Seqüències d’ordres, redireccions i filtres

# Índex

[Índex](#_6bldf3f1h8hg)

[Execució en seqüència](#_s6fe5ywm45iw)

[El valor de retorn i el valor lògic](#_354oa0gygu45)

[Les ordres true i false](#_eocvf5fqo1zz)

[El valor de retorn de l’execució aritmètica](#_bixtig3nsqqh)

[Execució condicional](#_1kq1qzg3r22o)

[L’ordre test i els claudàtors](#_89ad8mq5iqbh)

[Prova d’un valor](#_3clnyma0clj1)

[Comparació numérica](#_knn3kjl3u2nn)

[Comparació de cadenes de caràcters (strings)](#_gr9z2z5bijz3)

[Condicions de fitxers i directoris](#_dcqoin9dfif5)

[Redireccions: Canals d’entrada, sortida i error](#_2m5r2wup3ixf)

[Redirecció d’entrada](#_k7w30m3u6j4q)

[Document en línia (herescript o here document)](#_29cfvf8vq77)

[Herestring](#_fegd8krbw14z)

[Redirecció de sortida](#_k7w30m3u6j4q)

[Redirecció d’error](#_k7w30m3u6j4q)

[Redirecció dels errors al dispositiu null](#_5hx7q5c3aesy)

[Redirecció de sortida i d’error](#_k7w30m3u6j4q)

[Altres canals](#_k7w30m3u6j4q)

[El terminal](#_n2jvjzw3p7ck)

[La consola: /dev/console](#_axgb2bwlec4p)

[El terminal virtual: /dev/tty0](#_gd2gtt5p1020)

[Els pseudoterminals](#_oothndpcizt8)

[Altres terminals](#_badjjrx03rtb)

[El terminal: /dev/tty (TeleTYpewriter)](#_gfi9ez7yt8i3)

[Expansió d’ordres](#_io6v2scjy1l5)

[Substitució de processos](#_5uch6evndu6n)

[Canonades (pipes)](#_mvjbyzj0ntfv)

[Per saber-ne més:](#_nxyxprz18lf5)

[Agrupació d’ordres](#_2jw9vxmml0a8)

[Els parèntesi](#_moyy4f9c5j9h)

[Reflexions: subshells que executen scripts i ordres entre parèntesi](#_emtz9kabl7ow)

[Les claus](#_cf2ciset6aud)

# Execució en seqüència

Es poden executar ordres en seqüència fent servir el punt i coma. És equivalent a escriure les ordres en diferents línies, però amb el punt i coma les escrivim en una única línia. Bash rep tota la línia, però separa cadascuna de les ordres delimitades amb el punt i coma per executar-les en seqüència.

| [cesc@cport-virtualbox b]$ mkdir -p p1/a; cd p1/a [cesc@Skye a]$ |
| --- |

Quan parlem de [canonades (pipes)](#_mvjbyzj0ntfv) veurem que podem agrupar ordres amb els parèntesi i les claus, i el punt i coma separarà les ordres executades en seqüència.

## El valor de retorn i el valor lògic

A més de l’entrada i sortida estàndard, i el canal de sortida per als errors, els programes retornen un **byte** que indica un **codi d’error**. El valor zero indica que l’execució ha estat exitosa, i els altres 255 valors restants permeten una indicació del tipus d’error que s’ha produït.

Aquest codi de retorn el podem obtenir amb la variable especial ***$?***.

| [cesc@cport-virtualbox b]$ ls p1a.sh ls: no s'ha pogut accedir a 'p1a.sh': El fitxer o directori no existeix [cesc@cport-virtualbox b]$ echo $? 2 [cesc@cport-virtualbox b]$ touch p1a.sh [cesc@cport-virtualbox b]$ ls p1a.sh p1a.sh [cesc@cport-virtualbox b]$ echo $? 0 |
| --- |

En realitat existeixen molt pocs codis d’error predefinits, i si volem fer servir un codi d’error de retorn faríem servir:

* Si volem indicar el valor lògic **false**, el **codi de retorn 1**, reservat per a ***errors generals i imprevisibles***.
* Pel cas de no existir un fitxer, un directori, o no cumplir-se una comparació entre fitxers… és a dir, ***quan no es compleixen les precondicions necessàries***, es fa servir el **codi de retorn 2**, excepte si es vol diferenciar entre les diferents situacions, que farem servir un codi personalitzat.
* Si volem definir ***codis d’error personalitzats*** farem servir **codis de retorn entre el 3 i el 125**, que són codis que no tenen cap significat predefinit.
  + El codi 126 és un error de permisos.
  + El codi 127 és un error d’”ordre no trobada”.
  + Els codis 128 i 255 indiquen codi erroni en l’ordre exit.
  + Els codis 129-165 queden reservats per finalització deguda a una senyalització al procés (aturat per ^C és el codi 130=128+2)
  + Els codis 166-254 també són codis disponibles per a codis personalitzats.

