Scripts i control de flux en bash. Funcions.

# Índex

[Índex](#_vfpmjffvx8b8)

[Scripts de bash](#_4iq2vfnoamb9)

[Edició](#_ik92u2kuhbp)

[Indicador del shell d’execució per de l’script (Shebang)](#_ff2u95jah15v)

[Shebang](#_fqwtmcqcl1di)

[Scripting](#_364dtiv1ipxi)

[Observació](#_uvr9n0h054jf)

[Comentari preliminar de l’script](#_62g41dtdeaxi)

[Exemple](#_qkjh2xwbqq5w)

[Execució](#_b7020eoe4cl0)

[Observació](#_24yvawo4cnuw)

[Permís d’execució](#_br4i2yeurxyo)

[Els arguments](#_wm2dnnpklpcr)

[Redefinició dels arguments](#_ik9sxcqln9ud)

[Desplaçament dels arguments](#_vra50wl4obkz)

[Finalització i codi d’error de retorn](#_q1fw2otgyfja)

[Codi d’error de sortida](#_wrtldgcb86tz)

[Entrada de dades: read](#_yphz9wwu6pcy)

[Control de flux](#_yfax71aq3hl)

[Condicionals](#_12iyrb6xsq2h)

[construcció condicional: if](#_4hghk7mv9ykl)

[Exemple](#_73uij2ecfypl)

[Avaluació múltiple: case](#_9c0tbmnz1zqb)

[selecció per número entre un conjunt d’opcions: select](#_xpd00rahin4e)

[Bucles](#_os65f2k7s6df)

[Bucle for](#_lzbcebigze4i)

[Una variable (de tipus array o diccionari)](#_q9z6wr6bvehr)

[Llista implicita (arguments de l‘script)](#_7ftzodmyooyr)

[Llista d’elements explícita](#_fyfgkvyucyk6)

[Criteri per a noms d’arxiu](#_fyfgkvyucyk6)

[Interval de valors](#_2ln97g8rgnkd)

[Fent servir una substitució d’una ordre](#_fyfgkvyucyk6)

[Bucle while](#_khhx6futcxdc)

[Bucle until](#_khhx6futcxdc)

[Ordres break i continue](#_apm9zhrv7akf)

[break](#_vmofxssrbo75)

[continue](#_ntn7ilxmkfbp)

[Funcions](#_nqw6t4cyfyw0)

[Declaració de funcions:](#_r9irbt7nlbw1)

# Scripts de bash

## Edició

### Indicador del shell d’execució per de l’script (Shebang)

Tot script ha de començar amb una línia que comença amb els símbols **#!** indicant a continuació el shell que ha d’executar l’script, que en el nostre cas serà **/bin/bash**

**#!/bin/bash**

#### Shebang

El prefix ***#!*** s’anomena ***shebang*** o també ***hash-bang***, i és part de la tècnica del *magic-number* de Linux (Unix), i en aquest cas indica que és un script, i a continuació s’indica el programa que executa l’script.

### Scripting

Un script de bash consisteix en ***línies que s’executen en seqüència***. També sovint trobarem definicions de funcions en els arxius de script.

Ja s’ha vist que es pot escriure en una única línia una seqüència d’ordres que s’executen seqüencialment en ordre, separant-les amb punt i coma (***;***).

També hem vist l’execució en seqüència condicional, fent servir el doble ampersand (***&&***) i la doble barra recta (***||***).

Existeixen diverses ordres que permeten el control del flux de la seqüència i que implementen bucles i condicionals. Aquestes ordres són més pensades per l'execució en scripts, tot i que sempre es poden fer servir en una mateixa línia.

#### Observació

També podem fer servir **/bin/sh** amb resultats molt similars, però quan es crida a ***bash*** com a ***sh*** s’activa el mode de compatibilitat **POSIX**, i algunes característiques canvien.

