

Elements de Sistemes Operatius

Capítol 3

Apologia del *shell*

J. Fornés M. Farreras Y. Becerra

Facultat d'Informàtica de Barcelona
Universitat Politècnica de Catalunya

Grau en Enginyeria Informàtica — Sistemes Operatius

Versió de prova per a lectors · 23 de setembre de 2026



UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

© Universitat Politècnica de Catalunya

Aquesta obra està subjecta a la llicència de Creative Commons Reconeixement-NoComercial-CompartirIgual 4.0 Internacional (CC BY-NC-SA 4.0):

«By Universitat Politècnica de Catalunya — BarcelonaTech (UPC), Any 2026»



<https://creativecommons.org/licenses/by-nc-sa/4.0/deed.ca>

Aquest text s'ha escrit amb l'ajuda de Claude, el model de llenguatge d'Anthropic. Tots els errors que quedin són dels autors.

3 Apologia del *shell*

3.1 Què és un *shell*

Tot sistema operatiu ha de proveir, per definició, d'una interfície de programació i d'una interfície d'usuari per interactuar amb els serveis del kernel. Per al cas de la programació tenim les crides al sistema, que estudiarem a molts dels capítols d'aquest llibre. Per al cas de l'usuari tenim l'entorn de finestres, però també, en mode text, podem executar les comandes.

Un *shell* és l'interpret de comandes. Ras i curt: un bucle infinit que llegeix línies — cadenes de caràcters acabades en salt de línia —, les interpreta (les separa en programes i opcions) i executa els programes amb les opcions interpretades.

Fixeu-vos que en aquesta definició no hi surt el kernel enlloc. El *shell* no en forma part: és un programa d'usuari com qualsevol altre, que fa `fork()`, `exec()` i `wait()` com els faríeu vosaltres. Al capítol 10 n'escriurem un de mínim, capaç d'executar `ls | wc`, i veureu que hi caben trenta línies de C.

Tot programador de Unix ha tingut la temptació de crear-ne un. Alguns — molts — caigueren en la temptació, com ara Steve Bourne, creador del *Bourne shell* (`sh`), origen del *Bourne Again shell* (`bash`). Les distribucions de Linux acostumen a portar el `bash` per defecte, i el macOS, el `zsh`; però n'hi ha versions de tots dos —i de molts altres— per als dos sistemes. No és estrany que en trobeu d'altres als ordinadors del laboratori: `csh`, `ksh` o fins i tot el `sh` pelat, el més simple de tots.

Us preguntareu quina diferència hi ha. En les coses que ens ocupen

en aquest llibre — executar programes, redirigir-ne l’entrada i la sortida, connectar-los amb pipes — gairebé cap: el que aprendreu aquí funciona a tots. Les diferències són de comoditat i de llenguatge de guions, i s’aprenen millor fent-los servir que llegint-ne comparacions.

3.2 El manual

Abans de cap altra comanda, la que us les explica totes. Tot Unix porta el manual instal·lat, i es consulta amb **man**:

```
$ man ls
```

La pàgina ocupa el terminal sencer i es llegeix amb un paginador, normalment el **less**. N’hi ha prou amb poques tecles:

Tecla	Què fa
espai, b	avança o retrocedeix una pantalla
/ paraula	cerca paraula endavant
n , N	va a la coincidència següent o a l’anterior
q	surt
h	l’ajuda del paginador, amb totes les altres

El manual està dividit en **seccions** numerades, i en aquest llibre en fareu servir quatre:

Secció	Què hi ha	Per exemple
1	comandes	ls , chmod , man
2	crides al sistema	write() , fork()
3	funcions de la llibreria de C	printf , strlen
7	visions de conjunt i convencions	signal , pipe , inode

La 1 i la 2 són les dues interfícies del capítol 2: la d’usuari i la de programació, cadascuna amb la seva secció. Per això a la documentació trobareu els noms amb un número al costat, **write(2)** o **printf(3)**: vol dir «a la secció 2» o «a la 3», i no és cap argument.

