Bucles

Los bucles permiten

la **repetición**
de comandos

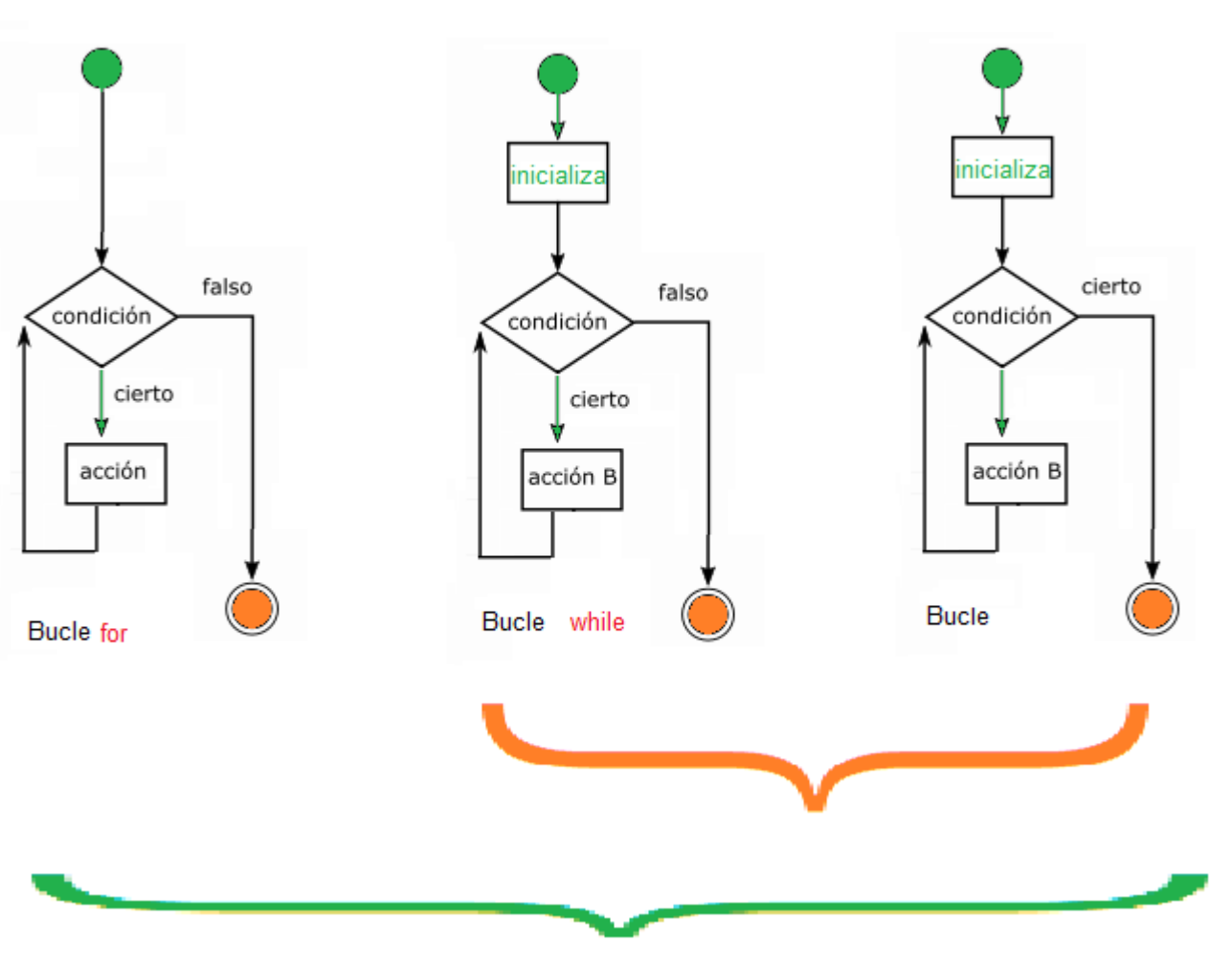
siempre que se cumpla
con una expresión
condicional

