echo "poder en la terminal" | awk ’{print "Mazinger AWK,"$0}’

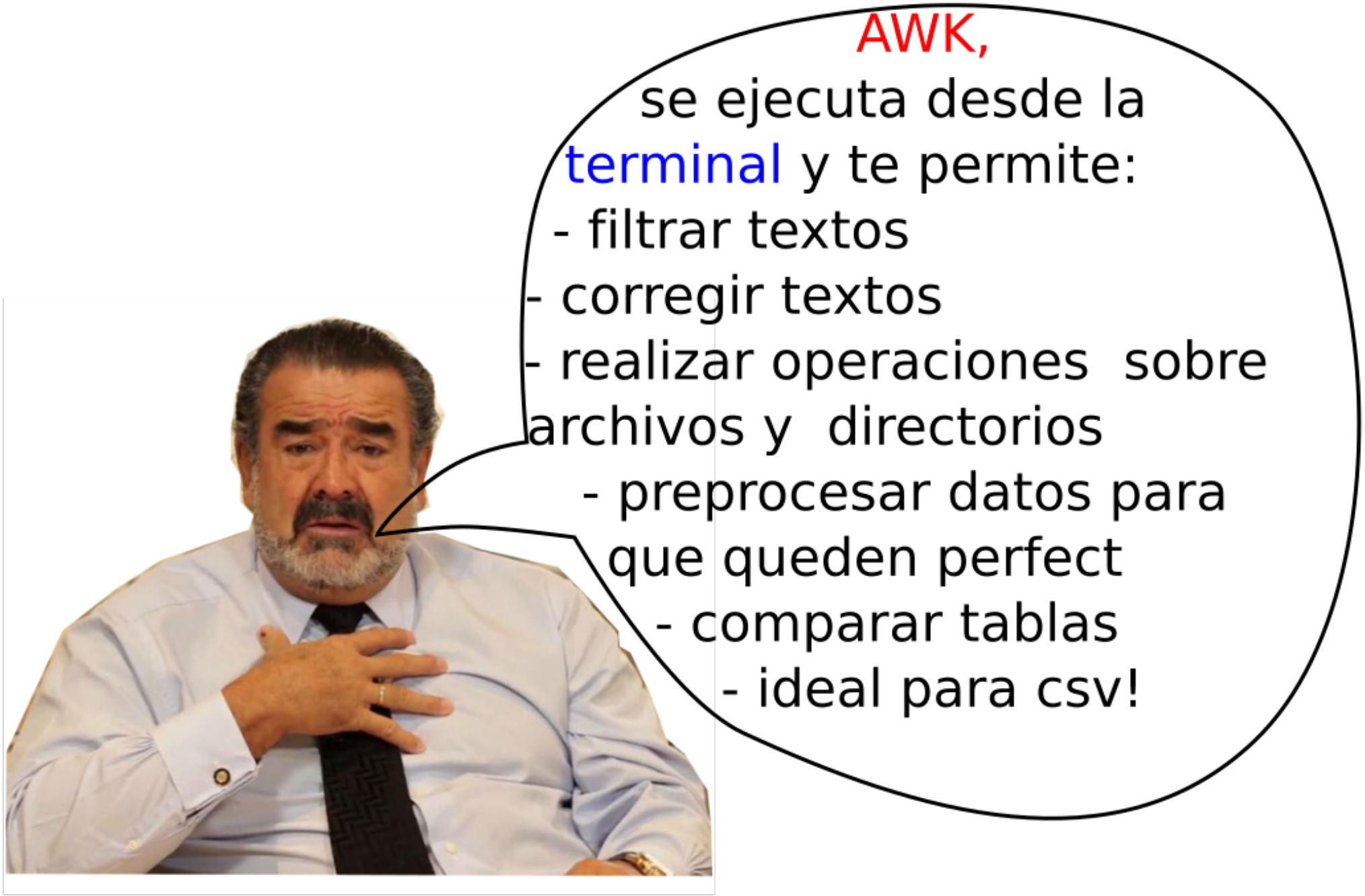
David Pineda Osorio

30 marzo 2019

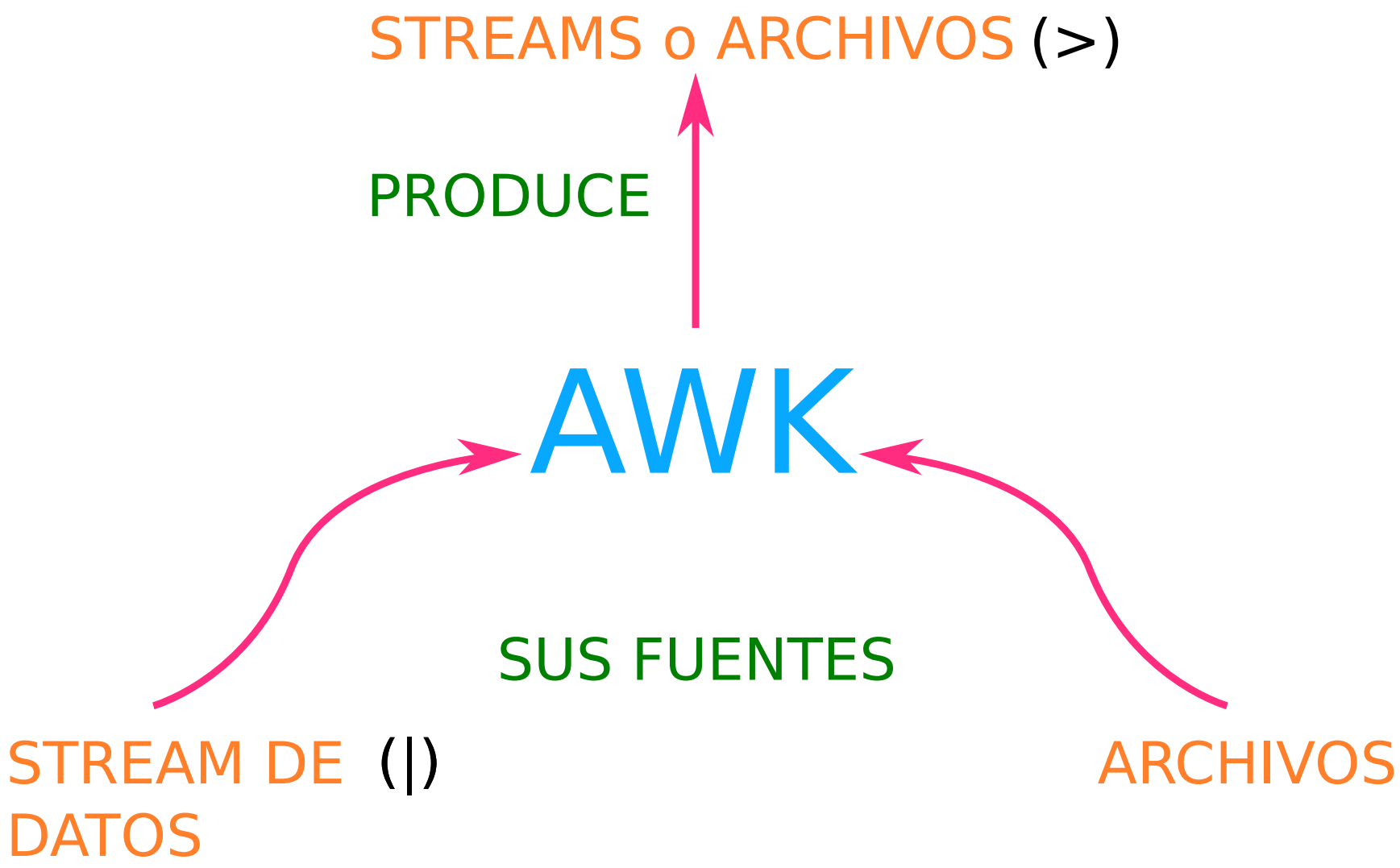
Recoleta, Santiago de Chile

jornadas.cuatrolibertades.org

AWK un lenguaje sencillo, pero poderoso

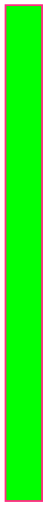