i Compte

Un mateix nom pot ser a més d'una secció, i si no li dieu quina, el **man** us ensenya la primera que troba —i comença per la 1. Proveu-ho amb la crida del capítol 2:

```
$ man -f write
write (1)          - send a message to another user
write (1p)         - write to another user
write (2)          - write to a file descriptor
write (3p)         - write on a file
```

Un **man write** us explica una comanda per enviar missatges a un altre usuari, que no té res a veure amb la crida; la crida és **man 2 write**. Les seccions amb una **p** (**1p**, **3p**) són les pàgines de l'estàndard POSIX: diuen què ha de fer la comanda o la funció a qualsevol Unix, no el que fa Linux en concret. Passa el mateix amb **printf** i amb **kill**. Quan busqueu una crida al sistema o una funció, poseu-hi sempre el número.

El **man -f** que acabau de veure diu a quines seccions és un nom que ja sabeu. Quan no sabeu el nom, **man -k** busca una paraula a la descripció de totes les pàgines:

```
$ man -k 'create pipe'
pipe (2)          - create pipe
pipe2 (2)         - create pipe
```

I si voleu saber tot el que fa el mateix **man**, ja sabeu on mirar: **man man**.

 Per saber-ne més

Si algun dia voleu més que la sintaxi d'una crida, la secció 7 és la que es deixa llegir sencera. No descriu cap comanda ni cap funció: explica com encaixa una part del sistema amb la resta, i quines crides hi intervenen. **man 7 signal** i **man 7 pipe**, per exemple, són de les millors visions de conjunt que trobareu enlloc. Llegiu-les quan arribeu als capítols 7 i 10: us diran el mateix que el llibre, amb el vocabulari exacte del sistema.

3.3 Comandes per obrir boca

Només cal que feu una ullada al directori `/bin` per adonar-vos de la quantitat de comandes de què disposeu. Totes les podeu cridar des d'un *shell* i les trobareu, si fa no fa amb els mateixos arguments, a tots els sistemes Unix. Aquí només us en presentem un petit tast. Esperem que us enganxi al teclat.

La comanda **ls** llista el contingut d'un directori. Aquest contingut — el bloc de dades d'un directori — consisteix en parelles *[inode, nom]*. Però aquests noms no identifiquen els fitxers: podeu tenir el mateix nom en directoris diferents i que corresponguin a fitxers totalment diferents. El que de debò identifica un fitxer és el seu índex de node, o *inode*. Dins d'aquesta estructura de dades, el sistema de fitxers guarda la informació de gestió del fitxer — propietari, permisos, dates d'accés — i també quins blocs de dades té assignats.

Sense arguments, **ls** mostra només els noms, és a dir, un element de cada parella. Si hi afegiu l'opció **-l** (de *long*), per mostrar més informació de cada fitxer, li caldrà accedir no només al bloc de dades del directori sinó també al contingut dels inodes.

Vegem amb un exemple tota la informació que mostra aquesta opció. Hi hem afegit les opcions **-i** (per mostrar el número d'inode) i **-a** (per mostrar tots els fitxers, fins i tot el `.` i el `..`), per poder-nos explicar millor. El llistat és d'una altra màquina, amb el sistema en català; el `/dev` del laboratori no és igual, però es llegeix de la mateixa manera:

```
alumne@labS0:/dev$ ls -lia
total 4
  2 drwxr-xr-x 17 root root      3840 set 25 15:11 .
  2 drwxr-xr-x 20 root root     4096 set 15  2021 ..
342 crw-r--r--  1 root root    10, 235 set 25 15:06 autofs
215 drwxr-xr-x  2 root root      340 set 25 15:11 block
153 drwxr-xr-x  2 root root     3100 set 25 15:06 char
 15 crw--w----  1 root tty       5,   1 set 25 15:06 console
148 lrwxrwxrwx  1 root root       11 set 25 15:06 core ->
    /proc/kcore
328 brw-rw----  1 root disk    253,   0 set 25 15:06 dm-0
312 drwxr-xr-x  3 root root      100 set 25 15:06 dri
```

Anem a pams.