(c) Pino Mendoza

Bucles

Tres tipos o formas de hacer bucles

1. for
2. while
3. until

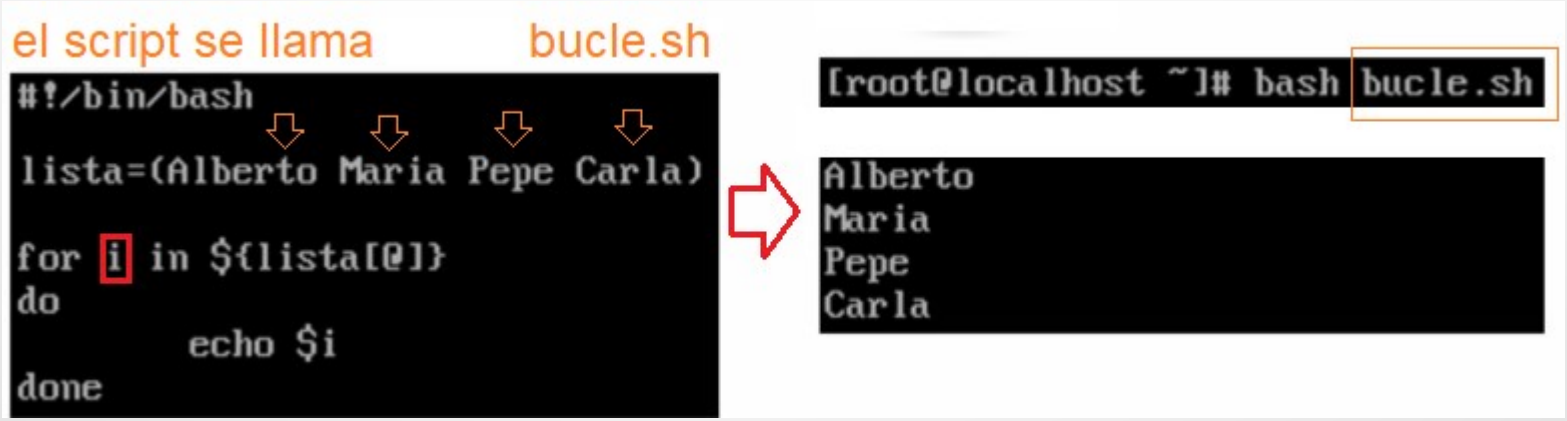


Bucles. 1. for

- También repite unas acciones de forma cíclica, mientras se cumpla una condición (como en el while)

- Con la excepción de que además nos permite recorrer tablas, listas, arrays.

- El valor del índice aumenta de forma "autoincremental" en cada ciclo/vuelta



(c) Pino Mendoza

Bucles. 2. while

•Los bucles “while” ejecutan unas acciones en bucle siempre que se cumpla una condición.

•**No** es “autoincremental”, debemos producir un cambio nosotros. (el bucle for sí lo es)

```
#!/bin/bash
```

```
i=5

echo "Cuenta atrás..."

while (( i >= 1 ))
do
        echo "$i"
        sleep 1
        ((i--))
done
```

Las variables numéricas que forman parte de la condición no se incrementa automáticamente.

Por lo que tenemos que poner una instrucción para incrementarla ((i++)) o ((i--))

Bucles. 3. until

- Hemos dicho de while:
  - Los bucles “while” ejecutan unas acciones en bucle siempre que se cumpla una condición.
  - **No** es “autoincremental”, debemos producir un cambio nosotros.

El bucle until

- Empieza ejecutando HASTA QUE se cumpla la condición

```
#!/bin/bash
result=1
i=1
read -p " introduce un número para el valor máximo " x

until [ $i -gt $x ]
do
        let "result*=$i"
        (( i++ ))
done
echo " $result"
```

Si introduces x igual a 6

Resultado

1*2*3*4*5*6

Las variables numéricas que forman parte de la condición no se incrementa automáticamente.

Por lo que tenemos que poner una instrucción para incrementarla ((i++)) o ((i--))

Probar Scripts máquina virtual + online

Probamos los scripts en la máquina virtual de Linux.

Pero si quieres practicar en cualquier lugar y desde el móvil, solo para scripts muy sencillos, los del principio.