### Comentari preliminar de l’script

A continuació de la línia ***#!/bin/bash*** que determina el shell que ha d’executar l’script, és una pràctica comú incorporar comentaris, indicant:

* El nom de l’script
* Un comentari detallant la funcionalitat general de l’script.
  + La sintaxi: paràmetres i opcions que s’accepten.
  + Si es requereix detallar les funcionalitats de les diferents opcions, es recomana incorporar l’opció --help
* També aquí, si escau, es fa constar l’autor de l’script i data de creació.
  + Quan n’hi ha el nom de l’autor i la data, es fa constar a sota qualsevol modificació fent servir el mateix format.

Aquests comentaris són opcionals, però **és un requisit bàsic de bones pràctiques**.

#### Exemple

Copiem directament l’script des del terminal; recordar que per acabar cal fer servir ^D (CTRL-D), que assenyala el final del fitxer.

[cesc@Skye a]$ cat > p1a.sh

#!/bin/bash

#

# p1a.sh

#

# El meu primer script

#

echo 'El meu primer script'

[cesc@Skye a]$

## Execució

Els scripts s’han d’executar cridant-los des de bash:

[cesc@Skye a]$ bash p1a.sh

El meu primer script

Sembla obvi que estem cridant a una nova instància de bash, i per tant s’executa en un subshell. Podem requerir que l’script s’executi en el shell actual, fent servir el punt:

[cesc@Skye a]$ . p1a.sh

El meu primer script

Per veure la diferència anem a incorporar en el text que escriu l’script, l’identificador de procés del shell que l’executa. Editem l’script amb l’editor **nano** (no es detallen les accions dins de l’editor):

[cesc@Skye a]$ nano p1a.sh

#!/bin/bash

#

# p1a.sh

#

# El meu primer script

#

echo 'El meu primer script: ' $$

[cesc@Skye a]$ bash p1a.sh //*Per executar un script o bash*

El meu primer script: 13676

[cesc@Skye a]$ . p1a.sh *// o un punt i l’script*

El meu primer script: 11382

[cesc@Skye a]$ echo $$

11382

Efectivament, quan l’executem fent servir el punt, és el mateix identificador de procés del shell en què ens trobem. D’aquesta manera l’script pot modificar variables de manera visible pel shell actual.

### Observació

També podem fer servir ***sh*** per executar l’script, amb resultats molt similars, però quan es crida a ***bash*** com a ***sh*** s’activa el mode de compatibilitat **POSIX**, i algunes característiques canvien.

### Permís d’execució

Un script és un arxiu que es pot executar; si li donem permís d’execució el podem executar directament sense necessitat de executar explícitament bash i fer servir l’script com a paràmetre.

Amb aquesta ordre afegim permisos d’execució:

[cesc@Skye a]$ chmod +x p1a.sh

Ara podem executar l’script, només que, com la nostra carpeta no hi és en el $PATH, haurem d’indicar el camí explícitament, bé sigui absolut o relatiu:

[cesc@Skye a]$ ./p1a.sh

El meu primer script: 15610

[cesc@Skye a]$ echo $$

11382

Una opció seria incorporar el directori en el $PATH; generalment farem servir un directori comú per als scripts que volem siguin disponibles, i l’afegirem en el $PATH:

* ${HOME}/.local/bin

Per als scripts de l’usuari, que no comparteix.

* /usr/local/bin

Per als scripts compartits per als usuaris de la màquina local. Aquest directori, en les diferents distribucions de Linux, generalment hi és buit.

Observem, en l’execució d’exemple de dalt, que el ***PID*** (*process identifier*) és diferent (ens informa del valor 15610, de mentre que el shell des d’on executem veiem que té el 11382), i per tant també en aquest cas s’executa en un subshell.

*// Avantatges: no té efectos sobre el nostre shell. relativament protegits.*

## Els arguments

Els arguments en un script són posicionals; és a dir, tindrem el primer argument, el segon argument…

| $0 | Nom del script |
| --- | --- |
| $1-9 | $1, $2, $3 … Els nou primers paràmetres rebuts. |
| $# | Número total de paràmetres. |
| $\* | Llista de tots els paràmetres concatenats i separats amb espais. |
| $@ | Tots els paràmetres, com a vector (*array*) |

### Redefinició dels arguments

Ampliació: característica poc usual.