¿Qué hace AWK?

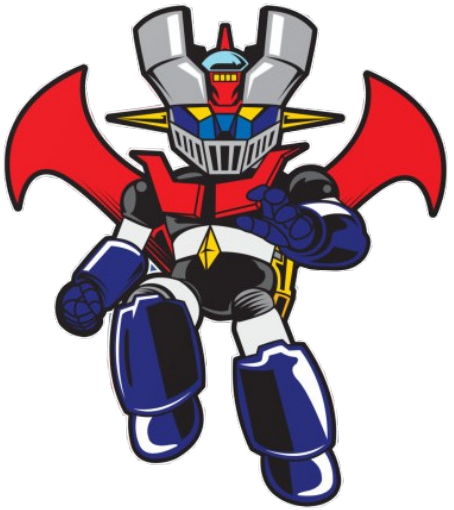

STREAMS o ARCHIVOS (>)
PRODUCE
AWK
SUS FUENTES
STREAM DE DATOS (|)
ARCHIVOS

Ejercicio 1: Leer desde un stream y reconocer columnas

cat archivo



(pleca)



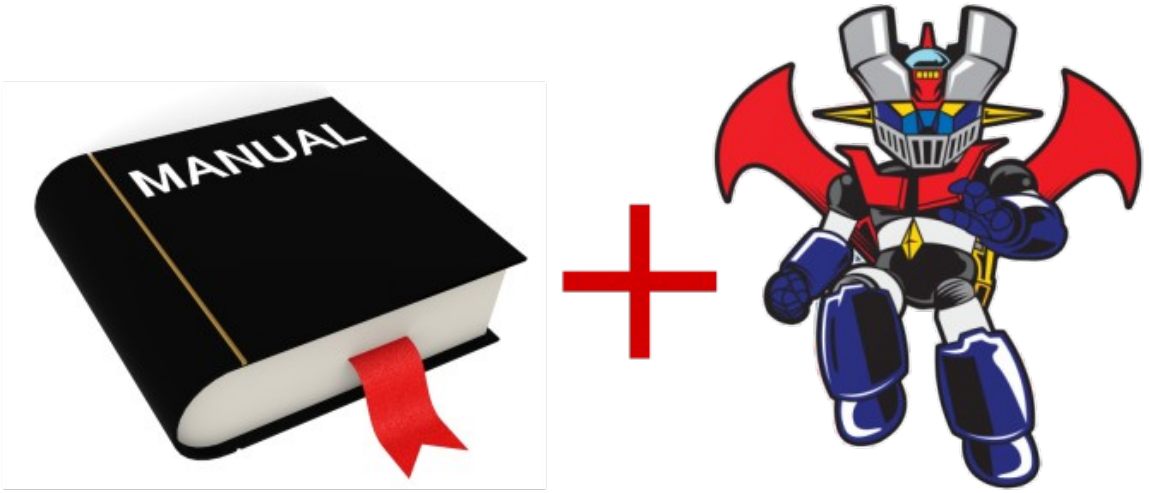
```
cat exp_vino.csv | awk -F',' '{print $0}'
```

¿Cómo cuenta columnas awk?