Puedes probar los scripts sencillos con:

https://www.tutorialspoint.com/execute_bash_online.php

O con

https://rextester.com/l/bash_online_compiler (más rápido)

Bucles for PRUÉBALO

el script se llama bucle.sh

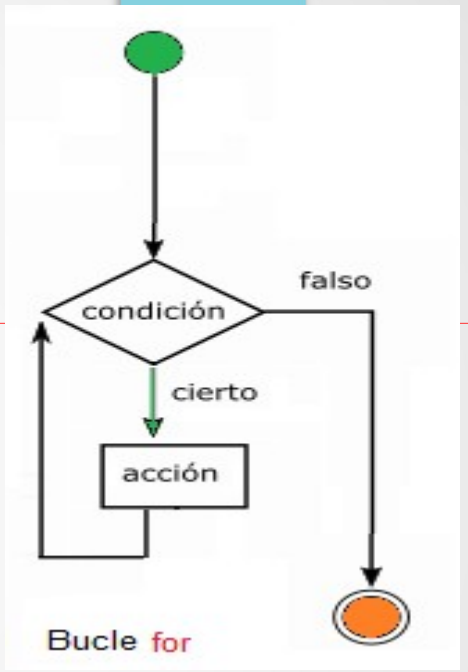
```
#!/bin/bash
lista=(Alberto Maria Pepe Carla)

for i in ${lista[@]}
do
        echo $i
done
```

```
[root@localhost ~]# bash bucle.sh

Alberto
Maria
Pepe
Carla
```

PRUEBA
1forbucle.sh




```
for hora in {1..4}
do
   echo "en el  for  → la hora vale $hora"
done
```

```
1  # Probando bucles
2
3  for hora in {1..4}
4  do
5     echo "en el for --> hora vale
6          $hora"
   done
```

RESULTADO

```
en el for -- hora vale 1
en el for -- hora vale 2
en el for -- hora vale 3
en el for -- hora vale 4
```

Bucles for PRUÉBALO

```
for hora in {1..4}
do
   echo "en el  for  → la hora vale $hora"
done
```

PRUEBA
A CAMBIAR COSAS

1

AÑADIR ESPACIOS por EJEMPLO
En el rango

{ 1 .. 4 }

2

A añadir
una línea en blanco
entre
el for y el do

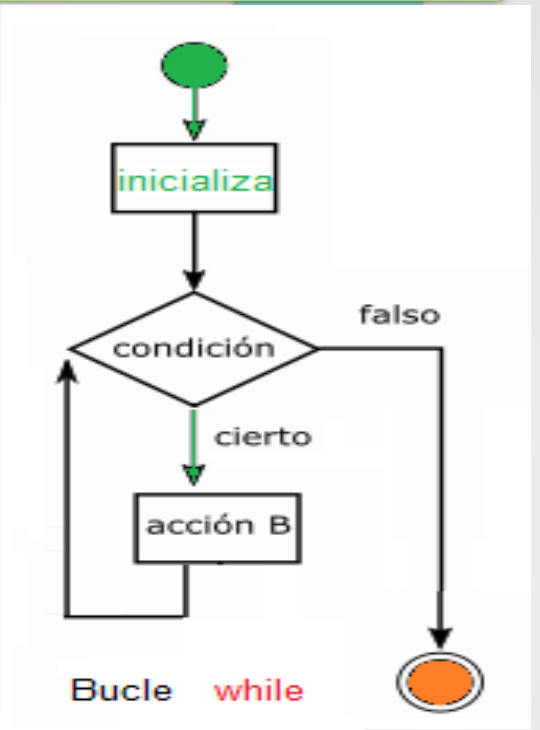
for hora in {1..4}

do

Bucles while PRUÉBALO

```
hora=1
while (( hora<5 ))
do
    echo "en el while --> la hora vale $hora"
     (( hora++ ))
done
```