L’ordre **set** permet redefinir tots els arguments, redefinit també el valor de **$#** en conseqüència. Així, la següent ordre defineix tots els arguments, que seran quatre després d’executar-se, siguessin quants i quins siguessin en número i valor els arguments originals:

**set** *valor1 valor2 valor2 valor4*

### Desplaçament dels arguments

Ampliació: característica avançada, per a scripts molt llargs i amb moltes opcions

L’ordre **shift** suprimeix el primer argument i desplaça la resta d’arguments en una posició; així el **$2** serà el **$1**, **$3** el **$2** i així successivament.

També es pot fer servir un argument per a **shift**, fent un desplaçament de més posicions; així **shift 4** fa que **$5** sigui **$1**, **$6** serà **$2**, i així successivament.

## Finalització i codi d’error de retorn

L’ordre **exit** produeix la sortida del subshell que executa l’script.

### Codi d’error de sortida

L’ordre exit incorpora un paràmetre, que serà el valor de retorn. Recordem els diferents valors de retorn ja comentats:

* El paràmetre per a **exit** ha de ser un **número enter**; en cas contrari retornarà el codi d’error 128.
* El paràmetre per a **exit** ha de ser un número enter **positiu i menor o igual a 255**; en cas contrari retornarà el codi d’error 255.
* **Retornar zero** per a una execució correcta.
* La conveniència de fer servir el **valor de retorn 2** quan no es compleixen les precondicions, excepte si es vol retornar un codi personalitzat.
  + Els codis personalitzats per fer servir es troben entre el 3 i el 125.
* Quan no s’indica un valor de retorn es retornarà el **valor de retorn de l’última ordre executada**.

<https://tldp.org/LDP/abs/html/exitcodes.html>

## Entrada de dades: read

L’ordre **read** permet la lectura de dades des del canal d’entrada estàndard. L’opció **-p** mostra un prompt per indicar a l’usuari quina dada o dades ha d’entrar. L’ordre read permet la introducció de valors per a diverses variables, però totes les variables es llegeixen en una única línia, i la lectura de dades acaba amb el final de línia.

[cesc@cport-virtualbox ~]$ read -p "dades: " A B C

dades: x y z

[cesc@cport-virtualbox ~]$ echo $C

z

Quan la intenció és llegir dades des del terminal de l’usuari, és a dir, demanar a l’usuari la introducció de dades, farem servir l’opció **-p** i llegirem només una variable.

## Control de flux

### Condicionals

#### construcció condicional: if

La construcció **if** permet l’execució d’ordres condicional, que començaran amb la paraula **then**; la construcció acaba amb la paraula **fi** (*if a l’inrevés*).

Podem indicar també múltiples sentències **elif** amb condicions alternatives, què quan no es compleix la primera condició s'avaluen en cascada, una per una, les diferents condicions **elif**, fins trobar una condició certa.

Opcionalment es pot indicar la sentència final **else** per executar una acció alternativa, quan no es compleix cap de les condicions.

Cada bloc d’ordres pot incloure diverses ordres, cadascuna en una línia, o bé separades fent servir el punt i coma.

Només el bloc d’ordres d’una de les opcions de la construcció (***if***, un dels ***elif***, ***else***) s’executarà.

**if** **[[** *CONDICIÓ1* **]]**;

**then**

*ORDRES1* # si es compleix la condició1

**elif** **[[** *CONDICIÓ2* **]]**;

**then**

*ORDRES2* # quan no es compleix condició1 però es compleix condició2

**else**

*ORDRES3* # en altre cas

**fi**

##### Exemple

Aquesta construcció la podem escriure en una única línia en el terminal, tot i que generalment és una construcció que es fa servir dins dels scripts i s’escriu en múltiples línies per millorar la llegibilitat:

[cesc@Skye pràctiques]$ if [[ -f p1a.sh ]]; then echo SI; else echo NO; fi

SI

#### Avaluació múltiple: case

Permet verificar un valor (una variable o un resultat) de manera múltiple amb diferents models. Cada model a comparar serà, bé un text simple, o un patró format amb els caràcters especials.

