

# Programación con Shell Scripting: Sesión 11

Máster Universitario en Bioinformática



**Universidad**  
Internacional  
de Valencia

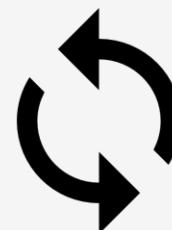
Dra. Paula Soler Vila  
[paula.solerv@professor.universidadviu.com](mailto:paula.solerv@professor.universidadviu.com)

De:  
 Planeta Formación y Universidades



## Aspectos a tratar

- 1 Finalizando el comando awk
- 2 Listas, vectores o Arrays
- 3 Sentencias de control de flujo:



### Bucles o *loops*

- *for*
- *while*

## Ejercicio final -> Comando awk

```
> $ cat data.txt  
seq1 chr1 19  
seq3 chr1 34  
seq2 chr2 182  
  
seq1 chr10 55  
seq2 chr11 33  
  
seq4 chr3 22
```

- 1. Imprime las líneas que no contengan el 33 en el campo 3.**

```
awk '{if($3!=33) print $0}' data.txt
```

- 2. Imprime las líneas que contengan el 182 en el campo 3.**

```
awk '{if($3==182) print $0}' data.txt
```

- 3. Imprime las líneas que contengan un valor mayor o igual a 50 en el campo 3.**

```
awk '{if($3>=50) print $0}' data.txt
```

## Ejercicio final -> Comando awk

```
> $ cat data.txt
seq1 chr1 19
seq3 chr1 34
seq2 chr2 182

seq1 chr10 55
seq2 chr11 33

seq4 chr3 22
```

### 4. Calcula el promedio del campo 3

```
sed '/^$/d' data.txt | awk 'BEGIN{valor=0}{valor += $3}END{print valor/NR}'
```

# Cursos de AWK Shell Scripting



Categorías



Buscar cualquier cosa

Desarrollo > Lenguajes de programación > Linux

## Awk Shell Scripting de novato a experto

Aprende AWK, el lenguaje de programación para manipular archivos planos en Linux.

4,5 ★★★★★ (67 calificaciones) 608 estudiantes

Creado por [Alejandro Guzman Castellanos](#)

Última actualización: 2/2022    Español    Español

### Lo que aprenderás

- ✓ Aprenderás hacer operaciones rápidas sobre archivos estructurados.
- ✓ Dominaras un lenguaje sencillo pero poderoso para realizar diversas tareas de básicas a completas en la administración de Sistemas.
- ✓ Aprenderás a utilizar un lenguaje de programación orientado a búsqueda de patrones y operaciones específicas sobre el resultado.

The screenshot shows the GitHub repository page for 'lh3/bioawk'. The repository is public and has 119 forks and 560 stars. The 'Code' tab is selected, showing the master branch with 1 branch and 1 tag. A recent merge commit from 'lh3' is visible, along with commits from '.gitignore' and 'FIXES'. The 'About' section describes Bioawk as a modified awk for biological data, with tags for bioinformatics and sequence-analysis. The 'Introduction' section explains Bioawk's purpose and dependencies. The footer indicates no packages have been published.

lh3 / bioawk Public

Code Issues 20 Pull requests 4 Actions Projects Wiki Security Insights

master 1 branch 1 tag Go to file Code

**lh3 Merge branch 'master' of github.com:lh3/bioawk** fd40150 on Sep 11, 2017 51 commits

.gitignore rename README; support CLI "bioawk -tc help" 10 years ago

FIXES The original BWK-awk. 12 years ago

**About**

BWK awk modified for biological data

bioinformatics sequence-analysis

Readme Activity

## Introduction

Bioawk is an extension to [Brian Kernighan's awk](#), adding the support of several common biological data formats, including optionally gzip'ed BED, GFF, SAM, VCF, FASTA/Q and TAB-delimited formats with column names. It also adds a few built-in functions and a command line option to use TAB as the input/output delimiter. When the new functionality is not used, bioawk is intended to behave exactly the same as the original BWK awk.

