GIE/ICE

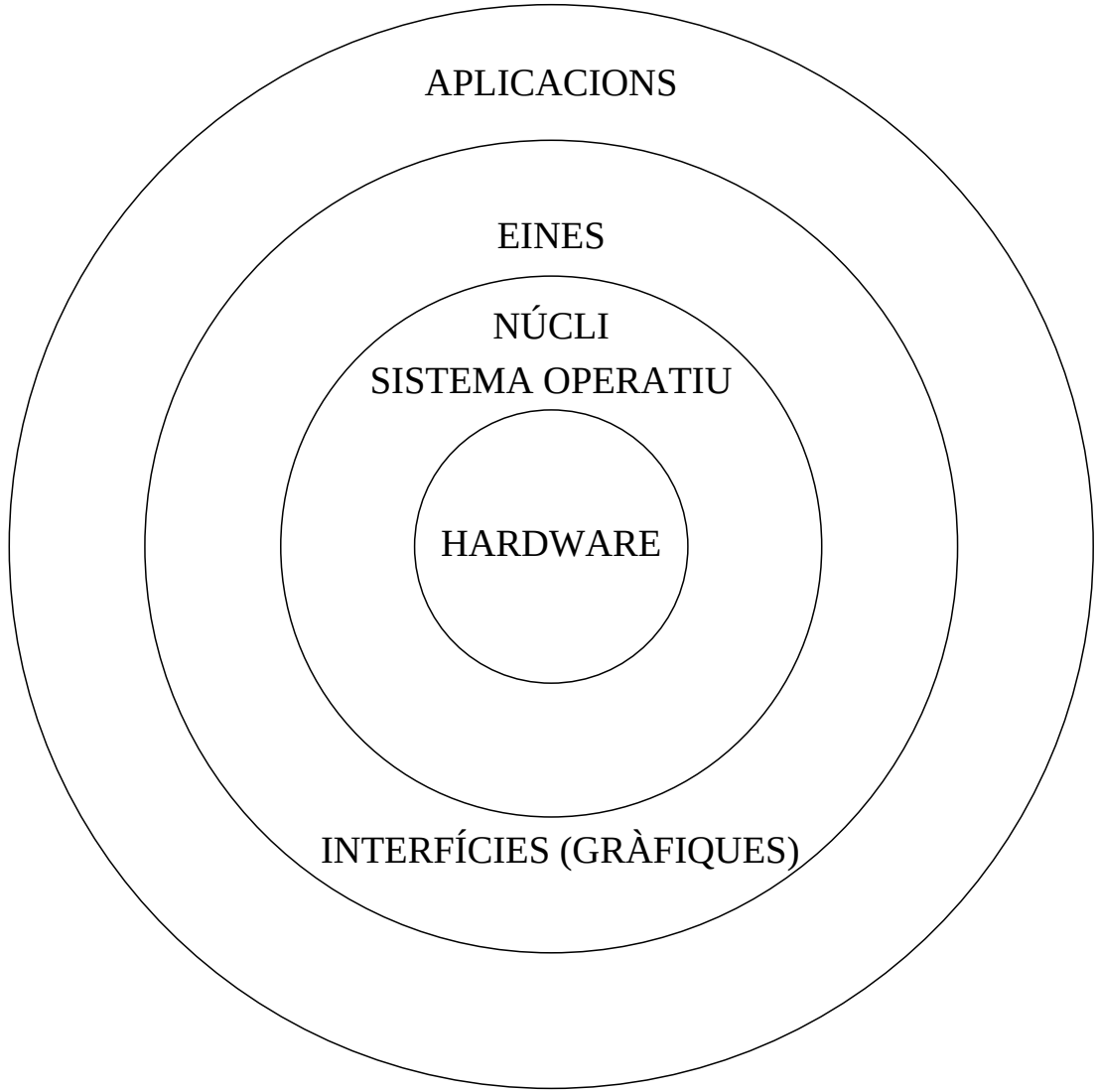
Introducció a Linux

Introducció al bash

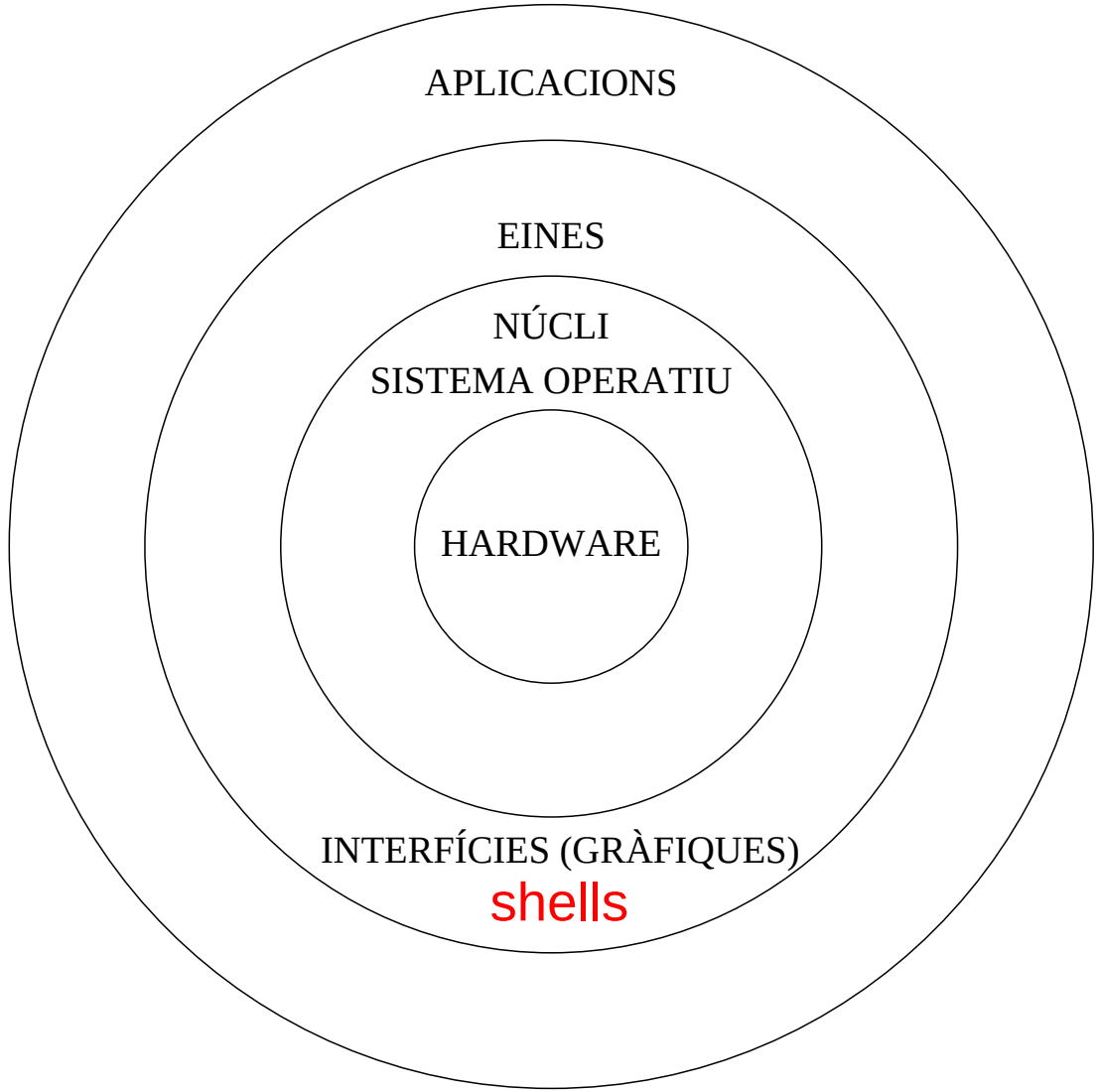
Josep Vilaplana

UPC/GIE

Nivells del sistema operatiu



Nivells del sistema operatiu



Un shell és un programa que interpreta la sintaxi de línies de caràcters rebudes per un canal d'entrada (per exemple, el teclat) i les fa a executar com a comandes del sistema (sempre que la línia sigui escrita correctament).
En Unix hi ha tres families de shells:

- sh: Bourne shell (bash). Va ser la primera

Un shell és un programa que interpreta la sintaxi de línies de caràcters rebudes per un canal d'entrada (per exemple, el teclat) i les fa a executar com a comandes del sistema (sempre que la línia sigui escrita correctament).
En Unix hi ha tres families de shells:

- sh: Bourne shell (bash). Va ser la primera
- csh: C shell (tcsh). S'apropa a la sintaxi del llenguatge C.

Un shell és un programa que interpreta la sintaxi de línies de caràcters rebudes per un canal d'entrada (per exemple, el teclat) i les fa a executar com a comandes del sistema (sempre que la línia sigui escrita correctament).
En Unix hi ha tres families de shells:

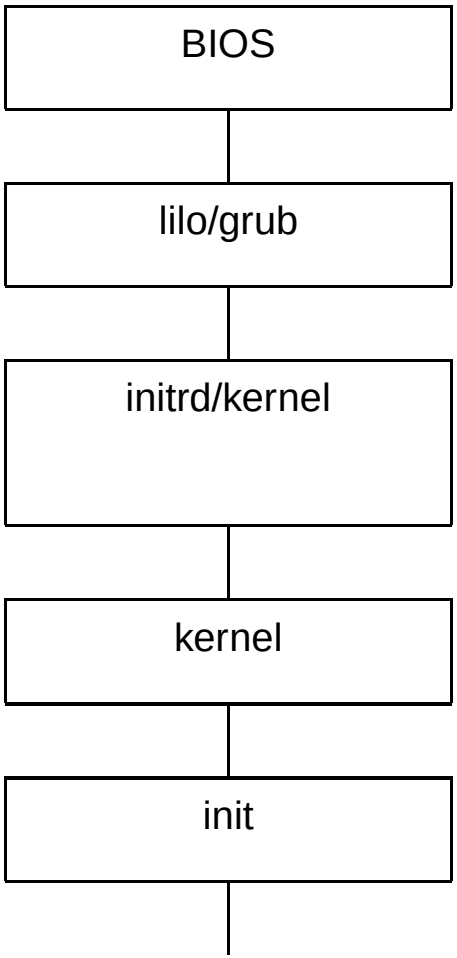
- sh: Bourne shell (bash). Va ser la primera
- csh: C shell (tcsh). S'apropa a la sintaxi del llenguatge C.
- ksh: Korn shell.

Un shell és un programa que interpreta la sintaxi de línies de caràcters rebudes per un canal d'entrada (per exemple, el teclat) i les fa a executar com a comandes del sistema (sempre que la línia sigui escrita correctament).
En Unix hi ha tres families de shells:

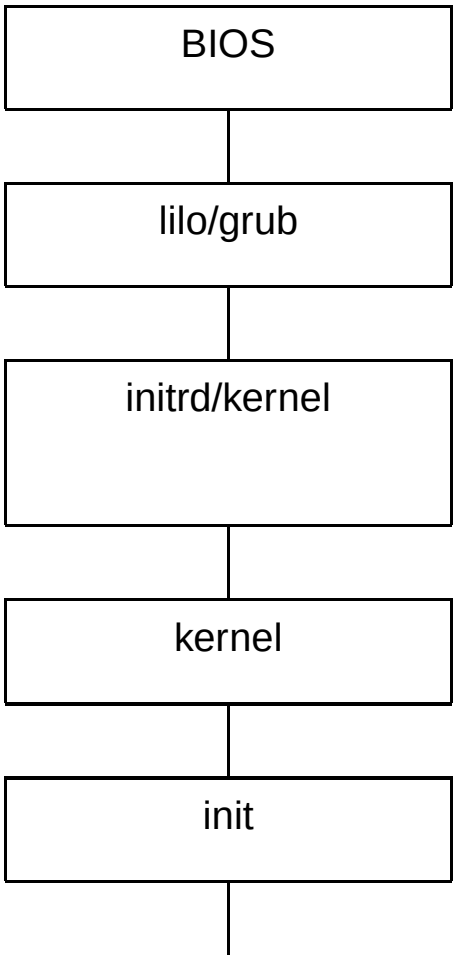
- sh: Bourne shell (bash). Va ser la primera
- csh: C shell (tcsh). S'apropa a la sintaxi del llenguatge C.
- ksh: Korn shell.

En el programari lliure, a més de les contribucions notables de bash i tcsh, s'han creat altres interprets de comandes que segueixen filosofies diferents i aporten altres funcionalitats: perl, tcl/tk, python, etc.

