

# Comandos GNU/Linux e SHELL BASH (/bin/bash)

terminator

```
user@computer: ~ 82x24
top - 23:30:36 up 3:26, 0 users, load average: 1,12, 1,18, 1,23
Tasks: 277 total, 2 running, 275 sleeping, 0 stopped, 0 zombie
%Cpu0 : 0,0 us, 0,0 sy, 0,0 ni, 100,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu1 : 1,7 us, 0,3 sy, 0,0 ni, 91,0 id, 7,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu2 : 99,3 us, 0,7 sy, 0,0 ni, 0,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
%Cpu3 : 0,7 us, 0,3 sy, 0,0 ni, 99,0 id, 0,0 wa, 0,0 hi, 0,0 si, 0,0 st
KiB Mem: 3890584 total, 2629160 used, 1261424 free, 333924 buffers
KiB Swap: 2559996 total, 0 used, 2559996 free. 984516 cached Mem

  PID USER      PR  NI   VIRT   RES   SHR  S  %CPU  %MEM     TIME+ COMMAND
 3118 user        20   0   5764   1328  1240 R 100,0  0,0   74:14.74 yes
1365 root         20   0  447536  8768  7668 S   0,7  0,2   0:33.04 gvfsd-trash
1974 user        20   0 1981900 227836 55620 S   0,7  5,9   2:18.69 gnome-shell
 411 root        -51   0     0     0     0 S   0,3  0,0   0:06.97 irq/30-iwlwi+
1294 root         20   0  533228  34392 23320 S   0,3  0,9   0:00.79 nm-applet
1705 root         20   0  297288  49880 36332 S   0,3  1,3   1:19.41 Xorg
2010 user        20   0  729972  48340 28644 S   0,3  1,2   0:04.91 nautilus
2087 user        20   0  447536  8480   7372 S   0,3  0,2   0:32.94 gvfsd-trash
5384 root         20   0     0     0     0 S   0,3  0,0   0:00.15 kworker/u8:0
5534 user        20   0  25800   3112  2496 R   0,3  0,1   0:01.30 top
   1 root         20   0  177548  5772  3092 S   0,0  0,1   0:01.67 systemd
   2 root         20   0     0     0     0 S   0,0  0,0   0:00.00 kthreadd
   3 root         20   0     0     0     0 S   0,0  0,0   0:00.92 ksoftirqd/0
   5 root         0 -20     0     0     0 S   0,0  0,0   0:00.00 kworker/0:0H

root@computer: ~ 8
user@computer:~$ id
uid=1000(user) gid=1001(user) grupos=1001(user),14
user@computer:~$ su -
Contraseña:
root@computer:~$ id
uid=0(root) gid=0(root) grupos=0(root)
root@computer:~$

user@computer: ~ 8
user@computer:~$ ping -c2 www.google.es
PING www.google.es (178.60.128.20) 56(84) bytes of
64 bytes from cache.google.com (178.60.128.20): ic
64 bytes from cache.google.com (178.60.128.20): ic

--- www.google.es ping statistics ---
2 packets transmitted, 2 received, 0% packet loss,
rtt min/avg/max/mdev = 50.088/50.432/50.777/0.411
user@computer:~$
```

gnome-terminal

```
user@computer: ~
Archivo Editar Ver Buscar Terminal Ayuda
Every 1,0s: ip addr show eth0 Sat Jan 16 00:04:22 2016
2: eth0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state DOWN group default qlen 1000
    link/ether 08:9e:01:11:9d:cc brd ff:ff:ff:ff:ff:ff

top - 00:04:21 up 4:00, 0 users, load average: 1,39, 1,25, 1,24
Tasks: 278 total, 2 running, 276 sleeping, 0 stopped, 0 zombie
%Cpu(s): 28,4 us, 3,3 sy, 0,0 ni, 65,9 id, 2,3 wa, 0,0 hi, 0,0 si, 0,0 st
KiB Mem: 3890584 total, 2637656 used, 1252928 free, 339636 buffers
KiB Swap: 2559996 total, 0 used, 2559996 free. 965112 cached Mem

  PID USER      PR  NI   VIRT   RES   SHR  S  %CPU  %MEM     TIME+ COMMAND
 3118 user        20   0   5764   1328  1240 R 100,0  0,0 107:59.28 yes
1974 user        20   0 1989872 237708 56120 S  14,3  6,1  3:09.74 gnome-shell
1705 root         20   0  300376  48400 33788 S   8,0  1,2  1:35.56 Xorg
7323 user        20   0  346828  30192 21864 S   1,7  0,8  0:00.42 gnome-scre+
1365 root         20   0  447536  8768  7668 S   1,0  0,2  0:47.33 gvfsd-trash
2087 user        20   0  447536  8488  7372 S   1,0  0,2  0:47.21 gvfsd-trash
 411 root        -51   0     0     0     0 S   0,3  0,0  0:08.04 irq/30-iwl+
 801 message+    20   0  43964  4968  2992 S   0,3  0,1  0:01.91 dbus-daemon
 840 redis       20   0  38200  3092  2272 S   0,3  0,1  0:07.21 redis-serv+
1881 user        20   0  42964  3340  2064 S   0,3  0,1  0:01.44 dbus-daemon
   1 root         20   0  177548  5772  3092 S   0,0  0,1  0:01.70 systemd
   2 root         20   0     0     0     0 S   0,0  0,0  0:00.00 kthreadd
   3 root         20   0     0     0     0 S   0,0  0,0  0:01.08 ksoftirqd/0
   5 root         0 -20     0     0     0 S   0,0  0,0  0:00.00 kworker/0:+
   7 root         20   0     0     0     0 S   0,0  0,0  0:06.90 rcu_sched
   8 root         20   0     0     0     0 S   0,0  0,0  0:00.00 rcu_bh
   9 root         rt   0     0     0     0 S   0,0  0,0  0:00.01 migration/0
```

Ricardo Feijoo Costa



This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/)

# arrays (vectores)

## arrays (man bash -> Vectores)

Bash proporciona variables vectores, monodimensionais. Calquera variable pode usarse como un vector; a orde interna **declare** declarará un vector explicitamente. Non hai un límite máximo no tamaño dun vector, nin ningún requisito para que os membros se indexen ou asignen de forma contigua. Os vectores indéxanse empregando enteiros e o seu primer elemento é o de índice cero, como en C. Se se indica o índice na declaración entón asignase a ese índice o valor esperado, senón asignarase o valor ao (último índice + 1)

Pódense especificar atributos para unha variable vector mediante as ordes internas **declare** e **readonly**, onde cada atributo aplícase a cada un dos membros do vector.

As variables de tipo vector non poden (aínda) exportarse.

### Declarar e asignar arrays en BASH:

- `nome[indice]=valor` onde **nome** é o nome do array, **índice** é a posición enteira onde gardarase o valor da variable e **valor** é o valor na posición índice do array
- `nome=( [índice0]=cadea0 [índice1]=cadea1 [índice2]=cadea2 ... [índiceN]=cadeaN )` onde **cadeaX** é o valor asignado ao índiceX
- `nome=(valor1 valor2 valor3 ... valorN)` onde **valor** é da forma `[índice]=cadea` ou `cadea`, co cal se non existe referenciado o índice os valores iranse gardando consecutivamente no (último índice + 1)
- `declare -a nome` onde **nome** é o nome do array
- `declare -a nome[indice]` onde **nome** é o nome do array e **índice** non se ten en conta
- `declare -a nome=( [índice0]=cadea0 [índice1]=cadea1 [índice2]=cadea2 ... [índiceN]=cadeaN )` onde **cadeaX** é o valor asignado ao índiceX
- `declare -a nome=(valor1 valor2 valor3 ... valorN)` onde **valor** é da forma `[índice]=cadea` ou `cadea`, co cal se non existe referenciado o índice os valores iranse gardando consecutivamente no (último índice + 1)

