

Flujos de datos

Todos los procesos en ejecución tienen asociados tres flujos de datos:

0. **stdin** – representa la entrada estándar, donde se reciben datos desde el teclado, otro programa, etc.
1. **stdout** – representa la salida estándar, donde se escriben los resultados de la ejecución de un programa.
2. **stderr** – representa la salida de errores, donde un proceso vuelca los posibles errores que puedan surgir durante su ejecución.



```
user@users-desktop:~$ ls -l /dev/{stdin,stdout,stderr}
lrwxrwxrwx 1 root root 15 abr  8 10:25 /dev/stderr -> /proc/self/fd/2
lrwxrwxrwx 1 root root 15 abr  8 10:25 /dev/stdin -> /proc/self/fd/0
lrwxrwxrwx 1 root root 15 abr  8 10:25 /dev/stdout -> /proc/self/fd/1
```

Flujos de datos

Los flujos de datos, al igual que en otras consolas, se pueden redirigir libremente hacia otro fichero.

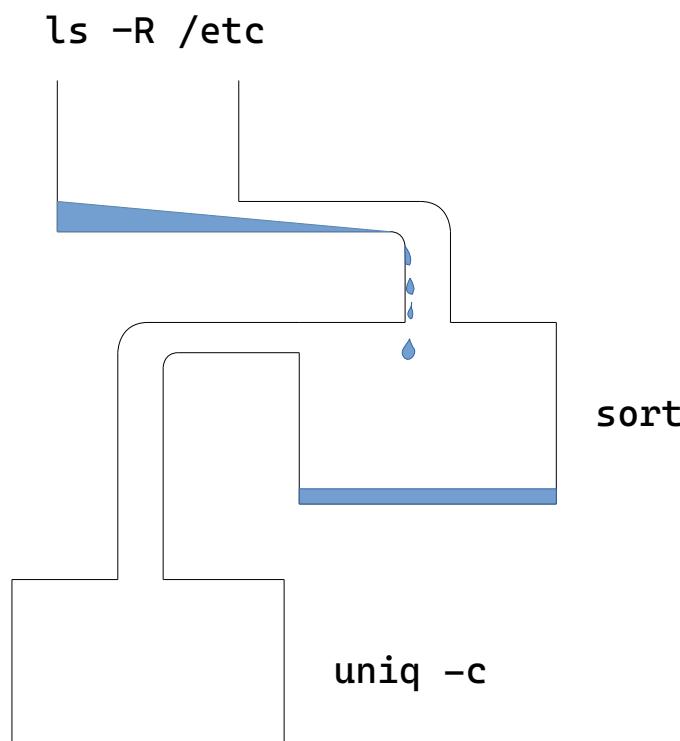
Esto se traduce en que la salida de la ejecución de un programa puede escribirse a un fichero en lugar de al terminal, así como la entrada puede ser tomada de un fichero.

Los redireccionamientos se realizan con los signos de comparación: < y >

- **instruc > salida:** redirecciona el flujo de datos desde “instruc” al fichero “salida”.
- **instruc < entrada:** redirecciona el contenido del fichero “entrada” a “instruc” como datos de entrada.
- **instruc 2> salida:** redirecciona el flujo de datos de errores desde “instruc” a “salida” – solo errores.

```
...  
  
# Redirecciona la salida estándar hacia un fichero con nombre "users"  
user@users-desktop:~$ cat /etc/passwd > users  
# Lee el contenido del fichero "alumnos" y lo añade al final de "users"  
user@users-desktop:~$ cat < alumnos >> users  
# Lee de la entrada estándar (teclado) y lo añade al final de "users"  
user@users-desktop:~$ cat >> users  
# Redirecciona la salida del comando "ls" al dispositivo "nulo" (desaparece)  
user@users-desktop:~$ ls -l > /dev/null  
# Muestra por pantalla el contenido de "/tmp" y redirige los errores a "errs"  
user@users-desktop:~$ ls /tmp 2> errs  
# Redirecciona los ficheros encontrados por "find" y lo guarda en "files".  
# Por otra parte, redirige los errores al fichero "err-files".  
user@users-desktop:~$ find / -iname *.sh > files 2> err-files  
# Redirección de la salida estándar a un fichero y de la salida de errores a  
# la salida estándar.  
user@users-desktop:~$ apt update > upd-file 2>1  
# Redirección de tanto la salida estándar como de la salida de errores al  
# mismo fichero "output"  
user@users-desktop:~$ apt upgrade &> output
```

Flujos de datos



Las tuberías suponen un mecanismo muy potente para enviar datos de un comando a otro, permitiendo concatenar tantos como se desee.

Estos mecanismos solo se encuentran disponibles en Unix y derivados. Su función radica en que muchos de los comandos, cuando no se especifica un fichero, toman datos de la entrada estándar.

Gracias al concepto anterior, se puede “redirigir” la salida estándar de un comando a la entrada estándar del siguiente, función que realizan las tuberías.

Destacar que una tubería acepta una cantidad “ilimitada” de comandos. Sin embargo, no se ejecutan secuencialmente (uno tras otro) sino que, a medida que tiene datos, van realizando sus operaciones.

Flujos de datos

Dada la característica única de las tuberías, no se debe confiar nunca en el orden de la ejecución de los comandos (para ello hay otros operadores). Por ello, hay ciertas órdenes que, en su documentación, indican que muy posiblemente requieran de otro comando antes. Este es el caso de `uniq`, el cual cuando lee por la entrada estándar lleva la cuenta de la línea actual y no del total. En su documentación declara que interesa previamente ordenar la salida con `sort`, ya que dicho comando espera a tener toda la entrada completa antes de producir una salida y cumple con los requisitos de `uniq`.

Nota: 'uniq' no detecta lineas repetidas a menos que sean adyacentes.
Puede que quiera ordenar la entrada primero, o usar 'sort -u' sin 'uniq'.



```
# Página el comando "ls" usando una tubería y el comando "less"
user@users-desktop:~$ ls -l /dev | less
# Filtra la salida del comando "ls" usando una tubería y el comando "grep"
user@users-desktop:~$ ls -l /dev | grep sd
# Lista de forma recursiva el directorio "/etc" y ordena su salida
user@users-desktop:~$ ls -R /etc | sort
# Lista de forma recursiva el directorio "/etc", ordena su salida y filtra
# por elementos únicos
user@users-desktop:~$ ls -R /etc | sort | uniq
# Lista de forma recursiva el directorio "/etc", ordena su salida, filtra por
# elementos únicos y lo paga con "less"
user@users-desktop:~$ ls -R /etc | sort | uniq | less
# Filtra los usuarios que tengan "bash" como consola principal, cuenta la
# cantidad de ocurrencias y guarda dicho valor en el fichero "bash-users"
user@users-desktop:~$ grep "/bin/bash" /etc/passwd | wc -l > bash-users
```

Gestión de procesos

¿Qué es un proceso?

