

La Shell (II)

Departamento de Sistemas Telemáticos y Computación (GSyC)

gsync-profes (arroba) gsync.es

Octubre de 2013



©2013 GSyC
Algunos derechos reservados.
Este trabajo se distribuye bajo la licencia
Creative Commons Attribution Share-Alike 3.0

- 1 Usos no estándar de la barra
- 2 Ordenes internas
- 3 Permisos especiales
 - SUID
 - Sticky bit
- 4 Umask
- 5 source
- 6 Invocación de la shell
- 7 Tareas
- 8 Screen

Usos no estándar de la barra

Un principio básico para hacer buenos programas es *se laxo con lo que aceptas y estricto con lo que generas*

- `/d1//d2///d3/d4`

En rigor es un nombre incorrecto. Aunque normalmente se admite, porque la shell y las librerías lo *limpian* y generan `/d1/d2/d3/d4`

No hay garantía de que funcione siempre, es mucho mejor evitarlo

- `/d1/`

Algunas órdenes y algunos documentos muestran una barra al final de un directorio para indicar que se trata de un directorio y no un fichero ordinario (de la misma manera que puede usarse un color distinto)

Algunas órdenes pueden esperar que un nombre acabado en barra sea un directorio

Pero no es un nombre estándar, es preferible evitarlo

- Para la orden cp de Mac OS
cp -r d/ .
significa
cp -r d/* .
(pero solo para cp -r y solo para Mac OS)

Ordenes internas

La mayoría de las órdenes son externas

Pero todas las shell interpretan ciertas órdenes por sí mismas: Las órdenes internas (*builtin commands*)

- Por razones de eficiencia: `echo`, `kill`, `pwd`, `test`...
Son internas aunque también tienen versión externa
- Necesariamente internas:
`cd`, `export`, `alias`, `unset`, `exit`...
Realizan funciones que tienen que hacerse forzosamente en el proceso de la shell, harían algo completamente diferente si se implementan como ejecutables externos

```
koji@mazinger:~$ type echo  
echo es una función integrada en la shell
```

alias

Reemplaza una cadena por otra

- `alias c='clear'`
Expande `c`, se convierte en `clear`
- `alias`
Muestra todos los alias
- `unalias c`
Deshace el alias

alias suele definirse en `.bashrc`

Hay ataques/bromas basados en alias

Funcionamiento de la shell

- 1 La shell lee texto de cierto fichero (stdin). Frecuentemente el texto lo está escribiendo el usuario, así que aporta algunas facilidades (borrar, autocompletar, history)
- 2 Analiza el texto (expande metacaracteres, variables, alias)
- 3 Busca la primera palabra, para ver si se trata de un ejecutable
 - Primero la busca entre las órdenes internas
 - Si no es interna, busca el ejecutable en ciertos directorios (los indicados en el PATH)
- 4 Aplica las redirecciones que correspondan
- 5 Ejecuta, pasando el resto de palabras como argumento
- 6 Duerme
 - A menos que lancemos el ejecutable en *background*
`acoread file.1 &`

History

Facilita la entrada de líneas

- (cursor arriba y abajo)
Muestra, una a una, las órdenes introducidas
- !<cadena>
Repite la última orden que empiece por <cadena>
- history
Muestra el historial de órdenes introducidas
- !<n>
Repite la orden <n>

SUID

Sea un fichero perteneciente a un usuario

```
-rwxr-xr-x 1 koji koji 50 2009-03-24 12:06 holamundo
```

Si lo ejecuta un usuario distinto

```
invitado@mazinge:~$ ./holamundo
```

El proceso pertenece al usuario que lo ejecuta, no al dueño del fichero

```
koji@mazinge:~$ ps -ef |grep holamundo
invitado  2307  2260  22  12:16 pts/0    00:00:00 holamundo
koji      2309  2291   0  12:16 pts/1    00:00:00 grep holamundo
```

Este comportamiento es el normal y es lo deseable habitualmente

Pero en ocasiones deseamos que el proceso se ejecute con los permisos del dueño del ejecutable, no del usuario que lo invoca

- Esto se consigue activando el bit SUID (*set user id*)

```
chmod u+s fichero
```

```
chmod u-s fichero
```

En un listado detallado aparece una *s* en lugar de la *x* del dueño (o una *S* si no había *x*)