- La entrada, línea a línea pasa por la tubería a awk
- La línea completa es $0
- La primera columna es $1
- La última columna es $NF
- ¿Cómo se cuantos campos tiene una línea? → NF
- ¿Cómo defino el separador? → opción -F'separador'

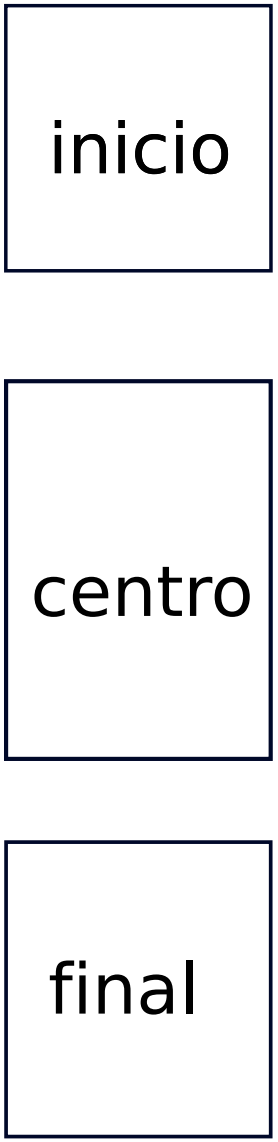
Descubriendo las variables de sistema

- Para conocerlas todas ver el manual de awk



- NF : número de campos en la fila actual
- NR: número de fila actual
- FS: separador de campo
- OFS: separador de campo en salida
- ENVIRON: arreglo con las variables de ambiente de sistema

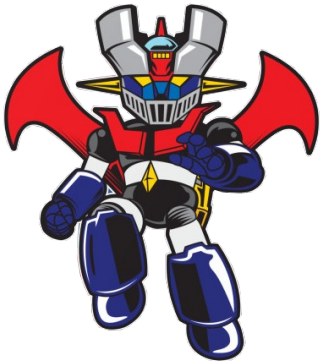
Estructura de un programa AWK




```
BEGIN{
<inicializar variables>
<acciones al inicio>
}
{
<procesar datos>,
<cargar arrays>,
<mprimir información>
}
END{
<procesar listas o arrays>
<finalizar acciones>
<imprimir cierre>
}
```

Encontrar Coincidencias

(.*)
expresiones
regulares



Usando la función
match

- 0 o False
  (si no hay coincidencia)
- 1 o True
  (si hay coincidencia)

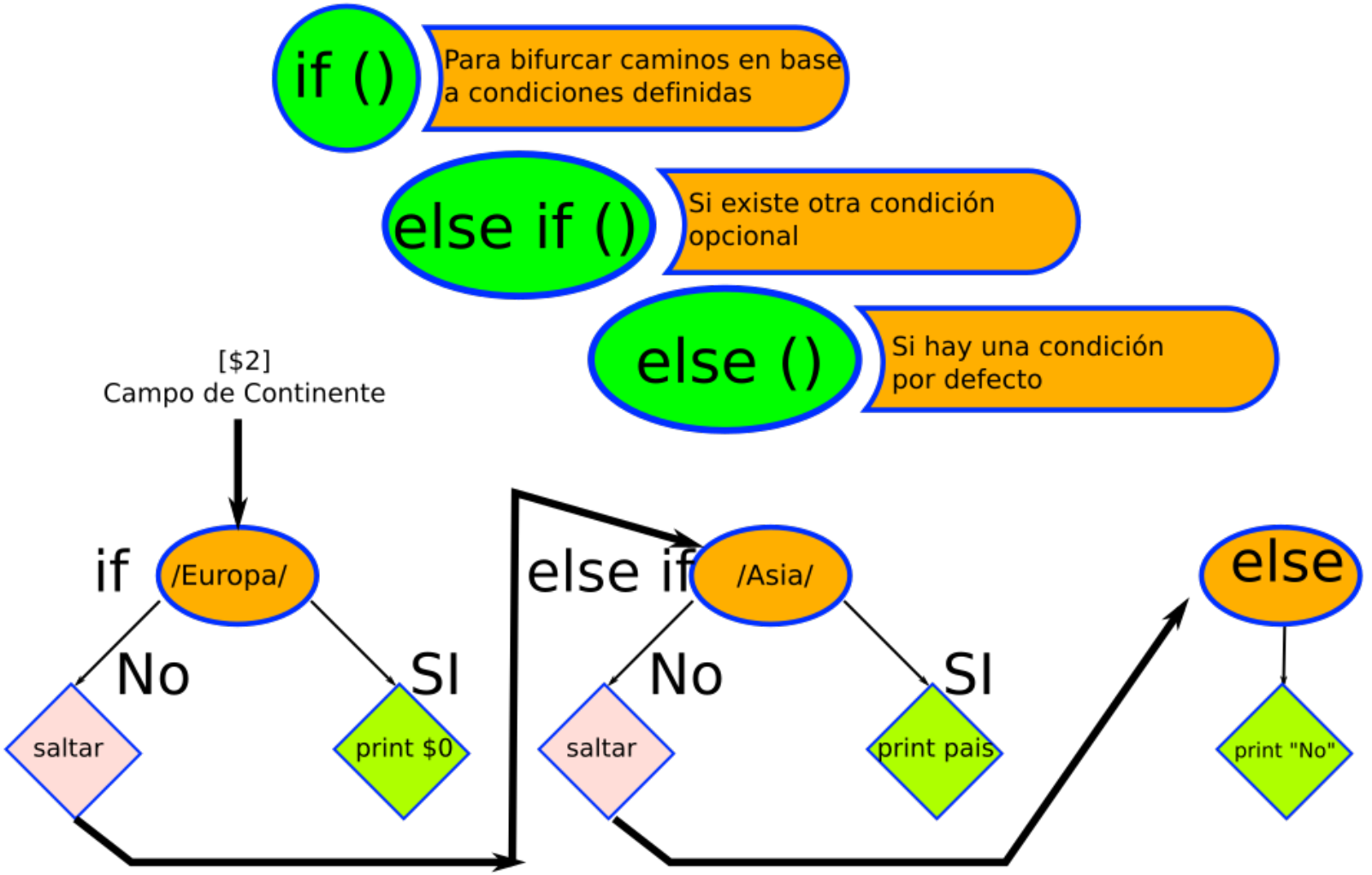
```
cat archivo | awk -F',' '(match($2,/Europa/)){print $0}'
```