Un proceso es una instancia de un programa en ejecución. En un sistema operativo multitarea se ejecutan múltiples procesos de forma simultánea, aunque solo una cantidad finita accede al procesador.

¿Qué quiere decir esto? Si se cuenta con un procesador mononúcleo, solo se podrá realizar una ejecución simultánea. Sin embargo, con procesadores multinúcleos se podrían llegar a realizar tantas ejecuciones simultáneas como núcleos hubiese disponibles.

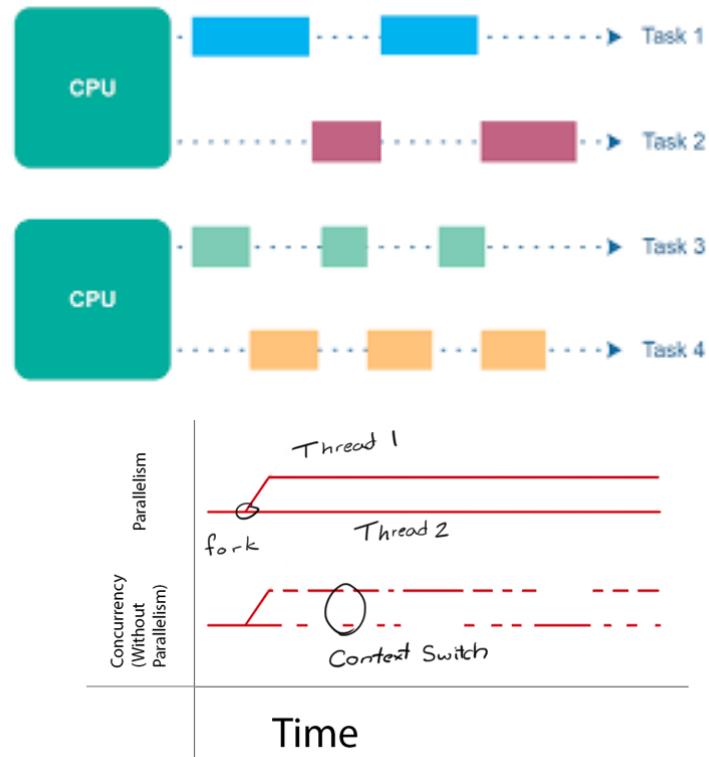
Name	22% CPU	31% Memory	0% Disk	0% Network
Apps (6)				
> HeavyLoad	0%	7.6 MB	0 MB/s	0 Mbps
e Microsoft Edge	0%	10.8 MB	0 MB/s	0 Mbps
> 🎨 PicPick (32 bit)	0%	56.7 MB	0 MB/s	0 Mbps
> 📄 Task Manager	0.5%	29.7 MB	0 MB/s	0 Mbps
> ⌘ Windows Command Processor	0%	0.4 MB	0 MB/s	0 Mbps
▼ Windows Explorer (5)	0.3%	58.5 MB	0 MB/s	0 Mbps
Between PCs				
Between PCs				
HomeGroup				
picpick_portable				
View and print your homegroup password				
Background processes (44)				
Application Frame Host	0%	12.7 MB	0 MB/s	0 Mbps
Browser_Broker	0%	2.4 MB	0 MB/s	0 Mbps
Fewer details				
			Restart	

Concurrencia, paralelismo... ¿No son lo mismo?

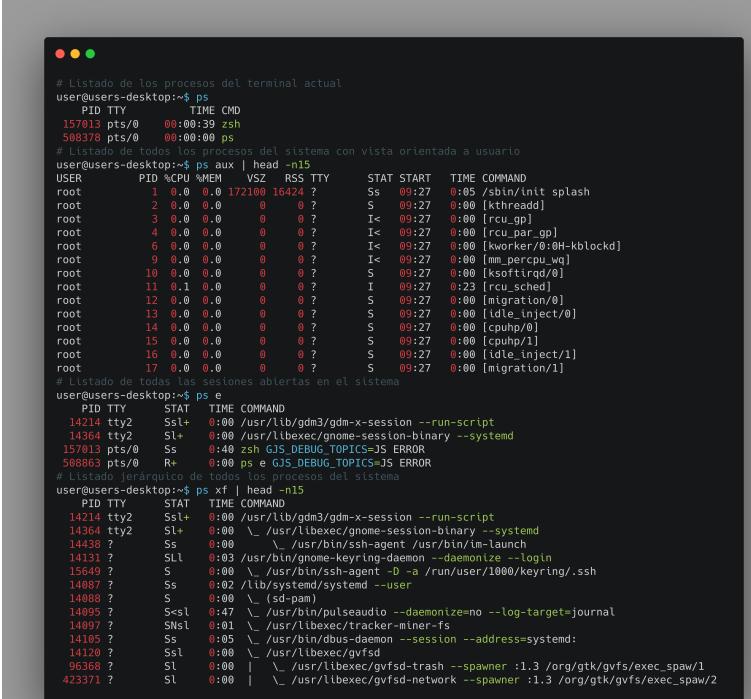
En realidad no: un sistema puede ser concurrente pero no ser paralelo; y ser paralelo y concurrente a la vez. Por lo general, se dice que un sistema operativo es tanto concurrente como paralelo (si cuenta con varios núcleos) porque permite manejar multitud de procesos a la vez y permite realizar multitud de acciones de una sola vez.

A system is said to be concurrent if it can support two or more actions in progress at the same time. A system is said to be parallel if it can support two or more actions executing simultaneously.