El model asterisc en la última posició és opcional, i cobreix l’acció per defecte si no es compleix cap dels models.

**case** *valor* **in** // valor = será una variable o una expressió.

*Model1*) ordres ;;

*Model2*) ordres ;;

**\*)** acció\_per\_defecte ;;

**esac**

// diferencia amb el if, aqui esta el valor. → és com el switch.

| **patrons pels models en *case*** | |
| --- | --- |
| **\*** | cadena de longitud variable, fins i tot buida |
| **?** | un caràcter. |
| **[ … ]** | un caràcter, entre els indicats dins dels claudàtors. |
| **[! … ]** | un caràcter, que no es troba entre els indicats entre claudàtors |
| **|** | o lògic, per indicar diferents patrons alternatius |

#### 

#### selecció per número entre un conjunt d’opcions: select

// seudo menu. que el usuari seleccioni una opció entre las que li donés. Ho fa com si fos una manera de bucle. Només podem sortir d’aqui amb un break.

La construcció **select** és en realitat un bucle, enfocat a l’entrada de dades per pantalla, i es repetirà fins executar l’ordre **break**. La seva funció és triar entre un conjunt d’opcions, que l’usuari selecciona per número, però el valor de la variable rep el valor explícit de l’opció, no pas el número introduït per l’usuari.

**select** variable **in** llista\_valors

**do**

ordres

**done**

Si no s’indica la llista de valors, es fa servir la llista de paràmetres de l’script.

*// L’usuari ha de seleccionar o Juli, Antoni, Francesc o Pere o sortir. (Es fa automátic).*

*select* ***resposta*** *→ será una variable*

*Obligatòriament posar* ***break*** *sinó seguirá demanant.*

*exit 0 → quan tot va bé, s’acaba amb* ***exit 0***

***PS3=”Digues: “*** *(PS1: prompt de bash normal , PS2: quan no s’ha acabat la linea (“.. ) → (****>****). PS3: demanar dades.*

| [cesc@Skye a]$ cat > select.sh #!/bin/bash # # select.sh #  # demostració ordre select # PS3="Digues: " select resposta in Juli Antoni Fancesc Pere '(sortir)' do  if [[ "$resposta" == '(sortir)' ]]  then  break  fi  echo "$USER ha triat $resposta" done echo "Adeu!" exit 0 |
| --- |

*//En executar:*

| [cesc@Skye a]$ bash select.sh 1) Juli // Això automaticament  2) Antoni 3) Fancesc 4) Pere 5) (sortir) Digues: 1 //Ho diu la màquina //ho escriu la persona cesc ha triat Juli Digues: 2 cesc ha triat Antoni Digues: 3 cesc ha triat Fancesc Digues: 4 cesc ha triat Pere Digues: 5 Adeu! |
| --- |

Observar que en aquest script s’ha definit la variable **$PS3**, que és el prompt que es fa servir per la introducció de dades.

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### Bucles

Els bucles, tots defineixen les ordres que s’executen entre les ordres **do** i **done**. És el mateix que hem vist per a **select**, però realment select també és un bucle.

#### Bucle for

Permet l’execució per a cadascú dels valors que es recullen a la llista; quan no s’indica la llista es fa servir la llista de paràmetres.