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

| [cesc@cport-virtualbox b]$ ordrequeminvento bash: ordrequeminvento: no s'ha trobat l'ordre [cesc@cport-virtualbox b]$ echo $? 127 |
| --- |

Les condicions en bash es fonamenten en el valor de retorn, perquè, en l’execució d’una ordre, informa quan s’ha produït un error retornant en un byte un codi d’error diferent de zero. D’aquesta manera queda definit el valor lògic:

* El **zero**, l’**absència d’error**, el valor “**cert**”.
* Qualsevol valor diferent de zero, el valor “**fals**”.

### Les ordres **true** i **false**

Efectivament existeixen les ordres **true** i **false**, que no tenen cap altra funcionalitat que retornar un valor d’error zero, en el cas de **true**, o un valor diferent de zero, l’u, en el cas de **false**.

***Compte!*** En aquesta lògica el **valor cert és zero**, ***el contrari del definit en la lògica booleana***, i, en general, en qualsevol llenguatge de programació. Bash és un llenguatge d’scripting on cada ordre s’ha d’executar, i l’error pot determinar la lògica del flux del programa; Aquest és el motiu d’aquesta inversió. Altres llenguatges, tot i ser interpretats, compten amb sofisticats mecanismes per a la gestió d’errors, però en bash aquesta gestió s’ha simplificat, quedant incorporada a la pròpia lògica.

| [cesc@cport-virtualbox b]$ type true true és una ordre interna [cesc@cport-virtualbox b]$ type false false és una ordre interna [cesc@cport-virtualbox b]$ true [cesc@cport-virtualbox b]$ echo $? 0 [cesc@cport-virtualbox b]$ false [cesc@cport-virtualbox b]$ echo $? 1 |
| --- |

### El valor de retorn de l’execució aritmètica

L’execució d’una expressió aritmètica fent servir dobles parèntesi **(( *expressió* ))**

* Retorna un codi d’error zero (***true***) quan l’expressió aritmética resultant ***no és zero***.
* Retorna un codi d’error 1 (***false***) quan l’expressió aritmètica resultant ***és zero***.

Pot resultar una mica desconcertant el fet que es retorni el valor canviat; això ens ha de fer recordar que el valor de retorn (que aquí hem enfatitzat anomenant-lo *codi d’error*) és un valor diferent del resultat, i això també és així quan s’escriuen funcions. Recordem que el valor de retorn es troba limitat a un byte, amb un valor màxim de 255.

## Execució condicional

* El connector ***&&*** executa l’ordre que li segueix si l’ordre anterior s’executa correctament:

| [cesc@Skye a]$ cd ~/pràctiques [cesc@Skye pràctiques]$ cd p1/a && ls p1a.sh |
| --- |

però si e directori no existeix, l’ordre cd falla i no s’executa ls:

| [cesc@cport-virtualbox b]$ cd ~/pràctiques [cesc@cport-virtualbox pràctiques]$ cd p1/b && ls bash: cd: p1/b: El fitxer o directori no existeix |
| --- |

* El connector ***||*** executa l’ordre que li segueix només si l’ordre anterior falla. Veiem un exemple:

| [cesc@cport-virtualbox b]$ cd ~/pràctiques/p1/a [cesc@cport-virtualbox pràctiques]$ cat p1a2.sh || touch p1a2.sh cat: p1a2.sh: El fitxer o directori no existeix |
| --- |

El missatge d’error que apareix es produeix en la primera ordre, però com no s’ha executat correctament s’executa la segona ordre i es crea l’arxiu.

| [cesc@cport-virtualbox b]$ cd ~/pràctiques [cesc@cport-virtualbox pràctiques]$ cd p1/b 2>/dev/null || mkdir -p p1/b && cd p1/b |
| --- |

En aquest cas hem fet una mica de “*trampa*” per no veure el missatge d’error que es produeix: hem redireccionat ( ***2>/dev/null*** ) el canal d’error al dispositiu null, ignorant qualsevol missatge d’error.