Tamén se poden declarar arrays como variables locais, as cales soamente se poden usar nunha función:

- `local -a nome=( [índice0]=cadea0 [índice1]=cadea1 [índice2]=cadea2 ... [índiceN]=cadeaN )` onde **cadeaX** é o valor asignado ao índiceX
- `local -a nome=(valor1 valor2 valor3 ... valorN)` onde **valor** é da forma `[índice]=cadea`

Tamén coa orde interna **read** pódese asignar unha lista de palabras recollidas dende a entrada estándar a un array:

- `read -a nome` recolle dende a entrada estándar(teclado) a lista de palabras como índices no array de nome **nome**, empezando por 0. Os campos separadores das palabras, que serán recollidas nos índices do array, están determinados polo valor da variable separador de campos **\$IFS**
- `read -p 'mensaxe ' -a nome` Aмоса a mensaxe **mensaxe** ' e recolle dende a entrada estándar(teclado) a lista de palabras como índices no array de nome **nome**, empezando por 0. Os campos separadores das palabras, que serán recollidas nos índices do array, están determinados polo valor da variable separador de campos **\$IFS**
- `read nome[indice]` recolle dende a entrada estándar(teclado) a lista de palabras dentro do índice indicado no array de nome **nome**. Con esta sintaxe o valor da variable separador de campos **\$IFS** non se ten en conta.
- `read -p 'mensaxe ' nome[indice]` Aмоса a mensaxe **mensaxe** ' e recolle dende a entrada estándar(teclado) a lista de palabras dentro do índice indicado no array de nome **nome**. Con esta sintaxe o valor da variable separador de campos **\$IFS** non se ten en conta.

## Referenciar arrays en BASH:

As chaves son necesarias para evitar conflitos co **globbing (expansión de caracteres)**

- `${nome[índice]}` onde **nome** é o nome do array e **índice** é a posición enteira que contén o seu valor correspondente.
- `${nome[@]}` onde **nome** é o nome do array e o **índice @** representa tódolos elementos do vector. Devolve unha cadea cos elementos separados por espazo.
- `${nome[*]}` onde **nome** é o nome do array e o **índice \*** representa tódolos elementos do vector. Devolve unha cadea cos elementos separados por espazo.
- `"${nome[@]}"` onde **nome** é o nome do array e o **índice @** representa tódolos elementos do vector. Devolve unha cadea cos elementos separados por espazo.
- `"${nome[*]}"` onde **nome** é o nome do array e o **índice \*** representa tódolos elementos do vector. Devolve unha cadea cos elementos separados polo primeiro carácter da variable separador de campos IFS
- `${nome}` equivale a `${nome[0]}`

As variables de tipo vector como calquera variable soporta a referencia indirecta (tamén denominada expansión indirecta):

### Referencia indirecta dunha variable:

```
$ nome='Ricardo'
```

```
$ NAME=nome
```

```
$ echo ${NAME} #Dará como resultado o valor da variable NAME
nome
```

```
$ echo ${!NAME} #Referencia indirecta (válida dende a versión 2 de BASH), co cal equivale a: echo ${nome}, que
dará como resultado o valor da variable nome
Ricardo
```

```
$ echo \${NAME} #Sustitúe ${NAME} polo seu valor nome. E unha vez substituído mediante o comando echo
amósa o texto $nome, xa que escápase o primeiro carácter dolar mediante a expresión \$
```

```
$ eval echo \${NAME} #Referencia indirecta, co cal equivale a: echo $nome, que dará como resultado o valor da
variable nome
Ricardo
```

### Referencia indirecta dun array:

```
$ nome=(Anxo Brais)
```

```
$ NAME=(nome[@])
```

```
$ echo ${NAME} #Dará como resultado o valor da variable NAME
nome[@]
```

```
$ echo ${!NAME} #Referencia indirecta (válida dende a versión 2 de BASH), co cal equivale a: echo ${nome[@]},
que dará como resultado o valor da variable nome
Anxo Brais
```

```
$ echo \${NAME} #Sustitúe ${NAME} polo seu valor nome[@]. E unha vez substituído mediante o comando
echo amósa o texto ${nome[@]}, xa que escápase o primeiro carácter dolar mediante a expresión \$
```

```
$ eval echo \${NAME} #Referencia indirecta, co cal equivale a: echo ${nome[@]}, que dará como resultado o
valor da variable nome
Anxo Brais
```

## Eliminar arrays en BASH:

- `unset nome` #Elimina o array declarado como **nome** sempre e cando non esté declarado en modo lectura
- `unset nome[índice]` Elimina o índice do array **nome** da posición [índice] sempre e cando non esté declarado en modo lectura
- `unset nome[@]` #Elimina o array declarado como **nome** sempre e cando non esté declarado en modo lectura. Equivale a **unset nome**
- `unset nome[*]` #Elimina o array declarado como **nome** sempre e cando non esté declarado en modo lectura. Equivale a **unset nome**

## Percorrer arrays en BASH:

- ```
for i in "${nome[*]}"
do
    echo $i
done
```

#Percorre os índices do array declarado como **nome** e amosa cadanseu valor por pantalla. **Empregar sempre as comiñas dobres** para impedir erros de separadores coma espazos existentes nos valores dos índices.
- ```
for i in "${nome[@]}"
do
    echo $i
done
```

#Percorre os índices do array declarado como **nome** e amosa cadanseu valor por pantalla. **Empregar sempre as comiñas dobres** para impedir erros de separadores coma espazos existentes nos valores dos índices.

## Lonxitudes dos arrays e índices en BASH:

- `echo ${#nome[@]}` #Amosa o número de elementos existentes no array declarado como **nome**, é dicir, dá o número de posicións ocupadas no array
- `echo ${!nome[@]}` #Amosa os índices dos elementos que son **NON NULOS** no array declarado como **nome**
- `echo ${#nome[índice]}` #Amosa o número da lonxitude de caracteres do índice **índice** para o array declarado como **nome**

## Arrays como parámetros de funcións:

Na chamada a función empregamos como parámetro a notación "`nome[@]`" e dentro do corpo da función empregamos a referencia indirecta `${!number}`, onde `number` é o número do parámetro na chamada á función: 1, 2, 3...

```
f_name(){
    parametros=("${!1}")
    echo "${parametros[@]}"
}

nome=(Anxo Brais)

f_name "nome[@]"
```

#Definición da función de nome **f\_name**  
#Referencia indirecta do parámetro **\$1** dentro da definición do array **parametros**  
#Amósa todos os valores contidos nos índices do array de nome **parametros**  
# Fin do corpo da función **f\_name**  
#Declárase a variable array **nome** cos valores **Anxo** e **Brais** nos índices 0 e 1 respectivamente.  
#Chamada á execución da función de nome **f\_name** onde o primeiro parámetro **\$1** toma o valor "**nome[@]**", o cal é equivalente aos valores de todos os índices do array de nome **nome**

