
> El único martillo que vas a necesitar
>   (si todos tus problemas son tablas)

Basado en [1]

Temario

- ¿Qué es `AWK`?
- ¿Para que aprenderlo?
- El *main loop* de `AWK`:
  - Leyendo registro, separando campos.
  - Reglas: condiciones y acciones.
- Funciones básicas.
- Definición de funciones.
- Como escribir programas:
  - *one-liners*.
  - *scripts*.
- **¡Práctica!**

¿Qué es AWK?

AWK es un lenguaje orientado al procesamiento de registros de datos en archivos de texto. Fue creado en 1977 por *Alfred Aho*, *Peter Weinberger*, y *Brian Kernighan* en los *Laboratorios Bell*.

Actualmente forma parte obligatoria de todo sistema que quiera llamarse a si mismo UNIX, y parte del *Linux Standard Base*.

¿Para que aprenderlo?

- Automáticamente separa el archivo en registros y campos, sin dejar de ser flexible en la definiciones de estos.
- Orientado procesamiento de flujos de datos:
  - Se integra fácilmente a otras utilidades que siguen la filosofía UNIX.
  - Aprender distintos enfoques nos ayuda a ser mejores programadores.
- Siempre esta disponible.
- Sus distintos métodos de invocación, y sus comportamientos *default* lo hacen ideal para la creación de *one-liners*.

El *main loop* de `AWK`

Dada una entrada formada por archivos o *stdin*, y un programa que contiene un conjunto de reglas, se ejecutara el siguiente bucle hasta terminar:

1. Se obtiene un registro.
2. Se obtienen los campos del registro.
3. Por cada regla: si se cumple con la condición, se ejecuta el cuerpo de la regla.
4. Si no se llego al final de la entrada, volver al paso 1.

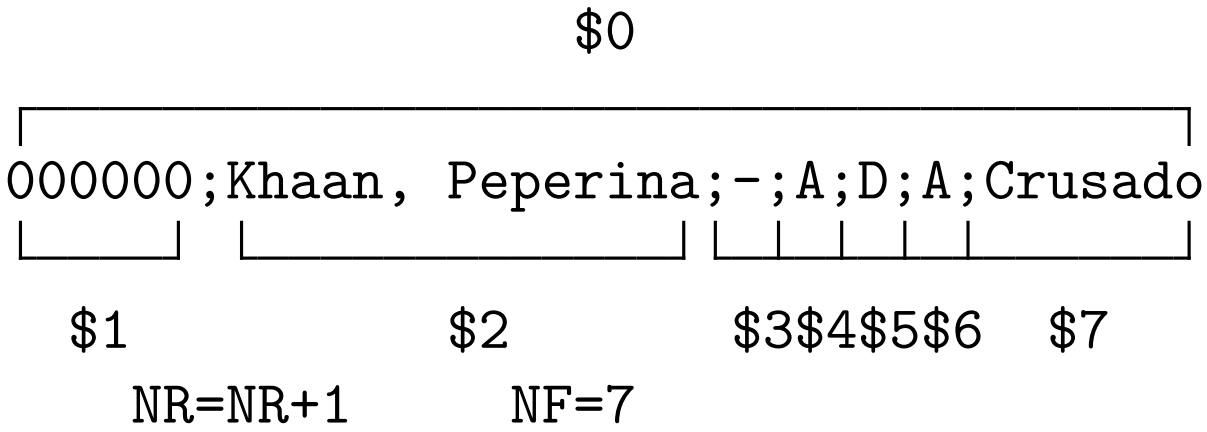
El *main loop* de AWK

Leyendo registro, separando campos

La variable RS determina como se determina la separación de los registros (*default*: salto de linea). Se almacena en la variable $0 y se incrementa NR

Una vez leído un registro, se separa en campos (*fields*) según lo determinado por FS (*default*: espacio en blanco). La cantidad se almacena en la variable NF, y cada uno de ellos en las variables $1, $2, ... $NF.