PRUEBA
1whilebucle.sh



Bucle while

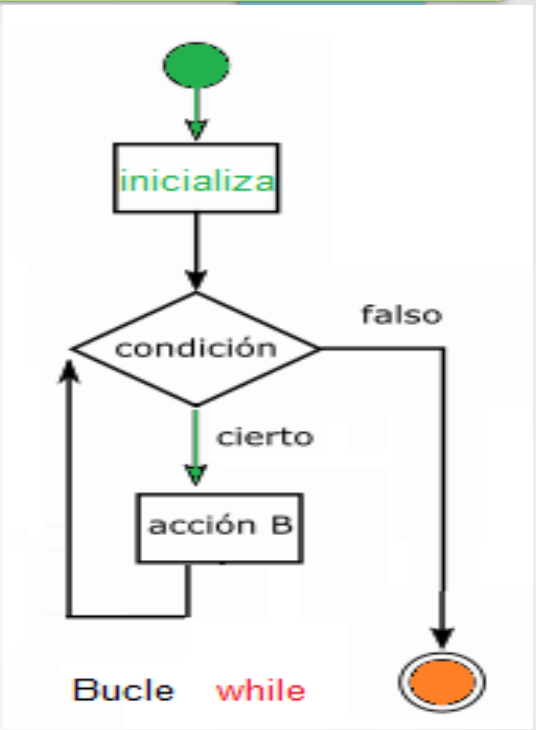
```
1  # Probando bucles while
2
3  hora=1
4  while (( hora<5 ))
5  do
6    echo "en el while -->  vale $hora"
7     (( hora++ ))
8  done
```

```
en el while --  vale 1
en el while --  vale 2
en el while --  vale 3
en el while --  vale 4
```

Bucles while PRUÉBALO

```
hora=1
while (( hora<5 ))
do
     echo "en el while --> la hora vale $hora"
      (( hora++ ))
done
```

PRUEBA
A CAMBIAR COSAS



1 Separa
hora < 5

2 une
((hora < 5))

3 Une todo
((hora<5))

4 Pon un $
while (($hora <5))

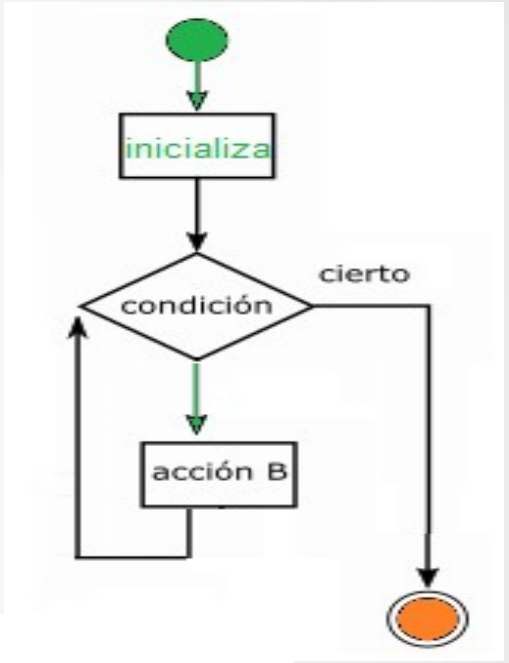
5 Pon un $
(($hora++))

Bucles **until** PRUÉBALO

```
hora=1
until [ $hora -gt 4 ]
do
   echo "en el until  vale $hora"
     (( hora++))

done
```

PRUEBA
1untilbucle.sh




```
1  # Probando bucles until
2
3  hora=1
4  until [ $hora -gt 4 ]
5  do
6     echo "en el until  vale $hora"
7       (( hora++))
8
9  done
```

```
en el until  vale 1
en el until  vale 2
en el until  vale 3
en el until  vale 4
```

hasta que $hora

sea mayor que 4

Bucles **until** PRUÉBALO

```
hora=1
until [ $hora -gt 4 ]
do
   echo "en el until  vale $hora"
     (( hora++))
done
```

PRUEBA
A CAMBIAR COSAS

Prueba a cambiar

[por (

until ($hora -gt 4)

Pon un $
(($hora++))

Prueba a cambiar

[por (y añade >

until ($hora > 4)

Prueba a cambiar

[por (y quita $ añade >

until (hora > 4)

Los tres códigos --- comparados

```
for hora in {1..4}
do
   echo "en el for --> hora vale
       $hora"
done
```