The original awk requires a YACC-compatible parser generator (e.g. Byacc or Bison). Bioawk further depends on [zlib](#) so as to work with gzip'd files.

kseq.h updated kseq.h to bring in empty line bug fix from klib/37b020f9db, a... 10 years ago No packages published

<https://github.com/lh3/bioawk>

# Instalación de BIOAWK

```
$ bioawk  
bash: bioawk: command not found
```

```
$ conda install -c bioconda bioawk
```

```
Retrieving notices: ...working... done  
Collecting package metadata (current_repodata.json): done  
Solving environment: done
```

```
$ bioawk -c help  
bed:  
 1:chrom 2:start 3:end 4:name 5:score 6:strand 7:thickstart 8:thickend 9:rgb 10:blockcount 11:blocksizes 12:blockstarts  
sam:  
 1:qname 2:flag 3:rname 4:pos 5:mapq 6:cigar 7:rnext 8:pnext 9:tlen 10:seq 11:qual  
vcf:  
 1:chrom 2:pos 3:id 4:ref 5:alt 6:qual 7:filter 8:info  
gff:  
 1:seqname 2:source 3:feature 4:start 5:end 6:score 7:filter 8:strand 9:group 10:attribute  
fastx:  
 1:name 2:seq 3:qual 4:comment
```





## Aspectos a tratar

- 1 Finalizando el comando awk
- 2 Listas, vectores o Arrays
- 3 Sentencias de control de flujo:  
  


### Bucles o *loops*

- *for*
- *while*

# Arrays (listas) con Bash

```
array=(a1.fasta chr1 35 "human genome" sox13)
```

- 1 Variable con múltiples valores (**distinta naturaleza**)
- 2 Valores secuenciales
- 3 **NO** hay límite máximo
- 4 El índice del campo comienza en **cero**

# ¿Cómo declaramos una lista de valores?

1

```
array=(comp1 comp2 comp3)
```



```
$ numeros=(1 2 3 4 5)
```

0 1 2 3 4 (índices de posición)

2

```
array[0]=comp1  
array[1]=comp2  
array[2]=comp3
```



```
$ num[0]=1  
$ num[1]=2  
$ num[2]=3
```

*El nombre de los arrays siguen las mismas normas, convenciones y pautas, que los nombres de las variables*

# Mostrar el contenido completo de una lista



```
echo ${lista[@]}
```



```
echo ${lista[*]}
```

```
$ chrom=(chr1 chr2 chr3 chr4 chr5)
```

```
$ echo ${chrom[*]}  
chr1 chr2 chr3 chr4 chr5
```

```
$ echo ${chrom[@]}  
chr1 chr2 chr3 chr4 chr5
```

# Acceder a un elemento de una lista

```
echo ${lista[ÍNDICE]}
```

```
$ chrom=(chr1 chr2 chr3 chr4 chr5)
```

```
$ echo ${chrom[*]}  
chr1 chr2 chr3 chr4 chr5
```

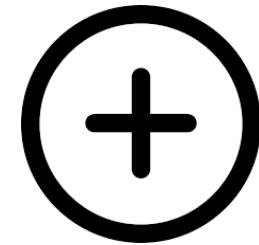
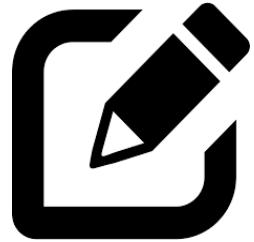
```
$ echo ${chrom[@]}  
chr1 chr2 chr3 chr4 chr5
```

```
$ echo ${chrom[2]}  
chr3
```

```
$ echo ${chrom[10]}
```

# *Añadir o modificar un elemento del array*

lista[índice o posición]=nuevo\_valor



```
$ echo ${chrom[@]}

chr1 chr2 chr3 chr4 chr5

$ chrom[2]=chr21

$ echo ${chrom[@]}

chr1 chr2 chr21 chr4 chr5
```

```
$ echo ${chrom[@]}

chr1 chr2 chr3 chr4 chr5

$ chrom[5]=chrX

$ echo ${chrom[@]}

chr1 chr2 chr21 chr4 chr5 chrX
```

## *Añadir al final, sin necesidad de especificar el índice*

lista`+=(nuevo_valor(es))`

```
$ chrom=(chr1 chr2 chr3 chr4 chr5)
$ chrom+=(sox13 baf1 ctcf)
$ echo ${chrom[@]}
chr1 chr2 chr3 chr4 chr5 sox13 baf1 ctcf

$ pos=(10 256 8900 10000)
$ concatenado=(${chrom[@]} ${pos[@]}) 
$ echo ${concatenado[@]}
chr1 chr2 chr3 chr4 chr5 sox13 baf1 ctcf 10 256 8900 10000
```