## Práctica1 arrays: declarar, eliminar

```
$ unset curso #Elimina a variable curso se non está declarada en modo lectura

$ declare -p | grep curso #Amosa o valor da variable de nome curso e como está declarada no caso de existir

$ curso[0]='2015-2016' #Crea automaticament o array de nome curso onde no índice 0 toma o valor 2015-2016.

$ declare -p | grep curso #Amosa o valor da variable array de nome curso e como está declarada

$ curso[1]='2016-2017' #Como o array de nome curso xa está creado asigna no índice 1 do array o valor 2016-2017.

$ declare -p | grep curso #Amosa o valor da variable array de nome curso e como está declarada

$ curso=('2017-2018' '2018-2019') #Xera un novo array de nome curso onde o índice 0 e 1, toman os valores 2017-2018 e 2018-2019 respectivamente.

$ declare -p | grep curso #Amosa o valor da variable array de nome curso e como está declarada

$ curso=([2]=2019-2020) #Aínda que non existe o índice 2 no array de nome curso, a sintaxe do comando xera un novo array con soamente un índice: o índice 2 que toma o valor 2019-2020.

$ curso=([0]='2015-2016' [1]=2019-2020) #Xera un novo array de nome curso onde non existe o índice 2 e onde o índice 0 e 1 toman os valores 2015-2016 e 2019-2020 respectivamente.

$ declare -p | grep curso #Amosa o valor da variable array de nome curso e como está declarada

$ echo ${curso[0]} #Amosa o valor do índice 0 da variable array de nome curso

$ echo ${curso[1]} #Amosa o valor do índice 1 da variable array de nome curso

$ curso=([0]='1111-1112' 1112-1113 1113-1114 1114-1115) #Xera un novo array de nome curso onde existe soamente referenciado na declaración o índice 0, polo que o resto dos elementos do array irán gardando os seus valores nos índices 1, 2 e 3. Así os índices 0,1,2,3 toman os valores 1111-1112, 1112-1113, 1113-1114, 1114-1115 respectivamente.

$ declare -p | grep curso #Amosa o valor da variable array de nome curso e como está declarada

$ curso=([0]='1111-1112' [5]=1112-1113 1113-1114 1114-1115) #Xera un novo array de nome curso onde existen soamente referenciados na declaración os índices 0 e 5, polo que o resto dos elementos do array irán gardando os seus valores a partir do índice 5. Así os índices 0,5,6,7 toman os valores 1111-1112, 1112-1113, 1113-1114, 1114-1115 respectivamente.

$ declare -p | grep curso #Amosa o valor da variable array de nome curso e como está declarada

$ declare -r curso #Declara en modo lectura o variable array de nome curso

$ unset curso #Non pode eliminar a variable curso xa que está declarada en modo lectura

$ curso[0]='1990-1991' #Como a variable array de nome curso está en modo lectura NON pode asignar o valor 1990-1991 ao índice 0.
```

## Práctica2 arrays: modo lectura

```
$ unset curso2 #Elimina a variable curso2 se non está declarada en modo lectura

$ declare -a curso2=('2017-2018' '2018-2019') #Xera un novo array de nome curso2 onde o índice 0 e 1, toman os valores 2017-2018 e 2018-2019 respectivamente.

$ declare -r curso2[0]='1990-1991' #Declara en modo lectura a variable array de nome curso2 e NON se lle asigna o valor 1990-1991.

$ declare -p | grep curso2 #Amosa o valor da variable array de nome curso2 e como está declarada
```

### Práctica3 arrays: modo lectura

```
$ unset curso3 #Elimina a variable curso3 se non está declarada en modo lectura

$ declare -r curso3[0]='1990-1991' #Declara en modo lectura a variable array de nome curso3 e NON se lle asigna o valor 1990-1991.

$ declare -p | grep curso3 #Amosa o valor da variable array de nome curso3 e como está declarada
```

### Práctica4 arrays: declarar, eliminar

```
$ unset curso4 #Elimina a variable curso4 se non está declarada en modo lectura

$ declare -p | grep curso4 #Amosa o valor da variable nome curso4 e como está declarada no caso de existir

$ curso4=(1800-1801 1801-1802) #Xera un novo array de nome curso4 onde o índice 0 e 1, toman os valores 1800-1801 e 1801-1802 respectivamente.

$ declare -p | grep curso4 #Amosa o valor da variable array de nome curso4 e como está declarada

$ unset curso4[1] #Elimina o índice 1 do array de nome curso4 se non está declarada en modo lectura

$ declare -p | grep curso4 #Amosa o valor da variable nome curso4 e como está declarada no caso de existir

$ curso4[2]='1990-1991' #Asigna o valor 1990-1991 á variable array de nome curso4 no índice 2.

$ declare -p | grep curso4 #Amosa o valor da variable nome curso4 e como está declarada no caso de existir

$ unset curso4[*] #Elimina a variable curso4 se non está declarada en modo lectura

$ declare -p | grep curso4 #Amosa o valor da variable nome curso4 e como está declarada no caso de existir

$ curso4=(1800-1801 1801-1802) #Xera un novo array de nome curso4 onde o índice 0 e 1, toman os valores 1800-1801 e 1801-1802 respectivamente.

$ declare -p | grep curso4 #Amosa o valor da variable nome curso4 e como está declarada no caso de existir

$ unset curso4[@] #Elimina a variable curso4 se non está declarada en modo lectura

$ declare -p | grep curso4 #Amosa o valor da variable nome curso4 e como está declarada no caso de existir
```

### Práctica5 arrays: read e IFS

```
$ set | grep ^IFS #Ver valor variable separador de campos IFS (espazo, tabulado, nova liña)
IFS=$'\t\n'

$ unset curso5 #Elimina a variable curso5 se non está declarada en modo lectura

$ declare -p | grep curso5 #Amosa o valor da variable array de nome curso5 e como está declarada

$ read -a curso5 #Recole por teclado unha lista que irá asignando aos índices do array de nome curso5.
1101-1102 1102-1103 1103-1104 #Como o valor de IFS=$'\r\n' e o número de palabras da lista insertada é 3, substitúense os
valores dos índices 0, 1 e 2 por 1101-1102, 1102-1103, 1103-1104 respectivamente

$ declare -p | grep curso5 #Amosa o valor da variable array de nome curso5 e como está declarada

$ read -p 'Escriba valores do array nunha lista: ' -a curso5 #Amosa a mensaxe Escriba valores do array nunha lista: e
recole dende a entrada estándar(teclado) a lista de palabras como índices nun array de nome curso5, empezando por 0. Os
campos separadores das palabras, que serán recollidas nos índices do array, están determinados polo valor da variable separador
de campos $IFS
1 2 3 #Como o valor de IFS=$'\r\n' e o número de palabras da lista insertada é 3, substitúense os valores dos índices 0, 1 e 2 por
estes novos: 1, 2 e 3 respectivamente

$ declare -p | grep curso5 #Amosa o valor da variable array de nome curso5 e como está declarada

$ read curso5[0] #Recole por teclado o valor do índice 0 do array de nome curso5
modificado1 #Asigna o valor modificado1 ao índice 0

$ declare -p | grep curso5 #Amosa o valor da variable array de nome curso5 e como está declarada
```



\$ read -p 'Escriba valor para o índice 2: ' curso5[2] #Recolle por teclado o valor do índice 2 do array de nome **curso5** modificado3 #Asigna o valor **modificado3** ao **índice 2**

\$ declare -p | grep curso5 #Amosa o valor da variable array de **nome curso5** e como está declarada

\$ read -p 'Escriba valor para o índice 1: ' curso5[1] #Recolle por teclado o valor do índice 1 do array de nome **curso5**  
4 5 6 #Asigna o valor **4 5 6** ao **índice 1** xa que na asignación ao índice non se ten en conta a variable **IFS**

\$ declare -p | grep curso5 #Amosa o valor da variable array de **nome curso5** e como está declarada

## Práctica6 arrays: export, shell e subshells

\$ unset curso6 #Elimina a variable **curso6** se non está declarada en modo lectura

\$ declare -p | grep curso6 #Amosa o valor da variable array de **nome curso6** e como está declarada

\$ curso6=('2017-2018' '2018-2019') #Xera un novo array de **nome curso6** onde o **índice 0** e **1**, toman os valores **2017-2018** e **2018-2019** respectivamente.

\$ declare -p | grep curso6 #Amosa o valor da variable array de **nome curso6** e como está declarada

\$ export curso6 #Non exporta a variable array de nome **curso6**.

\$ bash #Executa unha subshell do usuario que executa o comando

\$ declare -p | grep curso6 #Non amosa o valor da variable array de **nome curso6** xa que está por ser vector non puido ser exportada da shell pai

\$ exit #Sae da subshell actual e volta á shell pai

\$ declare -p | grep curso6 #Amosa o valor da variable array de **nome curso6** e como está declarada, e aínda que aparece como exportada por ser vector non é posible.