FOR

```
en el for -- hora vale 1
en el for -- hora vale 2
en el for -- hora vale 3
en el for -- hora vale 4
```

```
# Probando bucles while

hora=1
while (( hora<5 ))
do
  echo "en el while -->  vale $hora"
   (( hora++ ))
done
```

```
en el while -- vale 1
en el while -- vale 2
en el while -- vale 3
en el while -- vale 4
```

```
hora=1
while [ $hora -lt '5' ]
do
     echo "en el while --> la hora vale $hora"
      (( hora++ ))
done
```

```
en el while --> la hora vale 1
en el while --> la hora vale 2
en el while --> la hora vale 3
en el while --> la hora vale 4
```

```
# Probando bucles until

hora=1
until [ $hora -gt 4 ]
do
   echo "en el until  vale $hora"
      (( hora++))

done
```

```
en el until  vale 1
en el until  vale 2
en el until  vale 3
en el until  vale 4
```

hasta que $hora sea mayor que 4

Bucles.

Repetir lo mismo con diferentes valores

```
 i=0
while [ $i -lt 1000 ]
do
    echo 'Cemento Fresco, no hay letrero más bello… bueno, sólo Alto Voltaje.'
    ((i++))
done
```

- Aquí tienes que fijarte en la **forma de incrementar** el contador **i**, que has utilizado ((i++))

- Nota: Las operaciones matemáticas se encerraban entre dobles paréntesis.

  - Otra observación: Tienes que **inicializar la variable i**, si utilizar corchetes simples.

  - Si utilizas corchetes dobles **no** necesitas inicializar la variable.

Amplia … condiciones

https://www.atareao.es/tutorial/scripts-en-bash/condicionales-en-bash/

En Bash, esto de los condicionales se materializa con **if then else** y con **case.**
Se parte de resolver una **cuestión**, una prueba, un test, al final una **comparación**.

PRUEBA
1if.sh

```
#!/bin/bash

if [[ $1 =~ (.*)a$ ]]
then
    echo Sra. $1
else
    echo Sr. $1
fi
```

En el caso [Tiene que estar **separado** el corchete de la expresión que viene a continuación, porque es un comando.

Caso dobles corchetes [[Son una mejora respecto a los simples.

Te recomiendo ampliar conocimientos. Cadenas:
https://poesiabinaria.net/2015/10/9-trucos-para-manejar-cadenas-de-caracteres-en-bash-y-no-morir-en-el-intento/

(c) Pino Mendoza

Amplia … condiciones

Mira estos ejemplos:
https://www.atareao.es/tutorial/scripts-en-bash/condicionales-en-bash/

1- No tienes que utilizar las **comillas** con las variables, los **dobles corchetes** trabajan perfectamente con los espacios.
Así [-f **"**$file**"**] es equivalente a [[-f $file]].

2- Con **[[** puedes utilizar los operadores **|| y &&,** así como <y>`
para las **comparaciones** de cadena.

3- Puedes utilizar el **operador =~** para expresiones regulares,
Por ejemplo [[$respuesta =~ ^s(i)?$]]
También puedes utilizar **comodines** como por ejemplo en la expresión
[[abc = a*****]]

4- Es posible que te **preguntes** por la razón para seguir utilizando **[simple** corchete en lugar de doble. La cuestión es por **compatibilidad**.

Si utilizas Bash en diferentes equipos es posible que te encuentres alguna incompatibilidad.
Así que depende de ti y de donde lo vayas a utilizar.

(c) Pino Mendoza

Amplia … condiciones

[o [[Así, las diferencias entre uno y otro son las siguientes,

COMPARANDO CADENAS o strings

[[	[
>	>
<	**<**
=	=
!=	**!=**

COMPARANDO ENTEROS o números

[[	[	
-gt	-gt	mayor
-lt	**-lt**	**menor**
-ge	-ge	mayor o igual
-le	-le	menor o igual
-eq	-eq	igual
-ne	**-ne**	**distinto**

Aquí quiero que te des cuenta que [[001 = 1]] es falso
mientras que [[001 -eq 1]] es cierto.

OPERADORES BOOLEANOS o valores lógicos

[[	[
&&	-a
||	-o

(c) Pino Mendoza

¿Preguntas?