Inici Linux

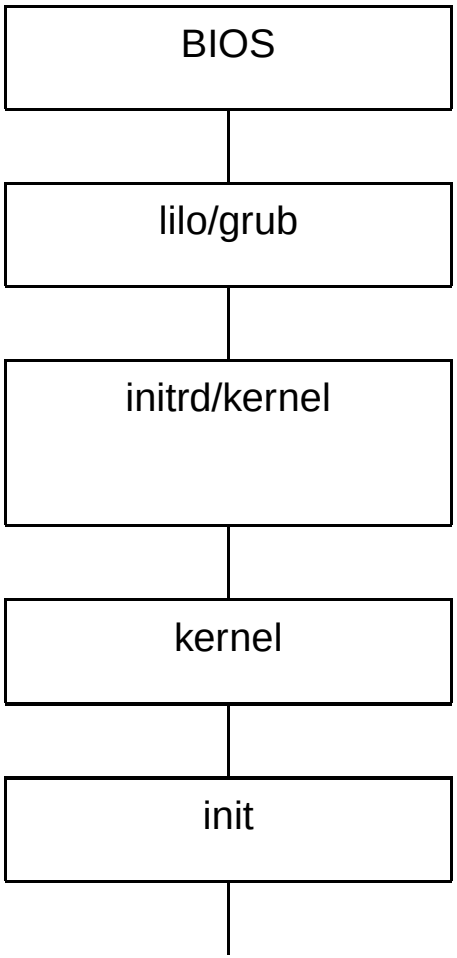


Inici Linux



Inicialitza dispositius de hardware

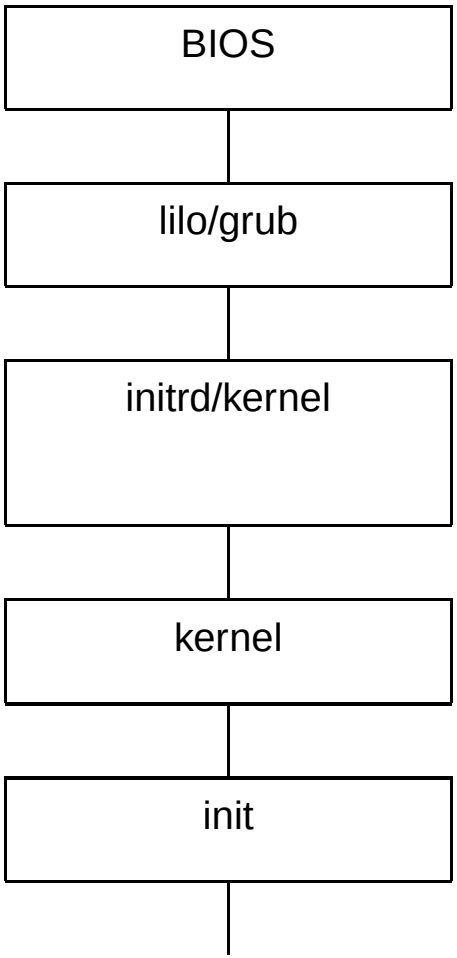
Inici Linux



Inicialitza dispositius de hardware

Carrega programa per carregar kernel

Inici Linux

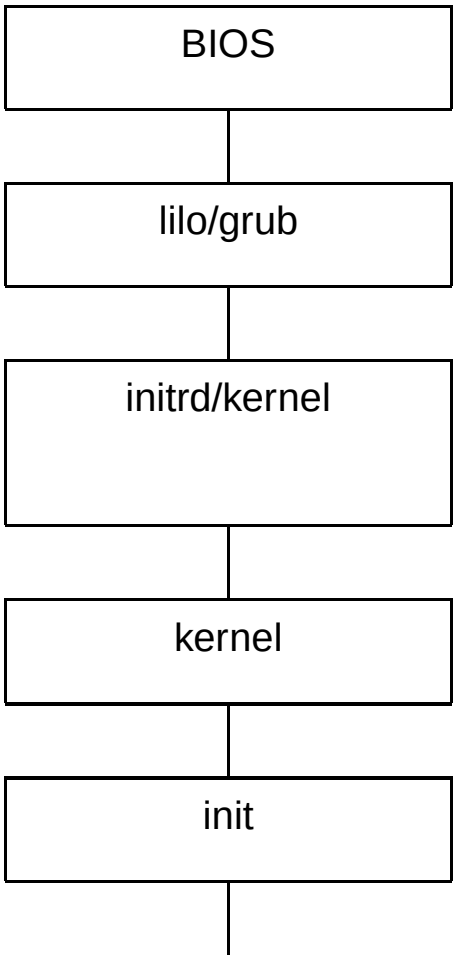


Inicialitza dispositius de hardware

Carrega programa per carregar kernel

Crea un ramdisk i carrega mòduls necessaris que no estan en el kernel per poder accedir al root autèntic

Inici Linux



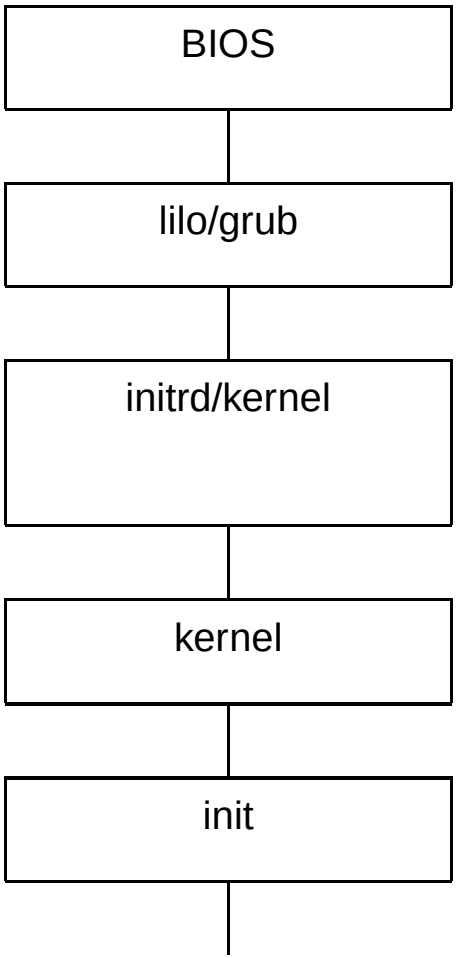
Inicialitza dispositius de hardware

Carrega programa per carregar kernel

Crea un ramdisk i carrega mòduls necessaris que no estan en el kernel per poder accedir al root autèntic

Completa inicialització dispositius i engega el procés *init*

Inici Linux



Inicialitza dispositius de hardware

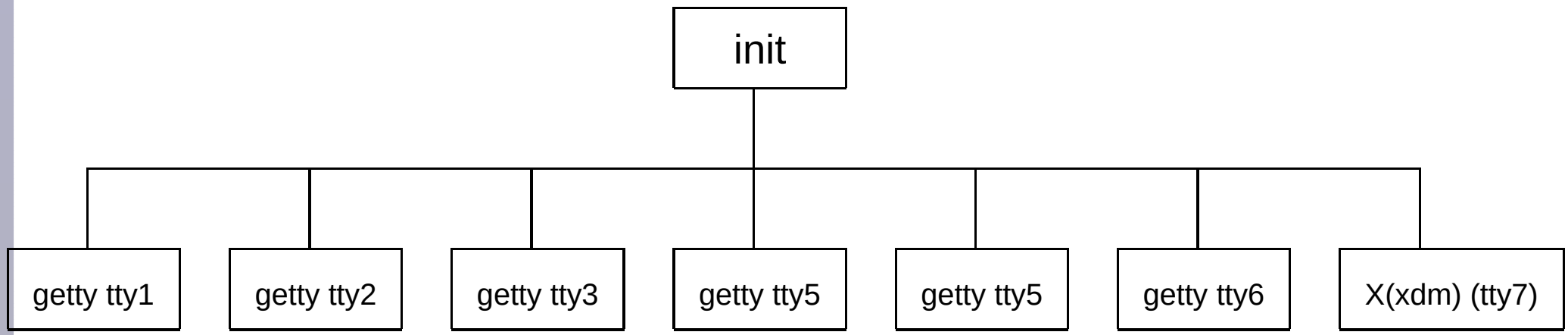
Carrega programa per carregar kernel

Crea un ramdisk i carrega mòduls necessaris que no estan en el kernel per poder accedir al root autèntic

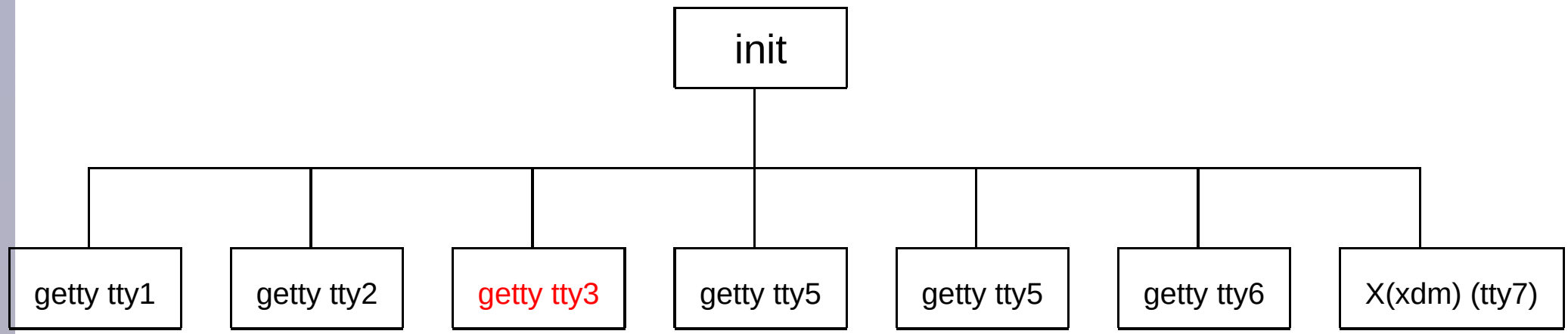
Completa inicialització dispositius i engega el procés *init*

Procés pare

En una situació freqüent tindrem (hi han diversos nivells d'arrancada) i d'acord a l'arxiu */etc/inittab* :



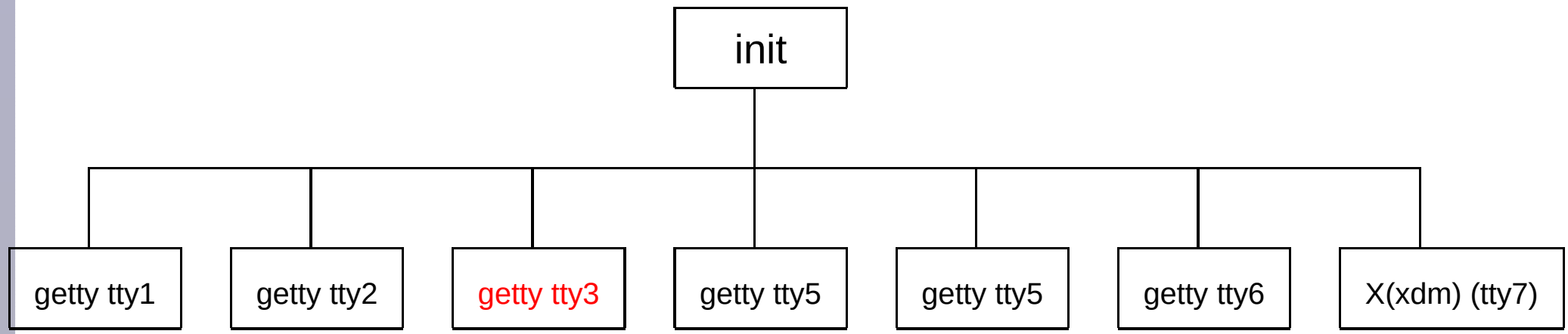
En una situació freqüent tindrem (hi han diversos nivells d'arrancada) i d'acord a l'arxiu */etc/inittab* :



Debian GNU/Linux fpc tty3

fpc login:

En una situació freqüent tindrem (hi han diversos nivells d'arrancada) i d'acord a l'arxiu */etc/inittab* :

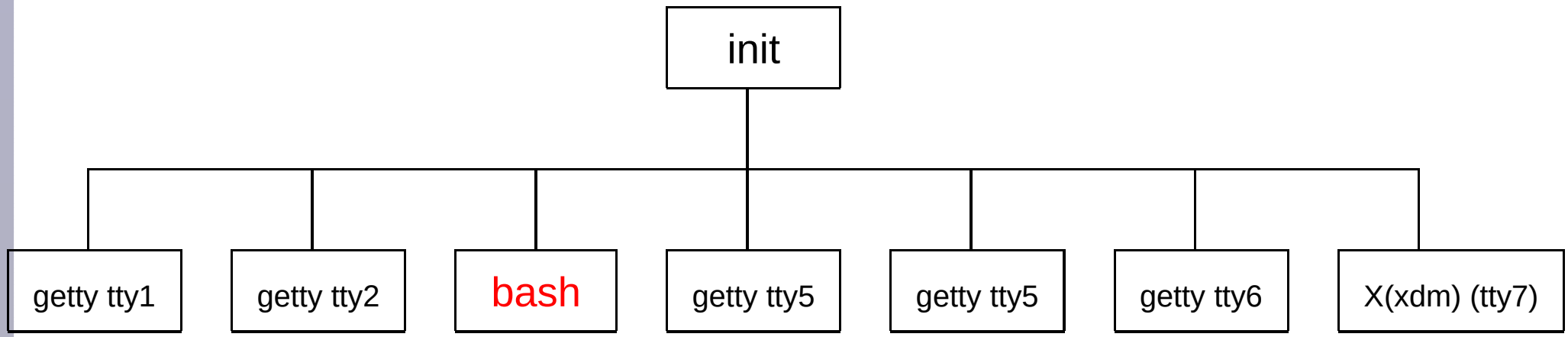



```
Debian GNU/Linux fpc tty3

fpc login: alumne

password:
```