## Práctica7 arrays: \*, @, " e IFS

\$ unset curso7 #Elimina a variable **curso7** se non está declarada en modo lectura

\$ declare -p | grep curso7 #Amosa o valor da variable array de **nome curso7** e como está declarada

\$ curso7=(A B) #Xera un novo array de **nome curso7** onde o **índice 0** e **1**, toman os valores **A** e **B** respectivamente.

\$ declare -p | grep curso7 #Amosa o valor da variable array de **nome curso7** e como está declarada

\$ set | grep ^IFS #Ver valor variable separador de campos IFS (espazo, tabulado, nova liña)

\$ oldIFS=\$IFS #Gardamos o valor da variable IFS

\$ IFS=, #Modificamos o valor da variable IFS para que o separador de campos sexa soamente o caracter ','

\$ echo \${curso7[\*]} #Amosa os valores de tódolos índices do array separados por espazo

\$ echo \${curso7[@]} #Amosa os valores de tódolos índices do array separados por espazo

\$ echo "\${curso7[\*]}" #Amosa os valores de tódolos índices do array, pero debido ás comillas dobres "\${curso7[\*]}" expande unha soa palabra co valor de cada membro do array separados polo primeiro caracter da variable separador de campos **IFS**, nesta caso o caracter ','

\$ echo "\${curso7[@]}" #Amosa os valores de tódolos índices do array separados por espazo.

\$ IFS=c, #Modificamos o valor da variable IFS para que o separador de campos sexa o caracter 'c' e o caracter ','

\$ echo \${curso7[\*]} #Amosa os valores de tódolos índices do array separados por espazo

\$ echo \${curso7[@]} #Amosa os valores de tódolos índices do array separados por espazo

\$ echo "\${curso7[\*]}" #Amosa os valores de tódolos índices do array, pero debido ás comillas dobres "\${curso7[\*]}" expande unha soa palabra co valor de cada membro do array separados polo primeiro caracter da variable separador de campos **IFS**, nesta caso o caracter 'c'

\$ echo "\${curso7[@]}" #Amosa os valores de tódolos índices do array separados por espazo.

```
$ IFS=$oldIFS #Recuperamos o valor por defecto da variable IFS
```

## Práctica8 arrays: espazos, declarar e percorrer

```
$ unset ficheiros ficheiros2 ficheiros3 ficheiros4 #Elimina as variables ficheiros, ficheiros2, ficheiros3, ficheiros4 se non están declaradas en modo lectura
```

```
$ declare -p | grep ficheiros #Amosa o/s valor/es da/s variable/s que conteña/n no nome o patrón ficheiros e como está/n declarada/s
```

```
$ rm -rf /tmp/arrays #Eliminar /tmp/arrays
```

```
$ mkdir /tmp/arrays && cd /tmp/arrays #Crear o cartafol /tmp/arrays e acceder a este
```

```
$ touch script1.sh 'script 2.sh' #Creamos dous ficheiros baleiros onde o segundo deles contén no nome un caracter espazo
```

```
$ ficheiros=$(ls) #Declaramos e asignamos valor á variable ficheiros, non sendo esta declarada como array
```

```
$ declare -p | grep ficheiros #Amosa o/s valor/es da/s variable/s que conteña/n no nome o patrón ficheiros e como está/n declarada/s
```

```
$ ficheiros2=$(ls) #Declaramos e asignamos valor á variable ficheiros2, sendo esta declarada como array. O array de nome ficheiros2 conterá 3 índices: 0, 1, 2 cuxos valores respectivamente son: script, 2.sh, script1.sh
```

```
$ declare -p | grep ficheiros #Amosa o/s valor/es da/s variable/s que conteña/n no nome o patrón ficheiros e como está/n declarada/s
```

```
$ ficheiros3=(*) #Declaramos e asignamos valor á variable ficheiros3, sendo esta declarada como array. O array de nome ficheiros3 conterá 2 índices: 0 e 1 cuxos valores respectivamente son: script 2.sh e script1.sh
```

```
$ declare -p | grep ficheiros #Amosa o/s valor/es da/s variable/s que conteña/n no nome o patrón ficheiros e como está/n declarada/s
```

```
$ for i in "${ficheiros3[@]}"; do echo $i;sleep 1;done #Amosa os valores de tódolos índices do array, e debido ás comillas dobres "${ficheiros3[@]}" non ten en conta o espazo no índice 2 e amosa 2 liñas, unha por índice, como era o esperado
```

```
$ for i in ${ficheiros3[@]}; do echo $i;sleep 1;done #Amosa os valores de tódolos índices do array, e como non existen as comillas dobres "${ficheiros3[@]}" ten en conta o espazo no índice 2 e amosa 3 liñas, unha máis do esperado
```



### Script 1 (arrays1.sh): Comprobar portos TCP a múltiples hosts

```
1 #!/bin/bash
2
3 ##FUNCTIONS
4 f_port() {
5     for i in "${array_IPs[@]}"
6     do
7         nc -vz $i "${array_ports_TCP[@]}"
8     done
9 }
10
11 ##VARIABLES
12 array_ports_TCP=(21 22 23 80 443 445)
13 array_IPs=('127.0.0.1' 192.168.1.1)
14
15 ##main()
16 f_port
```

### Script 2 (arrays2.sh): Comprobar portos TCP a múltiples hosts soamente se os hosts están activos e indicar que hosts están activos e cales non

```
1 #!/bin/bash
2
3 ##FUNCTIONS
4 f_port() {
5     unset IPs_on
6     unset IPs_off
7     ports_TCP=("${!1}")
8     index=0
9     for i in "${array_IPs[@]}"
10    do
11        ping -c4 $i 2>/dev/null
12        if [ $? -eq 0 ]; then
13            IPs_on[$index]=$i
14        else
15            IPs_off[$index]=$i
16        fi
17        index=$((index+1))
18    done
19    for i in "${IPs_on[@]}"
20    do
21        nc -vz $i "${ports_TCP[@]}"
22    done
23    declare -p IPs_on
24    declare -p IPs_off
25 }
26
27 ##VARIABLES
28 array_ports_TCP=(21 22 23 80 443 445)
29 array_IPs=('127.0.0.1' 192.168.56.101 10.0.2.10)
30
31 ##main()
32 f_port "array_ports_TCP[@]"
```

### Script 3 (arrays3.sh): Comprobar portos TCP a múltiples hosts soamente se os hosts están activos e indicar que hosts e portos están activos e cales non

```
1 #!/bin/bash
2
3 ##FUNCIONS
4 f_port() {
5     clear
6     unset IPs_ports_TCP_on IPs_ports_TCP_off
7     ports_TCP=("${!1}")
8     index_IPs_on=0
9     index_IPs_off=0
10    index_ports_TCP_on=0
11    index_ports_TCP_off=0
12    for i in "${array_IPs[@]}"
13    do
14        ping -c4 $i 1>/dev/null 2>&1
15        if [ $? -eq 0 ]; then
16            IPs_on[${index_IPs_on}]=$i
17            for j in "${ports_TCP[@]}"
18            do
19                nc -vz $i $j 1>/dev/null 2>&1
20                if [ $? -ne 0 ]; then
21                    ports_TCP_off[${index_ports_TCP_off}]=$j
22                    IPs_ports_TCP_off[${index_ports_TCP_off}]="$i $j"
23                    index_ports_TCP_off=$((index_ports_TCP_off+1))
24                else
25                    ports_TCP_on[${index_ports_TCP_on}]=$j
26                    IPs_ports_TCP_on[${index_ports_TCP_on}]="$i $j"
27                    index_ports_TCP_on=$((index_ports_TCP_on+1))
28                fi
29            done
30            index_IPs_on=$((index_IPs_on+1))
31        else
32            IPs_off[${index_IPs_off}]=$i
33            index_IPs_off=$((index_IPs_off+1))
34        fi
35    done
36    echo "#####"
37    echo "IPs on e portos on:"
38    declare -p IPs_ports_TCP_on
39    ##echo "${IPs_ports_TCP_on[@]}"
40    echo
41    echo "IPs on e portos off:"
42    declare -p IPs_ports_TCP_off
43    ##echo "${IPs_ports_TCP_off[@]}"
44    echo
45    echo "IPs off:"
46    declare -p IPs_off
47    ##echo "${IPs_off[@]}"
48    echo "#####"
49 }
50
51 ##VARIABLES
52 array_ports_TCP=(21 22 23 80 443 445)
53 array_IPs=('127.0.0.1' 192.168.56.101 10.0.2.10)
54
55 ##main()
56 f_port "array_ports_TCP[@]"
```

Ricardo Feijoo Costa



This work is licensed under a **Creative Commons Attribution-ShareAlike 4.0 International License**

# Scripts e SHELL BASH: funcións (man bash -> function)

## echo \$SHELL

- Orde de preferencia: 5 tipos

alias --> keywords --> functions --> **builtin** --> file onde:

*alias* --> \$HOME/.bashrc, /etc/bashrc

*keywords* --> palabras clave: function, if, for...