The Art of Concurrency, de Clay Breshears



Gestión de procesos



```
# Listado de los procesos del terminal actual
user@users-desktop:$ ps
  PID TTY      STAT   TIME CMD
157013 pts/0    00:00:39 zsh
508378 pts/0    00:00:00 ps
# Listado de todos los procesos del sistema con vista orientada a usuario
user@users-desktop:$ ps aux | head -n15
USER     PID %CPU %MEM    VSZ RSS TTY      STAT START  TIME COMMAND
root      1  0.0  0.0 172100 16424 ?        Ss  09:27  0:05 /sbin/init splash
root      2  0.0  0.0      0  0 ?        Ss  09:27  0:00 [kthreadd]
root      3  0.0  0.0      0  0 ?        I<  09:27  0:00 [rcu_gp]
root      4  0.0  0.0      0  0 ?        I<  09:27  0:00 [rcu_par_gp]
root      6  0.0  0.0      0  0 ?        I<  09:27  0:00 [kworker/0:0H-kblockd]
root      9  0.0  0.0      0  0 ?        I<  09:27  0:00 [mm_percpu_wq]
root     10  0.0  0.0      0  0 ?        Ss  09:27  0:00 [ksoftirqd/0]
root     11  0.1  0.0      0  0 ?        I  09:27  0:23 [rcu_sched]
root     12  0.0  0.0      0  0 ?        Ss  09:27  0:00 [migration/0]
root     13  0.0  0.0      0  0 ?        Ss  09:27  0:00 [idle_inject/0]
root     14  0.0  0.0      0  0 ?        Ss  09:27  0:00 [cpuhp/0]
root     15  0.0  0.0      0  0 ?        Ss  09:27  0:00 [cpuhp/1]
root     16  0.0  0.0      0  0 ?        Ss  09:27  0:00 [idle_inject/1]
root     17  0.0  0.0      0  0 ?        Ss  09:27  0:00 [migration/1]
# Listado de todas las sesiones abiertas en el sistema
user@users-desktop:$ ps e
  PID TTY      STAT   TIME COMMAND
14214 tty2    Ssl+  0:00 /usr/lib/gdm3/gdm-x-session --run-script
14364 tty2    S+    0:00 /usr/libexec/gnome-session-binary --systemd
157013 pts/0    Ss  0:48 zsh GJS_DEBUG_TOPICS=JS ERROR
508363 pts/0    R+  0:00 ps e GJS_DEBUG_TOPICS=JS ERROR
# Listado jerárquico de todos los procesos del sistema
user@users-desktop:$ ps xf | head -n15
  PID TTY      STAT   TIME COMMAND
14214 tty2    Ssl+  0:00 /usr/lib/gdm3/gdm-x-session --run-script
14364 tty2    S+    0:00 \_ /usr/libexec/gnome-session-binary --systemd
14438 ?      Ss  0:00 \_ \_ /usr/bin/ssh-agent /usr/bin/im-launch
14131 ?      Sll  0:03 /usr/bin/gnome-keyring-daemon --daemonize --login
15649 ?      Ss  0:00 \_ /usr/bin/ssh-agent -D -a /run/user/1000/keyring/.ssh
14087 ?      Ss  0:02 /lib/systemd/systemd --user
14088 ?      Ss  0:00 \_ (sd-pam)
14095 ?      Ss1  0:47 \_ /usr/bin/pulseaudio --daemonize=no --log-target=journal
14097 ?      SNs1 0:01 \_ /usr/libexec/tracker-miner-fs
14105 ?      Ss  0:05 \_ /usr/bin/dbus-daemon --session --address=systemd:
14120 ?      Ssl  0:00 \_ /usr/libexec/gvfsd
96368 ?      Sl  0:00 | \_ /usr/libexec/gvfsd-trash --spawner :1.3 /org/gtk/gvfs/exec_spaw/1
423371 ?      Sl  0:00 | \_ /usr/libexec/gvfsd-network --spawner :1.3 /org/gtk/gvfs/exec_spaw/2
```

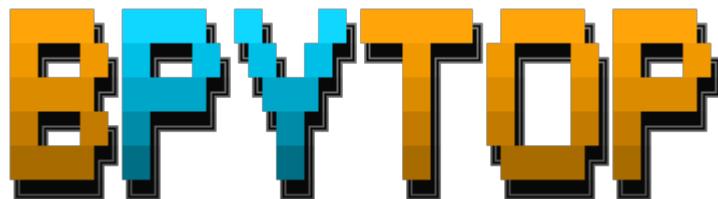
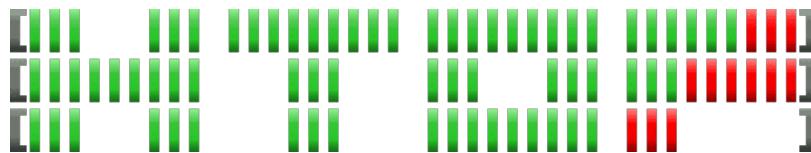
Cuando se crea un proceso, se asocia cierta información fundamental al mismo:

1. Usuario que lo ejecuta.
2. Hora de inicio.
3. Línea de comandos asociada a la ejecución.
4. Estado del proceso: **sleep, running, zombie, stopped**.
5. La prioridad asociada al proceso: [-20, 19].
6. La terminal desde la que fue invocado, si está asociado.

El comando fundamental para ver información sobre los procesos en ejecución es “ps”. Por defecto, muestra el PID, la consola desde donde se ejecutó, el tiempo de CPU que ha consumido y el comando que lo desencadenó. Algunas opciones muy útiles son:

- **x:** muestra todos los procesos del usuario actual (sin distinción de terminal).
- **a:** muestra todos los procesos de todos los usuarios.
- **f:** muestra las relaciones jerárquicas entre los procesos.
- **e:** muestra el entorno de cada proceso.
- **l:** utiliza un formato más largo (más información).
- **u:** utiliza un formato orientado a usuario.

top



Pese a que el comando “ps” es muy útil, existen multitud de herramientas interactivas que permiten ver, identificar y gestionar los procesos que se están ejecutando en el sistema. Se llaman “top” (*table of processes*).

Algunas destacadas son:

- top, herramienta nativa de Linux que se actualiza en tiempo real mostrando información de la CPU.
- htop, herramienta que “mejora top” añadiendo personalización y más opciones.
- bpytop, herramienta diseñada en Python que añade gráficos y distinta información del sistema.