Usando la expresión
comparadora A~/regex/

- 0 o False
  (si no hay coincidencia)
- 1 o True
  (si hay coincidencia)

```
cat archivo | awk -F',' '($2 ~/Europa/){print $0}'
```

Si, las clásicas estructuras de control




```
cat exp_vino.csv | awk -F',' '{if($2 ~ /
Europa/){print $0}else if($2 ~ /Asia/)
{pais=$4;print pais}else{print "no"}}'
```

¿Qué pasa si el comando se hace muy largo?
---> crear un script awk!

- Se crea un archivo con extensión “*.awk”
- Para ejecutarlo, usar el comando con la opción -f (minúscula) “awk -f script.awk”
- Para definir separador y otras variables de sistema, usar la ejecución al principio BEGIN{}
- Para terminar un programa y realizar algo al final, programar en la ejecución al final END{}

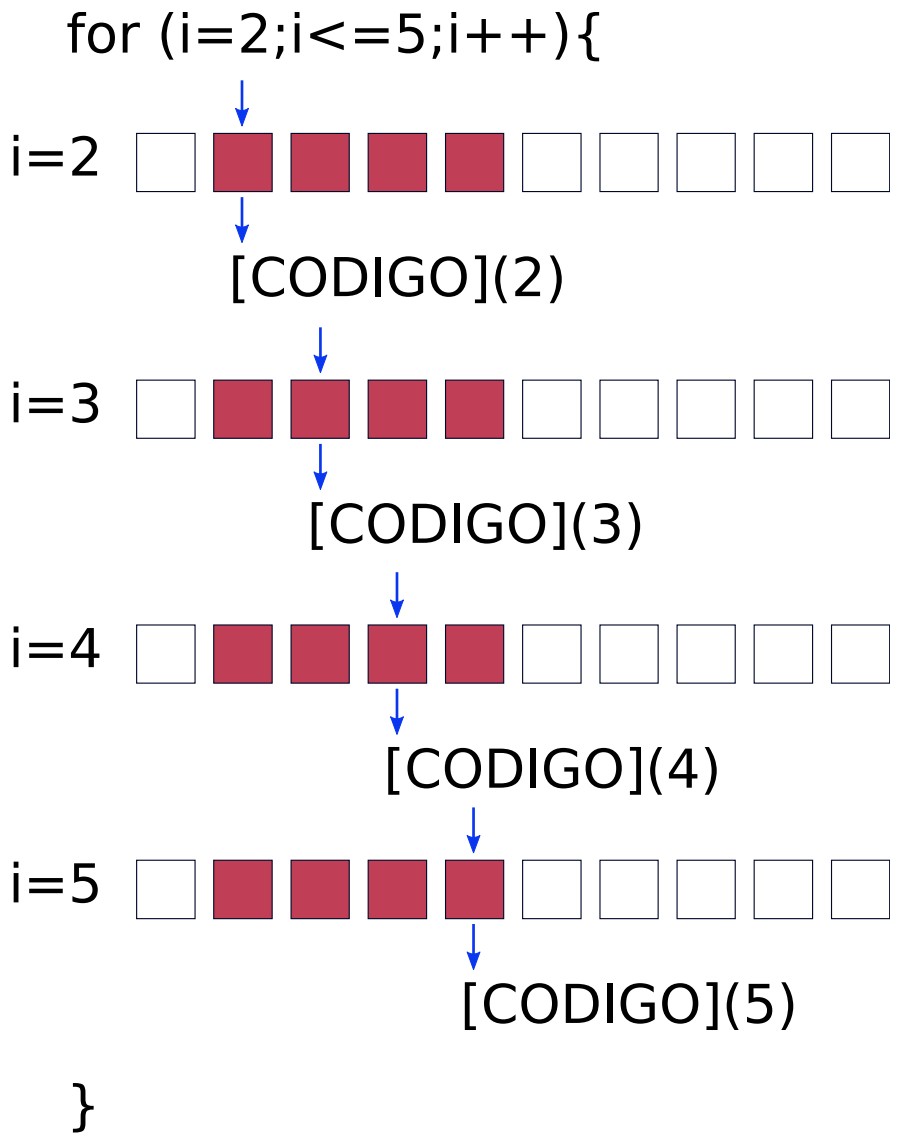
(un archivo con extensión *.awk)

```
#! /usr/bin/awk -f
BEGIN{FS=","}
{
if($2 ~ /Europa/){
        print $0
}
else if($2 ~ /Asia/){
pais=$4;
print pais}
else{
print "no"
}
}
```