## *Eliminar un elemento del array*

unset lista[índice o posición]

```
$ echo ${chrom[@]}

chr1 chr2 chr3 chr4 chr5 sox13 baf1 ctcf
```

```
$ unset chrom[0]

$ echo ${chrom[@]}

chr2 chr3 chr4 chr5 sox13 baf1 ctcf
```

```
$ unset chrom

$ echo ${chrom[@]}
```

# Arrays (listas) con Bash

## Características del array

1. Verificar el índice de cada elemento -> `echo ${!numeros[*]}`

```
numeros=(1 2 3 4 5)
numeros[10]=6
echo "Contenido del array:"
echo ${numeros[*]}
echo "Índices del array:"
echo ${!numeros[*]}
```

Y mostraría:

Contenido del array:

1 2 3 4 5 6

Índices del array:

0 1 2 3 4 10

# *Arrays (listas) con Bash*

## Características del array

2. Verificar la longitud del array-> **echo \${#numeros[\*]}**

```
$ echo ${numeros[*]}
```

```
1 100 3 4 5 6
```

```
$ echo ${numeros[*]} | wc -w
```

```
6
```

```
$ echo ${#numeros[*]}
```

```
6
```

## *Creación de rangos dentro de listas*

{inicio..fin}

```
cifrasLetras=( {A..Z} {a..z} {0..9} )  
  
$ echo ${cifrasLetras[*]}  
  
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z  
a b c d e f g h i j k l m n o p q r s t u v w x y z  
0 1 2 3 4 5 6 7 8 9
```

*¿Cómo generarías una lista con todos los cromosomas humanos?*

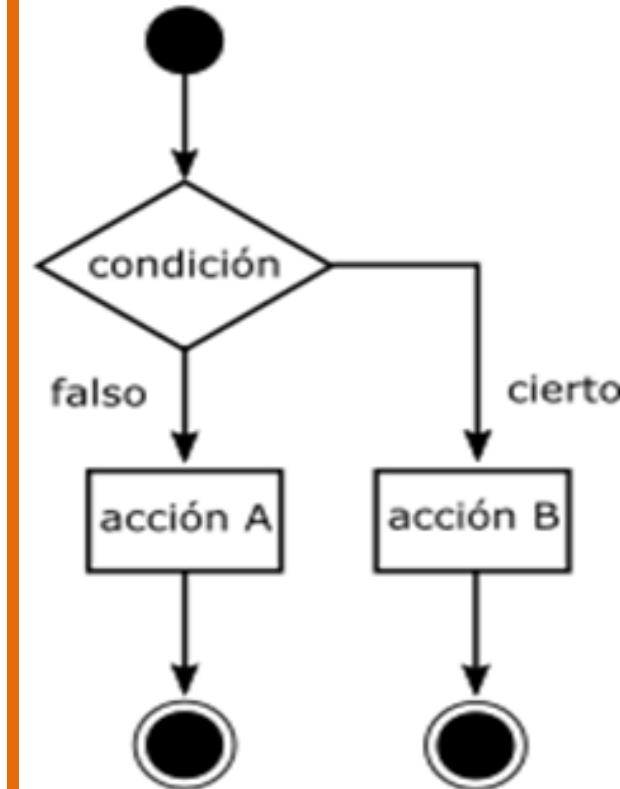
```
$ cromosomas=(chr{1..22} chrX chrY)  
  
$ echo ${cromosomas[*]}  
chr1 chr2 chr3 chr4 chr5 chr6 chr7 chr8 chr9 chr10 chr11 chr12 chr13 chr14 chr15 chr16 chr17 chr18 chr19  
chr20 chr21 chr22 chrX chrY
```

# *Listado de archivos en un array*

```
$ ls  
genes.txt seq1.fasta seq2.fasta seq3.fasta  
  
$ fichero=(`ls *.fasta`)  
con los acentos graves está diciendo quiero que listes todos los archivos que tengan una expansión .fasta  
$ echo ${fichero[*]}  
seq1.fasta seq2.fasta seq3.fasta  
  
$ echo ${#fichero[*]}  
3  
  
$ echo ${fichero[0]}  
seq1.fasta
```

→ ¿Interacción con **cada elemento** del array ?