## L’ordre test i els claudàtors

Veiem que, efectivament, el codi d’error permet la construcció lògica del control de flux, i es defineixen les ordres ***true*** i ***false***.

Quan volem avaluar una condició, fem servir l’ordre ***test***. L’ordre test es pot escriure també fent servir els claudàtors (***[ ]***). La versió més moderna incorpora també els dobles claudàtors (***[[ ]]***), que són paraula clau de bash, i permeten simplificar les expressions evitant l’escapament d’alguns caràcters. Tot i això, és important ***deixar sempre espais abans i després dels claudàtors*** perquè siguin correctament identificats.

Excepte per motius de compatibilitat enrere, es recomana **fer servir sempre els dobles claudàtors**, però els dobles claudàtors no són part de l’estàndard POSIX, i per tant **no es reconeixen** en ***sh*** ni en l’opció ***bash -POSIX***.

| [cesc@cport-virtualbox b]$ type test test és una ordre interna [cesc@cport-virtualbox b]$ type [ [ és una ordre interna [cesc@cport-virtualbox b]$ type [[ [[ és una paraula clau de l'intèrpret [cesc@cport-virtualbox b]$ which test /usr/bin/test [cesc@cport-virtualbox b]$ which [ /usr/bin/[ [cesc@cport-virtualbox b]$ ls -l /usr/bin/{test,[} -rwxr-xr-x 1 root root 59528 6 març 2020 '/usr/bin/[' -rwxr-xr-x 1 root root 55432 6 març 2020 /usr/bin/test |
| --- |

<https://wiki.bash-hackers.org/commands/classictest>

### Prova d’un valor

Per a una cadena de text (string) es retorna el valor **cert** (***true***) si la cadena no és buida, i **fals** (***false***) quan està buida.

| [cesc@cport-virtualbox b]$ [[ "x" ]] && echo SI || echo NO SI [cesc@cport-virtualbox b]$ [[ "" ]] && echo SI || echo NO NO |
| --- |

Cal anar en compte, perquè una cadena buida sense les cometes produirà error (i si és una variable buida, serà el resultat després de la expansió):

| [cesc@cport-virtualbox b]$ [[ $A ]] && echo SI || echo NO bash: error de sintaxi a prop de «]]»  [cesc@cport-virtualbox b]$ [[ ]] && echo SI || echo NO bash: error de sintaxi a prop de «]]» |
| --- |

Per tant, **cal fer servir les variables entre cometes**, per evitar el cas que siguin buides i doni error.

També s’ha d’anar amb compte, perquè els números no s’avaluen com a codis de retorn, i un valor no buit sempre retorna **cert** (***true***):

| [cesc@cport-virtualbox b]$ [[ 0 ]] && echo SI || echo NO SI [cesc@cport-virtualbox b]$[[ 1 ]] && echo SI || echo NO SI |
| --- |

### Comparació numérica

| **-eq** | equal |
| --- | --- |
| **-ne** | not equal |
| **-lt** | less than |
| **-le** | less or equal to |
| **-gt** | greater than |
| **-ge** | greater or equal to |

| [cesc@cport-virtualbox b]$ # -eq [cesc@cport-virtualbox b]$ [[ 5 -eq 05 ]] && echo "5 equals 05" 5 equals 05 [cesc@cport-virtualbox b]$ # -ne [cesc@cport-virtualbox b]$ [[ 6 -ne 20 ]] && echo "6 is not equal to 20" 6 is not equal to 20 [cesc@cport-virtualbox b]$ -lt [cesc@cport-virtualbox b]$ [[ 8 -lt 9 ]] && echo "8 is less than 9" 8 is less than 9 [cesc@cport-virtualbox b]$ [[ 3 < 4 ]] && echo "3 is less than 4" 3 is less than 4 [cesc@cport-virtualbox b]$ -le [cesc@cport-virtualbox b]$ [[ 3 -le 8 ]] && echo "3 is less than or equal to 8" 3 is less than or equal to 8 [cesc@cport-virtualbox b]$ -gt [cesc@cport-virtualbox b]$ [[ 5 -gt 10 ]] || echo "5 is not bigger than 10" 5 is not bigger than 10 [cesc@cport-virtualbox b]$ [[ 4 > 2 ]] && echo "4 is greater than 2" 4 is greater than 2 [cesc@cport-virtualbox b]$ -ge [cesc@cport-virtualbox b]$ [[ 3 -ge 3 ]] && echo "3 is greater than or equal to 3" 3 is greater than or equal to 3 |
| --- |