En una situació freqüent tindrem (hi han diversos nivells d'arrancada) i d'acord a l'arxiu */etc/inittab* :




```
Debian GNU/Linux fpc tty3

fpc login: alumne

password:

$
```

En el cas del servidor d'*X* i *xdm (gdm o kdm)*, després del login correcte s'engega un gestor de sessió *(gnome*, *kde)* o un gestor de finestres *(icewm*, *fvwm*, *etc)* que activa clients pel servidor *X*.
L'activació d'un *xterm* des d'un gestor de sessions o de finestres permetrà accedir des d'una finestra a una shell. La finestra es comporta com si fós un terminal.

cat Concatena fitxers al canal de sortida (Visualitza fitxers text en pantalla).
Exemple: $ cat fitxer1 fitxer2 fitxer3

cat Concatena fitxers al canal de sortida (Visualitza fitxers text en pantalla).
Exemple: $ cat fitxer1 fitxer2 fitxer3

cp Còpia un fitxer(s) a un altre fitxer
Exemple: $ cp fitxer1 fitxer4

cat Concatena fitxers al canal de sortida (Visualitza fitxers text en pantalla).
Exemple: $ cat fitxer1 fitxer2 fitxer3

cp Còpia un fitxer(s) a un altre fitxer
Exemple: $ cp fitxer1 fitxer4

mv Mou un fitxer(s) cap a un altre fitxer (Canvi nom fitxer, directori, etc)
Exemple: $ mv fitxer1 fitxer5

Operacions relatives a fitxers

cat Concatena fitxers al canal de sortida (Visualitza fitxers text en pantalla).
Exemple: $ cat fitxer1 fitxer2 fitxer3

cp Còpia un fitxer(s) a un altre fitxer
Exemple: $ cp fitxer1 fitxer4

mv Mou un fitxer(s) cap a un altre fitxer (Canvi nom fitxer, directori, etc)
Exemple: $ mv fitxer1 fitxer5

rm Esborra fitxer(s)
Exemple: $ rm -f fitxer5

chmod Canvia proteccions fitxer(s)

Exemple: $ chmod a-w fitxer4

Exemple: $ chmod g-r fitxer4

Exemple: $ chmod u-r fitxer4

Exemple: $ chmod 777 fitxer4

chmod Canvia proteccions fitxer(s)

Exemple: $ chmod a-w fitxer4

Exemple: $ chmod g-r fitxer4

Exemple: $ chmod u-r fitxer4

Exemple: $ chmod 777 fitxer4

chown Canvia propietari fitxer(s)

Exemple: $ chown lluis fitxer4

Operacions relatives a fitxers

chmod Canvia proteccions fitxer(s)
Exemple: $ chmod a-w fitxer4
Exemple: $ chmod g-r fitxer4
Exemple: $ chmod u-r fitxer4
Exemple: $ chmod 777 fitxer4

chown Canvia propietari fitxer(s)
Exemple: $ chown lluis fitxer4

chgrp Canvia grup propietari de fitxer(s)
Exemple: $ chgrp alumnes fitxer4

Operacions relatives a fitxers

chmod Canvia proteccions fitxer(s)
Exemple: $ chmod a-w fitxer4
Exemple: $ chmod g-r fitxer4
Exemple: $ chmod u-r fitxer4
Exemple: $ chmod 777 fitxer4

chown Canvia propietari fitxer(s)
Exemple: $ chown lluis fitxer4

chgrp Canvia grup propietari de fitxer(s)
Exemple: $ chgrp alumnes fitxer4

umask Estableix proteccions per defecte
Exemple: $ umask 022

Operacions relatives a fitxers

cmp Compara dos fitxers

Exemple: $ cmp fitxer1 fitxer2

Operacions relatives a fitxers

cmp Compara dos fitxers
Exemple: $ cmp fitxer1 fitxer2

diff Compara dos fitxers
Exemple: $ diff fitxer4 fitxer2
Exemple: $ diff -r dir1 dir2

Operacions relatives a fitxers

cmp Compara dos fitxers
Exemple: $ cmp fitxer1 fitxer2

diff Compara dos fitxers
Exemple: $ diff fitxer4 fitxer2
Exemple: $ diff -r dir1 dir2

find Cerca de fitxers que responen a un criteri
Exemple: $ find . -name "*.dat" -print

Operacions relatives a fitxers

cmp Compara dos fitxers
Exemple: $ cmp fitxer1 fitxer2

diff Compara dos fitxers
Exemple: $ diff fitxer4 fitxer2
Exemple: $ diff -r dir1 dir2

find Cerca de fitxers que responen a un criteri
Exemple: $ find . -name "*.dat" -print

od Exposa contingut fitxer en octal o hexadecimal en stdout
Exemple: $ od -c fitxer2

ls Visualitza els fitxers existents d'un o més directoris

Exemple: $ ls

Exemple: $ ls -la

ls Visualitza els fitxers existents d'un o més directoris
Exemple: $ ls
Exemple: $ ls -la

pwd Visualitza els directori de treball per defecte
Exemple: $ pwd

Operacions relatives a directoris

ls Visualitza els fitxers existents d'un o més directoris
Exemple: $ ls
Exemple: $ ls -la

pwd Visualitza els directori de treball per defecte
Exemple: $ pwd

mkdir Crea un directori
Exemple: $ mkdir dir1

GIE/ICE

Operacions relatives a directoris

ls Visualitza els fitxers existents d'un o més directoris
Exemple: $ ls
Exemple: $ ls -la

pwd Visualitza els directori de treball per defecte
Exemple: $ pwd

mkdir Crea un directori
Exemple: $ mkdir dir1

rmdir Esborra un directori
Exemple: $ rmdir dir1

Operacions sobre processos

ps	Status de processos
kill	Para bruscament un proces
nice	Executa comanda amb baixa prioritat
<ctrl-z>	Suspén l'execució de la comanda actual
bg	Continua execució comanda suspesa en el background.
fg	Torna a activar comanda suspesa.
sleep	Suspèn l'execució durant n segons.
at	Encua una comanda per ser executada a partir d'un temps especificat. Examina la cua de treballs pendents Esborra de la cua un treball.
nohup	Executa una comanda inmune a qualsevol senyal que intenti aturarla. Si hom surt de la sessió, la comanda seguirà executant-se
batch	Executa una comanda quan la càrrega del sistema ho permeti. El canal de sortida standard serà el correu electronic

Variables d'entorn

- El funcionament del bash es pot condicionar d'acord a unes variables establertes.

Variables d'entorn

- El funcionament del bash es pot condicionar d'acord a unes variables establertes.
- Per exemple, les variables $PS1$, $PS2$ i $PS3$ controlen la visualització del *prompt*. La variable $HOME$ indica quin directori de treball es té per defecte a l'entrar mitjançant login. La variable $TERM$ indica en quin tipus de terminal s'està treballant.

Variables d'entorn

- El funcionament del bash es pot condicionar d'acord a unes variables establertes.
- Per exemple, les variables $PS1$, $PS2$ i $PS3$ controlen la visualització del *prompt*. La variable $HOME$ indica quin directori de treball es té per defecte a l'entrar mitjançant login. La variable $TERM$ indica en quin tipus de terminal s'està treballant.
- Només hi ha un tipus de variable: cadena de caràcters.

Variables d'entorn

- El funcionament del bash es pot condicionar d'acord a unes variables establertes.
- Per exemple, les variables $PS1$, $PS2$ i $PS3$ controlen la visualització del *prompt*. La variable $HOME$ indica quin directori de treball es té per defecte a l'entrar mitjançant login. La variable $TERM$ indica en quin tipus de terminal s'està treballant.
- Només hi ha un tipus de variable: cadena de caràcters.
- No cal declararles: El primer ús que s'en faci, es crea la variable.

Variables d'entorn

- El funcionament del bash es pot condicionar d'acord a unes variables establertes.
- Per exemple, les variables $PS1$, $PS2$ i $PS3$ controlen la visualització del *prompt*. La variable $HOME$ indica quin directori de treball es té per defecte a l'entrar mitjançant login. La variable $TERM$ indica en quin tipus de terminal s'està treballant.
- Només hi ha un tipus de variable: cadena de caràcters.
- No cal declararles: El primer ús que s'en faci, es crea la variable.
- Hi han mecanismes per fer que les cadenes de caràcters dígits i simbols adients puguin interpretarse com una expressió numérica a avaluar. El resultat serà però una altre cadena de caràcters.

Variables d'entorn

- El funcionament del bash es pot condicionar d'acord a unes variables establertes.
- Per exemple, les variables $PS1$, $PS2$ i $PS3$ controlen la visualització del *prompt*. La variable $HOME$ indica quin directori de treball es té per defecte a l'entrar mitjançant login. La variable $TERM$ indica en quin tipus de terminal s'està treballant.
- Només hi ha un tipus de variable: cadena de caràcters.
- No cal declararles: El primer ús que s'en faci, es crea la variable.
- Hi han mecanismes per fer que les cadenes de caràcters dígits i simbols adients puguin interpretarse com una expressió numérica a avaluar. El resultat serà però una altre cadena de caràcters.
- El contingut d'una variable pot posar-se en una línia de comandes.

Variables d'entorn

- El funcionament del bash es pot condicionar d'acord a unes variables establertes.
- Per exemple, les variables $PS1$, $PS2$ i $PS3$ controlen la visualització del *prompt*. La variable $HOME$ indica quin directori de treball es té per defecte a l'entrar mitjançant login. La variable $TERM$ indica en quin tipus de terminal s'està treballant.
- Només hi ha un tipus de variable: cadena de caràcters.
- No cal declararles: El primer ús que s'en faci, es crea la variable.
- Hi han mecanismes per fer que les cadenes de caràcters dígits i simbols adients puguin interpretarse com una expressió numérica a avaluar. El resultat serà però una altre cadena de caràcters.
- El contingut d'una variable pot posar-se en una línia de comandes.
- El contingut d'una variable pot llegir-lo qualsevol programa de forma que pugui adaptar el seu funcionament a l'entorn en que s'executa.

- Per crear i assignar un valor a una variable cal fer: nomVariable=valor. Per exemple,

  $ LA_VARIABLE=valor

  $ LA_VARIABLE="valor1 valor2 valor3"

  Aquest darrer cas permet assignar vàries paraules. Després es veurà que la shell pot interpretar la variable com una llista de valors.

Variables d'entorn

- Per crear i assignar un valor a una variable cal fer: nomVariable=valor. Per exemple,

  $ LA_VARIABLE=valor

  $ LA_VARIABLE="valor1 valor2 valor3"

  Aquest darrer cas permet assignar vàries paraules. Després es veurà que la shell pot interpretar la variable com una llista de valors.

- Per posar el seu contingut en la línia de comandes cal posar el nom de la variable després del caràcter $. Per exemple,

  $ echo $LA_VARIABLE

  $ LA_VARIABLE=1

  $ LA_VARIABLE=$LA_VARIABLE+1

  En el darrer exemple, la variable contindrà "1+1".

Variables d'entorn

- Per crear i assignar un valor a una variable cal fer: nomVariable=valor. Per exemple,

  $ LA_VARIABLE=valor

  $ LA_VARIABLE="valor1 valor2 valor3"

  Aquest darrer cas permet assignar vàries paraules. Després es veurà que la shell pot interpretar la variable com una llista de valors.

- Per posar el seu contingut en la línia de comandes cal posar el nom de la variable després del caràcter $. Per exemple,

  $ echo $LA_VARIABLE

  $ LA_VARIABLE=1

  $ LA_VARIABLE=$LA_VARIABLE+1

  En el darrer exemple, la variable contindrà "1+1".

- Per avaluar una cadena de caràcters, hi han diverses formes. Per exemple,

  $ LA_VARIABLE=$(($LA_VARIABLE+1))

  la variable contindrà "3".

Execució de comandes en bash

- Les comandes en bash poden ser internes o externes. Per exemple, comandes com *cd*, *exit*, *pwd*, i *echo* les executa la pròpia shell.

Execució de comandes en bash

- Les comandes en bash poden ser internes o externes. Per exemple, comandes com *cd*, *exit*, *pwd*, i *echo* les executa la pròpia shell.
- Comandes com *ls*, *cp*, *mv* són programes externs.

Execució de comandes en bash

- Les comandes en bash poden ser internes o externes. Per exemple, comandes com *cd*, *exit*, *pwd*, i *echo* les executa la pròpia shell.
- Comandes com *ls*, *cp*, *mv* són programes externs.
- Per saber on són, la shell consulta la variable *PATH* per saber en quins directoris es poden trobar els programes a executar.

```
$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games
```

GIE/ICE

Execució de comandes en bash

- Les comandes en bash poden ser internes o externes. Per exemple, comandes com *cd*, *exit*, *pwd*, i *echo* les executa la pròpia shell.
- Comandes com *ls*, *cp*, *mv* són programes externs.
- Per saber on són, la shell consulta la variable *PATH* per saber en quins directoris es poden trobar els programes a executar.