Si RS="\n" y FS=";":



Existen tres tipos de reglas:

0. Sin cuerpo, se ejecuta el cuerpo para todos los registros.

   ```
   { # Acciones de la regla }
   ```

1. Con una condición, solo si es verdadera se ejecutara el cuerpo.

   ```
   (condición){ # Acciones de la regla }
   ```

2. Con dos condiciones, se ejecutara el cuerpo para todos los registros que se lean desde que la condición sea verdadera, hasta que deje de serlo (inclusive).

   ```
   inicio,fin{ # Acciones de la regla }
   ```

Las condiciones pueden ser tan complejas como se desee, hasta se pueden llamar funciones, pero deben evaluar a algo equivalente a falso: 0 o la cadena nula, o ser equivalente a verdadero: cualquier cosa que no sea falso.

Ademas de los operadores clásicos como `<=`, `||`, `!=`, *etc.*, se cuenta con los operadores:

- Coincidencia de la expresión regular: `~` y su negación: `!~`. Ejemplo: `("bash"~/sh/) == true`.
- Pertenencia a un arreglo: `in`. Ejemplo: `arr["user"]="pepe"; ("user" in arr) == true`

El *main loop* de AWK

Condiciones BEGIN y END

Existen dos condiciones especiales:

- BEGIN: el cuerpo se ejecutara antes de comenzar a leer los registros. Se puede utilizar para cambiar FS y RS.
- END: esta regla se ejecutara solo al finalizar la lectura de los registros.

Sigue el paradigma estructurado y tiene una sintaxis muy similar a C. Se pueden utilizar estructuras como `while`, `if`, `for(init; cond; inc)`, `for(i in array)`, entre otras.

Para declarar una variable, solo hace falta utilizarla. Comienzan con valor cero o la cadena nula ¡**CUIDADO**: son todas globales! La única excepción son los parámetros de las funciones.

Funciones básicas

- `print(s)`: imprime `s` por pantalla.
- `printf(...)`: permite controlar el formato de salida, equivalente a la función de `C` con el mismo nombre.
- `length(s)`: retorna la longitud de la cadena `s`.
- `gsub(regExp, string, res)`: remplaza las ocurrencias de la expresión regular `regExp`, en la cadena `res`, por la cadena `string`.
- `tolower(s)` y `toupper(s)`: retornan una copia de la cadena `s` convertida a minúsculas o mayúsculas.

¡Y muchísimas más! Leer `man awk`.

Declaración de funciones

Para definir funciones se debe usar la palabra clave `function`.

Tener en cuenta que solo los parámetros son variables locales, por otro lado no es necesario definirlos todos cuando se invoca una función.

```
function mucho_texto(cant, texto, _, i, res){
    for(i=0; i<cant; i++){
        res = res texto #concatena res y texto
    }
    return res
}
{ print mucho_texto($1,$2) }
```

Como escribir programas

Existen dos maneras de invocar programas:

- Incluyendo el texto del programa como uno de los parámetros:

```
$ cat /etc/password |
    awk 'BEGIN{FS=":"} ($7 ~ "sh$"){print $1":\t"$7}'
```

- Guardando el texto en un archivo:

```
$ cat /etc/password | awk -f checkShell.awk
```

  - También se puede hacer a través de un *shebang*:

```
$ head -n1 checkShell
    #!/usr/bin/awk -f
$ ./checkShell /etc/password
```

Temario

- ¿Qué es `AWK`?
- ¿Para que aprenderlo?
- El *main loop* de `AWK`:
  - Leyendo registro, separando campos.
  - Reglas: condiciones y acciones.
- Funciones básicas.
- Definición de funciones.
- Como escribir programas:
  - *one-liners*.
  - *scripts*.
- **¡Práctica!**

Atribuciones

[1] P. S. Foresman.
Psf a-60002.png.
https://commons.wikimedia.org/wiki/File:PSF_A-60002.png (public domain).