### Comparació de cadenes de caràcters (strings)

| **-z *string*** | *sting* és buit |
| --- | --- |
| **-z "${VAR+x}"** | *$VAR* no està definida.  Observar que ***-z $VAR*** també és certa quan *$VAR* és buit.  L’expressió ***${VAR+x}*** retorna ***x*** només quan *$VAR* està definida. |
| **-n *string*** | *string* no és buit |
| ***string* = *pattern***  *string1* == *pattern* | *string1* compleix el patró *pattern*. Només admet models senzills, els mateixos que els noms de fitxer, per la comprovació de patrons. |
| ***string =~ regexp*** | *string* compleix l’***expressió regular*** *pattern*. |
| *string1* = “*valor*”  ***string1* == “*valor*”** | *string1* té el *valor* (fa servir cometes). |
| ***string1* != *pattern*** | *string1* no compleix el patró *pattern*. Mateixos models que els noms de fitxer. |
| ***string1 != “valor”*** | *string1* no té el *valor*. |
| ***string1* > *string2*** | *string1* s’ordena (alfabèticament) abans que *string2* |
| ***string1* < *string2*** | *string1* s’ordena (alfabèticament) després que *string2* |

| [cesc@cport-virtualbox b]$ [[ '/home/cesc' == "$HOME" ]] && echo SI || echo NO SI [cesc@cport-virtualbox b]$ [[ hola == l\* ]] && echo SI || echo NO NO [cesc@cport-virtualbox b]$ [[ hola == \*l\* ]] && echo SI || echo NO SI [cesc@cport-virtualbox b]$ [[ hola == hola ]] && echo SI || echo NO SI [cesc@cport-virtualbox b]$ [[ caca < caça ]] && echo SI || echo NO SI [cesc@cport-virtualbox b]$ [[ cada < caça ]] && echo SI || echo NO NO |
| --- |

### Condicions de fitxers i directoris

| **-e *file*** | *file* existeix |
| --- | --- |
| **-f *file*** | *file* és un fitxer |
| **-s *file*** | *file* té mida més gran que zero |
| **-h *file*** | *file* és un *soft-link* (accés directe) |
| **-d *file*** | *file* és un directori |
| **-r *file*** | disponible permís de lectura per *file* |
| **-w *file*** | disponible permís d‘escriptura per *file* |
| **-x *file*** | disponible permís d’execució per *file* |
| **-N *file*** | *file* ha estat modificat des de que es va llegir |
| ***file1* -ef *file2*** | *file1* i *file2* són el mateix (*equal file*), degut a un *hard-link*. |
| ***file1* -nt *file2*** | *file1* més nou que *file2* (*newer than*) |
| ***file1* -ot *file2*** | *file1* més antic que *file2 (older than)* |
| ***-b file*** | *file* és un dispositiu de bloc |
| ***-c file*** | *file* és un dispositiu de caràcter |
| ***-p file*** | *file* és un pipe |
| ***-S file*** | *file* és un socket |
| ***-t file*** | *file* està associat a un terminal  Amb [[ -t 0 ]] sabem que stdin no està redirigit.  Amb [[ -t 1 ]] sabem que stdout no està redirigit. |

Referència: <https://tldp.org/LDP/abs/html/fto.html>

| [cesc@cport-virtualbox b]$[[ -e p1 ]] && echo SI || echo NO SI [cesc@cport-virtualbox b]$ [[ -f p1 ]] && echo SI || echo NO NO [cesc@cport-virtualbox b]$ [[ -d p1 ]] && echo SI || echo NO SI [cesc@cport-virtualbox b]$ [[ -e p1/a ]] && echo SI || echo NO SI [cesc@cport-virtualbox b]$ [[ -f p1/a ]] && echo SI || echo NO NO [cesc@cport-virtualbox b]$ [[ -d p1/a ]] && echo SI || echo NO SI [cesc@cport-virtualbox b]$ [[ -e p1/a/p1a.sh ]] && echo SI || echo NO SI [cesc@cport-virtualbox b]$ [[ -f p1/a/p1a.sh ]] && echo SI || echo NO SI |
| --- |

# Redireccions: Canals d’entrada, sortida i error

<https://baulderasec.wordpress.com/programacion/programacion-con-linux/2-programacion-shell/2-2-tuberias-y-redireccionamiento/>

Els programes tenen definits tres canals:

* Entrada estàndard (canal zero)
* Sortida estàndard (canal u)
* Sortida d’errors (canal dos)

## Redirecció d’entrada

El canal d’entrada és el canal pel que els programes esperen les dades o paràmetres. Les entrades poden fer-se des del teclat, però podem indicar un fitxer fent servir el caràcter ‘***<***’.