  $ echo $PATH

  /usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games
- El contingut de la variable és una llista de camins de directoris separats per ":".

GIE/ICE

Execució de comandes en bash

- Les comandes en bash poden ser internes o externes. Per exemple, comandes com *cd*, *exit*, *pwd*, i *echo* les executa la pròpia shell.
- Comandes com *ls*, *cp*, *mv* són programes externs.
- Per saber on són, la shell consulta la variable *PATH* per saber en quins directoris es poden trobar els programes a executar.

  $ echo $PATH

  /usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games
- El contingut de la variable és una llista de camins de directoris separats per ":".
- Per afegir altres directoris d'interés, es pot fer

  $ PATH=$PATH":."

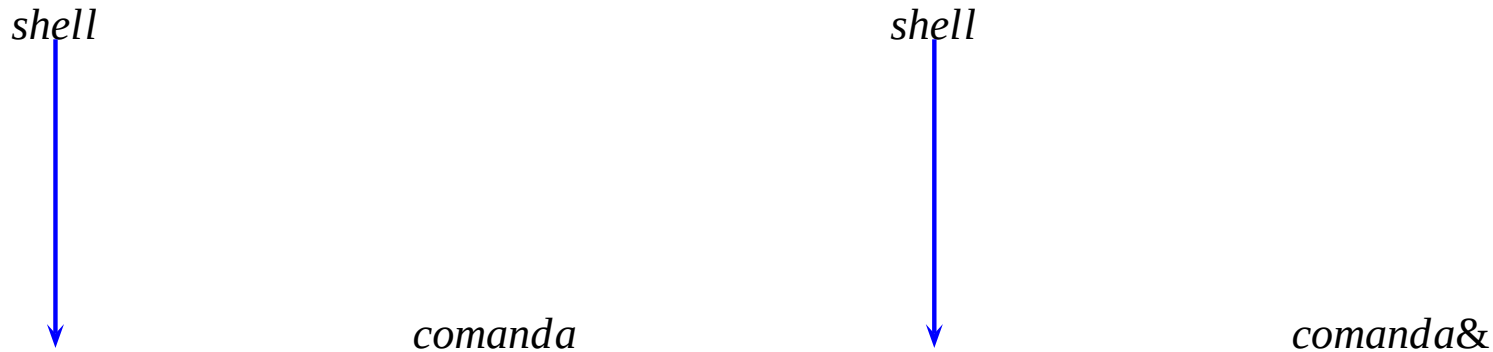
  En aquest cas s'afegeix el directori per defecte de treball.

Execució de comandes en bash

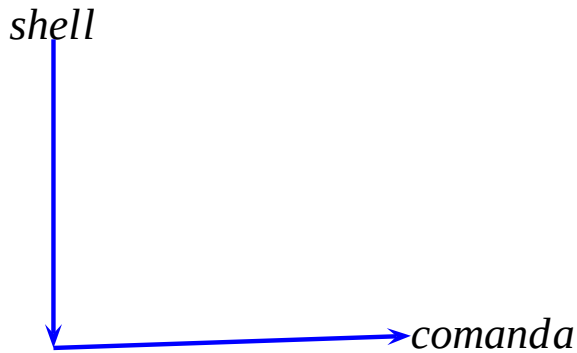
shell

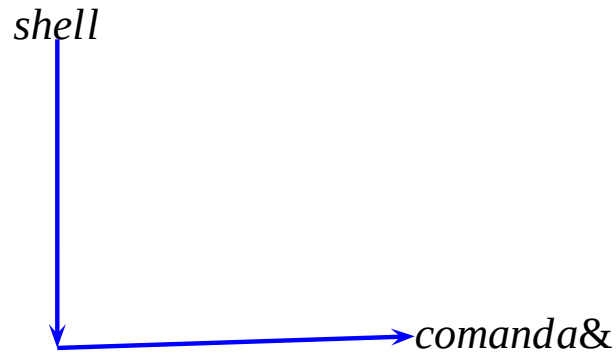
shell

Execució de comandes en bash

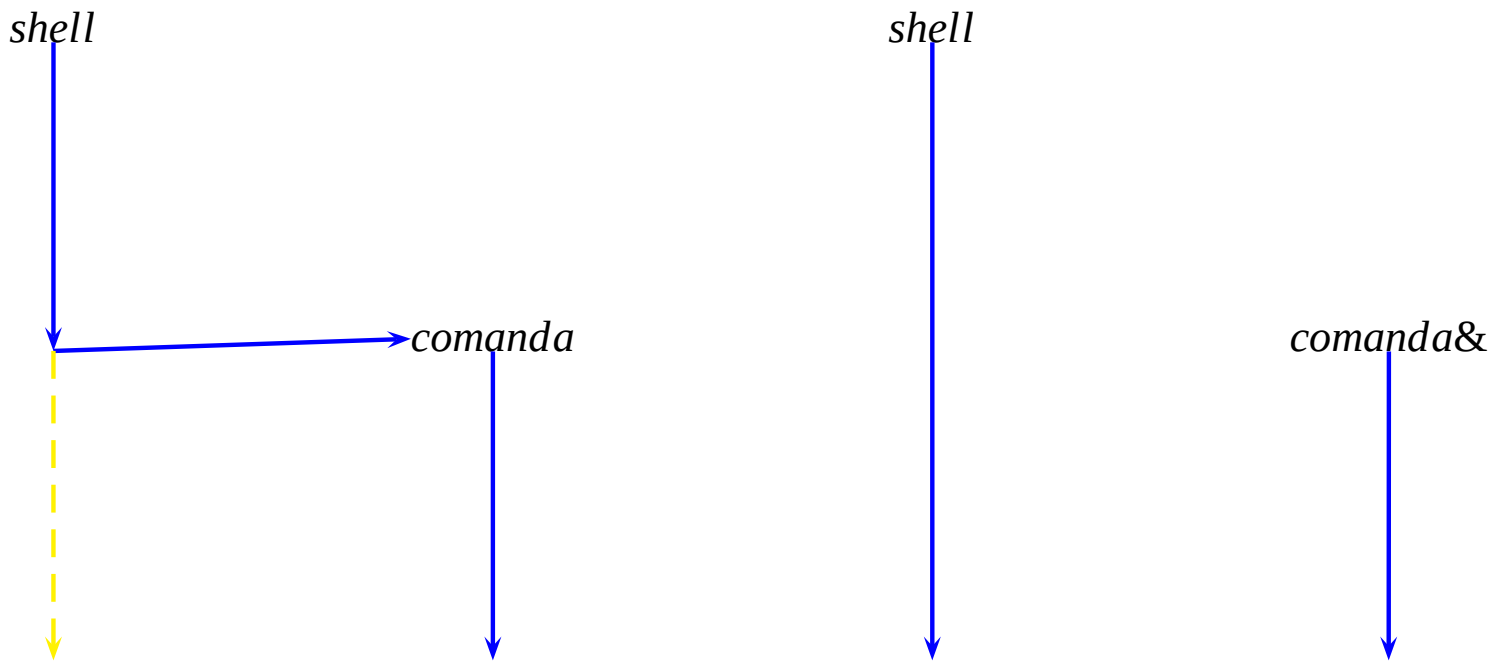


Execució de comandes en bash

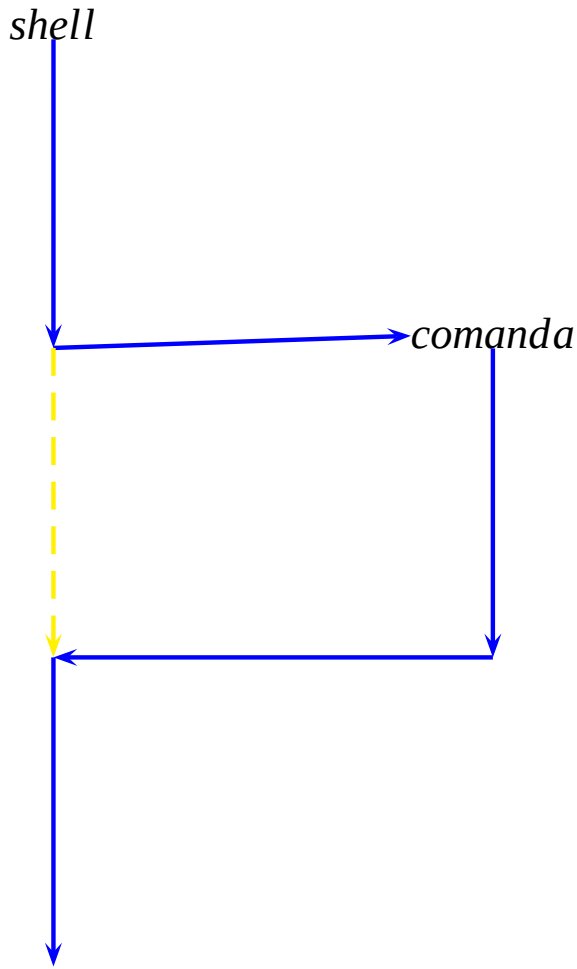


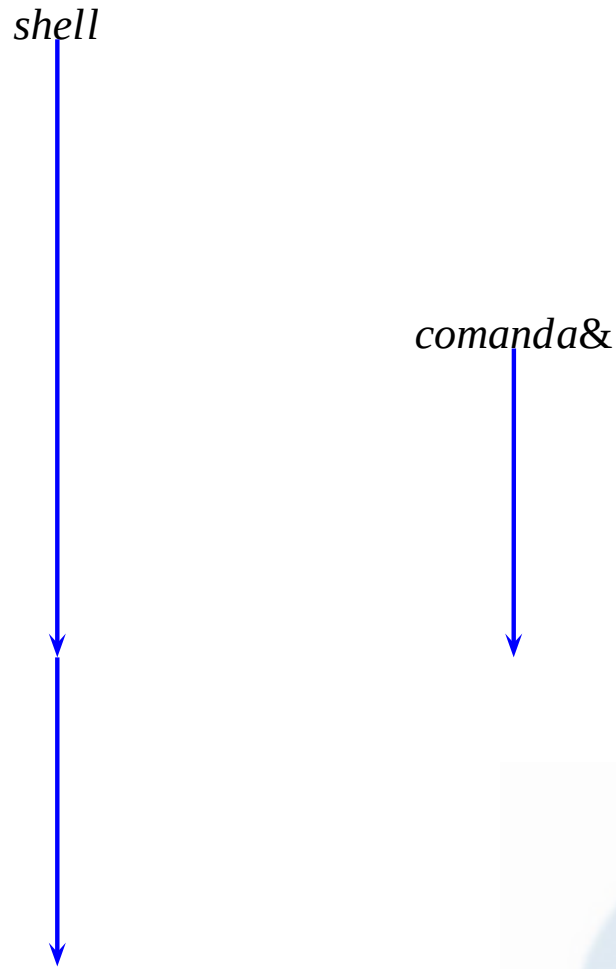


Execució de comandes en bash



Execució de comandes en bash





Els dos processos són bash.

Estructura d'una comanda en bash

- línia $\equiv$ seqüència de comandes acabada amb $< return >$.

Estructura d'una comanda en bash

- línia $\equiv$ seqüència de comandes acabada amb $< return >$.
- comanda $\equiv$ seqüència de paraules acabada amb $< return >$ o caràcters operadors de control.

Estructura d'una comanda en bash

- línia $\equiv$ seqüència de comandes acabada amb $< return >$.
- comanda $\equiv$ seqüència de paraules acabada amb $< return >$ o caràcters operadors de control.
- paraula $\equiv$ seqüència de caràcters alfabetics, numérics i _ finalitzada per un o més espais, o algun(s) caràcter(s) de control.
  També pot ser una seqüència de caràcters començada i acabada pel caràcter ". En aquest darrer cas, s'admeten espais dins de la paraula i d'altres caràcters reservats.

Estructura d'una comanda en bash

- línia $\equiv$ seqüència de comandes acabada amb $< return >$.
- comanda $\equiv$ seqüència de paraules acabada amb $< return >$ o caràcters operadors de control.
- paraula $\equiv$ seqüència de caràcters alfabetics, numérics i _ finalitzada per un o més espais, o algun(s) caràcter(s) de control.
  També pot ser una seqüència de caràcters començada i acabada pel caràcter ". En aquest darrer cas, s'admeten espais dins de la paraula i d'altres caràcters reservats.
- Les paraules de la línia s'anomenen arguments i s'enumeran a partir del 0.

$$\boxed{arg_0} \quad \boxed{arg_1} \quad \boxed{arg_2} \quad \boxed{\ldots}$$

Estructura d'una comanda en bash

- línia $\equiv$ seqüència de comandes acabada amb $< return >$.
- comanda $\equiv$ seqüència de paraules acabada amb $< return >$ o caràcters operadors de control.
- paraula $\equiv$ seqüència de caràcters alfabetics, numérics i _ finalitzada per un o més espais, o algun(s) caràcter(s) de control.
  També pot ser una seqüència de caràcters començada i acabada pel caràcter ". En aquest darrer cas, s'admeten espais dins de la paraula i d'altres caràcters reservats.
- Les paraules de la línia s'anomenen arguments i s'enumeran a partir del 0.