*functions* --> funcións: nome\_funcion() {...}

*builtin* --> comandos internos: cd, type... Son internos a Bash e están sempre cargados na súa memoria.

*file* --> scripts e programas executables (según PATH)

## Práctica

- type <nome\_comando>: amosa información sobre o tipo de comando
- type type: amosa información sobre o comando type
- type -t : amosa o tipo do/s comando/s pasado/s como argumento/s  
\$ type -t function  
keyword
- type -a: amosa tódolos valores nos tipos do/s comando/s pasado/s como argumento/s, é dicir, tódalas posibilidades de execución do/s comando/s.  
\$ type -a function  
function es una palabra clave del shell
- declare -f: amosa as funcións definidas nunha sesión ordeadas alfabeticamente
- declare -F: amosa soamente os nomes das funcións definidas nunha sesión ordeadas alfabeticamente

## Definir funcións:

O estado de saída (\$?) dunha función é o do último comando executado.

Se se executa a orde interna **return** nunha función ésta remata e a execución reanúdase coa seguinte orde tras a chamada á función. Cando unha función complétase, os valores dos parámetros posicionais e o parámetro especial **#** restáurase aos valores que tiñan antes da execución da función.

As funcións poden exportarse, mediante **export -f** de modo que as subshells as teñan definidas automaticamente.

As funcións poden ser recursivas. Non se impón ningún límite no número de chamadas recursivas.

- nomefuncion() { lista; } onde **nomefuncion** é o nome da función e o corpo da función é a lista de comandos dentro das chaves
- function nomefuncion() { lista; } onde **nomefuncion** é o nome da función e o corpo da función é a lista de comandos dentro das chaves
- function nomefuncion { lista; } onde **nomefuncion** é o nome da función e o corpo da función é a lista de comandos dentro das chaves

## Eliminar funcións:

- unset -f nomefuncion #Elimina a función definida co nome **nomefuncion**

## Executar funcións

- nomefuncion [\$1 \$2 \$3 ...] onde **nomefuncion** é o nome da función, onde [\$1 \$2 \$3 ...] son opcionais e son os argumentos, como cando executamos un comando, os cales actúan como parámetros da función na chamada á función

## Trap e funcións

- `function nomefuncion() {  
trap 'comando' sinal1 [sinal2] ... [sinalN] #Capturar sinais e definir como acción un comando, unha lista de comandos ou unha función  
}`  
As funcións execútanse no mesmo proceso que o script que as chama, co cal un **trap** definido dentro dunha función segue activo no script cando a función remata, polo que deberemos devolver ao estado inicial o **trap** para impedir que a súa execución permaneza vixente no resto do script.

### Práctica1 funcións: Definir, eliminar, executar

```
$ declare -f #Amosa as funcións definidas nesta shell ordeadas alfabeticamente

$ declare -f | less #Amosa as funcións definidas nesta shell ordeadas alfabeticamente e filtradas co comando less

$ declare -f f_listar1 #Amosa se existe a función(nome e corpo) de nome f_listar1

$ declare -F #Amosa soamente o nome e como están declaradas as funcións definidas nesta shell ordeadas alfabeticamente

$ declare -F | less #Amosa soamente o nomee como están declaradas as funcións definidas nesta shell ordeadas alfabeticamente e filtradas co comando less

$ declare -F f_listar1 #Amosa se existe soamente o nome da función de nome f_listar1

$ function f_listar1() {ls;id;} #Erro xa que debe existir separación entre a chave inicial e os comandos no corpo da función:
bash: error sintáctico cerca del elemento inesperado `{ls'

$ function f_listar1() { ls;id;} #Definición da función de nome f_listar1

$ declare -f f_listar1 #Amosa a función(nome e corpo) de nome f_listar1

$ function f_listar1() { #Definición da función de nome f_listar1
ls
id
} #Fin do corpo da función

$ declare -f f_listar1 #Amosa a función(nome e corpo) de nome f_listar1

$ declare -f f_listar1() #Erro: bash: error sintáctico cerca del elemento inesperado `('

$ declare -F f_listar1 #Amosa soamente o nome da función de nome f_listar1

$ declare -F f_listar1() #Erro: bash: error sintáctico cerca del elemento inesperado `('

$ f_listar1 #Execución da función definida como f_listar1

$ f_listar1() #Ao premer <Enter> amosa o prompt PS2 á espera de seguir definindo a función de nome f_listar1
> ^C

$ unset -f f_listar1 #Elimina a función definida co nome f_listar1

$ declare -f f_listar1 #Non amosa nada xa que a función de nome f_listar1 xa non está definida

$ declare -F f_listar1 #Non amosa nada xa que a función de nome f_listar1 xa non está definida

$ f_listar1 #Erro: bash: f_listar1: no se encontró la orden
```

## Práctica2 funcións: preferencias execución

```
$ unset -f f_listar1 #Elimina a función definida co nome f_listar1 no caso que exista

$ f_listar1() { #Definición da función de nome f_listar1
ls
id
} #Fin do corpo da función

$ alias f_listar1='cd /tmp' # Crea o alias f_listar1 que executará o comando cd /tmp

$ f_listar1 #Executa o alias f_listar1, é dicir, executa o comando cd /tmp

$ f_listar1() #Erro: bash: error sintáctico cerca del elemento inesperado `('

$ unalias f_listar1 #Elimina o alias f_listar1

$ f_listar1 #Execución da función definida como f_listar1

$ f_listar1() #Ao premer <Enter> amosa o prompt PS2 á espera de seguir definindo a función de nome f_listar1
> { #Comeza o corpo da función
> who #Comando pertencente á función que amosa todos os usuarios logueados no sistema (Logs de /var/run/utmp)
> } #Finaliza o corpo da función

$ f_listar1 #Execución da función definida como f_listar1, é dicir, amosar os usuarios logueados no sistema
```

