/bash echo Por favor, introduzca su nombre y primer apellido read NO AP echo "¡Hola \$AP, \$NO!"

10.2 Evaluación aritmética

Pruebe esto en la línea de comandos (o en una shell): echo 1 + 1

ċ

Si esperaba ver '2', quedará desilusionado. ¿Qué hacer si quiere que BASH evalúe unos números? La solución es ésta: echo \$((1+1)) Esto producirá una salida más 'lógica'. Esto se hace para evaluar una expresión aritmética. También puede hacerlo de esta manera: echo \$[1+1] Si necesita usar fracciones, u otras matemáticas, puede utilizar bc para evaluar expresiones aritméticas. Si ejecuta "echo \$[3/4]" en la línea de comandos, devolverá 0, porque bash sólo utiliza enteros en sus respuestas. Si ejecuta "echo 3/4|bc -l", devolverá 0.75. 11. Tablas 11.1 Operadores de comparación de cadenas s1 = s2s1 coincide con s2 s1! = s2s1 no coincide con s2 s1 < s2s1 es alfabéticamente anterior a s2, con el locale actual s1 > s2s1 es alfabéticamente posterior a s2, con el locale actual -n s1 s1 no es nulo (contiene uno o más caracteres)

s1 es nulo

-z s1

11.2 Ejemplo de comparación de cadenas

Comparando dos cadenas

11.3 Operadores aritméticos

```
+ (adición)
- (sustracción)
* (producto)
/ (división)
% (módulo)
```

11.4 Operadores relacionales aritméticos

```
-lt (<) less than
-gt (>) Greater than
less legand
-le (<=) Greater legand
-ge (>=) equal
-eq (==)
-ne (!=)
```

Los programadores de C tan sólo tienen que corresponder el operador con su paréntesis.