  $\boxed{arg_0}$ $\boxed{arg_1}$ $\boxed{arg_2}$ $\boxed{\ldots}$
- Caràcters separadors de paraules: `| & ; ( ) < > || && ;; <return>`.

Estructura d'una comanda en bash

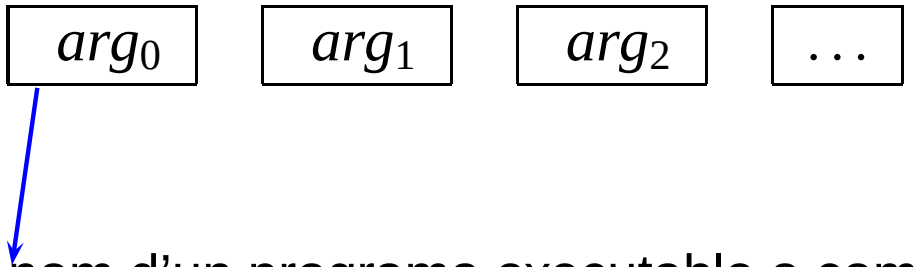
- línia $\equiv$ seqüència de comandes acabada amb $< return >$.
- comanda $\equiv$ seqüència de paraules acabada amb $< return >$ o caràcters operadors de control.
- paraula $\equiv$ seqüència de caràcters alfabetics, numérics i _ finalitzada per un o més espais, o algun(s) caràcter(s) de control.
  També pot ser una seqüència de caràcters començada i acabada pel caràcter ". En aquest darrer cas, s'admeten espais dins de la paraula i d'altres caràcters reservats.
- Les paraules de la línia s'anomenen arguments i s'enumeran a partir del 0.

  $\boxed{arg_0}$ $\boxed{arg_1}$ $\boxed{arg_2}$ $\boxed{\ldots}$
- Caràcters separadors de paraules: `| & ; ( ) < > || && ;; <return>`.
- Paraules reservades (com a primera paraula o tercera en les construccions de for i case): `! case do done elif else esac fi for function if in select then until while time [[ ]]`

Estructura d'una comanda en bash

- arg_0 arg_1 arg_2 . . .

Estructura d'una comanda en bash

- 

- nom d'un programa executable o comanda interna de bash

GIE/ICE

Estructura d'una comanda en bash

- nom d'un programa executable o comanda interna de bash
- arguments que rebrà el programa sol·licitat

GIE/ICE

Programes per bash

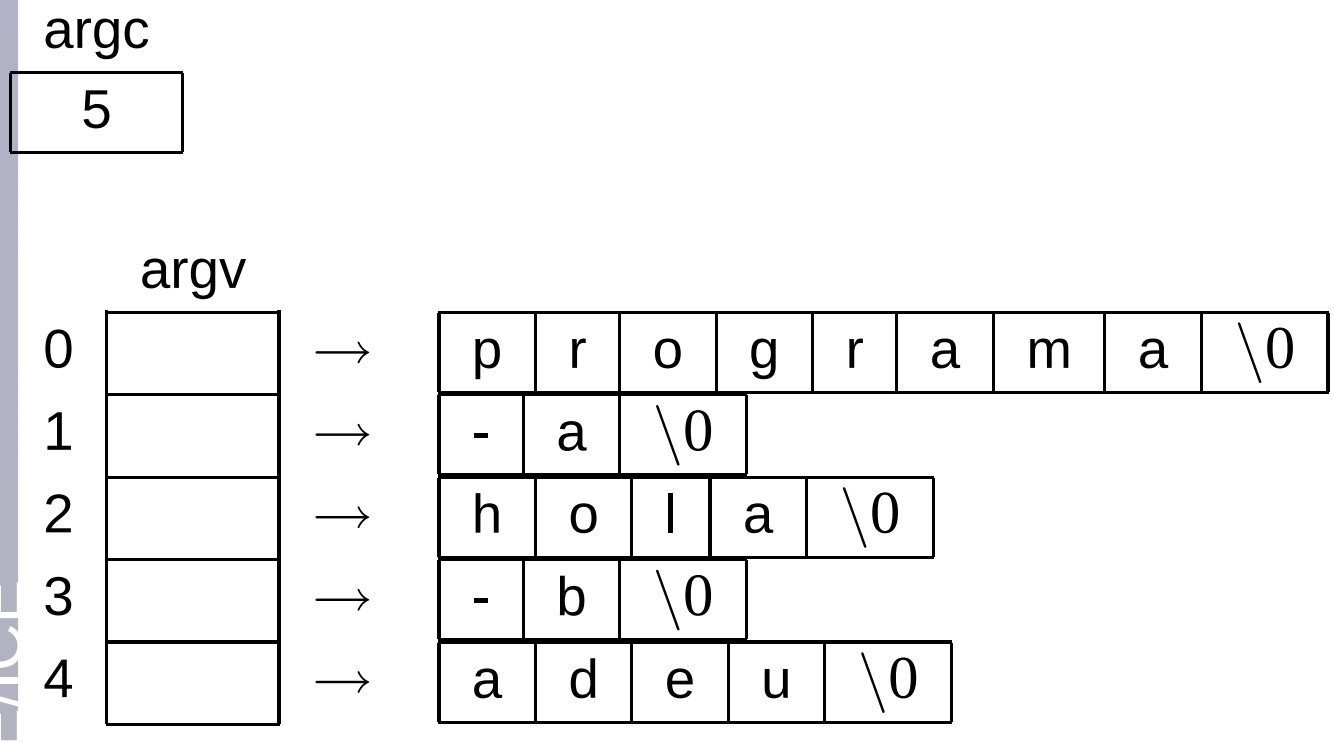
- Qualsevol programa pot rebre els arguments que hi han a la línia.
- De fet, per l'interpret de bash, un programa executable és una funció.
- Un programa en C es declararà:

```
int main(int argc, char * argv[]) {
    ...

    return .../* Enter indicant si ha anat bé o malament*/
}
```

Exemple programa C per bash

programa -a hola -b adeu



Exemple programa C per bash (2)

programa -a hola -b adeu

```
if (argc>1) {/* argument 0 es nom programa */
  i = 1;
  while (i<argc) {
    if (argv[i][0]=='-') {
      if (argv[i][1]=='a')        {/* Processem opcio a */
      } else if (argv[i][1]=='b') {/* Processem opcio b */
      } else { errorf(argv[0], "opcio invalida");
      }
    }
    i = i + 1; /* i pot haver canviat en les branques */
  }
} else errorf(argv[0], "falten arguments");
```

Expansió de la línia de bash

- La línia que arriba al programa no és la mateixa que es veu des del terminal.
- Exemple:
  La comanda *echo* escriu al canal de sortida tots els arguments que si l'han passat.

```
$ echo hola                mira
hola mira
$ echo "hola                 mira"
hola                mira
```

Expansió de la línia de bash

- La línia que arriba al programa no és la mateixa que es veu des del terminal.
- Exemple:
  La comanda *echo* escriu al canal de sortida tots els arguments que si l'han passat.
  Però,

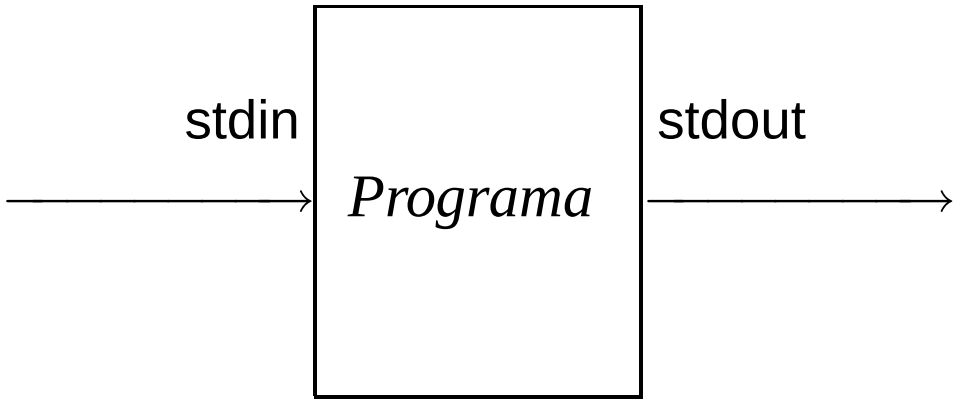
```
$ echo *
bin boot dev etc home lib lost+found  mnt proc root sbin srv
sys tmp usr var
```

  ... que són tots els fitxers visibles i presents en un directori de treball.
  En canvi,

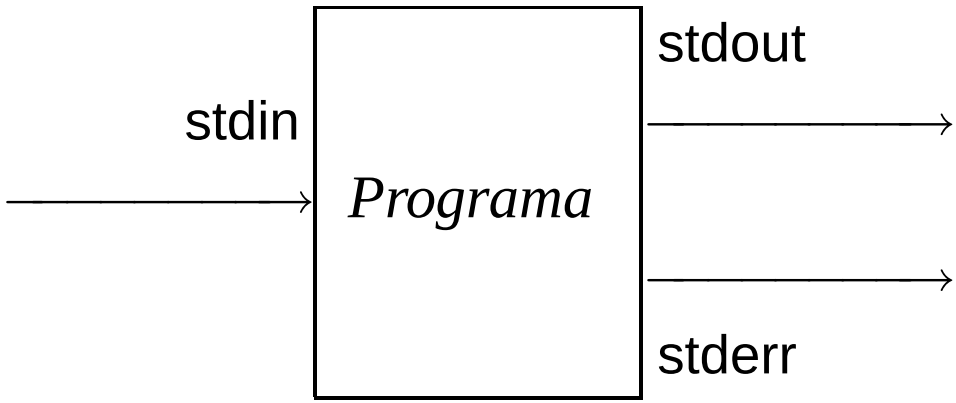
```
$ echo "*"
*
```

Canals estandard de comunicació dels programes i/o comandes

El model de programa que pot gestionar i/o reutilitzar la shell per connectar a altres dispositius o a altres programes que segueixin el mateix model passa per l'us de canals estandard d'entrada/sortida.

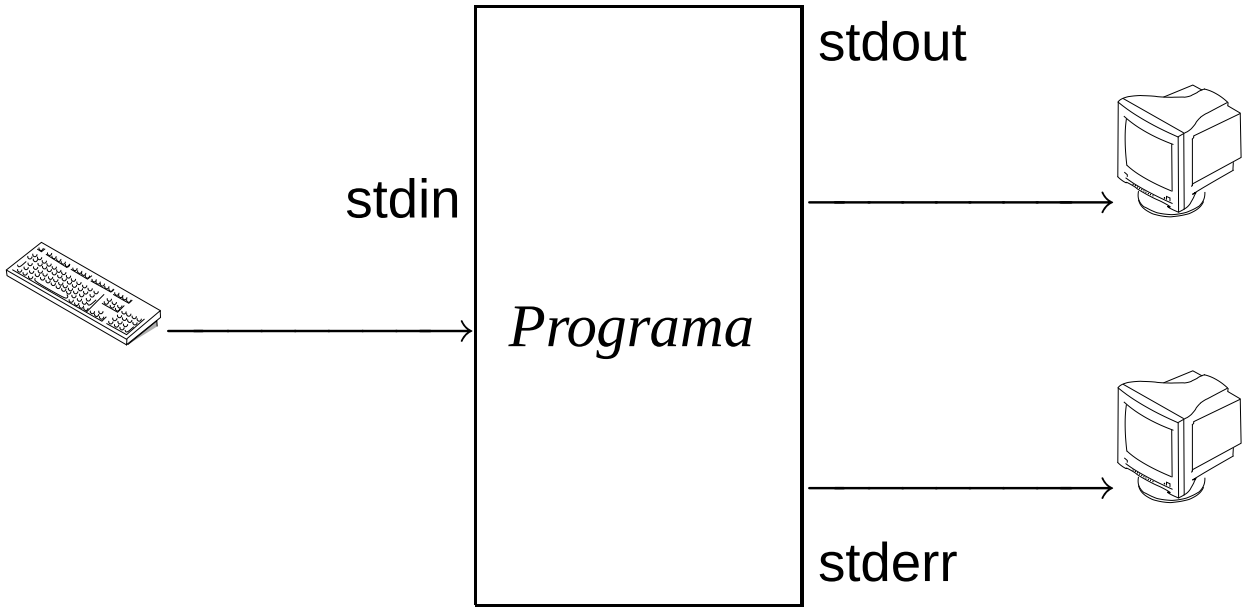


El model de programa que pot gestionar i/o reutilitzar la shell per connectar a altres dispositius o a altres programes que segueixin el mateix model passa per l'us de canals estandard d'entrada/sortida.