**for** *variable* **in** *llista*

**do**

*ordres*

**done**

En la llista podem tenir diferents variants:

##### **Una variable (de tipus array o diccionari)**

| [cesc@cport-virtualbox b]$ cat > for1.sh #!/bin/bash # # for1.sh # # bucle for, iterar una variable  # for params in $@  do  echo "$params" done |
| --- |

// Fa un *echo* de la llista que inserta.

| [cesc@cport-virtualbox b]$ bash for1.sh a1 a2 a3 a4 a5 a1 a2 a3 a4 a5 |
| --- |

##### 

##### 

##### **Llista implicita (arguments de l‘script)**

| [cesc@cport-virtualbox b]$ cat > for2.sh #!/bin/bash # # for1.sh # # bucle for, iterar els paràmetres d'entrada # for params  do  echo "$params" done |
| --- |

//Es lo mateix però t’estalvies $@

//Sortida:

| [cesc@cport-virtualbox b]$ bash for2.sh a1 a2 a3 a4 a5 a1 a2 a3 a4 a5 |
| --- |

##### **Llista d’elements explícita**

// Posem nosaltres els valors

| [cesc@cport-virtualbox b]$ cat > for3.sh #!/bin/bash # # for1.sh # # bucle for, llista explícita # for params in 1 2 3 do  echo "$params" done |
| --- |

//Sortida:

| [cesc@cport-virtualbox b]$ bash for3.sh 1 2 3 |
| --- |

##### **Criteri per a noms d’arxiu**

| [cesc@cport-virtualbox b]$ cat > for4.sh #!/bin/bash # # for1.sh # # bucle for, llista de noms d'arxiu # for params in \* do  echo "$params" done |
| --- |

// \*: tots els fitxers on està el directori

//Sortida:

| [cesc@cport-virtualbox b]$ bash for4.sh for1.sh for2.sh for3.sh for4.sh |
| --- |

##### 

##### **Interval de valors**

Tenim dues opcions:

* Fer servir l’ordre **seq**, que produeix una seqüència de números, començant per 1, fins el paràmetre que indiquem a **seq**

// seq: Escriu números de un en un, començant per un.

// Aquest for es com python

| [cesc@Skye a]$ for i in **$(seq 5)**; do echo $i; done 1 2 3 4 5 |
| --- |

* Fent servir una sintaxi semblant a C

// Aquest bucle és com java. Amb doble parentesís.

| [cesc@Skye a]$ for (( a=1 ; a<=5 ; a++ )); do echo $a; done 1 2 3 4 5 |
| --- |

##### 

##### **Fent servir una substitució d’una ordre**

Això ja l’hem vist quan veiem que podem fer servir l’ordre seq per generar una seqüència de números enters.

Es retalla la sortida d’aquest exemple, perquè mostra tots els usuaris, havent-hi molts usuaris de sistema.

// Això agafa tots els usuaris del nostre sistema.

**$(cat /etc/passwd | cut -d: -f1)**

for params in $(cat /etc/passwd | cut -d: -f1); do echo $params; done → seria lo mateix

| [cesc@cport-virtualbox b]$ cat for5.sh #!/bin/bash # # for1.sh # # bucle for, llista generada per l'execució d'ordres # for params in $(cat /etc/passwd | cut -d: -f1) do  echo "$params" done |
| --- |

// Sortida: tots el usuaris.

| [cesc@cport-virtualbox b]$ bash for5.sh root nobody dbus bin daemon mail ftp http systemd-journal-remote systemd-network systemd-resolve systemd-timesync systemd-coredump uuidd dhcpcd dnsmasq rpc ... usbmux cesc openvpn convidat |
| --- |

#### **Bucle while**

// Executar **mentre** la condició sigui certa (until: **fins)**

**while** *condició*

**do**

*ordres* // echo “$1”

**done**

El bucle ***while*** executará les *ordres* repetidament, comprovant abans de cada iteració si es compleix la *condició*; quan la *condició* deixa de complir-se, el bucle acaba: acaben les iteracions i l’execució segueix per la següent ordre que pugui haver després de **done**.

Executa les *ordres* si la *condició* és certa, i repetira l’execució d’ordres fins que la *condició* sigui falsa, comprovant cada vegada la *condició* abans de tornar a executar-les.

// sleep 1; → un segon sense fer res. Dir-li que esperi un segon abans de fer-ho. Espera en cada sortida un segon.

| [cesc@cport-virtualbox ~]$ i=5 [cesc@cport-virtualbox ~]$ while (( i >= 1 )); do echo "$i"; sleep 1; (( i-- )); done 5 4 3 2 1 |
| --- |

#### 

#### 

#### **Bucle until**

// Es el mateix que el while però al revés. Executar **fins** que la condició sigui certa

Es la condició contraria.