## Práctica3 funcións: exportar, shell e subshells

```
$ unset -f f_listar1 #Elimina a función definida co nome f_listar1 no caso que exista

$ f_listar1() { #Definición da función de nome f_listar1
ls
id
} #Fin do corpo da función

$ declare -f f_listar1 #Amosa a función(nome e corpo) de nome f_listar1

$ declare -F | grep f_listar1 #Amosa como está declarada a función de nome f_listar1

$ bash #Executa unha subshell do usuario que executa o comando

$ declare -f f_listar1 #Debido a que a función non foi exportada, non se herda na subshell e polo tanto non amosa a
función(nome e corpo) de nome f_listar1

$ exit #Sae da subshell actual e volta á shell pai

$ declare -f f_listar1 #Amosa a función(nome e corpo) de nome f_listar1

$ declare -F | grep f_listar1 #Amosa como está declarada a función de nome f_listar1

$ export -f f_listar1 #Exporta a función de de nome f_listar1

$ declare -F | grep f_listar1 #Amosa como está declarada a función de nome f_listar1

$ f_listar1 #Execución da función definida como f_listar1

$ bash #Executa unha subshell do usuario que executa o comando

$ declare -f f_listar1 #Debido a que a función está exportada, hérdase a subshell e polo tanto amosa a función(nome e corpo)
de nome f_listar1

$ f_listar1 #Execución da función definida como f_listar1

$ exit #Sae da subshell actual e volta á shell pai

$ bash #Executa unha subshell do usuario que executa o comando

$ unset -f f_listar1 #Elimina a función definida co nome f_listar1 no caso que exista

$ declare -f f_listar1 #Non amosa nada xa que a función de nome f_listar1 xa non está definida
```

```
$ declare -F f_listar1 #Non amosa nada xa que a función de nome f_listar1 xa non está definida

$ f_listar1 #Erro: bash: f_listar1: no se encontró la orden

$ exit #Sae da subshell actual e volta á shell pai

$ declare -f f_listar1 #Amosa a función(nome e corpo) de nome f_listar1

$ f_listar1 #Execución da función definida como f_listar1
```

#### Práctica4 funcións: \$?

```
$ unset -f f_listar1 #Elimina a función definida co nome f_listar1 no caso que exista

$ f_listar1() { who;ls;} #Definición da función de nome f_listar1

$ declare -f f_listar1 #Amosa a función(nome e corpo) de nome f_listar1

$ f_listar1 #Execución da función definida como f_listar1

$ echo $? #Amosa o valor do estado do último comando executado na función definida como f_listar1
0

$ f_listar1() { who;ls;lskdfjlsakdfj;} #Definición da función de nome f_listar1

$ declare -f f_listar1 #Amosa a función(nome e corpo) de nome f_listar1

$ f_listar1 #Execución da función definida como f_listar1

$ echo $? #Amosa o valor do estado do último comando executado na función definida como f_listar1
127

$ f_listar1() { who;ls;lskdfjlsakdfj;id;} #Definición da función de nome f_listar1

$ declare -f f_listar1 #Amosa a función(nome e corpo) de nome f_listar1

$ f_listar1 #Execución da función definida como f_listar1

$ echo $? #Amosa o valor do estado do último comando executado na función definida como f_listar1
0
```



## trap-in-function.sh

```
1 #!/bin/bash
2
3 #Atrapar sinais:
4
5 clear
6
7 function f_trap(){
8     ## Ignorar <Ctrl>+<c>: SIGGINT (2)
9     trap '' 2
10 }
11
12 f_trap
13 echo '-----'
14 echo 'Ignorando <Ctrl>+<c>. Inténtelo!'
15 echo '-----'
16 for i in $(seq 1 15)
17 do
18     echo $i
19     sleep 1
20 done
```

## trap-out-function.sh

```
1 #!/bin/bash
2
3 #Atrapar sinais:
4
5 clear
6
7 ## Ignorar <Ctrl>+<c>: SIGGINT (2)
8 trap '' 2
9
10 function f_trap(){
11     echo '-----'
12     echo 'Ignorando <Ctrl>+<c>. Inténtelo!'
13     echo '-----'
14     for i in $(seq 1 15)
15     do
16         echo $i
17         sleep 1
18     done
19 }
20
21 f_trap
```

### Script 1 (number1.sh):

Que acontece cando executamos o script e introducimos caracteres que non son números?

No Script 1 non se cumpren as 2 primeiras condicións por erro de comparación de tipos de variables, co cal ademais dos erros ao final amósase **PARABÉNS!!! Adiviñaches o número**. Como se podería solucionar ese problema? Ver Script 2

```
1 #!/bin/bash
2
3 declare -r NUMBER=$((1+RANDOM%999))
4 declare -i NUMBER
5
6 function f_read_number() {
7     read -p 'Adiviña o número de 3 cifras: ' -n3 number
8     declare -i number
9 }
10
11
12 function f_procure() {
13     if [ $number -gt $NUMBER ] ; then
14         echo -e '\n0 número é menor'
15         f_read_number
16     else
17         if [ $number -lt $NUMBER ] ; then
18             echo -e '\n0 número é maior'
19             f_read_number
20         else
21             echo -e '\nPARABÉNS!!! Adiviñaches o número'
22             exit
23         fi
24     fi
25 }
26 f_procure
27
28
29 f_main() {
30     clear
31     echo -e "\n $NUMBER $number"
32     f_read_number
33     f_procure
34 }
35
36 f_main
```

## Script 2 (number2.sh)

No Script 2 recóllese o valor do estado do comando executado, \$?, e faise unha condición que permita soamente pedir números. Ademais para que non amose erro por pantalla empregamos 2>/dev/null

Como se podería modificar esta solución para que se contabilicen os intentos de acerto? Ver Script 3

```
1 #!/bin/bash
2
3 declare -r NUMBER=$((1+RANDOM%999))
4 declare -i NUMBER
5
6 function f_read_number() {
7     read -p 'Adiviña o número de 3 cifras: ' -n3 number
8     declare -i number
9 }
10
11
12 function f_procure() {
13     if [ $number -gt $NUMBER ] 2>/dev/null; then
14         #echo -e "\naaa $?"
15         echo -e '\n0 número é menor'
16         f_read_number
17     else
18         if [ $number -lt $NUMBER ] 2>/dev/null; then
19             #echo -e "\nbbaa $?"
20             echo -e '\n0 número é maior'
21             f_read_number
22         else
23             #echo -e "\ncbaa $?"
24             if [ $? -ne 2 ]; then
25                 echo -e '\nPARABÉNS!!! Adiviñaches o número'
26                 exit
27             else
28                 echo -e '\nDebe introducir un número'
29                 f_read_number
30             fi
31         fi
32     fi
33 }
34
35
36
37 f_main() {
38     clear
39     echo -e "\n $NUMBER $number"
40     f_read_number
41     f_procure
42 }
43
44 f_main
```

### Script 3 (number3.sh)

Como se podría modificar esta solución para ter un tempo límite de acerto? Ver Script 4

```
1 #!/bin/bash
2
3 declare -r NUMBER=$((1+RANDOM%999))
4 declare -i NUMBER
5 cont=0
6
7 function f_read_number() {
8     read -p 'Adiviña o número de 3 cifras: ' -n3 number
9     declare -i number
10    cont=$((cont+1))
11 }
12
13
14 function f_procure() {
15     if [ $number -gt $NUMBER ] 2>/dev/null; then
16         #echo -e "\naaa $?"
17         echo -e '\n0 número é menor'
18         f_read_number
19     else
20         if [ $number -lt $NUMBER ] 2>/dev/null; then
21             #echo -e "\nbbb $?"
22             echo -e '\n0 número é maior'
23             f_read_number
24         else
25             #echo -e "\nccc $?"
26             if [ $? -ne 2 ]; then
27                 echo -e "\nPARABÉNS!!! Adiviñaches o número en $cont intentos"
28                 exit
29             else
30                 echo -e '\nDebe introducir un número'
31                 f_read_number
32             fi
33         fi
34     fi
35 }
36 f_procure
37
38
39 f_main() {
40     clear
41     echo -e "\n $NUMBER $number"
42     f_read_number
43     f_procure
44 }
45
46 f_main
```