# Control de flujo



## Control de flujo: Bucles

```
FOR (I IN 1:100) {  
  PRINT("I WILL NOT USE LOOPS IN R")  
}
```

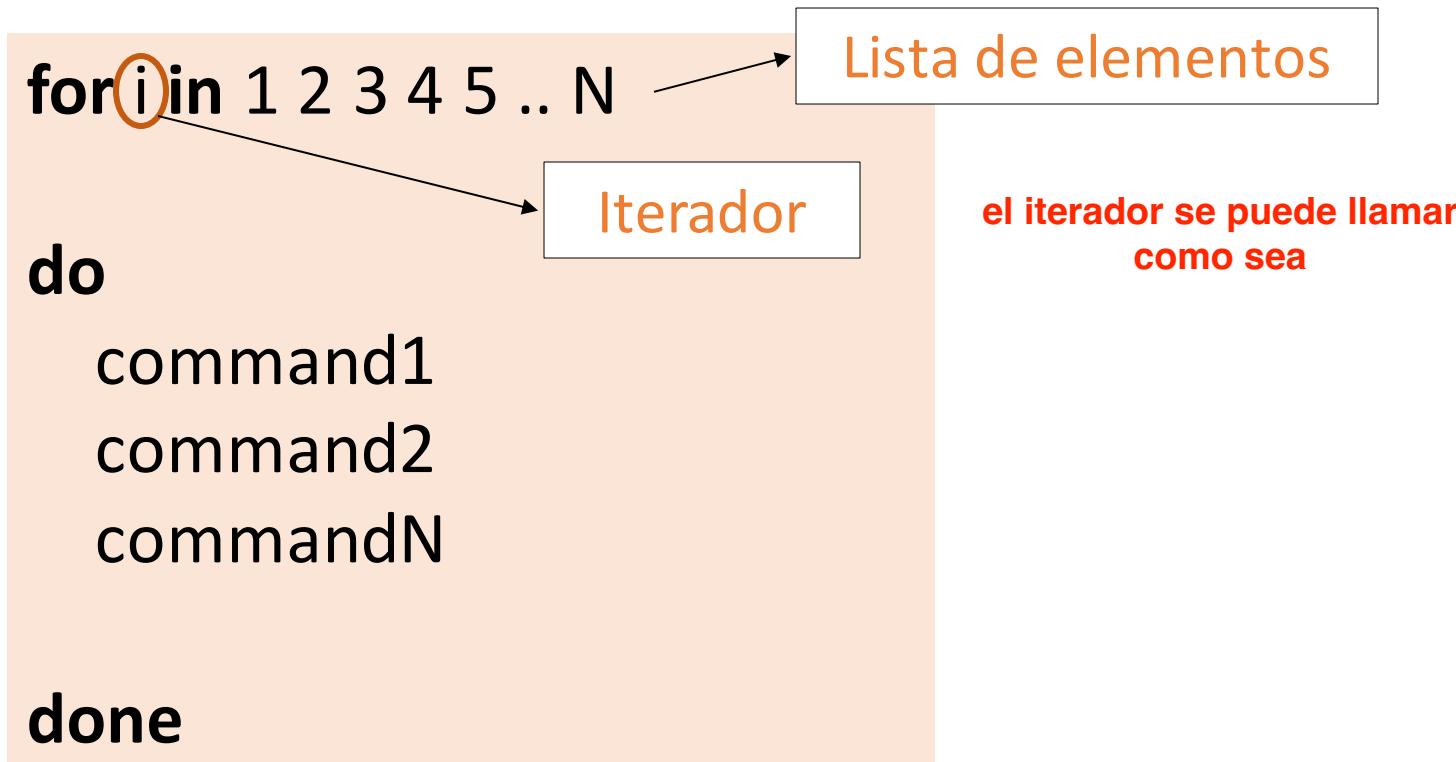


# Control de flujo: Bucles

# EL BUCLE FOR

# Sintaxis del bucle FOR

Se usa para ejecutar un conjunto de comandos dado un número **conocido** de veces



# Sintaxis del bucle FOR

Se usa para ejecutar un conjunto de comandos dado un número **conocido** de veces

**for i in 1 2 3 4 5 .. N**

**do**

- ↔ command1
- ↔ command2
- ↔ commandN

**done**

**Sangrado**  
**(NO es esencial)**

# Bucles for aplicado a listas de números

```
#!/usr/bin/bash
for num in 1 2 3 4 5
do
    echo "Hello $num"
done
```

script.sh

```
$ bash script.sh
Hello 1
Hello 2
Hello 3
Hello 4
Hello 5
```

# Bucles for aplicado a listas de números

Línea de comandos

```
$ for num in 1 2 3 4 5; do echo "Hello $num"; done  
Hello 1  
Hello 2  
Hello 3  
Hello 4  
Hello 5
```

```
$ for i in 3 43 44 11 9; do echo $i; done  
3  
43  
44  
11  
9
```