Per exemple, l’ordre wc (word count) compta el número de línies, paraules i caràcters d’un fitxer:

| [cesc@cport-virtualbox b]$ wc < f1  4 5 24 [cesc@cport-virtualbox b]$ cat f1 HOLA que tal? ... Adeu! |
| --- |

### Document en línia (*herescript* o *here document*)

La redirecció ‘***<<***’ és un tant particular, perquè permet la introducció d’un text i enviar el resultat a una ordre o un filtre. A continuació de ‘***<<***’ s’ha d’escriure la paraula que farà de clau per indicar el final del fitxer; en el exemple a continuació, es fa servir *:end*.

| [cesc@cport-virtualbox b]$ tr "[[:lower:]]" "[[:upper:]]" << :end > hola amigos > esto es un ejemplo > de herescript > :end HOLA AMIGOS ESTO ES UN EJEMPLO DE HERESCRIPT |
| --- |

### Herestring

Un cas molt semblant és ‘***<<<***’, que de vegades s’anomena “**here string**”. En aquest cas escrivim una cadena, i l’entrada acaba en acabar la línia; és a dir, en aquest cas no cal la paraula clau, perquè l’entrada acaba amb la línia.

| [cesc@cport-virtualbox b]$ cat <<< tutut tutut |
| --- |

Aquest string pot no estar directament escrit des del teclat, sinó ser generat per un altre programa:

TORNA A CREAR FITXER > (SOBREESCRIU)

>> ESCRIU A CONTINUACIÓ

| [cesc@cport-virtualbox b]$ tr "[a-z]" "[A-Z]" > archtest << :end hola amigos esto es un ejemplo de herescript :end |
| --- |

| [cesc@cport-virtualbox b]$ tr "[[:upper:]]" "[[:lower:]]" <<< $(cat archtest) hola amigos esto es un ejemplo de herescript |
| --- |

## Redirecció de sortida

El caràcter ‘***>***’ permet redireccionar la sortida a un fitxer. Quan s’executa la redirecció es crea el fitxer, i si existeix es crea de nou. Fins i tot quan l’ordre per la què dirigim la sortida produeix un error, l’arxiu ja s’haurà creat (buit).

| [cesc@cport-virtualbox b]$ bash p1a.sh > p1.txt |
| --- |

Quan volen afegir nova informació a un arxiu existent, en comptes de crear un de nou, fem servir ‘***>>***’:

| [cesc@cport-virtualbox b]$ bash p1b.sh >> p1.txt |
| --- |

## Redirecció d’error

La redirecció del canal 2 (canal d’error) la podem indicar amb ‘***2>***’. D’aquesta manera podem redireccionar la sortida cap a un fitxer i els errors a un fitxer diferent.

| [cesc@cport-virtualbox b]$ cat mequivocodefitxer 2> /dev/null echo $? 1 |
| --- |

### Redirecció dels errors al dispositiu null

Redireccionar els errors permet comprovar-los més tard. També, de vegades, volem ignorar els errors que es produeixen, potser perquè sabem que es produeixen texts d’error ja previstos i no significatius; en aquest cas els podem dirigir al dispositiu null, el “***forat negre***” que s’ho empassa tot.

| [cesc@Skye pràctiques]$ cd p1/b 2>/dev/null || mkdir -p p1/b && cd p1/b |
| --- |

## Redirecció de sortida i d’error

Potser volem que totes les sortides es dirigeixin a un mateix arxiu; això treu els missatges d’error de pantalla, i permet trobar els errors dins el context en que es donen; naturalment el fitxer contindrà tant els resultats com els missatges d’error, de manera que aquests missatges poden impedir el seu processament automàtic posterior.