(ejecutar en la terminal)

cat exp_vino.csv| awk -f ec_if.awk

For, iterar sobre un conjunto de elementos



El uso de <for (inicio; condicion; incremento)>

Si necesitamos imprimir solo las columnas 2 a 5, también se puede usar for.

```
cat exp_vino.csv| awk -F',' '{
for(i=2;i<=5;i++){
printf $i;
if (i<5){printf ";"}
else{printf "\n"}};}'
```

Sabemos que la información es siempre **Vino (Wine)** y no nos interesa saber los años. ¿Cómo hacer para mostrar toda la otra información y no la que no nos interesa?

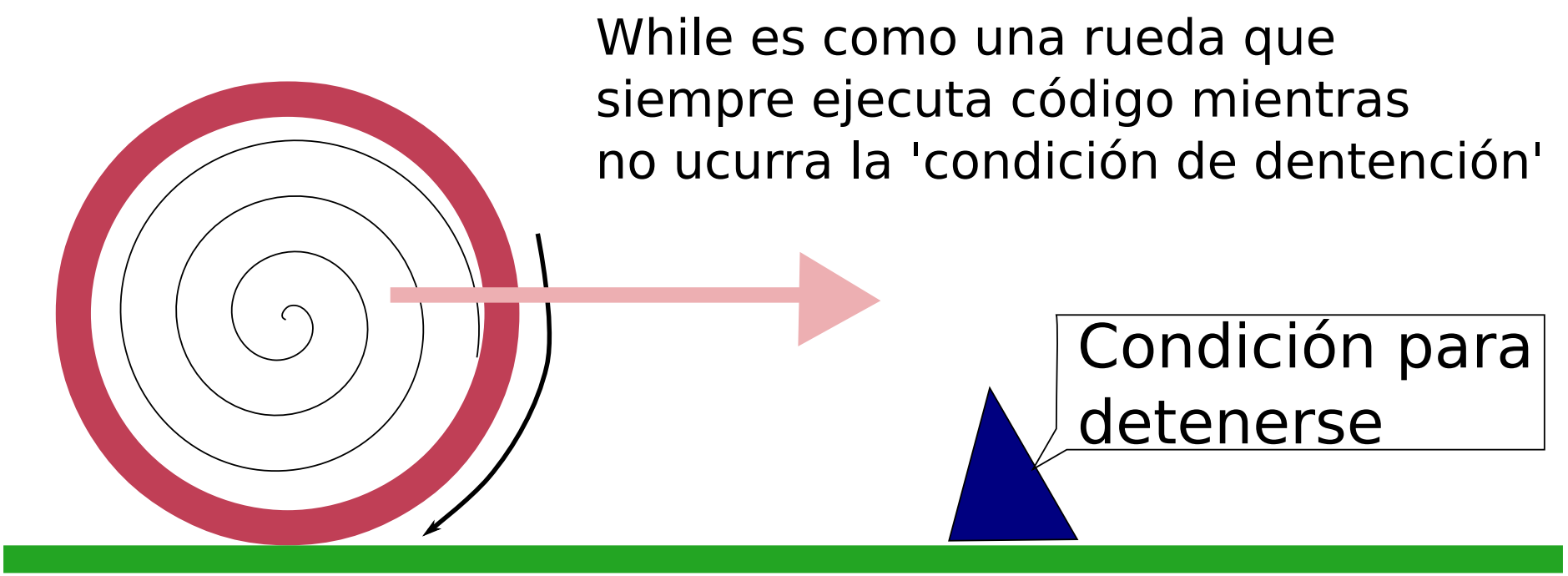
- Inspeccionamos que columnas corresponden a **Wine** y los **años**
- Hacemos una lectura columna a columna, omitiendo cuando se cumpla la condicion.

```
BEGIN{FS=","}
{
for (i=1;i<=NF;i++){
        if ($i !~/Wine/){
                printf $i
                if(i<NF){
                        printf ";"
                        }
        }
        if (i==NF){
                printf "\n"
        }

        }
}
END{print "Info solicitada"}
```

El uso de <while (condicion)>

- El mismo problema anterior se puede resolver utilizando la estructura de control **while**



While en Awk

- Por ejemplo, queremos imprimir todos los campos entre el campo 2 al 5, lo mismo que con for, pero con while

```
BEGIN{FS=","}
{
i=2;
while(i<5){
            printf $i";";
            i+=1};
printf $i"\n"
}
```

¡Ahora sí: a probar lo que sabemos!