Canals estandard de comunicació dels programes i/o comandes

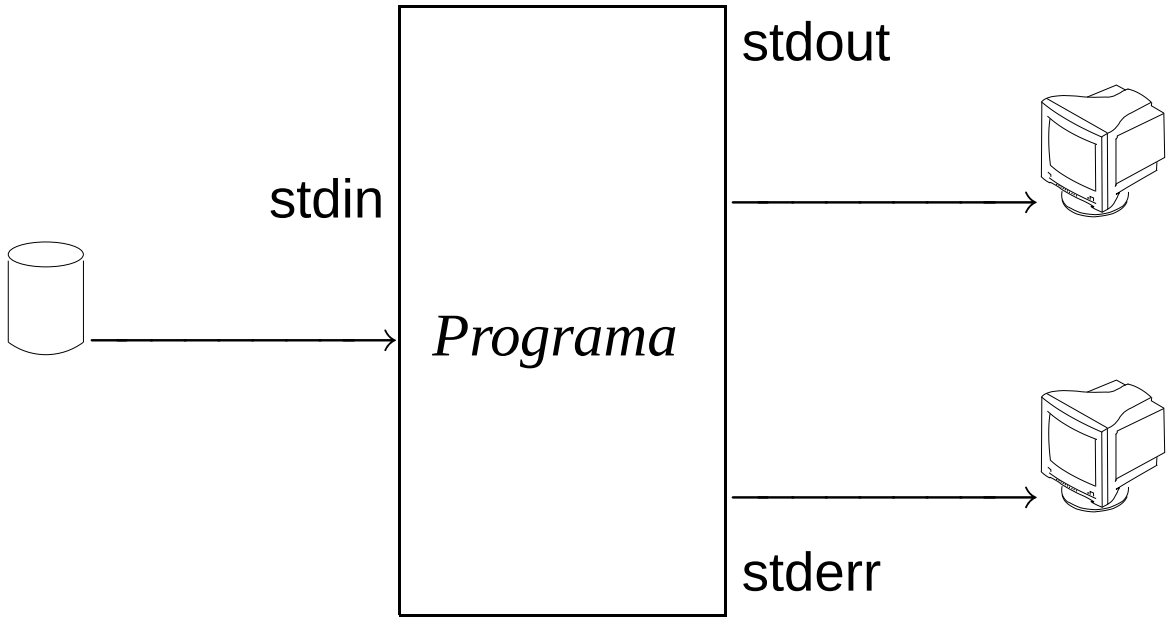
El model de programa que pot gestionar i/o reutilitzar la shell per connectar a altres dispositius o a altres programes que segueixin el mateix model passa per l'us de canals estandard d'entrada/sortida.




```
$ ./Programa
```

Canals estandard de comunicació dels programes i/o comandes

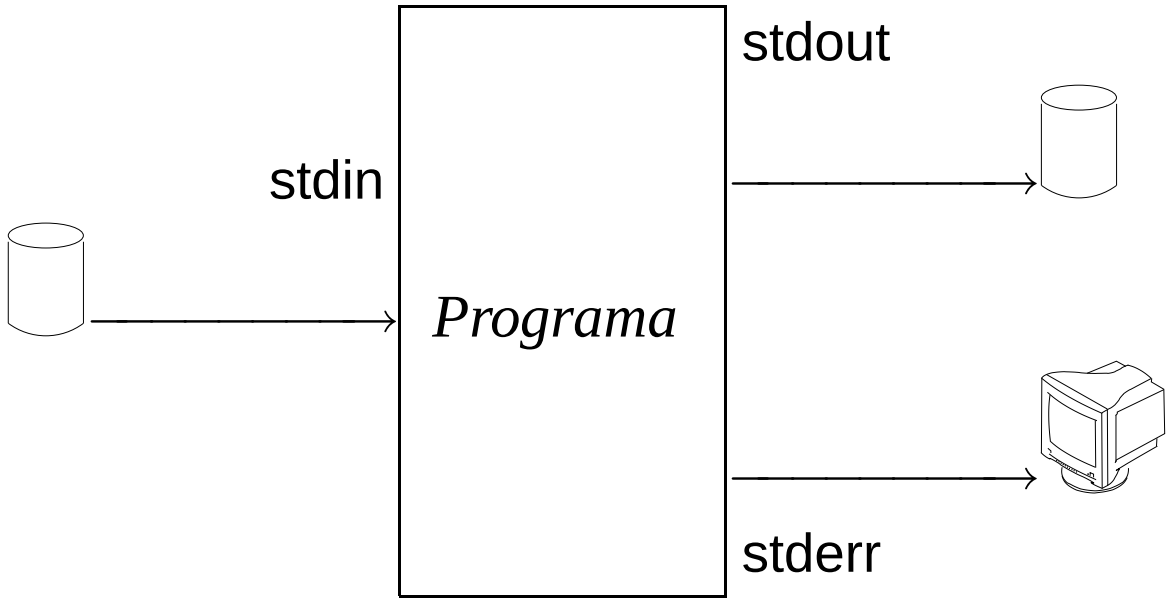
El model de programa que pot gestionar i/o reutilitzar la shell per connectar a altres dispositius o a altres programes que segueixin el mateix model passa per l'us de canals estandard d'entrada/sortida.




```
$ ./Programa < fitxerEntrada.dat
```

Canals estandard de comunicació dels programes i/o comandes

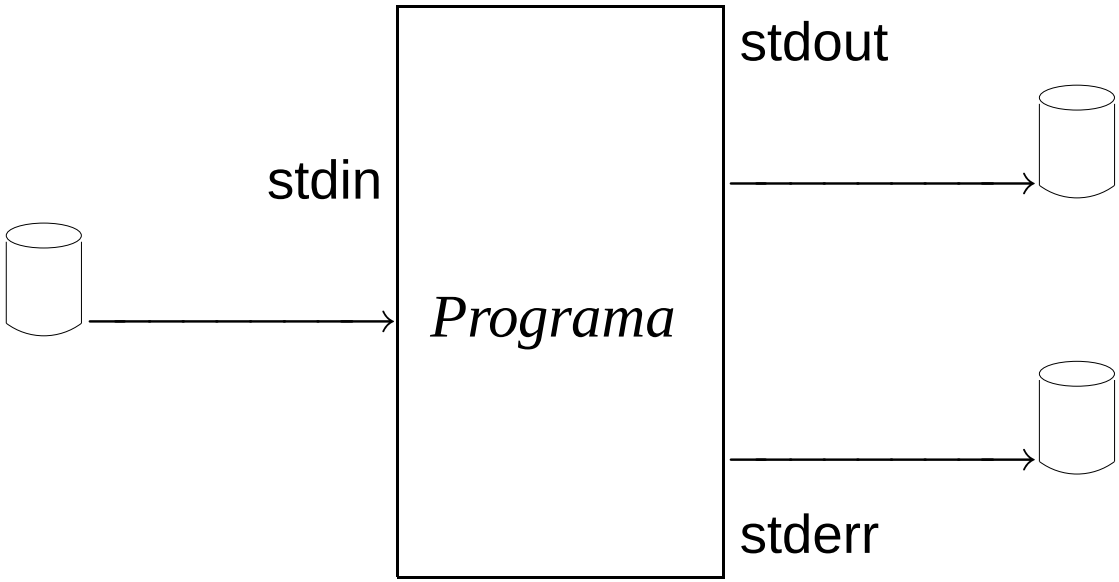
El model de programa que pot gestionar i/o reutilitzar la shell per connectar a altres dispositius o a altres programes que segueixin el mateix model passa per l'us de canals estandard d'entrada/sortida.




```
$ ./Programa < fitxerEntrada.dat > fitxerSortida.dat
```

Canals estandard de comunicació dels programes i/o comandes

El model de programa que pot gestionar i/o reutilitzar la shell per connectar a altres dispositius o a altres programes que segueixin el mateix model passa per l'us de canals estandard d'entrada/sortida.

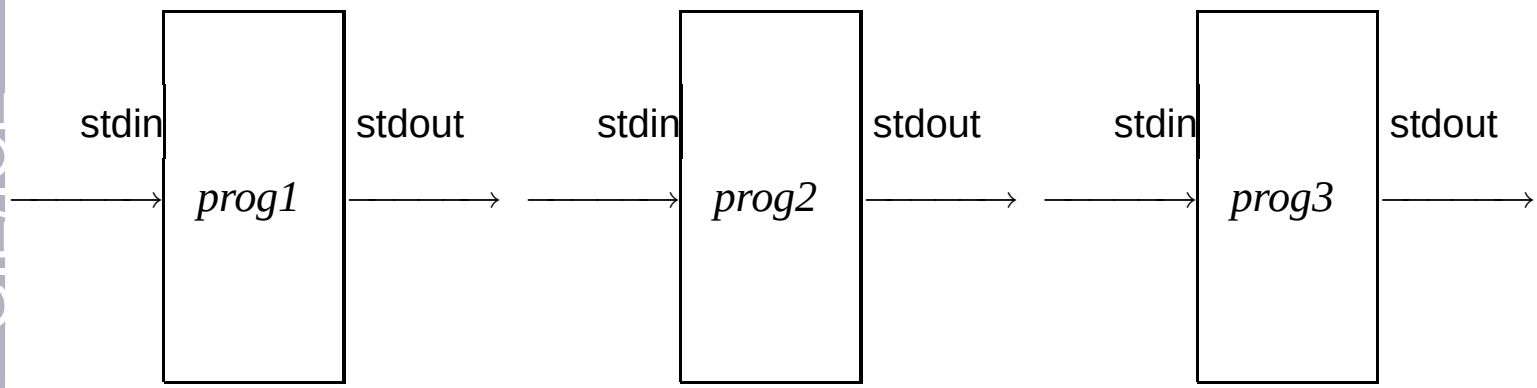



```
$ ./Programa < fitxerEntrada.dat 2>&1 > fitxerSortida.dat
```

Comunicació entre programes i/o comandes

Pipes (tuberies): Les sortides *sdtout* dels programes es connecten a les entrades *stdin* d'altres programes.
Entre una sortida i una entrada hi haurà un buffer o cua circular.

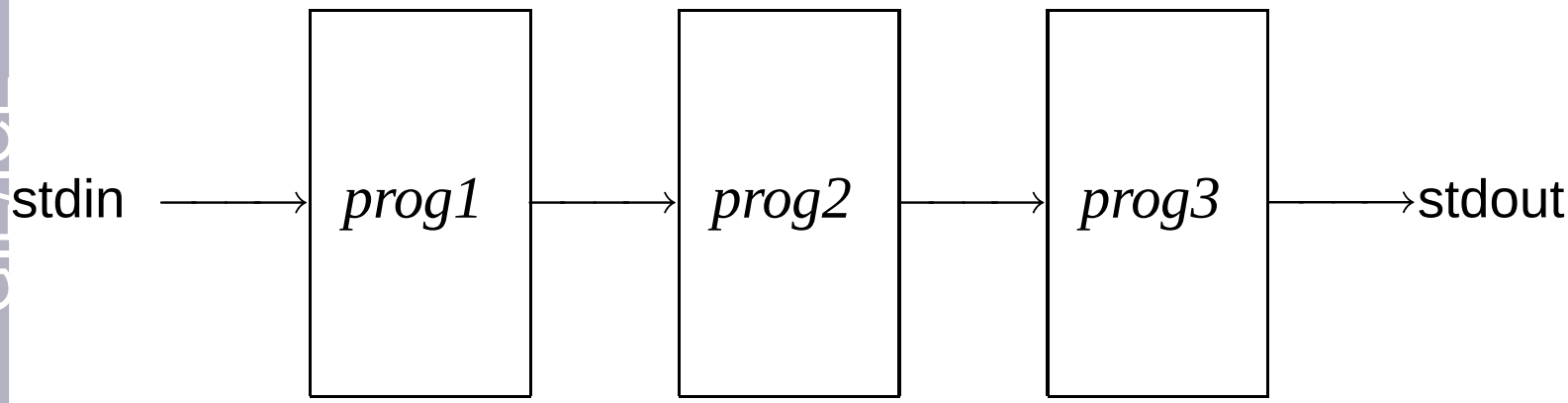
Per exemple, si tenim tres programes *prog1*, *prog2*, i *prog3* i cada programa també té arguments de línia:



Pipes (tuberies): Les sortides *sdtout* dels programes es connecten a les entrades *stdin* d'altres programes.
Entre una sortida i una entrada hi haurà un buffer o cua circular.