Per indicar que el canal 2 (d’error) es direcciona al mateix canal 1 (de sortida) ho indiquem amb ***2>&1***.

| [cesc@cport-virtualbox b]$ cat mequivocodefitxer \*.sh > sortida.txt 2>&1 echo $? 1 |
| --- |

També es pot fer servir l’abreviació ‘&>’ per indicar que tots dos canals es redirigeixen al mateix arxiu; així aquesta línia és equivalent a l’anterior exemple:

| [cesc@cport-virtualbox b]$ cat mequivocodefitxer \*.sh &> sortida.txt echo $? 1 |
| --- |

Per redirigir la sortida de echo al canal d’error, es redirigeix la sortida estàndard, que és on escriu l’ordre ***echo***, al canal d’error:

| [cesc@cport-virtualbox b]$ echo "error" >&2 |
| --- |

## Altres canals

En realitat ‘***0<***’ és el mateix que ‘***<***’, i ‘***1>***’ el mateix que ‘***>***’. En total n’hi ha nou canals, però els canals 3 al 9, tot i que són disponibles, s’han d’obrir per fer-los servir.

Per obrir un canal es fa servir l’ordre ***exec***. D’aquesta manera podem treure uns resultats per un canal, i altres per un altra.

| [cesc@cport-virtualbox b]$ # Obrim el canal 3 que es dirigirà al fitxer dump.log [cesc@cport-virtualbox b]$ exec 3>dump.log [cesc@cport-virtualbox b]$ # Dirigim la sortida al canal 3 [cesc@cport-virtualbox b]$ ls -l >&3 [cesc@cport-virtualbox b]$ cat dump.log total 64 -rw-r--r-- 1 cesc cesc 45 11 febr. 17:48 archtest -rw-r--r-- 1 cesc cesc 0 11 febr. 19:24 dump.log -rw-r--r-- 1 cesc cesc 24 10 febr. 16:49 f1 -rw-r--r-- 1 cesc cesc 14 10 febr. 18:29 foo.txt -rw-r--r-- 1 cesc cesc 102 10 febr. 16:09 for1.sh -rw-r--r-- 1 cesc cesc 102 10 febr. 16:12 for2.sh -rw-r--r-- 1 cesc cesc 103 10 febr. 16:13 for3.sh -rw-r--r-- 1 cesc cesc 104 10 febr. 16:16 for4.sh -rw-r--r-- 1 cesc cesc 153 10 febr. 16:24 for5.sh -rw-r--r-- 1 cesc cesc 10524 11 febr. 16:03 frases drwxr-xr-x 2 cesc cesc 4096 10 febr. 18:31 one -rw-r--r-- 1 cesc cesc 370 11 febr. 15:43 proves.txt -rw-r--r-- 1 cesc cesc 622 11 febr. 18:55 sortida.txt drwxr-xr-x 2 cesc cesc 4096 10 febr. 18:31 three drwxr-xr-x 2 cesc cesc 4096 10 febr. 18:31 two |
| --- |

El canal es pot tancar relacionant-lo amb el pseudocanal de tancament ‘***-***’.

| [cesc@cport-virtualbox b]$ exec 3>&- |
| --- |

## El terminal

El terminal, quan ens trobem en mode interactiu, es troba associat amb els tres canals:

* zero (0: entrada estàndard)
* u (1: sortida estàndard)
* dos (1: canal d’error estàndard)

Però podem fer servir diversos terminals simultàniament, i, de fet, també diversos pseudo terminals. Podem veure el dispositiu associat actualment executant l’ordre **tty**.

### La consola: /dev/console

La consola històricament tenia accés directament a l’ordinador, de mentre que la resta de terminals es trobaven connectats per línia sèrie. Actualment el dispositiu **/dev/console** és un dispositiu protegit, que només els processos de root poden accedir, i el sistema de vegades el fa servir internament.

### El terminal virtual: /dev/tty0

Quan ens trobem en un terminal virtual, el terminal assignat és **/dev/tty1**, **/dev/tty2**, **/dev/tty3**, **/dev/tty4**… (segons el número de terminals definits, el número pot seguir). El valor **/dev/tty0** es refereix al terminal en un terminal virtual, indistintament del terminal concret assignat; és a dir, en tots els terminals virtuals **/dev/tty0** es troba disponible, i en cadascú es refereix al seu propi terminal.

### Els pseudoterminals

En el cas dels psudoterminals, i també les connexions remotes, el dispositiu es crea quan es crea el pseudoterminal. Aquesta funció la fa el dispositiu **/dev/ptmx**, un multiplexor que crea terminals; crea un pseudoterminal master (PTM), i un pseudoterminal slave (PTS) que es crea en el directori **/dev/pts**. Així, cada pseudoterminal o connexió remota obté un nou pseudoterminal per interactuar amb l’usuari.