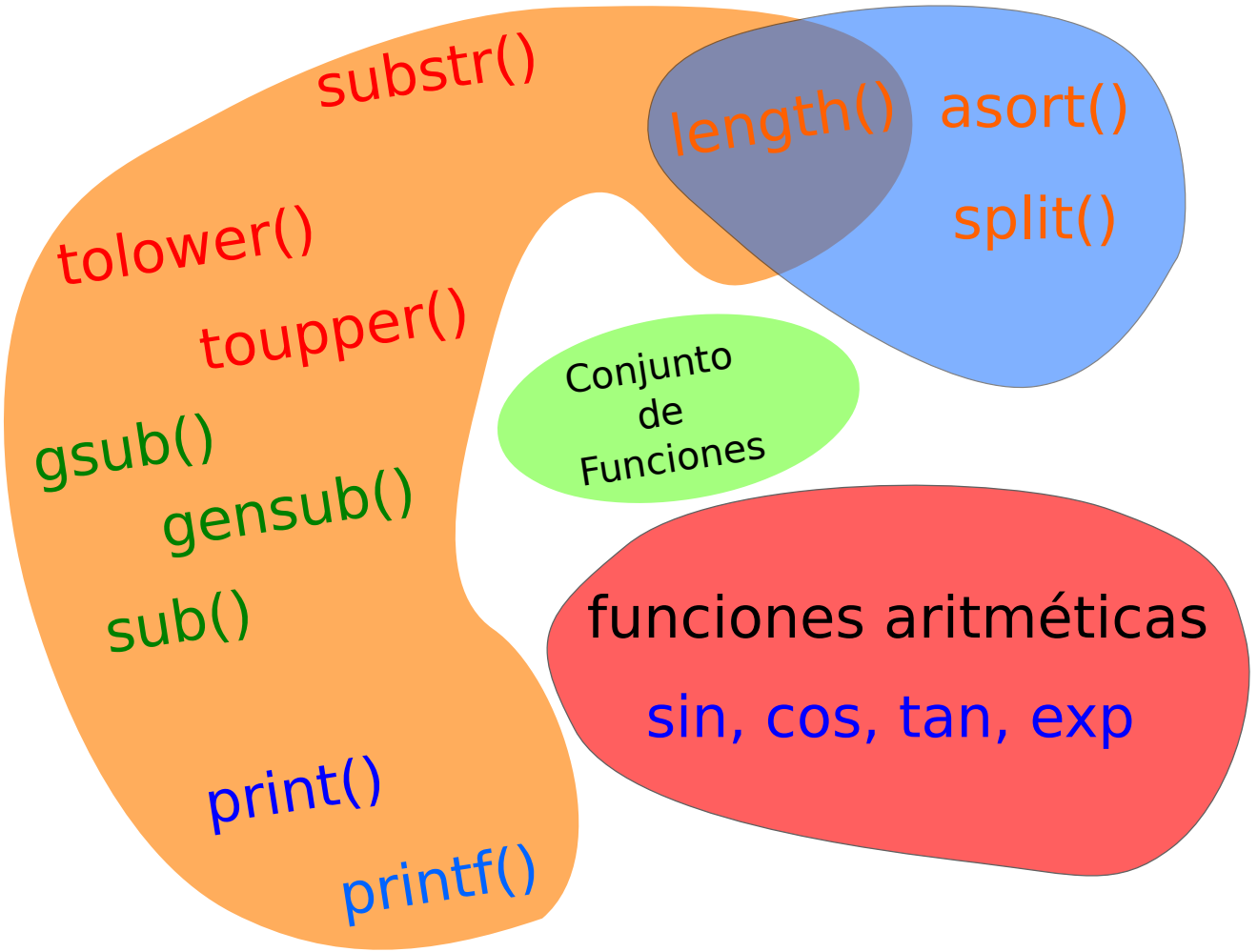
Realizar los siguientes ejercicios y crear el archivo con el resultado.

- Sin definir campo separador (-F',') imprime los campos {0,1,2,10,NF}
- ¿Cuantos datos tiene el archivo?
- ¿Es posible mostrar solo los datos desde la línea 100 a 150?

Definiendo como separador (-F',') encontrar:

- Muestra solo las exportaciones a Europa
- ¿Cuantos países recibieron vino chileno el 2014?
- ¿Cuantos países de Africa recibieron vino chileno el 2016?
- Mostrar las exportaciones a Australia y China

Nueva Iteración: Volver al Principio



Jugando con textos (strings)

Vamos a imprimir un texto y lo encadenamos a **awk**

Por defecto el separador es ‘ ‘ (espacio)

Podemos jugar a cambiar el separador a ‘,’ ‘;’ ‘|’

Aplicamos las funciones de **awk** sobre los campos

Texto
en
una
línea

```
echo "Mazinger Awk, un taller increíble. Me encanta|¡Es fabuloso!"
```

Algunos ejemplos

- Substituir un patrón (**regex)** por un carácter o texto, todos los espacios → =:

```
resultado = gensub(/regex/, reemplazo, "g", objetivo)
```

```
echo "Mazinger Awk, un taller increíble. Me encanta|¡Es fabuloso!"|
awk -F'|' '{a=gensub(/ /,"=","G",$0); print a}'
```

- Pasar a mayúsculas el último campo

```
resultado = toupper(objetivo)
```

```
echo "Mazinger Awk, un taller increíble. Me encanta|¡Es fabuloso!"|
awk -F'|' '{resultado=toupper($NF); print resultado}'
```

Ejercicio: Imprimir toda la línea, pasando a mayúsculas último campo.
Usa la estructura de control **for**

Crear una lista separando un string por un carácter:

- Puede ocurrir que, en algún momento, te encuentres con la necesidad de separar los elementos de un campo, cada elemento puede ir definido en relación a los otros con un carácter

  La separación de un **texto o string** produce un **arreglo o lista,** indexado desde la posición 1 a **length(arreglo)**

  Obtener el último elemento de cada campo, separandolos con '|' y '/' respectivamente

```
texto "estos|son|varios|campos /home/david/carpeta/archivo.txt"
```