**until** *condició*

**do**

*ordres*

**done**

El bucle ***until*** executará les *ordres* repetidament, comprovant abans de cada iteració si no es compleix la *condició*; quan la *condició* es compleix, el bucle acaba: acaben les iteracions i l’execució segueix per la següent ordre que pugui haver després de **done**.

Executa les *ordres* si la *condició* és falsa, i repetira l’execució d’ordres fins que la *condició* es compleixi, comprovant cada vegada la *condició* abans de tornar a executar-les.

// Condició contraria sería **i < 1**

| [cesc@cport-virtualbox ~]$ i=5 [cesc@cport-virtualbox ~]$ until (( i == 0 )); do echo "$i"; sleep 1; (( i-- )); done 5 4 3 2 1 |
| --- |

#### **Ordres break i continue**

// per sortir del select.

// tots els bucles es poden sortir amb un break

// si es un while, es preferiria no sortir amb un break

En els bucles, dins de les ordres que es troben entre ***do*** i ***done***, podem fer servir les ordres ***break*** i ***continue***. En general és preferible evitar continue, i fer servir break únicament per a casos imprescindibles o molt evidents, particularment quan encara no tenim soltura programant bucles.

##### **break**

L’ordre ***break*** acaba l’iteració del bucle, i surt per a executar la següent ordre a continuació de ***done***.

##### **continue**

L’ordre continue acaba l’actual iteració i torna a l’inici per avaluar la condició inicial del bucle i comprovar si ha d'executar una nova iteració.

## Funcions

### Declaració de funcions:

// Totes tres son lo mateix. la de la dreta és més antiga.

| Dues alternatives: | |
| --- | --- |
| **Preferible** | **Alternativa** |
| function\_name () {  commands  } | **function** function\_name {  commands  } |
| També pot fer-se en una única línia:  function\_name () { commands; } | En una única línia:  **function** function\_name { commands; } |
| **Sintaxi mixta** (tercera alternativa) | |
| **function** function\_name () {  commands  } | |
| En una única línia:  **function** function\_name() { commands; } | |

// ' '|\*[!0-9]\*)

Si està buit ‘ ‘

si qualsevol cosa \*

si no esta entre 0-9

Esta verificant que el primer paràmetre sigui un número

ho envia al canal dos: <&2 : error

Després verifica si el paràmetre de números rebuts (reb un, més el que tingui → per això +1)

esac (fi del case)

| # Funció que verifica que hagi tants paràmetres com indica el primer paràmetre function checkNparam() {  case $1 in  ''|\*[!0-9]\*)  # Mostra un error en el canal 2 (error)  echo 'checkNparam requires a numeric first parameter' >&2  return 1 ;;  # Retorna un 1 indicant un error.   \*)  if [[ $# -ne $(($1 + 1)) ]]  then  echo 'Invalid number of arguments: '"$(( $# - 1 ))"\  'Expected: '"$1" >&2  return 1  else  return 0  fi ;;  esac } |
| --- |
|  |

// Primer, verificar els paràmetres. → que només tinc un paràmetre (checkNparam)

| # Funció que crea el directori si no existeix, i canvia el directori de treball function mkdircd() {  if checkNparam 1 $@  then  if [[ ! -d $1 ]]  then  mkdir -p $1  fi  cd $1  fi ca} |
| --- |

## CUT

cut -d

-d → delimitador

cut -d:

: → el character que delimita

-f (quina posició) -f2

cat archiu | cut

**DECLARE**

**Si volem números enters: (-i)**

**declare -i number**

| **declare -i number *# The script will treat subsequent occurrences of "number" as an integer.*   number=3 echo "Number = $number" *# Number = 3*  number=three echo "Number = $number" *# Number = 0* *# Tries to evaluate the string "three" as an integer.*** |
| --- |