#### Script 4 (number4.sh)

Como se podría modificar esta solución para que o tempo máximo de acerto sexa 120 segundos e que cada intento reste 5segundos do tempo dispoñible? Ver Script 5

```
1 #!/bin/bash
2
3 ##man tput && man terminfo
4
5 PID=$$
6
7 declare -r NUMBER=$((1+RANDOM%999))
8 declare -i NUMBER
9 cont=0
10 declare -r tinicio=$(date +%s)
11 declare -r tlimite=30
12
13 bash cronometro2.sh $(date +%s) $tlimite $PID &
14
15 function f_read_number() {
16     read -p 'Adiviña o número de 3 cifras: ' -n3 number
17     declare -i number
18     cont=$((cont+1))
19 }
20
21
22 function f_procedure() {
23     CHRONEPID=$(pgrep -f cronometro2)
24     texecucion=$((`date +%s` - tinicio))
25     while [ $tlimite -gt $texecucion ]
26     do
27         echo -en "\t $(date -u --date @$(($tlimite - $texecucion)) +%M:%S)\r"
28         if [ $number -gt $NUMBER ] 2>/dev/null; then
29             #echo -e "\naaa $"
30             echo -e '\n0 número é menor'
31             f_read_number
32         else
33             if [ $number -lt $NUMBER ] 2>/dev/null; then
34                 #echo -e "\nbba $"
35                 echo -e '\n0 número é maior'
36                 f_read_number
37             else
38                 #echo -e "\ncba $"
39                 if [ $? -ne 2 ]; then
40                     echo -e "\nPARABÉNS!!! Adiviñaches o número en $cont intentos e $texecucion segundos"
41                     tput sc
42                     COLUMNS=$((`tput cols` - 5))
43                     tput cup 0 $COLUMNS
44                     COLOR_ORIGINAL=$(tput rmso)
45                     echo -n "${COLOR_ORIGINAL}      ${COLOR_ORIGINAL}"
46                     tput rc
47                     kill $CHRONEPID 1>/dev/null 2>&1
48                     exit
49                 else
50                     echo -e "\nDebe introducir un número"
51                     f_read_number
52                 fi
53             fi
54         fi
55     done
56     texecucion=$((`date +%s` - tinicio))
57     echo -e "\nRematouse o tempo!!!-----"
58     exit
59 }
60
61
62
63 f_main() {
64     clear
65     ## echo -e "\n $NUMBER $number"
66     f_read_number
67     f_procedure
68 }
69
70 f_main
```

## cronometro1.sh

```
1 #!/bin/bash
2
3 declare -r tinicio=$(date +%s)
4 declare -r tlimite=30
5
6 tagora=$(date +%s)
7 texecucion=$((tagora-tinicio))
8 while [ $tlimite -gt $texecucion ]
9 do
10 tagora=$(date +%s)
11 texecucion=$((tagora-tinicio))
12 echo -en "\t $(date -u --date @$(($tlimite - $texecucion)) +%M:%S)\r"
13 done
```

## cronometro2.sh

```
1 #!/bin/bash
2
3 tinicio=$1
4 tlimite=$2
5 FATHERPID=$3
6 while [ $tlimite -gt $((`date +%s`-$tinicio)) ]
7 do
8
9 tput sc
10
11 DATA=$(date -u --date @$(($tlimite - $((`date +%s`-$tinicio)))) +%M:%S)
12 COLUMNS=$((`tput cols` - 5))
13 tput cup 0 $COLUMNS
14
15 COLOR_CHANGE=$(tput smso)
16 COLOR_ORIGINAL=$(tput rmso)
17 echo -n "${COLOR_CHANGE}$DATA${COLOR_ORIGINAL}"
18
19 tput rc
20
21 sleep 1
22 done
23 tput sc
24 tput cup 0 $COLUMNS
25 echo -n "${COLOR_ORIGINAL}      ${COLOR_ORIGINAL}"
26 tput rc
27 kill $FATHERPID 1>/dev/null 2>&1
28 echo -e "\nRematouse o tempo!!!-----"
```



### Script 5 (number5.sh)

Como se podería modificar esta solución para que na mensaxe final de acerto apareza tamén o tempo de penalización? Ver SScript 6

```
1 #!/bin/bash
2
3 declare -r NUMBER=$((1+RANDOM%999))
4 declare -i NUMBER
5 cont=0
6 declare -r tinicio=$(date +%s)
7 ##declare -r tlimite=30
8 declare tlimite=120
9
10 function f_read_number() {
11     read -p 'Adiviña o número de 3 cifras: ' -n3 number
12     declare -i number
13     cont=$((cont+1))
14 }
15
16
17 function f_procure() {
18 texecucion=$((`date +%s` - tinicio))
19 while [ $tlimite -gt $texecucion ]
20 do
21     echo -en "\t $(date -u --date @$((($tlimite - $texecucion)) +%M:%S)\r"
22     if [ $number -gt $NUMBER ] 2>/dev/null; then
23         #echo -e "\naaa $"
24         echo -e '\n0 número é menor'
25         tlimite=$((tlimite-5))
26         f_read_number
27     else
28         if [ $number -lt $NUMBER ] 2>/dev/null; then
29             #echo -e "\nbba $"
30             echo -e '\n0 número é maior'
31             tlimite=$((tlimite-5))
32             f_read_number
33         else
34             #echo -e "\ncba $"
35             if [ $? -ne 2 ]; then
36                 echo -e "\nPARABÉNS!!! Adiviñaches o número en $cont intentos e $texecucion segundos"
37                 exit
38             else
39                 echo -e "\nDebe introducir un número"
40                 tlimite=$((tlimite-5))
41                 f_read_number
42             fi
43         fi
44     fi
45 ## f_procure
46 texecucion=$((`date +%s` - tinicio))
47 done
48 echo -e "\nRematouse o tempo!!!-----"
49 exit
50 }
51
52
53 f_main() {
54     clear
55     echo -e "\n $NUMBER $number"
56     f_read_number
57     f_procure
58 }
59
60 f_main
```

## Script 6 (number6.sh)

```
1 #!/bin/bash
2
3 declare -r NUMBER=$((1+RANDOM%999))
4 declare -i NUMBER
5 cont=0
6 declare -r tinicio=$(date +%s)
7 ##declare -r tlimite=30
8 declare tlimite=120
9 cont_penaliz=0
10
11
12
13 function f_read_number() {
14     read -p 'Adiviña o número de 3 cifras: ' -n3 number
15     declare -i number
16     cont=$((cont+1))
17 }
18
19
20
21 function f_procure() {
22     texecucion=$((`date +%s` - tinicio))
23     while [ $tlimite -gt $texecucion ]
24     do
25         echo -en "\t $(date -u --date @(($tlimite - $texecucion)) +%M:%S)\r"
26         if [ $number -gt $NUMBER ] 2>/dev/null; then
27             #echo -e "\naaa $?"
28             echo -e '\n0 número é menor'
29             tlimite=$((tlimite-5))
30             cont_penaliz=$((cont_penaliz+1))
31             f_read_number
32         else
33             if [ $number -lt $NUMBER ] 2>/dev/null; then
34                 #echo -e "\nbba $?"
35                 echo -e '\n0 número é maior'
36                 tlimite=$((tlimite-5))
37                 cont_penaliz=$((cont_penaliz+1))
38                 f_read_number
39             else
40                 #echo -e "\ncba $?"
41                 if [ $? -ne 2 ]; then
42                     tpenaliz=$((cont_penaliz*5))
43                     echo -e "\nPARABÉNS!!! Adiviñaches o número en $cont intentos e $texecucion segundos, sendo o tempo de
penalización $tpenaliz"
44                     exit
45                 else
46                     echo -e "\nDebe introducir un número"
47                     tlimite=$((tlimite-5))
48                     cont_penaliz=$((cont_penaliz+1))
49                     f_read_number
50                 fi
51             fi
52         fi
53     done
54     texecucion=$((`date +%s` - tinicio))
55     echo -e "\nRematouse o tempo!!!-----"
56     exit
57 }
58
59
60
61
62 f_main() {
63     clear
64     echo -e "\n $NUMBER $number"
65     f_read_number
66     f_procure
67 }
68
69
70
71 f_main
```

Ricardo Feijoo Costa



This work is licensed under a **Creative Commons Attribution-ShareAlike 4.0 International License**