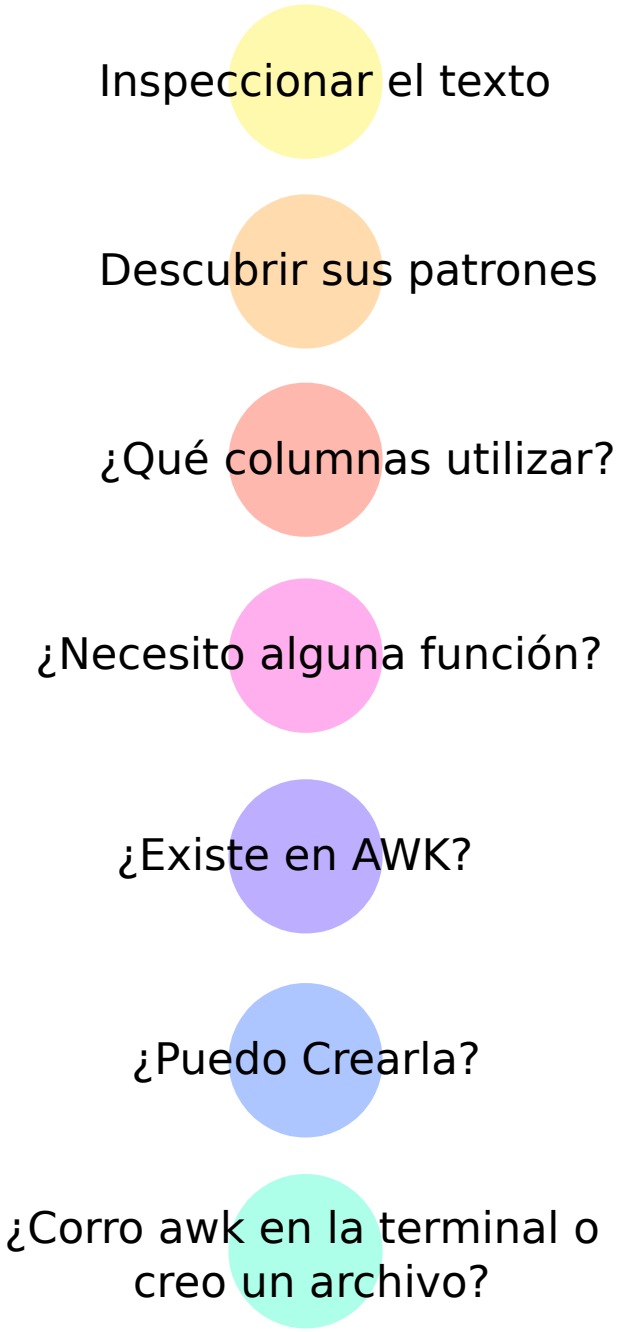
```
echo $texto|awk '{
split($1, primer, "|");
split($2, ruta, "/");
l1=length(primer);
l2=length(ruta);
print primer[l1],ruta[l2]}'
```

Un viejo clásico: Temperaturas °Celsius → °Farenheit

- Vamos a crear una función que transforma valores Celsius a Farenheit
- Trabajamos con un archivo **“./ejercicios/ej2/temperaturas.csv”**
- Usaremos funciones matemáticas
- Crearemos una función en **awk** y la usaremos
- Generaremos un archivo con las temperaturas en °Celcius
- Si una ciudad tiene una temperatura mayor a 68 farenheit, poner su nombre en mayúsculas
- Creamos un archivo con nuestro script

{Técnicas de Programación en AWK}

{Pasos elementales}



Usando la técnica descrita sobre el archivo

- Los paises están en campo 1
- Las ciudades en campo 2
- Las temperaturas Celsius en campo 3
- El separador es ‘,’
- Se escribirá un archivo ya que el código es largo y para mantener claridad

Estudiamos el archivo con el script “ciudades_c_f.awk”

Ejecutar en la terminal:

awk -f ciudades_c_f.awk temperaturas.csv

El script tiene dos partes importantes

- Declaración de funciones

```
#
#
# SE DEFINEN LAS FUNCIONES
#
#
function celsius2farenheit(celsius)
{
    farenheit = celsius*(9/5)+32;
    return farenheit
}

function alerta(ciudad, farenheit)
{
    city="";
    if (farenheit>=68)
    {
        city = toupper(ciudad)}
    else{
        city = ciudad}
    return city
}
```