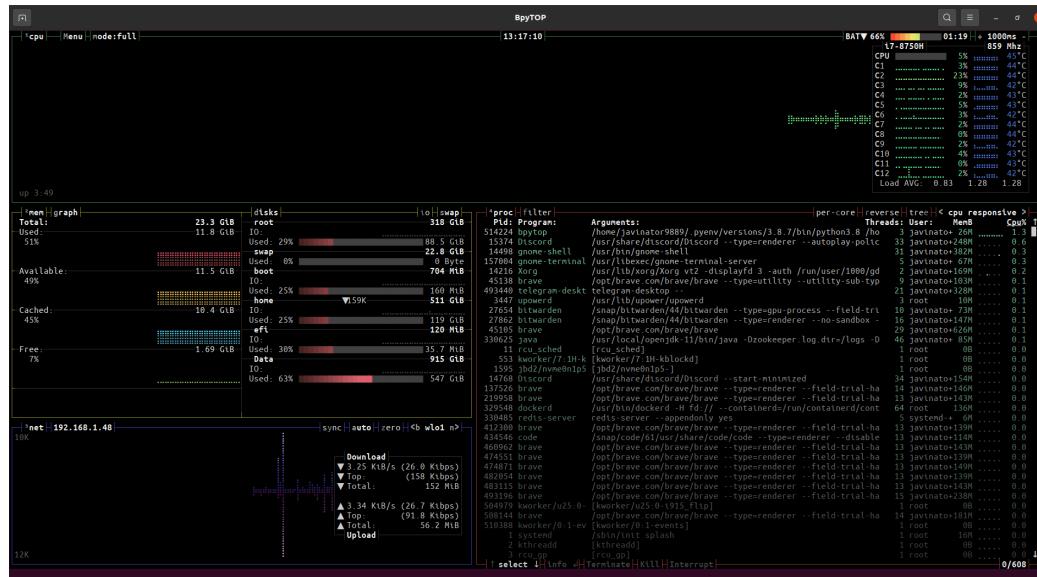
Gestión de procesos

```
Top - 13:18:12 up 3:50, 1 user, load average: 1.92, 1.51, 1.36
Tasks: 614 total, 4 ejecutar, 610 hibernar, 0 detener, 0 zombie
%cpu(s): 2,9 usuario, 1,6 sist, 0,1 adecuado, 0,2 inact, 0,1 en espera, 0,0 hardw int, 0,0 softw int, 0,0 robar tiempo
MiB Mem : 23902,7 total, 1718,1 libre, 9407,2 usado, 12777,3 buffer/caché
MiB Intercambio: 23437,0 total, 23437,0 libre, 0,0 usado. 11792,3 dispon Mem

PID USUARIO PR NI VIRT RES SHR S %CPU %MEM HORAS+ ORDEN
14499 javinat+ 20 0 6328092 391736 125632 S 12,3 1,6 13:05:93 gnome-shell
15374 javinat+ 20 0 24,9g 253628 12848g S 7,9 1,0 13:46:18 Discord
14216 root+ 20 0 595604 175168 11842 S 7,6 0,7 12:21:08 Xorg
4404 javinat+ 20 0 1000000 1000000 1000000 R 2,5 0,0 13:18:12 python3
45105 javinat+ 20 0 1993084 641580 212884 R 1,7 0,0 13:18:12 telegram-deskt...
493440 javinat+ 20 0 3024408 336324 144144 S 1,7 1,4 0:48:39 telegram-deskt...
329548 root+ 20 0 4918128 1409212 49588 S 1,3 0,6 1:33:74 dockerd
1912 root+ 35 15 30236 26928 248g S 1,0 0,1 2:12:38 preload
27654 javinat+ 20 0 559916 7468g 47720 S 1,0 0,3 2:04:18 bitwarden
45105 javinat+ 20 0 1993084 641580 212884 R 0,9 0,0 13:18:12 discord
77865 javinat+ 20 0 4816308 151084 100268 S 0,7 0,5 1:57:24 bitwarden
434546 javinat+ 20 0 4721248 116916 89348 S 0,7 0,5 0:33:09 code
514693 javinat+ 20 0 15172 4436 3324 R 0,7 0,0 0:00:07 top
11 root+ 20 0 0 0 0 I 0,3 0,0 0:25:75 rcu_sched
1967 root+ 20 0 2077680 48620 26688 S 0,3 0,2 0:27:78 containerd
```

```
Tasks: 363 2308 thr: 1 running
Load average: 1.34 1.54 1.36
Uptime: 03:48:01

PID USER PR NI VIRT RES SHW S %CPU %MEM TIME+ Command
113951 javinat+ 20 0 16716 7032 3376g S 1,0 0,8 0:00:05 http...
15374 javinat+ 20 0 24,9g 253628 12848g S 7,9 1,0 13:46:18 /usr/share/discord/Discord --type=renderer --autoplay-policy=no-user-gesture-required --...
15374 Discord 20 0 24,9g 253628 12848g S 7,9 1,0 13:46:18 /usr/share/discord/Discord --type=renderer --autoplay-policy=no-user-gesture-required --...
14498 javinat+ 20 0 6179M 381K 122g S 3,2 1,6 12:54:62 /usr/bin/gnome-shell
27833 javinat+ 20 0 582K 95424 16628 S 3,2 0,4 1:19:72 /snap/mailspring/408/usr/share/mailspring/resources/app.asar.unpacked/mailsync.bla...
27833 bitwarden 20 0 582K 95424 16628 S 3,2 0,4 1:19:72 /snap/mailspring/408/usr/share/mailspring/resources/app.asar.unpacked/mailsync.bla...
77865 javinat+ 20 0 4816308 151084 100268 S 0,7 0,5 1:57:24 bitwarden
493440 javinat+ 20 0 297M 328g 140g S 1,3 1,4 0:46:29 telegram-desktop --
45143 javinat+ 20 0 392K 103K 69976 S 1,3 0,4 1:28:92 /opt/brave.com/brave/brave --utility-sub-type=network.nojon.NetworkService...
45138 javinat+ 20 0 581K 74680 47716 S 1,0 0,6 0:00:07 /opt/brave.com/brave/brave --utility-sub-type=network.nojon.NetworkService...
14216 Xorg 20 0 581K 169M 110g S 0,6 0,7 11:59:83 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/108/gdm/Xauthority background none
45138 javinat+ 20 0 392K 103K 69976 S 0,6 0,4 1:34:50 /opt/brave.com/brave/brave --utility-sub-type=network.nojon.NetworkService...
14244 root+ 20 0 581K 169M 110g S 0,6 0,7 1:27:91 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/108/gdm/Xauthority background none
27862 javinat+ 20 0 4763K 147K 978 S 0,6 0,6 1:56:00 /snap/bitwarden/44/bitwarden --type=renderer --no-sandbox --field-trial-handle:114230949
```