De la primera línia (**total 4**), no us en preocupeu per ara. Mostra quant ocupen de debò aquests fitxers al disc, però d'això ja en parlarem al capítol 11.

La resta de línies segueixen totes el mateix patró. N'agafem una qualsevol per fixar idees:

```
15 crw--w----  1 root tty  5,  1 set 25 15:06 console
```

3.3.1 L'inode

```
15 crw--w----  1 root tty  5,  1 set 25 15:06 console
```

El 15 és l'inode del fitxer. Aquest número és únic a tot el sistema de fitxers i és el que de debò identifica el fitxer. Podem tenir molts noms diferents del mateix fitxer* i molts fitxers diferents amb el mateix nom, tot i que no dins del mateix directori. Aquest número només apareix amb l'opció `-i`.

3.3.2 El tipus i els permisos

```
15 crw--w----  1 root tty  5,  1 set 25 15:06 console
```

*Allò que anomenem *links*. Mireu la pàgina de manual de `ln`.

Primer ve la lletra `c`, que vol dir que aquest fitxer és un dispositiu de caràcter: la unitat mínima d'accés és el caràcter*.

A continuació venen nou caràcters, `rw--w----`, que heu d'agrupar de tres en tres per a *read*, *write* i *execute*. Són els permisos del fitxer per al propietari, per al grup d'usuaris al qual pertany, i per a la resta d'usuaris. Així, el propietari — l'usuari `root` — té permís de lectura i escriptura (`rw-` = 110) però no d'execució. Els membres del grup `tty` tenen permís d'escriptura, però ni de lectura ni d'execució (`-w-` = 010). I la resta del món no en té cap (`---` = 000).

i Compte

A Unix un fitxer només es pot executar si té permís d'execució, independentment de com es digui. Allò del `nom.extensió` de Windows aquí no vol dir res.

Per canviar els permisos teniu la comanda `chmod` (*change mode*). Si penseu els bits dels permisos en octal, ja teniu els arguments. Per exemple, si voleu afegir permís de lectura als membres del grup `tty` deixant la resta igual, voldríeu `rw-`, és a dir, $110_2 = 6_8$. Per al propietari també `rw-`, $110_2 = 6_8$, i per a la resta del món `---`, $000_2 = 0_8$. En resum:

```
$ chmod 660 console
$ ls -li console
15 crw-rw---- 1 root tty 5, 1 set 25 15:06 console
```

3.3.3 El nombre de links

```
15 crw--w---- 1 root tty 5, 1 set 25 15:06 console
```

Aquest 1 és el nombre de *links* del fitxer: quants noms diferents té aquest inode a tot el sistema de fitxers o, dit d'una altra manera,

*De tipus diferents de fitxer n'hi ha uns quants. Quants? Feu `man 7 inode` per veure'ls tots, i de passada la resta d'informació que es guarda a l'inode.

quants camins diferents acaben al mateix fitxer. En aquest cas, només un.

Però fixeu-vos en la primera línia del llistat, la del directori `.`: l'inode 2 té **disset** links. Com pot ser? De fet són pocs; aquell sistema de fitxers és petit, i trobareu instal·lacions on l'inode 2 en té molts més. La raó és que l'inode 2 és el directori arrel*.

Qüestió 1

Sabem que l'inode 2 és l'arrel perquè és l'únic directori on els fitxers `.` i `..` són el mateix inode. Per què? I per què cada directori nou que creeu fa pujar en un el comptador de links del seu pare?

Resposta

Perquè `..` és un nom del directori pare, i per tant un link més cap al seu inode. A l'arrel no hi ha pare, i per això el seu `..` apunta a ella mateixa. D'aquí surt també la resposta a la segona: un directori qualsevol té almenys dos links — el seu nom dins del pare i el seu propi `.` —, i en guanya un per cada subdirector que li pengi.

Aquí ho deixem. De com el sistema de fitxers guarda tot això, per què l'inode no conté el nom del fitxer i què passa exactament quan n'esborreu un, en parla el capítol 11.

3.4 Redireccions

Fins ara les comandes escriuen per pantalla i llegeixen del teclat. Això és mig veritat, i la mitja que falta és la important.