- El bit SUID permite que ciertos usuarios modifiquen un fichero, pero no de cualquier manera sino a través de cierto ejecutable

```
-rwsr-xr-x 1 root root 29104 2008-12-08 10:14 /usr/bin/passwd  
-rw-r--r-- 1 root root 1495 2009-03-23 19:56 /etc/passwd
```

- El bit SUID también puede ser un problema de seguridad
- En el caso de los scripts, lo que se ejecuta no es el fichero con el script, sino el intérprete
Un intérprete con bit SUID es muy peligroso, normalmente la activación del SUID en un script no tiene efecto
- Para buscar ficheros con SUID activo:
`find / -perm +4000`
- El bit SGID es análogo, cambia el GID
`chmod g+s fichero`

Sticky bit

- En ficheros ya no se usa
- En un directorio, hace que sus ficheros solo puedan ser borrados o renombrados por el dueño del fichero, del directorio o el *root*

Se representa con una *t*, en el listado y en `chmod`

```
chmod [+-]t directorio
```

```
drwxrwxrwt 15 root root 4096 2007-02-21 13:36 /tmp/
```

Si el directorio no tuviera permiso de ejecución, aparecería *T*

```
drwxrwx-wT
```

Umask

Orden interna que muestra y cambia la variable umask
(*user file creation mode mask*)

- `umask`
Devuelve el valor umask
- `umask nuevo-valor`
Cambia el valor umask

¿Qué permisos tiene por omisión un fichero recién creado?

- Ficheros: `666 and not umask`
- Directorios: `777 and not umask`

Ejemplo. Creación de un fichero
 Calculamos el valor de umask negado

umask	022	000	010	010
not umask	755	111	101	101

Hacemos *and lógico* entre 666 y el valor de umask negado

	666	110	110	110
		and		
not umask	755	111	101	101

	644	110	100	100
		rw-	r--	r--

source

Ejecuta un fichero en el entorno de la shell actual, que no muere. Las variables usadas en el fichero importado serán por tanto variables del proceso actual

El mandato *punto* (.) es equivalente, (aunque puede resultar menos legible)

- `. ~/.bashrc` # Ejecuta el código de `.bashrc`
 # en el entorno actual
- `source ~/.bashrc` # Forma equivalente

Invocación de la shell

- Es frecuente desear que todas nuestras sesiones ejecuten o configuren algo, sin necesidad de teclearlo a mano cada vez. Para hacer esto necesitamos saber cómo funciona la *invocación de la shell*
- Cada vez que se invoca una shell, esta ejecuta (con source) cierto fichero
- Típicamente esto se emplea para definir y exportar variables de entorno, modificar el prompt, declarar alias...
- Cada tipo de shell ejecuta un fichero diferente
 - Una shell puede ser de login o no de login
 - Una shell puede ser interactiva o no interactiva

- Una **shell de login** es aquella en la que el usuario ha introducido login y contraseña
- En general, una **shell interactiva** es aquella que tiene `stdin` redirigida desde la consola de un usuario, y `stdout` y `stderr` redirigidos a la consola de un usuario

Bash interactivo y de login

Ejemplos:

- Una sesión en una máquina sin gráficos (p.e. un Unix antiguo, un router...)
- Una sesión sin gráficos en una máquina con gráficos, que se inicia pulsando Ctrl+Alt+F1
- Entrar por ssh en una máquina

En este caso, la shell

- Lee y ejecuta `/etc/profile`
- Después, ejecuta el primero que encuentre de
 - `~/.bash_profile`
 - `~/.bash_login`
 - `~/.profile`

No se ejecuta `.bashrc`, a menos que `.bash_profile` lo llame.

Al terminar ejecuta

`~/.bash_logout`

Bash interactivo, no de login

Ej: Un terminal en Gnome o en Fluxbox

Se ejecuta

- `~/.bashrc`

No se ejecuta `~/.bash_profile`

Bash no interactivo, no de login

Ej: Un script

- Se ejecuta el fichero `$BASH_ENV`

- Antes del `.bashrc` de cada usuario, se ejecuta `/etc/bash.bashrc`, común para todos los usuarios
- Cuando se crea un usuario con `adduser`, se copia en su *home* todos los fichero que haya en `/etc/skel` (aquí se guardan los ficheros de configuración por omisión para cada usuario)
- Hablamos siempre del inicio de la shell. No debemos confundir todo esto con los niveles de ejecución, que se refieren al inicio de la máquina (directorios `/etc/rc2.d`, `/etc/rcS.d`, etc)