Es poden connectar així:

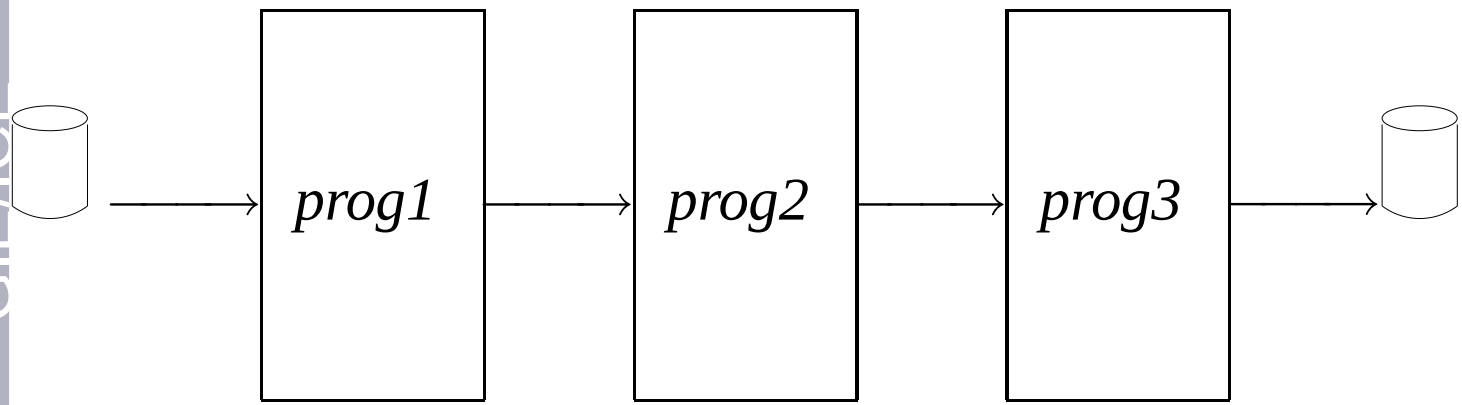
```
$ prog1 a1 a2 | prog2 b1 | prog3 c1 c2 c3
```



Pipes (tuberies): Les sortides *sdtout* dels programes es connecten a les entrades *stdin* d'altres programes.
Entre una sortida i una entrada hi haurà un buffer o cua circular.

I també:

```
$ prog1 a1 a2 < ent.dat | prog2 b1 | prog3 c1 c2 c3 > sor.dat
```



Comunicació entre programes i/o comandes

Pipes (tuberies): Les sortides *sdtout* dels programes es connecten a les entrades *stdin* d'altres programes.
Entre una sortida i una entrada hi haurà un buffer o cua circular.
Aventatges:

- S'utilitza la memòria principal com a intercanvi de dades en comptes del disc: La diferència de costos de temps entre un programa que ho faci tot respecte fer-ho en programes separats executats concurrents és mínima (En molts sistemes operatius la manera de comunicar-se entre programes era mitjançant fitxers).
- Simplifica el desenvolupament de programes al tenir cadascun d'ells una especificació més senzilla que la d'integrar-ho tot en un sol programa.
- La funcionalitat de cada programa pot ser reutilizada per altres objectius composant altres *pipes*.
- Simplicitat en el disseny de la concurrència entre processos.

Llista de comandes

En el que segueix entendrem per comanda la invocació d'un programa o comanda interna del bash amb els seus paràmetres o la invocació d'un pipeline.
Sigui C_i ($1 \leq i \leq n$) una comanda, en una línia es poden composar diverses comandes mitjançant els operadors ;, &, &&, o $\|$ formant una llista:

- ; composició seqüèncial de comandes: $C_1; C_2; \ldots C_n$ primer s'executa C_1, després C_2, ...
- & executa la comanda sense que la shell esperi la seva finalització. Així, $C_1 \& C_2 \& \ldots C_n \&$ s'executen tots alhora i la shell espera la següent línia a executar. No es tindrà en compte l'estat retornat d'aquestes comandes ja que s'executen en un segon pla (background). Per la shell, sempre acaban bé.
- && Operador i lògic ($\wedge$). En el cas de $C_1 \&\& C_2$, la comanda C_2 s'executarà si i només si C_1 ha acabat amb éxit.
- $\|$ Operador o lògica ($\vee$). En el cas de $C_1 \| C_2$, la comanda C_2 s'executarà si i només si C_1 ha acabat amb fallida.

Composició de llistes

(llista) La llista s'executa en una altra shell. Qualsevol canvi en l'entorn d'aquesta altra shell es perd al tornar.

Composició de llistes

(llista) La llista s'executa en una altra shell. Qualsevol canvi en l'entorn d'aquesta altra shell es perd al tornar.

{ llista; } s'executa en el mateix entorn de la shell. Els caràcters { i } són tractats com a paraules (cal la separació d'un espai).

Composició de llistes

(llista) La llista s'executa en una altra shell. Qualsevol canvi en l'entorn d'aquesta altra shell es perd al tornar.

{ llista; } s'executa en el mateix entorn de la shell. Els caràcters { i } són tractats com a paraules (cal la separació d'un espai).

((expresió)) Si el valor de l'expressió no és zero, el retorn de l'estat és zero (éxit).

Composició de llistes

(llista) La llista s'executa en una altra shell. Qualsevol canvi en l'entorn d'aquesta altra shell es perd al tornar.

{ llista; } s'executa en el mateix entorn de la shell. Els caràcters { i } són tractats com a paraules (cal la separació d'un espai).

((expresió)) Si el valor de l'expressió no és zero, el retorn de l'estat és zero (éxit).

[[expresió]] El valor retornat dependrà de l'expressió condicional. No es fan totes les expansions previstes de la shell. Hi han els operadors ! ($\neg$) && ($\wedge$) || ($\vee$) i () (control de prioritats d'operadors).

Composició de llistes

- for nom ; do llista ; done S'executa tants cops com arguments tingui la shell en el moment d'invocar. La variable nom contindrà a cada pas un dels arguments.
  Per exemple:
  Editeu un fitxer anomenat exemple1 que contingui

```
#!/bin/bash
for pep ; do echo $pep ; done
```

  Un cop salvat, escriviu

```
$ chmod +x exemple1
$ ./exemple1 hola que tal
$ ./exemple1 *
```

Composició de llistes

- for nom in paraula/es ; do llista ; done S'executa per cada paraula que hagi en la variable de nom paraula

```
$ for pep in "això sembla que funciona, oi?"; do echo $pep ; done
$ for pep in això sembla que funciona, oi?; do echo $pep ; done
$ paraules="això sembla que funciona, oi?"
$ for pep in $paraules; do echo $pep ; done
```

Composició de llistes

- for nom in paraula/es ; do llista ; done S'executa per cada paraula que hagi en la variable de nom paraula

```
$ for pep in "això sembla que funciona, oi?"; do echo $pep ; done
$ for pep in això sembla que funciona, oi?; do echo $pep ; done
$ paraules="això sembla que funciona, oi?"
$ for pep in $paraules; do echo $pep ; done
```

- for ((expr1 ; expr2 ; expr3)) ; do llista ; done Inicialment s'avalúa expr1. Després s'avalúa expr2: Si retorna diferent de zero, s'executa llista i expr3 fins que expr2retorni 0.

```
$ for ((i=1; $i<=5 ; i++)); do echo $i; done
```

Composició de llistes

- for nom in paraula/es ; do llista ; done S'executa per cada paraula que hagi en la variable de nom paraula

```
$ for pep in "això sembla que funciona, oi?"; do echo $pep ; done
$ for pep in això sembla que funciona, oi?; do echo $pep ; done
$ paraules="això sembla que funciona, oi?"
$ for pep in $paraules; do echo $pep ; done
```

- for ((expr1 ; expr2 ; expr3)) ; do llista ; done Inicialment s'avalúa expr1. Després s'avalúa expr2: Si retorna diferent de zero, s'executa llista i expr3 fins que expr2retorni 0.

```
$ for ((i=1; $i<=5 ; i++)); do echo $i; done
```

- select nom in paraula/es ; do llista ; done Demana pel canal estandard d'entrada un nombre corresponent a l'ordre de la llista de paraules i en el cas de que sigui correcte executa la llista. Acaba a l'entrar fi de fitxer (control-D). També es pot fer select nom ; do llista ; done funcionant de forma anàloga a la del for.

```
$ select nom in joan lluis nuria dolors; do echo has indicat $nom ; done
```

- case paraula in patró | patró . . .) llista ;; . . . esac Expansiona paraula i mira de trobar coincidències amb els patrons. Quan troba una coincidència executa la llista i ja no busca més.
  Escriviu:

```
$ pep="ala bola coco adidas"
$ case $pep in a* | b* ) echo trobat en pep "a*"o "b*";;
> c*) echo trobat en pep ç*";;
> d*) echo trobat en pep "d*";;
> esac
```

Composició de llistes

- case paraula in patró | patró . . .) llista ;; . . . esac Expansiona paraula i mira de trobar coincidències amb els patrons. Quan troba una coincidència executa la llista i ja no busca més.
  Escriviu:

```
$ pep="ala bola coco adidas"
$ case $pep in a* | b* ) echo trobat en pep "a*"o "b*";;
> c*) echo trobat en pep ç*";;
> d*) echo trobat en pep "d*";;
> esac
```

- if llista; then llista; fi
  if llista; then llista; elif llista; then llista; . . . ; fi
  if llista; then llista; elif llista; then llista; . . . else llista; fi
  if llista; then llista; else llista; fi
  Si l'execució d'if llista retorna èxit (0) s'executa el then, sino s'executa un elif o bé else.
  Exemple:

```
$ if gcc -o prova.c ; then ./prova; else echo "programa incorrecte"; fi
```

Composició de llistes

- while llista; do llista; done
  until llista; do llista; done
  El “mentre” del bash. Acaba quan la llista que hi ha després del while falla (retorna estat diferent de 0).
  Escriviu:

```
i=1; while [ $i -le 5 ]; do echo $i; i=$(($i+1)); done
```

Composició de llistes

- while llista; do llista; done
  until llista; do llista; done
  El "mentre" del bash. Acaba quan la llista que hi ha després del while falla (retorna estat diferent de 0).
  Escriviu:
  ```
  i=1; while [ $i -le 5 ]; do echo $i; i=$(($i+1)); done
  ```
- function nom () { llista ; }
  nom () { llista ; }
  Es poden declarar funcions. S'invoquen com qualsevol comand i admeten arguments. El valor de retorn és el de la darrera comanda executada de la llista.
  Exemple:
  ```
  $ function quants () {
  > i=1; for a in $*; do i=$(($i+1)); done
  > echo la funció $0 ha detectat $i arguments
  > }
  $ quants avui farem 4
  ```

Substitució de comandes

- Es pot reemplaçar la comanda per la sortida de la comanda.

Substitució de comandes

- Es pot reemplaçar la comanda per la sortida de la comanda.
- bash permet dues formes de fer-ho:

Substitució de comandes

- Es pot reemplaçar la comanda per la sortida de la comanda.
- bash permet dues formes de fer-ho:
  - `$(comanda)`

Substitució de comandes

- Es pot reemplaçar la comanda per la sortida de la comanda.
- bash permet dues formes de fer-ho:
  - `$(comanda)`
  - `` `comanda` ``

Substitució de comandes

- Es pot reemplaçar la comanda per la sortida de la comanda.
- bash permet dues formes de fer-ho:
  - `$(comanda)`
  - `` `comanda` ``
- Qualsevol retorn de línia que generi la sortida de la comanda s'esborra.

Substitució de comandes

- Es pot reemplaçar la comanda per la sortida de la comanda.
- bash permet dues formes de fer-ho:
  - `$(comanda)`
  - `` `comanda` ``
- Qualsevol retorn de línia que generi la sortida de la comanda s'esborra.
- En el cas de `` ` `` el caràcter `\` té algunes interpretacions. Mentre que en el cas de `$()` cap caràcter té un tractament especial.

Activitats

- Localització arxius insegurs.
- Personalització (.bash_profile)
- Backup personal.
- Lliurament electrònic de pràctiques.
- Extracció i tests de pràctiques.