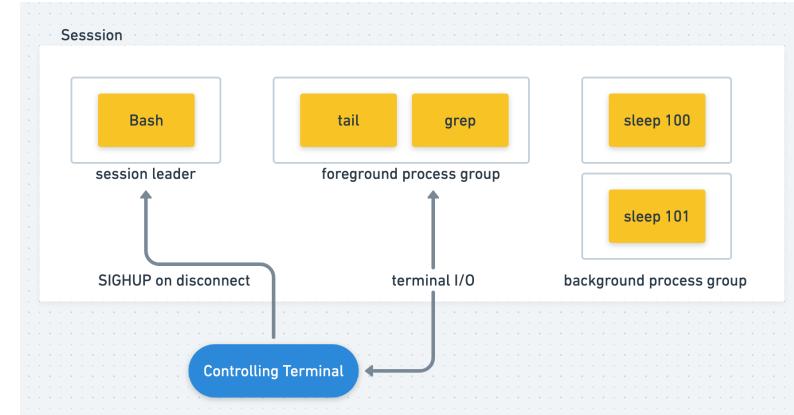
Gestión de procesos

Las consolas de comandos actuales permiten ejecutar múltiples procesos tanto en **foreground** (primer plano) como en **background** (segundo plano).

Los procesos en **foreground** son únicos por terminal (no puede haber más de un proceso en primer plano en un terminal) y se devuelve el control al usuario cuando dicho proceso ha terminado, sea interrumpido o detenido.

Los procesos en **background** por su parte son “ilimitados”: se pueden lanzar tantos procesos como capacidad tenga la máquina. Para indicar que un comando se quiere ejecutar en segundo plano se usa la siguiente sintaxis:

<comando> &



```
javinator9889@Tony-MkIII ~ > sudo updatedb &
[1] 521967
  * javinator9889@Tony-MkIII ~ > jobs
[1] + running    sudo updatedb
  * javinator9889@Tony-MkIII ~ >
[1] + 521967 done      sudo updatedb
```

Gestión de procesos

```
...
user@users-desktop:~$ cat /tmp/quijote
DON QUIJOTE DE LA MANCHA
Miguel de Cervantes Saavedra

PRIMERA PARTE
CAPÍTULO 1: Que trata de la condición y ejercicio del famoso hidalgo D. Quijote
de la Mancha
En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no ha mucho
tiempo que vivía un hidalgo de los de lanza en astillero, adarga antigua, rocin
flaco y galgo corredor. Una olla de algo más vaca que carnero, salpicón las más
noches, duelos y quebrantos los sábados, lentejas los viernes, algún palomino
de añadidura los domingos, consumían las tres partes de su hacienda.
^C
user@users-desktop:~$ cat /tmp/quijote
DON QUIJOTE DE LA MANCHA
Miguel de Cervantes Saavedra

PRIMERA PARTE
CAPÍTULO 1: Que trata de la condición y ejercicio del famoso hidalgo D. Quijote
de la Mancha
En un lugar de la Mancha, de cuyo nombre no quiero acordarme, no ha mucho
tiempo que vivía un hidalgo de los de lanza en astillero, adarga antigua, rocin
flaco y galgo corredor. Una olla de algo más vaca que carnero, salpicón las más
noches, duelos y quebrantos los sábados, lentejas los viernes, algún palomino
de añadidura los domingos, consumían las tres partes de su hacienda.
^Z
[1] + 529478 suspended cat /tmp/quijote
user@users-desktop:~$ jobs
[1] + suspended cat /tmp/quijote
user@users-desktop:~$ cp -r Downloads /tmp &
[2] 529974
user@users-desktop:~$ jobs
[1] + suspended cat /tmp/quijote
[2] - running cp -r Downloads /tmp
user@users-desktop:~$ fg %2
[2] 529974 running cp -r Downloads /tmp
^Z
[2] + 529974 suspended cp -r Downloads /tmp
user@users-desktop:~$ bg %2
[2] - 529974 continued cp -r Downloads /tmp
user@users-desktop:~$ 
[2] - 529974 done cp -r Downloads /tmp
user@users-desktop:~$ fg
El resto de la conclusión sayo de velarte, calzas de velludo para las fiestas
con sus puntiflors de lo mismo, los días de entre semana se honraba con su
vellori de lo más fino. Tentó en su casa una ama que pasaba de los cuarenta,
una sobrina que no llegaba a los veinte, y un mozo de campo y plaza, que así
ensillaba el rocin como tomaba la podadera.
^C
```

Los procesos se pueden gestionar desde la consola de dos formas:

1. Ctrl + C intenta interrumpir el proceso en *foreground*. Si es efectivo, el proceso finaliza su ejecución.
2. Ctrl + Z intenta detener un proceso en *foreground*. Si es efectivo, el proceso sigue activo pero no se ejecuta (no accede al procesador).

La lista de procesos detenidos o en segundo plano se puede obtener con el comando “jobs”. Por otra parte, un trabajo detenido se puede llevar a primer plano ejecutando el comando “fg” o que se ejecute en *background* usando el comando “bg”.

Comunicación de procesos

Hasta ahora solo hemos visto cómo crear procesos en segundo plano, cómo interrumpir un proceso y cómo “suspender” un proceso. Sin embargo, no nos hemos podido comunicar con un proceso para que haga cierta acción más allá de su tarea. ¿O sí?

Las señales son una gran forma de comunicarnos con un proceso. Son siempre asíncronas y presentan diversos comportamientos:

1. Si la señal es “capturada” (*caught*), el proceso puede decidir ignorarla o cambiar su comportamiento.
2. Si la señal no es capturada, provocará la terminación del proceso de forma abrupta.

Signal	Value	Action	Comment
SIGHUP	1	Term	Hangup detected on controlling terminal or death of controlling process
SIGINT	2	Term	Interrupt from keyboard
SIGQUIT	3	Core	Quit from keyboard
SIGILL	4	Core	Illegal Instruction
SIGABRT	6	Core	Abort signal from abort(3)
SIGFPE	8	Core	Floating point exception
SIGKILL	9	Term	Kill signal
SIGSEGV	11	Core	Invalid memory reference
SIGPIPE	13	Term	Broken pipe: write to pipe with no readers
SIGNALRM	14	Term	Timer signal from alarm(2)
SIGTERM	15	Term	Termination signal
SIGUSR1	30,10,16	Term	User-defined signal 1
SIGUSR2	31,12,17	Term	User-defined signal 2
SIGCHLD	20,17,18	Ign	Child stopped or terminated
SIGCONT	19,18,25	Cont	Continue if stopped
SIGSTOP	17,19,23	Stop	Stop process
SIGTSTP	18,20,24	Stop	Stop typed at terminal
SIGTTIN	21,21,26	Stop	Terminal input for background process
SIGTTOU	22,22,27	Stop	Terminal output for background process

The signals **SIGKILL** and **SIGSTOP** cannot be caught, blocked, or ignored.