### Altres terminals

Segueix sent possible connectar un terminal per línia sèrie (avuí dia, ja no hi és la línia sèrie, però queda l’alternativa del USB), o bé un altre ordinador emulant un terminal; aquestes connexions sèrie s’anomenen **/dev/ttyS** o bé **/dev/ttyUSB**, però generalment no es troben actius per defecte, i no entrarem en detall.

### El terminal: /dev/tty (TeleTYpewriter)

El terminal representa el terminal que actualment s’està fent servir, indiferentment del tipus de terminal.

Els processos llançats interactivament des d’un terminal sempre tenen el terminal **/dev/tty** disponible (fins i tot si s’ha creat un subshell per exectuar un script, i la sessió ja no és interactiva), excepte que hagin estat explícitament desvinculats del procés interactiu que els ha creat; és indiferent que referenciem **/dev/tty** des de un pseudoterminal o des d’un terminal remot, perquè en tots dos casos obtindrem un descriptor vàlid.

Es pot fer servir quan requerim que una acció sigui directament interactiva en un script, però, naturalment, per això el script s’ha d’iniciar des d’un terminal.

# Expansió d’ordres

Recordem l’expansió d’ordres que permet incorporar el resultat d’una execució en la línia d’entrada, quan escrivim entre parèntesi i fent servir el ‘***$***’ de prefix. Hem vist ja algun exemple, per exemple quan veiem les redireccions d’entrada:

| [cesc@cport-virtualbox b]$ tr "[[:upper:]]" "[[:lower:]]" <<< $(cat archtest) hola amigos esto es un ejemplo de herescript |
| --- |

Tota la sortida de l’execució es converteix en part de la línia d’entrada. És per això que cal anar amb compte, perquè la línia d’entrada té una mida finita. Podem veure les limitacions preguntant-li a xargs, que efectivament converteix la entrada que rep en part de la línia d’entrada; en el meu sistema (5.10.15-1-MANJARO):

| [cesc@Skye ~]$ xargs --show-limits Your environment variables take up 3241 bytes POSIX upper limit on argument length (this system): 2091863 POSIX smallest allowable upper limit on argument length (all systems): 4096 Maximum length of command we could actually use: 2088622 Size of command buffer we are actually using: 131072 Maximum parallelism (--max-procs must be no greater): 2147483647 |
| --- |

Efectivament és una longitud gran (1.995 MiB), però si fem servir indiscriminadament l’expansió d’ordres amb ***$( )***, així com ***xargs***, fàcilment trobarem fitxers amb una mida que excedeixi les limitacions.

# Substitució de processos

Es donen dos casos de la substitució de procesos. La substitució de processos permet construir amb comandes un paràmetre d’una ordre que no admet les dades pel canal d’entrada o de sortida:

* ***<(****procés****)***

La sortida de l’execució de *procés* queda disponible com a paràmetre fitxer; és a dir, es pot fer servir on s’admet un nom de fitxer en mode lectura.

* ***>(****procés****)***

L’entrada per l’execució de *procés* queda disponible com a paràmetre fitxer; és a dir, es pot fer servir on s’admet un nom de fitxer en mode escriptura.

# Canonades (pipes)

Una canonada (pipe) connecta la sortida d’un programa amb l’entrada d’un altre, evitant crear un fitxer. Així:

| [cesc@cport-virtualbox b]$ # Fent servir un fitxer per les dades d'intercanvi [cesc@cport-virtualbox b]$ ls -1 > resultado.txt [cesc@cport-virtualbox b]$ wc < resultado.txt 15 15 113 |
| --- |

| [cesc@cport-virtualbox b]$ # El mateix, però més senzill fent servir una canonada [cesc@cport-virtualbox b]$ ls -1 | wc 15 15 113 |
| --- |

El resultat final es pot dirigir finalment a un arxiu de resultat:

| [cesc@cport-virtualbox b]$ ls -1 | wc -l > resultat.txt |
| --- |

També es pot cronometrar el temps que triga tot el procés:

| [cesc@cport-virtualbox b]$ time ls -1 | wc -l > resultat.txt |
| --- |

real 0m0,003s

user 0m0,005s

sys 0m0,000s

| [cesc@cport-virtualbox b]$ cat /etc/passwd | sort |
| --- |

<https://linuxhint.com/bash_pipe_tutorial/>

## Per saber-ne més:

<http://blog.petersobot.com/pipes-and-filters>

# Agrupació d’ordres

Imaginem que volem construir un arxiu resultat de l’execució d’una seqüència d’ordres:

| [cesc@cport-virtualbox b]$ uname -a > r.txt; pwd >> r.txt; ls -l >> r.txt |
| --- |

Aquesta seqüència d’ordres, a més de llarga d’escriure, comporta la repetida manipulació de l’arxiu de sortida r.txt.