Orden establecido



# Bucles for aplicado a listas de números: Rangos

```
#!/usr/bin/bash
for num in 1 2 3 4 5
do
echo "Hello $num"
done
```

```
#!/usr/bin/bash
for num in {1..5}
do
echo "Hello $num"
done
```

```
$ for num in {1..10..2}; do echo $num; done
1
3
5
7
9
```

# Bucles for aplicado a listas de números: Creación de arrays

```
#!/usr/bin/bash
for num in 1 2 3 4 5
do
echo "Hello $num"
done
```

```
#!/usr/bin/bash
for num in {1..5}
do
echo "Hello $num"
done
```

```
#!/usr/bin/bash
num=(1 2 3 4 5) / num=(`seq 1 5`)
for i in ${num[@]}
do
echo "Hello $i"
done
```

# Bucles for aplicado a listas de palabras

```
#!/usr/bin/bash
genes=(SOX13 PAX5 TC1 ADF)
for gen in ${genes[@]}
do
echo "La longitud de $gen es ${#gen}"
done
```

```
$ for gen in ${genes[@]}; do echo "La longitud de $gen es ${#gen}"; done
```

La longitud de SOX13 es 5

La longitud de PAX5 es 4

La longitud de TC1 es 3

La longitud de ADF es 3

# Bucles for aplicado a listas de archivos

```
#!/usr/bin/bash
for archivo in *.fasta
do
    grep ATGC $archivo
    echo "El patron se ha encontrado en el archivo $archivo"
done
```

El patron se ha encontrado en el archivo fast1.fasta

El patron se ha encontrado en el archivo fast2.fasta

El patron se ha encontrado en el archivo fast3.fasta

El patron se ha encontrado en el archivo fast4.fasta

ATGC

El patron se ha encontrado en el archivo fast5.fasta

# Bucles for aplicado a varios comandos

```
#!/usr/bin/bash

nombre=`cat /etc/passwd | cut -d : -f1` 

for user in ${nombre[@]}
do
    echo "El nombre del usuario es: $user"
done
```

```
(base) [UNIVERSIDADVIU\paula.soler@a-3edhijmqygwxr ~]$ bash usuario.sh
El nombre de usuario es: root
El nombre de usuario es: bin
El nombre de usuario es: daemon
El nombre de usuario es: adm
El nombre de usuario es: lp
El nombre de usuario es: sync
El nombre de usuario es: shutdown
...
```



# Creando un bucle de tres expresiones

```
for (( EXP1; EXP2; EXP3 ))
```

```
do
```

```
    command1
```

```
    command2
```

```
    command3
```

```
done
```

Este bucle se compone de **tres** expresiones escritas:

- un inicializador (**EXP1**).
- una condición (**EXP2**).
- una expresión de conteo (**EXP3**).

# Creando un bucle de tres expresiones

```
#!/usr/bin/bash  
  
for (( c=1; c<=5; c++ ))  
  
do  
  
    echo "Loop number: $c"  
  
done
```

Este bucle se compone de tres expresiones escritas:

- un inicializador (**EXP1**) -> `c=1`
- una condición (**EXP2**) -> `c<=5`
- una expresión de conteo (**EXP3**) -> `c++`

# Creando un bucle de tres expresiones

```
$ for ((c=1; c<=5; c++)); do echo "Loop number: $c"; done
```

c+=1

Loop number: 1

Loop number: 2

Loop number: 3

Loop number: 4

Loop number: 5

# Creando un bucle de tres expresiones

```
$ for ((c=15; c>0; c-=1)); do echo "Loop number: $c"; done
```

```
Loop number: 15
```

```
Loop number: 14
```

```
Loop number: 13
```

```
Loop number: 12
```

```
Loop number: 11
```

```
Loop number: 10
```

```
Loop number: 9
```

```
Loop number: 8
```

```
Loop number: 7
```

```
Loop number: 6
```

```
Loop number: 5
```

```
Loop number: 4
```

```
Loop number: 3
```

```
Loop number: 2
```

```
Loop number: 1
```

# El bucle FOR: RESUMEN

```
for i in 1 2 3 4 5 6;
```

```
for i in {1..6};
```

```
for i in archivo1 archivo2 archivo3;
```

```
for i in ${lista[@]};
```

```
for i in ((exp1; exp2; exp3));
```



**viu**

**Universidad  
Internacional  
de Valencia**

[universidadviu.com](http://universidadviu.com)

De:  
 Planeta Formación y Universidades