Gestión de procesos

```
user@users-desktop:~$ kill -l
HUP INT QUIT TERM TRAP ABRT BUS FPE KILL USR1 SEGV USR2 PIPE ALRM TERM STKFLT CHLD CONT STOP TSTP TTIN TTOUT
SIG XCPU XFSZ VTALM PWR WINCH POLL PWR SYS
user@users-desktop:~$ ps ux
USER          PID %CPU %MEM   VSZ RSS TTY      STAT START  TIME COMMAND
javinat+    8475  0.0  0.0 19936 11016 ?        Ss  00:07: 0:01 /lib/systemd/systemd --user
javinat+   14420  0.0  0.0 284972 9444 ?        S  00:07: 0:00 /usr/share/discord/Discord --type=zygote
javinat+   31222  0.0  0.0 12116 3140 ?        S  00:07: 0:00 /bin/bash /usr/bin/brave-browser-stable
javinat+   31224  0.0  0.0 10852  592 ?        S  00:07: 0:00 cat
javinat+  141649  0.4  0.0 25988 12124 pts/0   Ss  00:22: 0:04 zsh
javinat+  256623  0.0  0.0 14200  3372 pts/0   R+  00:38: 0:00 ps ux
# Envio señal de terminación a "cat" (SIGTERM - 15)
user@users-desktop:~$ kill 31228
# Envio señal de kill a "Discord" (SIGKILL - 9)
user@users-desktop:~$ kill -9 14420
# Envio señal de HUP a "zsh" (SIGHUP - 1)
user@users-desktop:~$ kill -HUP 141649
# Lista de procesos en segundo plano
user@users-desktop:~$ jobs
[1] + suspended cat /tmp/quijote
[2] - running cp -r Downloads /tmp
# Envio señal de suspensión al segundo proceso (SIGSTOP - 17,19,23 | SIGTSTP - 18,20,24)
user@users-desktop:~$ kill -STOP %2
user@users-desktop:~$ jobs
[1] + suspended cat /tmp/quijote
[2] + suspended cp -r Downloads /tmp
# Continuación de cada uno de los procesos (SIGCONT - 19,18,25)
user@users-desktop:~$ kill -CONT %1
user@users-desktop:~$ kill -CONT %2
user@users-desktop:~$ jobs
[1] - running cat /tmp/quijote
[2] - running cp -r Downloads /tmp
```

En Linux, las señales se envían con el comando “kill” (tener en cuenta que el nombre no es necesariamente representativo). Sin argumentos, envía una señal de terminación a un proceso (SIGTERM – 15).

La señal para forzar la parada de un proceso es SIGKILL – 9, la cual no se puede evitar (pero es necesario ser administrador o el dueño del proceso). Por otra parte, SIGHUP – 1 suele provocar la “recarga” del proceso.

La sintaxis de kill es la siguiente:

kill PID

kill -signal PID

kill -s signal PID

Gestión de procesos

El comando **kill** es muy potente a la hora de comunicarnos con procesos pero es lento: hay que ir proceso a proceso enviando la señal. ¿Qué sucede si quiero enviar una misma señal a un grupo de procesos? ¿Y si quiero enviar una señal a un proceso según su nombre?

Entra en juego **killall**, comando por el cual se envía una señal a todos los procesos que tengan ese nombre.

```
user@users-desktop:~$ killall --help
Uso: killall [ -Z CONTEXTO ] [ -u USUARIO ] [ -y TIEMPO ] [ -o TIEMPO ] [ -eIgiqrw ]
      [ -s SEÑAL | -SIGNAL ] NOMBRE...
      killall -l, --list
      killall -V, --version

-e,--exact      require exact match for very long names
-I,--ignore-case case insensitive process name match
-g,--process-group kill process group instead of process
-y,--younger-than kill processes younger than TIME
-o,--older-than  kill processes older than TIME
-i,--interactive ask for confirmation before killing
-l,--list        list all known signal names
-q,--quiet       dont print complaints
-r,--regexp      interpret NAME as an extended regular expression
-s,--signal SIGNAL send this signal instead of SIGTERM
-u,--user USER   kill only process(es) running as USER
-v,--verbose     report if the signal was successfully sent
-V,--version     display version information
-w,--wait        wait for processes to die
-n,--ns PID      match processes that belong to the same namespaces
                  as PID
-Z,--contexto REGEXP matar solamente proceso(s) teniendo contexto
                  (debe preceder otros argumentos)
```

```
# Envío señal de terminación a todos los procesos que se llamen "Discord"
user@users-desktop:~$ killall Discord
# Finalizo, de forma forzosa, todas las instancias de Google Chrome
user@users-desktop:~$ killall -9 chrome
# Finalizo todas las sesiones de GNOME de forma interactiva
user@users-desktop:~$ killall -9 -i gdm3
```

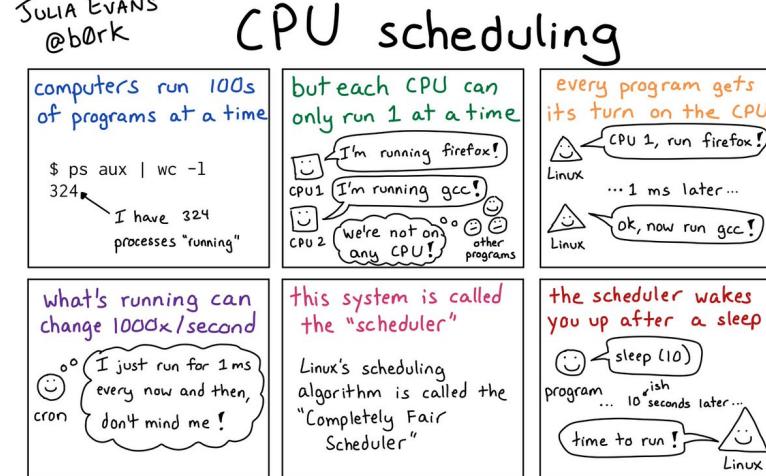
Prioridades de los procesos

Ya hemos visto cómo crear procesos y cómo finalizar la ejecución de los mismos. Además, se han introducido los distintos tipos de procesos así como que se ha mencionado que cuentan con prioridades.