Podriem executar les ordres en segon pla, però llavors, en alguns casos, ens podem trobar resultats inesperats, donat que pels programes en segon pla no queda definit l’ordre en què s’executen; justament s’executen concurrentment.

| [cesc@cport-virtualbox b]$ # compte! pot tenir resultats inesperats. [cesc@cport-virtualbox b]$ uname -a > r.txt & pwd >> r.txt & ls -l >> r.txt & |
| --- |

## Els parèntesi

El podem simplificar, si li redirigim la sortida a tota la seqüència d’ordres:

| [cesc@cport-virtualbox b]$ (uname -a; pwd; ls -l) > r.txt |
| --- |

En realitat l’agrupació crea un subshell on s’executen les ordres que es troben en parèntesi; en aquest cas, a més, tenim l'avantatge que el podem executar en segon pla:

| [cesc@cport-virtualbox b]$ (uname -a; pwd; ls -l) > r.txt & [1] 11150 [cesc@cport-virtualbox b]$ echo $$ 1862 [1]+ Fet ( uname -a; pwd; ls --color=auto -l ) > r.txt |
| --- |

Com que es fa servir un subshell cal anar amb compte amb les variables, perquè qualsevol modificació al valor de les variables o nova variable no es veurà reflectida.

| [cesc@cport-virtualbox b]$ (A=HOLA) [cesc@cport-virtualbox b]$ echo $A |
| --- |

### Reflexions: subshells que executen scripts i ordres entre parèntesi

El fet d’executar-se les ordres en un subscript presenta alguns problemes, no només relatius a les variables, que ja hem comentat, però principalment en la definició de funcions. Si volem crear l’ordre ***mkdircd***, que crea un directori i, a continuació, canvia el directori de treball, un script no serà una opció (cert que podem forçar l’execució en el shell actual fent servir el punt, però no sembla una solució pràctica), perquè l’ordre cd és interna, i el directori de treball és un estat de la instància de bash en execució que no es pot canviar des de un subshell.

Veurem que aquest problema està previst, i les funcions es poden incloure en la instància actual fent servir l’ordre ***source***, evitant l’execució en un subshell. Per a determinades accions són les funcions l’extensió natural de les ordres de bash, reservant els scripts per casos més complexes que poden executar-se en un subshell.

| [cesc@cport-virtualbox b]$ **function** **mkdircd** () { mkdir -p "$@" && cd "$@"; } |
| --- |

Naturalment, l’execució dels scripts en un subshell també incorpora avantatges, perquè proporciona un entorn d’execució aïllat que limita els efectes laterals que pot produir en el shell principal.

## Les claus

A més de l’agrupació amb els parèntesi, existeix l’agrupació amb les claus; en aquest cas no es crea un subshell per l’execució, i per tant serà preferible si s’han de manipular variables.

S’ha d’anar en compte de deixar espais en blanc al voltant de les claus, i finalitzar l'última ordre amb punt i coma.

| [cesc@cport-virtualbox b]$ { A=HOLA; } [cesc@cport-virtualbox b]$ echo $A HOLA |
| --- |

La diferència es fa evident si provem l’execució de l’ordre **exit**; ***(exit)*** no té cap efecte, perquè finalitza el subshell que executa les ordres entre parèntesi, però ***{ exit; }*** surt del shell actual, perquè no s’executa en un subshell.

Referències

* Bash One-Liners Explained
  + Bash One-Liners Explained, Part I: Working with files

<https://catonmat.net/bash-one-liners-explained-part-one>

* + Bash One-Liners Explained, Part II: Working with strings

<https://catonmat.net/bash-one-liners-explained-part-two>

* + Bash One-Liners Explained, Part III: All about redirections

<https://catonmat.net/bash-one-liners-explained-part-three>

* + Bash Redirections Cheat Sheet

<https://catonmat.net/bash-redirections-cheat-sheet>

* Illustrated Redirection Tutorial

<https://wiki.bash-hackers.org/howto/redirection_tutorial>

read -p ‘digues: ‘ > x /dev/tty