*La majoria de sistemes de fitxers de Unix reserven els inodes 0 i 1 per a gestió. En l'Ext4, el 0 vol dir error — buit, nul, no existeix — i l'1 conté tots els blocs dolents que ha anat trobant el sistema; assignar-los a l'inode 1 evita tornar-los a intentar. Vegeu [Ext4](#) — [Special inodes](#).

Tot procés de Unix comença la vida amb tres canals oberts: el 0, l'entrada estàndard; l'1, la sortida estàndard, i el 2, la sortida estàndard d'error*. Normalment tots tres van a parar al terminal, però el programa no ho sap ni li importa: ell escriu al canal 1 i prou.

I com que no ho sap, el *shell* pot canviar-los-hi abans d'executar-lo. Això és una redirecció:

```
$ ls > llistat.txt      # el canal 1 va al fitxer, no a la pantalla
$ ls >> llistat.txt    # igual, però afegint al final
$ wc -l < llistat.txt  # el canal 0 surt del fitxer, no del teclat
$ gcc prog.c 2> errors # el canal 2 va al fitxer; l'1 es queda al
                      terminal
```

Noteu que el `>` crea el fitxer si no existeix i el buida si existeix. El `>>` no el buida. És una distinció que es paga cara una sola vegada.

Els canals es poden redirigir a un altre canal en comptes d'a un fitxer, amb la notació `n>&m`, que vol dir «que el canal *n* vagi on va el *m*»:

```
$ gcc prog.c > tot 2>&1 # primer l'1 al fitxer, després el 2 on
                      va l'1
$ gcc prog.c 2>&1 > tot # NO és el mateix: mireu-vos l'ordre
```

Qüestió 2

Les dues línies de l'exemple anterior es diferencien només en l'ordre de les redireccions. On acaben els missatges d'error de cadascuna?

*Aquest tercer canal no hi era al principi. L'inventaren en Thompson i en Ritchie la mateixa nit que reescrigueren totes les comandes del sistema per fer-hi cabre les pipes; vegeu la secció 1.7.1.

Resposta

A la primera, el canal 1 va al fitxer **tot** i tot seguit el 2 es fa anar «on va l'1», que ja és el fitxer. Els errors acaben al fitxer.

A la segona, el 2 es fa anar on va l'1 *quan encara* és el terminal, i només després es canvia l'1 cap al fitxer. Els errors es queden a la pantalla. El *shell* processa les redireccions d'esquerra a dreta, i cada una fa la seva còpia en el moment en què s'executa.

Fixeu-vos en què acabem de fer: hem canviat d'on llegeix i on escriu un programa **sense tocar-lo**. El **ls** no sap que està escrivint en un fitxer, i el **wc** no sap que està llegint-ne un.

Aquesta idea — que els programes parlin amb canals numerats i que sigui algú altre qui decideixi on van a parar — és la que fa possible tot el capítol 10. Quan en McIlroy demanava que els programes es poguessin acoblar «com manegues de jardí», demanava exactament això: que la sortida d'un es pogués enganxar a l'entrada d'un altre. Amb les redireccions ja fetes, en Thompson va poder escriure la crida al sistema en una hora.

3.5 Exercicis

1. Feu `ls /bin | wc -l` i mireu quantes comandes teniu a l'abast. Trieu-ne tres que no conegueu i llegiu-ne la pàgina de manual.
2. Al llistat de `/dev` hi ha una línia que comença per `l` i una altra que comença per `b`. Què són? (Pista: `man 7 inode`.)
3. Creeu un directori buit i mireu-ne el nombre de links amb `ls -ldi`. Creeu-hi un subdirectori a dins i torneu a mirar-lo. Quadra amb la resposta de la qüestió 1?
4. Què fa `ls > llistat.txt` si `llistat.txt` apareix al llistat? Proveu-ho i expliqueu per què la mida del fitxer és la que és.

5. Redirigiu la sortida d'error d'una comanda a `/dev/null` i expliqueu, mirant el llistat de `/dev` d'aquest capítol, què és exactament aquest fitxer.