La prioridad va a determinar, a la hora de acceder a la CPU, qué proceso necesita un acceso más inmediato a la misma. Una CPU no funciona nunca sola sino que va acompañada siempre de un gobernador (*governor*) el cual define qué proceso se ejecuta.

La política del gobernador suele ser expulsora, esto es, cuando ha consumido su rodaja de tiempo pasa a la cola de procesos en espera a su nuevo turno.

SUZA EVANS
@b0rk

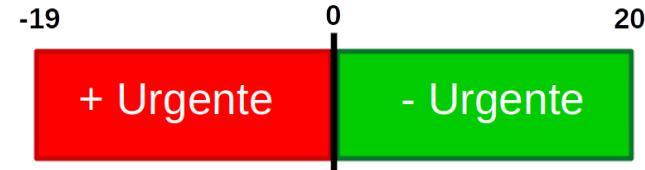


<https://twitter.com/b0rk/status/1215753312324157441>

Gestión de procesos

En Linux las prioridades se asignan en el rango de valores desde -19 hasta 20, donde el valor más pequeño es el más urgente y valores mayores son los menos urgentes (prioritarios). Dichas prioridades se asignan con el comando “nice” y se pueden cambiar en tiempo de ejecución con el comando “renice”.

Reducir la prioridad de un proceso propio se puede hacer directamente sin necesidad de permisos de administrador. Sin embargo, aumentar dicha prioridad (valores negativos) requiere siempre de acceso root.



```
# 'nice' sin parámetros incrementa la prioridad en 10
user@users-desktop:~$ nice top
Tareas: 525 total, 0 ejecutar, 524 hibernar, 0 detener, 0 zombie
%Cpus): 2,4 usuario, 0,9 sist, 1,2 adecuado, 94,3 inact, 0,0 en espera, 0,0 hardw int, 0,0 softw int, 0,0 robar tiempo
MIB Mem : 23902,7 total, 2113,2 libre, 5706,9 usado, 11082,9 búfer/caché
MIB Intercambio: 23437,0 total, 23437,0 libre, 0,0 usado. 16706,6 dispon Mem

PID USUARIO PR NI VIRT RES SHR S %CPU %MEM HORA+ ORDEN
826957 javinat+ 30 10 15040 4512 3408 R 16,7 0,0 0:00.05 top
^C
# Realiza una descarga con "wget" de forma silenciosa y con la mínima prioridad
user@users-desktop:~$ nice -n19 wget -q https://s.javinator9899.com/setup-jail-ssh"
# Ejecuta la compilación de un programa con mayor prioridad al resto de procesos
user@users-desktop:~$ sudo nice -n10 make -j12 all
```

```
# Cambia la prioridad del proceso 1001 a -19
user@users-desktop:~$ sudo renice -19 1001
# Cambia la prioridad del proceso 602 a 1
user@users-desktop:~$ renice 1 602
# Cambia la prioridad a todos los procesos del usuario "ext-users" a 5
user@users-desktop:~$ sudo renice 5 -u ext-users
# Cambia la prioridad de todos los procesos de los usuarios miembros del grupo "www-data"
user@users-desktop:~$ sudo renice 10 -g www-data
```

Gestión de procesos

```
● ● ●  
# Realizamos una descarga en segundo plano y la desasociamos del terminal  
user@users-desktop:~$ nohup wget https://github.com/quijsote.txt &  
[1] 878377  
nohup: se descarta la entrada y se añade la salida a 'nohup.out'  
user@users-desktop:~$ jobs  
[1] + running nohup wget  
user@users-desktop:~$ exit
```

```
● ● ●  
# Comprobamos en un nuevo terminal que efectivamente la descarga continúa  
user@users-desktop:~$ ps ux | grep wget  
javinat+ 878377 0.0 0.0 10704 592 ? SN 13:00 0:00 wget quijsote.txt  
user@users-desktop:~$
```

```
● ● ●  
# Iniciamos un proceso en segundo plano que espera 15 segundos  
user@users-desktop:~$ sleep 15 &  
[1] 893070  
user@users-desktop:~$ jobs  
[1] + running sleep 15  
# Desasociamos el proceso del terminal actual  
user@users-desktop:~$ disown  
# La lista de procesos activos en segundo plano está vacía  
user@users-desktop:~$ ps ux | grep sleep  
# Sin embargo, el proceso se sigue ejecutando en segundo plano  
user@users-desktop:~$ ps ux | grep sleep  
javinat+ 893070 0.0 0.0 10704 524 pts/0 SN 13:03 0:00 sleep 15
```

Los procesos, por defecto, se asocian siempre al terminal en ejecución (la terminal actual). Cuando se cierra un terminal, todos los trabajos en ejecución finalizan siguiendo el flujo siguiente:

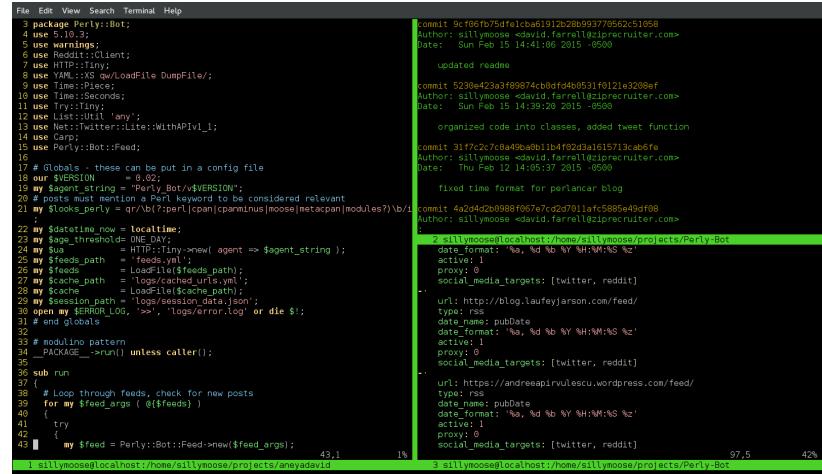
1. El proceso recibe la señal SIGHUP – 1.
2. Si no trata dicha señal, entonces recibe un SIGTERM – 15.
3. Si el proceso no finaliza, se le envía un SIGKILL – 9.