# Estruturas

## Percorrer ficheiros

```
while read LINE
do
...
done < file
```

```
cat file | while read LINE
do
...
done
```

```
#!/bin/bash

while read LINE
do
    USUARIO=$(echo ${LINE} | cut -d ':' -f1)
    USUARIO_ID=$(echo ${LINE} | cut -d ':' -f3)
    GRUPO_ID=$(echo ${LINE} | cut -d ':' -f4)
    DIRECTORIO=$(echo ${LINE} | cut -d ':' -f6)
    CONSOLA=$(echo ${LINE} | cut -d ':' -f7)
    echo -e "USER=${USUARIO}\tUID=${USUARIO_ID}\tGID=${GRUPO_ID}\tHOME=${DIRECTORIO}\tSHELL=${CONSOLA}"
    sleep 1
done < /etc/passwd
```

## Control de fluxo

### if then fi

```
if test CONDICION;then ...;fi
equivale a
if [ CONDICION ];then ...;fi
equivale a
[ CONDICION ] && comando
```

### if then else fi

```
if test CONDICION;then ...;else ...;fi
equivale a
if [ CONDICION ];then ...;else ...;fi
equivale a
[ CONDICION ] && comando1 || comando2
```

```
#!/bin/bash
##-----
if test -f /etc/passwd; then
    echo Existe
fi
##-----
if [ -f /etc/passwd ]; then
    echo Existe
fi
##-----
[ -f /etc/passwd ] && echo Existe
```

```
#!/bin/bash
##-----
if test ! -f /etc/passwd; then
    echo Non existe
else
    echo Existe
fi
##-----
if [ ! -f /etc/passwd ]; then
    echo Non existe
else
    echo Existe
fi
##-----
[ ! -f /etc/passwd ] && echo Non existe || echo Existe
```

```
#!/bin/bash
##-----
CADEA='string'
##-----
if test -n ${CADEA}; then
    echo Lonx. non nula
fi
##-----
if [ -n ${CADEA} ]; then
    echo Lonx. non nula
fi
##-----
[ -n ${CADEA} ] && echo Lonx. non nula
```

```
#!/bin/bash
##-----
CADEA='string'
##-----
if test ! -z ${CADEA}; then
    echo Lonx. non nula
else
    echo Lonx. nula
fi
##-----
if [ ! -z ${CADEA} ]; then
    echo Lonx. non nula
else
    echo Lonx. nula
fi
##-----
[ ! -z ${CADEA} ] && echo Lonx. non nula || echo Lonx. nula
```

## Menú

### echo read case ... esac

```
echo Opción 1...
echo Opción 2...
echo Opción 3...
read -p 'Elixe opción: 1,2,3? ' opcion
case $opcion in
    1) comandos
    ;;
    2) comandos
    ;;
    3) comandos
    ;;
    *) echo Opción non correcta
    ;;
esac
```

### select do case ... esac done

```
PS3='Elixe opción:1,2,3? '
OPCION1='Texto1'
OPCION2='Texto2'
OPCION3='Texto3'
select opcion in "${OPCION1}" "${OPCION2}" "${OPCION3}"
do
    case ${opcion} in
        ${OPCION1}) comandos
        ;;
        ${OPCION2}) comandos
        ;;
        ${OPCION3}) comandos
        ;;
        *) echo Opción non correcta
        ;;
    esac
done
```

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

```
echo Opcion1. Ver directorio actual
echo Opcion2. Ler /tmp
echo Opcion3. Sair
read -p 'Elixe opción: 1,2,3? ' opcion
case $opcion in
    1) pwd
    ;;
    2) ls
    ;;
    3) exit
    ;;
    *) echo Non elixiches nin 1,2,3
    ;;
esac
```

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

```
PS3='Elixe opción:1,2,3? '
OPCION1='Ver directorio actual'
OPCION2='Ler /tmp'
OPCION3='Saír'

select opcion in "${OPCION1}" "${OPCION2}" "${OPCION3}"
do
    case ${opcion} in
        ${OPCION1}) pwd
        ;;
        ${OPCION2}) ls
        ;;
        ${OPCION3}) exit
        ;;
        *) echo Non elixiches nin 1,2,3
        ;;
    esac
done
```

### array: select do case esac done

```
PS3='Opción? '
options=("Texto1" "Texto2" "Texto3")
select opcion in "${options[@]}"
do
    case $opcion in
        "Texto1") comandos
        ;;
        "Texto2") comandos
        ;;
        "Texto3") break
        ;;
        *) echo "Non elixiches ningunha opción válida"
        ;;
    esac
done
```

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

```
PS3='Opción? '
options=("Ver directorio actual" "Ler /tmp" "Saír")
select opcion in "${options[@]}"
do
    case $opcion in
        "Ver directorio actual") pwd
        ;;
        "Ler /tmp") ls /tmp
        ;;
        "Saír") break
        ;;
        *) echo "Non elixiches ningunha opción válida";;
    esac
done
```

# Funcións

## Definir e Invocar

```
function f_name() {
  comandos
}

f_name
```

## Menú

```
function f_op1() { comandos;}
function f_op2() { comandos;}
function f_menu(){
  echo Opcion1. Texto1
  echo Opcion2. Texto1
  read -p 'Elixex opcion:1,2? ' opcion
  case $opcion in
    1) comandos;;
    2) comandos;;
    *) echo Opción non correcta && f_menu;;
  ;;
  esac
}
function f_main(){
  f_menu
}
f_main
```

```
#!/bin/bash

function f_suma() {
  read -p 'Introduce número: ' n1
  read -p 'Outro número: ' n2
  echo Suma: $n1 + $n2 = $((n1+n2))
}

f_suma
```

```
#!/bin/bash

function f_op1(){ pwd;}
function f_op2(){ ls /tmp;}
function f_op3(){ exit;}

function f_menu() {
  echo Opcion1. Ver directorio actual
  echo Opcion2. Ler /tmp
  echo Opcion3. Sair
  read -p 'Elixex opcion:1,2,3? ' opcion
  case $opcion in
    1) f_op1;;
    2) f_op2;;
    3) f_op3;;
    *) echo Non elixiches nin 1,2,3 && f_menu;;
  esac
}

function f_main() {
  f_menu
}

f_main
```

## Invocar parámetros

```
function f_help() {
  echo "Exemplo execución: $1 $2" && exit
}
function f_parametros() {
  [ $# -ne 2 ] && f_help
}
function f_main() {
  f_parametros $*
}
f_main $*
```

```
#!/bin/bash

function f_help() {
  echo "Exemplo execución: 4 17"
  exit
}
function f_parametros() {
  [ $# -ne 2 ] && f_help
}
function f_suma() {
  echo A suma de $1 + $2 é: $((n1+n2))
}
function f_main() {
  f_parametros $*
  f_suma $*
}

f_main $*
```

## Errorlevel: \$?

```
comando
[ $? -eq 0 ] && echo OK || echo KO
```

```
#!/bin/bash

ping -c2 127.0.0.1
[ $? -eq 0 ] && echo OK || echo KO

nc -vz 127.0.0.1 80
[ $? -eq 0 ] && echo OK || echo KO

read -p 'Número: ' n1
[[ $n1 =~ ^-[0-9]+$ ]] && echo SI || echo NON
```

## Arrays

```
array_name=(1 2 3)
for i in "${array_name[@]}"
do
    echo $i
done
```

```
#!/bin/bash

array_ports_TCP=(21 22 23 80 443 445)
array_IPs=(127.0.0.1 127.127.127.127)

function f_port() {
    for i in "${array_IPs[@]}"
    do
        nc -vz $i "${array_ports_TCP[@]}"
    done
}

f port
```

## Contadores

```
for i in $(seq 1 10)
do
...
done
```

```
for ((i=1;i<=10;i++))
do
...
done
```

```
#!/bin/bash

for i in $(seq 1 10)
do
    echo Valor de i: $i
done
```

```
#!/bin/bash

for ((i=1;i<=10;i++))
do
    echo Valor de i: $i
done
```