Exemple 1: Localització de fitxers “insegurs”

- Inspeccionant fitxers que tenen permisos SUID/GUID"
- Inspeccionant fitxers que tenen permis de tothom per ser escrits"
- Inspeccionant fitxers que tenen permis de tothom per ser llegits"
- Inspeccionant fitxers sense propietaris (Sospitosos)"
- Inspeccionant fitxers .rhosts"

Exemple 1: Localització de fitxers “insegurs”

- Inspeccionant fitxers que tenen permisos SUID/GUID"
  `find . -type f \( -perm -04000 -o -perm -02000 \)`
- Inspeccionant fitxers que tenen permis de tothom per ser escrits"
- Inspeccionant fitxers que tenen permis de tothom per ser llegits"
- Inspeccionant fitxers sense propietaris (Sospitosos)"
- Inspeccionant fitxers .rhosts"

Exemple 1: Localització de fitxers “insegurs”

- Inspeccionant fitxers que tenen permisos SUID/GUID"
```
find .  -type f \( -perm -04000 -o -perm -02000 \)
```
- Inspeccionant fitxers que tenen permis de tothom per ser escrits"
```
find .  -perm -2 !  -type l -ls -path "./c-prune
```
- Inspeccionant fitxers que tenen permis de tothom per ser llegits"

- Inspeccionant fitxers sense propietaris (Sospitosos)"

- Inspeccionant fitxers .rhosts"

GIE/ICE

Exemple 1: Localització de fitxers “insegurs”

- Inspeccionant fitxers que tenen permisos SUID/GUID"
```
find .  -type f \( -perm -04000 -o -perm -02000 \)
```
- Inspeccionant fitxers que tenen permis de tothom per ser escrits"
```
find .  -perm -2 !  -type l -ls -path "./c-prune
```
- Inspeccionant fitxers que tenen permis de tothom per ser llegits"
```
find .  \( -perm -4 -o -perm -44 \) !  -type l -ls -path
"./c-prune
```
- Inspeccionant fitxers sense propietaris (Sospitosos)"

- Inspeccionant fitxers .rhosts"

GIE/ICE

Exemple 1: Localització de fitxers “insegurs”

- Inspeccionant fitxers que tenen permisos SUID/GUID"
```
find .  -type f \( -perm -04000 -o -perm -02000 \)
```
- Inspeccionant fitxers que tenen permis de tothom per ser escrits"
```
find .  -perm -2 !  -type l -ls -path "./c-prune
```
- Inspeccionant fitxers que tenen permis de tothom per ser llegits"
```
find .  \( -perm -4 -o -perm -44 \) !  -type l -ls -path
"./c-prune
```
- Inspeccionant fitxers sense propietaris (Sospitosos)"
```
find .  -nouser -o -nogroup -print
```
- Inspeccionant fitxers .rhosts"

Exemple 1: Localització de fitxers “insegurs”

- Inspeccionant fitxers que tenen permisos SUID/GUID"
```
find .  -type f \( -perm -04000 -o -perm -02000 \)
```
- Inspeccionant fitxers que tenen permis de tothom per ser escrits"
```
find .  -perm -2 !  -type l -ls -path "./c-prune
```
- Inspeccionant fitxers que tenen permis de tothom per ser llegits"
```
find .  \( -perm -4 -o -perm -44 \) !  -type l -ls -path
"./c-prune
```
- Inspeccionant fitxers sense propietaris (Sospitosos)"
```
find .  -nouser -o -nogroup -print
```
- Inspeccionant fitxers .rhosts"
```
find .  -name .rhosts -print
```

GIE/ICE

Exemple 2: perfil d'entrada: bash_profile

```
case $TERM in
   xterm*)
     local BARRATITOL='\[\033]0;\u@\h:\w\007\]'
     ;;
   *)
     local BARRATITOL=''
     ;;
esac
PS1="${BARRATITOL}\u@\h:\w\$ "
PS2=''
PS4='+'
umask 077
export LANG=ca_ES
if [ -d ~/bin ] ; then
    PATH=~/bin:"${PATH}"
fi
eval `dircolors`
alias ls='ls --color=auto'
```

Exemple 3: Backup

- Un usuari té un portàtil o un ordinador personal que no és mantingut per l'administrador de sistemes per raons diverses.

Exemple 3: Backup

- Un usuari té un portàtil o un ordinador personal que no és mantingut per l'administrador de sistemes per raons diverses.
- L'usuari té compte en un ordinador central mantingut per l'administrador de sistemes i té connexió d'internet mitjançant el seu ordinador. Les dades importants de la seva feina les té en el disc local del seu ordinador i no té servei NFS. Es connecta a l'ordinador central mitjançant ssh o rsh. La connexió la fa sense password.

Exemple 3: Backup

- Un usuari té un portàtil o un ordinador personal que no és mantingut per l'administrador de sistemes per raons diverses.
- L'usuari té compte en un ordinador central mantingut per l'administrador de sistemes i té connexió d'internet mitjançant el seu ordinador. Les dades importants de la seva feina les té en el disc local del seu ordinador i no té servei NFS. Es connecta a l'ordinador central mitjançant ssh o rsh. La connexió la fa sense password.
- L'usuari vol assegurar les seves dades fent el tradicional *backup*, però li agradaria no tenir que pensar cada dia en fer-ho.

Exemple 3: Backup: Organitzant el disseny

- Crea un directori (per exemple, *linux*) per tenir aquelles adaptacions que vol proposar-se.

Exemple 3: Backup: Organitzant el disseny

- Crea un directori (per exemple, *linux*) per tenir aquelles adaptacions que vol proposar-se.
- Dins d'aquest directori crearà un altre directori anomenat *cron.daily*. El contingut d'aquest directori hi hauran tots aquells guions que tinguin que executar-se diariament.

Exemple 3: Backup: Organitzant el disseny

- Crea un directori (per exemple, *linux*) per tenir aquelles adaptacions que vol proposar-se.
- Dins d'aquest directori crearà un altre directori anomenat *cron.daily*. El contingut d'aquest directori hi hauran tots aquells guions que tinguin que executar-se diariament.
- Edita un fitxer anomenat backup

Exemple 3: Backup: Organitzant el disseny

- Crea un directori (per exemple, *linux*) per tenir aquelles adaptacions que vol proposar-se.
- Dins d'aquest directori crearà un altre directori anomenat *cron.daily*. El contingut d'aquest directori hi hauran tots aquells guions que tinguin que executar-se diariament.
- Edita un fitxer anomenat backup
- Editar un fitxer anomenat crontab o invocar crontab -e

Exemple 3: Backup: fitxer crontab

```
# .crontab

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
HOME=/home/cvs

# m h dom mon dow command
5 0     * * *   run-parts ${HOME}/linux/cron.daily
```

Que vol dir que s'activarà cada dia a les 0h 5' i executarà tots els guions trobats al directori cron.daily

Si fessim

'* * 1,15 * Sun' correrà el dia 1, el dia 15 i el diumenge.

'* * 1,15 * *' només ho farà el dia 1 i 15 de cada mes.

'* * * * Sun' només el diumenge.

Exemple 3: Backup: guió Backup

```
export MAILTO="cursice"
usuari=cursice
home=/home/cursice
nomhost=festuc
dirBackup=elsMeusBackups
aGuardar=/home/cursice/
arxiuBackup=$dirBackup/cursice$(date +%Y%m%d).tgz

guarda=3
rsh  -l$usuari $nomhost ls ">" $ArxiuBackup
tar  -zcf $usuari@$nomhost:$backupfile $aGuardar
echo Backup $arxiuBackup finalitzat
quants=$(rsh  $nomhost -l$usuari ls -t $dirBackup/"cursice*.tgz"| wc -l )
num=$(($quants-$guarda))
echo $quants $num
if  [ $num -gt 0 ]
   then l=$(rsh  $nomhost -l$usuari ls -t $dirBackup/"cursice*.tgz" | tail -n $num )
; echo esborrant $l ; rsh  $nomhost -l$usuari "\rm" -f $l
fi
```

Exemple 4: Lliurament pràctiques

- **lliura** Ajuda a l'enviament de pràctiques a la bústia del professor. Comprova dades professor, comprova l'existència d'un fitxer identificador, elimina fitxers innecessaris, empaqueta la solució i l'envia per correu.

Exemple 4: Lliurament pràctiques

- **lliura** Ajuda a l'enviament de pràctiques a la bústia del professor. Comprova dades professor, comprova l'existència d'un fitxer identificador, elimina fitxers innecessaris, empaqueta la solució i l'envia per correu.
- **compara** Detecció de pràctiques idèntiques tenint en compte fitxers prèvis i comuns a la pràctica.

Exemple 4: Lliurament pràctiques

- **lliura** Ajuda a l'enviament de pràctiques a la bústia del professor. Comprova dades professor, comprova l'existència d'un fitxer identificador, elimina fitxers innecessaris, empaqueta la solució i l'envia per correu.
- **compara** Detecció de pràctiques idèntiques tenint en compte fitxers prèvis i comuns a la pràctica.
- **extreuAdjuntsBustia** Extreu de la bústia entrada, els adjunts de cadascun dels correus i els posa en un directori. Crea un directori arrel per desplegar cadascun dels adjunts i tenint en compte el nombre de pràctica i el nombre d'usuari, desempaqueta la pràctica en un subdirectori. Té en comptres possibles repeticions de lliuraments.

Exemple 4: Lliurament pràctiques

- **lliura** Ajuda a l'enviament de pràctiques a la bústia del professor. Comprova dades professor, comprova l'existència d'un fitxer identificador, elimina fitxers innecessaris, empaqueta la solució i l'envia per correu.
- **compara** Detecció de pràctiques idèntiques tenint en compte fitxers prèvis i comuns a la pràctica.
- **extreuAdjuntsBustia** Extreu de la bústia entrada, els adjunts de cadascun dels correus i els posa en un directori. Crea un directori arrel per desplegar cadascun dels adjunts i tenint en compte el nombre de pràctica i el nombre d'usuari, desempaqueta la pràctica en un subdirectori. Té en comptres possibles repeticions de lliuraments.
- **test** Guions de comprovació de pràctiques.

Exemple 4: Lliurament pràctiques: lliura

```
#
# make $(PROGRAM).tgz
# Genera el fitxer tar comprimit que cal lliurar
#

$(PROGRAM).tgz: WHO Makefile compte.c estatCompte.c estatCompte.h \
moviment.c moviment.h data.c data.h entsort.c entsort.h
tar zcvf $@ $^

#
# make lliurament
# Demana les dades necessàries pel lliurament
# i envia el correu electrònic
#
.PHONY: lliurament
lliurament: $(PROGRAM).tgz WHO
TAR=$$(printf "$(PROGRAM)_%s-%s.tgz" $$(egrep -o '[0-9]{3,}' WHO)); \
mv $< $${TAR}; \
mpack -s $${TAR} $${TAR} inf@lsi.upc.edu
```

Exemple 4: Lliurament pràctiques: compara (1)

```
#!/bin/bash
function compara {
if diff -B -i --ignore-all-space -q -s $1 $2 > pep
       then
         echo fitxers iguals $1 $2
fi
}
function filtre {
fitxers=${*/*$1/}
}
function filtres {
filtre entsort.c $fitxers
filtre entsort.h $fitxers
filtre finestraX.h $fitxers
filtre finestraX.c $fitxers
filtre color.c $fitxers
filtre color.h $fitxers
}
```

Exemple 4: Lliurament pràctiques: compara (2)

```
fitxers=‘find .   -name "*.c"  -print‘
filtres
for f in $fitxers; do
   for g in $fitxers; do
      if [ $f != $g ] ; then compara $f $g; fi
   done
done

fitxers=‘find .   -name "*akefil*"  -print‘
for f in $fitxers; do
   for g in $fitxers; do
      if [ $f != $g ] ; then compara $f $g; fi
   done
done
```

extreuAdjuntsBustia (1)

```
#!/bin/bash
function idUsuari {
   local NOM1=${1/josep.ecso$PRACTICA.ecso/}
   local NOM2=${NOM1/.tar.gz*/}
   USUARI=$NOM2
}

function idPractica {
   local NOM1=${1/josep.ecso/}
   local NOM2=${NOM1/.ecso*.tar.gz*/}
   PRACTICA=$NOM2
}

function extreuMissatgesBustia {
    local BUSTIA_CORREU=$1
    local DIR_EXTRACCIO=$2
    cat "$BUSTIA_CORREU" | formail -q -s munpack -q -C "$DIR_EXTRACCIO"
}
```

extreuAdjuntsBustia (2)

```
function creaDirectori
{
   local DIRPRACTICA=$1
   local DIRUSUARI=$2
   local DIRAFER=$2

   if [ -e "$DIRPRACTICA" ]; then cd .;  else mkdir $DIRPRACTICA;   fi
   cd $DIRPRACTICA
   while [ -e "$DIRUSUARI" ]; do
         if [ -e "$DIRUSUARI.b" ]
            then DIRUSUARI=$DIRUSUARI.b; echo "Practica repetida $DIRUSUARI"
            else mv  $DIRAFER $DIRUSUARI.b; DIRUSUARI=DIRAFER
         fi
   done
   mkdir $DIRAFER
   cd ..
}
```

extreuAdjuntsBustia (3)

```
if [ -e P ]; then cd . ; else mkdir P;   fi
if [ -e tars ]; then cd .; else mkdir tars;   fi
extreuMissatgesBustia $1 tars
cd tars
mkdir desc
mv *.desc desc/
FITXERS=‘ls *.tar.gz*‘
cd ..
for NOM in ${FITXERS[*]};
  do
     idPractica $NOM
     idUsuari $NOM
     cd P
     creaDirectori $PRACTICA $USUARI
     cd $PRACTICA/$USUARI
     echo "Extracció pràctica $PRACTICA $USUARI" en ‘pwd‘ fitxer $NOM
     tar zxvf ../../../tars/$NOM
     cd ../../../
  done
```

Exemple 4: Lliurament pràctiques: test

```
TESTEXEC=echo Test $(subst etst,,$(word 2,$^)): inici; \
        $(PROGRAM) < $(word 2,$^) > stst && \
        diff -b stst $(word 3,$^) && \
        echo Test $(subst etst,,$(word 2,$^)): correcte

.PHONY: test test0 test1 test2 test3
test: test0 test1 test2 test3

test0: $(PROGRAM) etst0 stst0
        $(TESTEXEC)

test1: $(PROGRAM) etst1 stst1
        $(TESTEXEC)

test2: $(PROGRAM) etst2 stst2
        $(TESTEXEC)

test3: $(PROGRAM) etst3 stst3
        $(TESTEXEC)
```