Usando el comando nohup se evita lo anterior ya que: el proceso deja de recibir datos desde `stdin`; redirige la salida estándar y de errores al fichero `nohup.out`; y previene que reciba la señal SIGHUP. Sin embargo, el comando sigue estando presente en la lista de trabajos de la shell. Si se quiere eliminar del todo, se puede usar el comando `disown` para tales efectos.

screen: una manera efectiva de ejecutar procesos en segundo plano

Hasta ahora, lo que hemos visto parecen más trucos: que si el proceso está en segundo plano, que si le quito las interrupciones, que si lo disocio del terminal, etc. Pero cada alternativa tiene su problema, y en común todos fallan en lo mismo: se pierde la entrada estándar. ¿Qué sucede si quiero ejecutar un terminal en segundo plano? ¿Y varios? Viene el comando screen al rescate.

screen permite crear “pantallas virtuales” en donde se inicia un nuevo terminal el cual está completamente disociado de la sesión de usuario.



The screenshot shows a terminal window with several tabs open, each displaying a different screen session. The sessions are labeled with commit hash, author, date, and a brief description of the changes made. The sessions include:

- Commit 9c706fb75d7e1cb61912b2899370562c5058, Author: sillymose <david.farrell@ziprecruiter.com>, Date: Sun Feb 15 14:41:06 2015 -0500, updated readme.
- Commit 5290e423a89874cb8df4b051f0121e3206ef, Author: sillymose <david.farrell@ziprecruiter.com>, Date: Sun Feb 15 14:38:20 2015 -0500, organized code into classes, added tweet function.
- Commit 31f7c2cc8d49ba011b4f02d3a1615713cabfe, Author: sillymose <david.farrell@ziprecruiter.com>, Date: Thu Feb 12 14:05:37 2015 -0500, fixed time format for perlancar blog.
- Commit 4a2d4d2b09888f067e7c2d7011af588fe49d108, Author: sillymose <david.farrell@ziprecruiter.com>, Date: Sun Feb 15 14:38:20 2015 -0500, 2 sillymose@localhost:/home/sillymose/projects/Perly-Bot.
- date_format: "%a %d %b %Y %H:%M:%S %z", active: 1, proxy: 0, social_media_targets: [twitter, reddit], url: http://blog.laufeyjarson.com/feed/, type: rss, date_name: pubDate, date_format: "%a, %d %b %Y %H:%M:%S %z", active: 1, proxy: 0, social_media_targets: [twitter, reddit], url: https://andreeapirulescu.wordpress.com/feed/, type: rss, date_name: pubDate, date_format: "%a, %d %b %Y %H:%M:%S %z", active: 1, proxy: 0, social_media_targets: [twitter, reddit], 1 sillymose@localhost:/home/sillymose/projects/Perly-Bot, 18, 49, 1, 97, 5, 4%

```
screen -S session_name
screen -d session_name
screen -ls
screen -r session_name
```

Gestión de procesos

Las sesiones de screen se gestionan siempre usando la combinación de teclas:

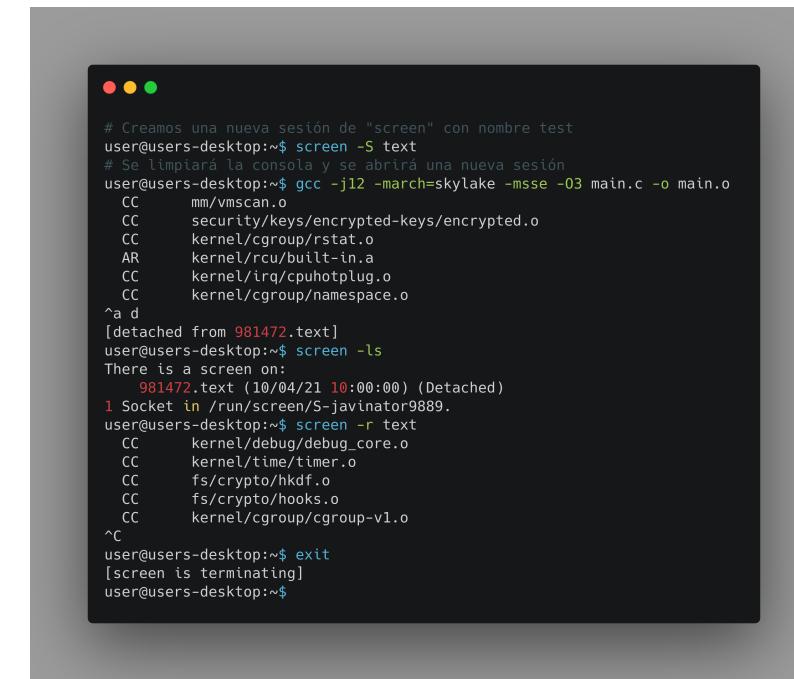
Ctrl + a <opción>

Por ejemplo, para salir de una sesión de screen pero dejar lo que sea ejecutando en segundo plano será Ctrl + a d.

Las sesiones de screen son persistentes, esto es, sobreviven a abrir/cerrar un terminal múltiples veces. Constituyen una forma muy útil de ejecutar aplicaciones interactivas en segundo plano que utilizarían la consola e impedirían que pudiésemos hacer múltiples acciones mientras tanto.

Pese a ser una gran utilidad, no viene instalado por defecto. Se hará con el comando:

sudo apt install screen



The screenshot shows a terminal window with a black background and white text. It demonstrates the creation of a new screen session named 'test' and then exiting it. The session contains a build command for a C program. The terminal also shows the listing of the screen sessions and the exit command.

```
# Creamos una nueva sesión de "screen" con nombre test
user@users-desktop:~$ screen -S test
# Se limpiará la consola y se abrirá una nueva sesión
user@users-desktop:~$ gcc -j12 -march=skylake -msse -O3 main.c -o main.o
CC      mm/vmscan.o
CC      security/keys/encrypted-keys/encrypted.o
CC      kernel/cgroup/rstat.o
AR      kernel/rcu/built-in.a
CC      kernel/irq/cpuhotplug.o
CC      kernel/cgroup/namespace.o
^a d
[detached from 981472.text]
user@users-desktop:~$ screen -ls
There is a screen on:
  981472.text (10/04/21 10:00:00) (Detached)
  1 Socket in /run/screen/S-javinator9889.
user@users-desktop:~$ screen -r text
CC      kernel/debug/debug_core.o
CC      kernel/time/timer.o
CC      fs/crypto/hkdf.o
CC      fs/crypto/hooks.o
CC      kernel/cgroup/cgroup-v1.o
^C
user@users-desktop:~$ exit
[screen is terminating]
user@users-desktop:~$
```

Cheat sheet de screen:
<https://gist.github.com/jctosta/af918e1618682638aa82>