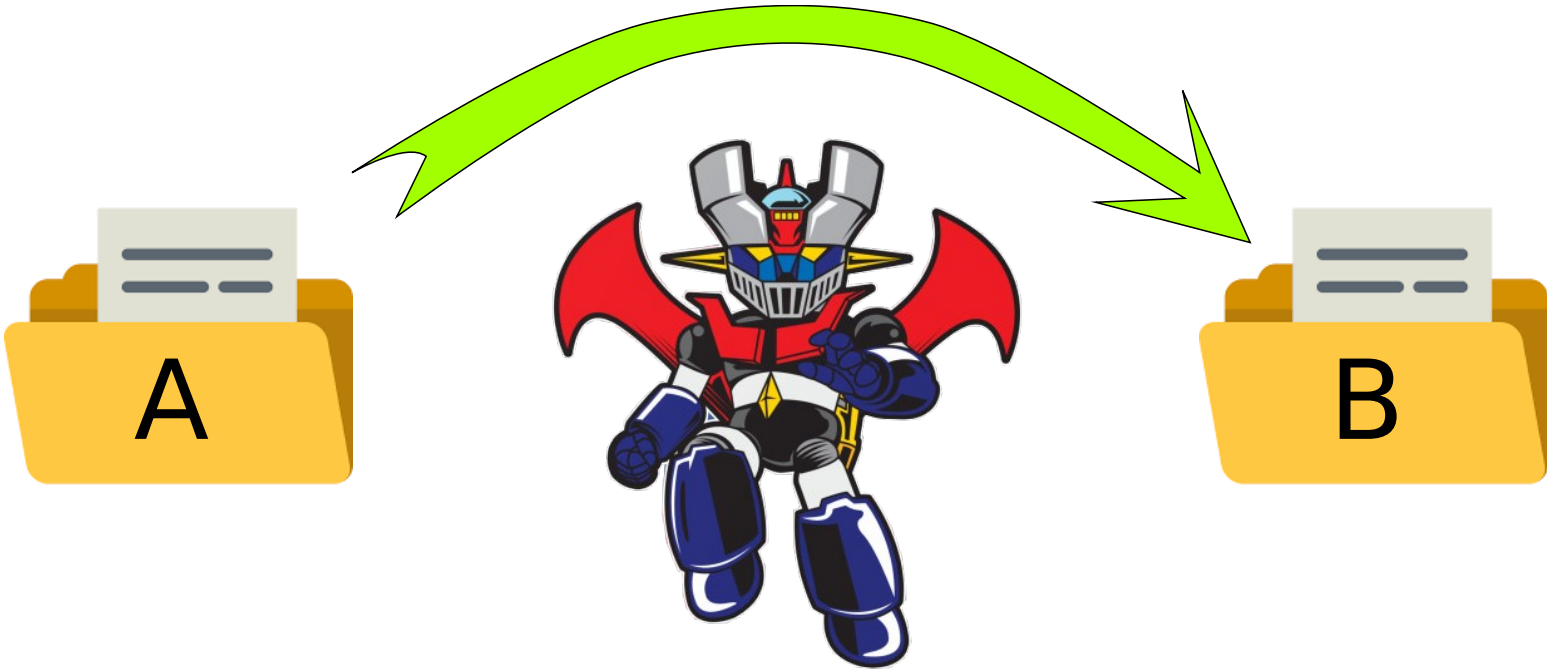
Se lee el archivo y procesan los datos

```
#
# SE PROGRAMA LA LECTURA DEL ARCHIVO
#
#
BEGIN{
vPAIS=1;
vCIUDAD=2;
vTEMP=3;
FS=",";
OFS=";";
print "país","ciudad","temperatura °[F]"
}
(NR>1){
FAR=celsius2farenheit($vTEMP);
CIUDAD=alerta($vCIUDAD,FAR);
print $vPAIS,CIUDAD,FAR
}
```

Ejecutar Comandos de Bash desde AWK: Dos caminos impíos

- Una de las grandes virtudes de **awk** es que es posible construir comandos de manera sencilla en base a la información que extraemos de un stream creado en base a algún comando, luego ejecutarlos y realizar tareas de manera programática y fabulosa.
- Hay dos formar de ejecutar un comando construir con **awk**
- Una es la función **system(comando)**
- Otra es capturar una línea y ejecutarla: **comando|getline d; close(d)**
- Se cierra la ejecución para no sobrecargar la memoria
- Si necesitas capturar el resultado usa → **getline**
- Si solo necesitas ejecutar un comando → **system**

BASH+AWK: Un ejercicio ejemplar



{un ejercicio ejemplar}

Copiar todos los archivos de texto de la **carpeta A,** que contengan la palabra **padre,** A la **carpeta B.** Comprimir la carpeta B.

Desarrollo del Programa A→B (1)

Paso 0: Buscar los textos que contengan "padre"

```
find -iname "*.txt"|xargs grep "padre"
```

Paso 1: Seleccionar solo aquellos de la carpeta A

```
awk -F':' 'BEGIN{OFS="|"}{
split($1,path,"/");
if(path[2]=="A"){print $1,$2}}'
```

(*): ¿Vale hacer un script?

Desarrollo del Programa A→B (2)

Paso 2: crear el comando y mostrarlo

```
find -iname "*.txt"|xargs grep "padre"|awk -F':' 'BEGIN{
 OFS="|"
 }{
  split($1,path,"/");
  if(path[2]=="A"){
   comando="cp "$1" ./B"
   print comando
   }
   }'
```

Paso 3: Ejecutar script A_B.bash y probarlo

```
bash A_B.sh |bash
ls B/*
rm B/*
```

Desarrollo del Programa A→B (3)

Paso 4: Integrar con system o getline en awk

```
comando="cp "$1" ./B"
comando|getline d;
close(d);
```

Paso 5: Finalizar comprimiendo carpeta B

```
END{
 print "Comprimiendo B";
 comprimir="tar -czvf textos_con_padre.tar.gz ./B";
 system(comprimir)}
```

¡Revisemos el script y lo ejecutamos!

Cosas interesantes que se pueden hacer

- Usar varios archivos a la vez y comparar información entre ellos
- Gestionar archivos y directorios
- Hacer filtros de información
- Crear listas para bash
- Combinar con las herramientas POSIX

Referencias, textos, ayudas

- Descargar Libros Técnicos en pdf: http://gen.lib.rus.ec/
- Sed and awk
- Awk, a pattern matching language
- The awk manual
- Effective Awk Programming
- Curso de Programación, nivel1 sesion4: https://gitlab.com/pineiden/curso_programacion
- Presentación Mazinger Awk

  https://gitlab.com/pineiden/mazinger-awk
- Hecho integramente con Software Libre {bash, awk, Impress, Inkscape, Gimp}
- Las imágenes y figuras sacadas de internet, bla!