- Actualmente la diferencia entre shell de login y shell no de login es algo artificial ¹
- Hoy no suele resultar conveniente tener un fichero para las de login (`~/.bash_profile`) y otro distinto para las que son no de login (`~/.bashrc`)
- Por tanto, lo normal es configurar todo lo necesario en `~/.bashrc` y tener en `~/.bash_profile` únicamente una llamada a `~/.bashrc`, de la siguiente manera:

```
if [ -f ~/.bashrc ]; then    # si existe .bashrc
    . ~/.bashrc            # ejecuta .bashrc
fi
```

O lo que es lo mismo

```
if test -f ~/.bashrc ; then # si existe .bashrc
    source ~/.bashrc        # ejecuta .bashrc
fi
```

¹En un linux con gráficos, una sesión ordinaria no ejecuta ninguna shell de login, mientras que en MacOS todas las shell que ejecuta el usuario son de login

Control de tareas (jobs)

- Para lanzar varios procesos que se ejecuten en paralelo lo más cómodo suele ser abrir varias shells (una nueva terminal o una nueva pestaña en el terminal o un multiplexor de terminales como screen)
- Pero también es posible desde una única shell manejar varios procesos simultáneamente: mediante el control de tareas (jobs)

Un proceso puede ejecutarse en primer o en segundo plano

- En primer plano (*foreground*) recibe órdenes desde el teclado, como Ctrl Z (detener temporalmente) o Ctrl C (finalizar)
Cada shell solo puede tener un proceso en primer plano
- En segundo plano no tiene vinculada su entrada estándar desde el teclado, no recibe las señales Ctrl Z o Ctrl D. Es necesario emplear *kill*

Puede haber varios procesos en segundo plano

De la misma manera, un proceso detenido puede estar tanto en primer como en segundo plano

- La orden *jobs* indica, en cada línea, número de tarea, estado y nombre

```
koji@mazinger:~$ jobs
[1]  Ejecutando          xcalc &
[2]- Ejecutando          evince &
[3]+ Detenido           gedit
```

- El signo + indica tarea por omisión, aquella que se sobreentiende si no se indica número de tarea. Si la tarea por omisión muere, la siguiente será la marcada con el signo -
- `kill %n` envía señal al proceso con el job `n` (El símbolo de porcentaje indica nº de job, su ausencia indica pid)
- Algunos programas multiproceso, complejos, aunque los lancemos desde una shell no son hijos de esa shell y no figurarán en la lista de tareas. Por ejemplo firefox o nautilus

- `fg n`
pone la tarea `n` en ejecución en primer plano
- `bg n`
pone la tarea `n` en ejecución en segundo plano
El resultado es el mismo que si hubiéramos lanzado la orden con el símbolo `&`

Las órdenes `bg` y `fg` pueden lanzarse sin indicar `n`, entonces se sobreentiende la tarea por omisión.

La orden `kill` necesita que se le indique siempre explícitamente el número de tarea o el número de `pid`

```
vmstat 1 Lanzo vmstat, indicando que se actualice cada 1 segundo.  
Ctrl Z Detengo el proceso. La shell me indica su número de trabajo.  
fg 1 El trabajo 1 vuelve a primer plano. No puedo usar la shell.  
Ctrl Z Vuelvo a detenerlo.  
jobs Listado de todos los trabajos.  
bg 1 El trabajo 1 se ejecuta en segundo plano. Sigue escribiendo  
en stdout, pero puedo usar la shell.  
En este momento no puedo matarlo con ctrl C.  
fg El trabajo pasa a primer plano, puedo matarlo.
```

nohup

- Normalmente, cuando un usuario cierra una sesión, todos sus procesos reciben la señal SIGHUP y mueren
- Si tenemos procesos que queremos que se continúen ejecutando aunque el usuario cierre la sesión, podemos usar `nohup`
 - `nohup <orden>`
`<orden>` ignorará la señal SIGHUP. Escribirá stdout en `./nohup.out` (o en `~/nohup.out`)
 - Si necesitamos stdin, es necesario redirigirla desde un fichero

Screen

- Screen es una alternativa a `nohup` mucho más potente: Además de mantener el proceso vivo cuando el usuario se desconecta, posteriormente se puede seguir usando interactivamente `stdin` y `stdout`
- Otra ventaja:
 - Normalmente, si deseo tener `n` sesiones en una máquina remota, es necesario abrir `n` conexiones mediante `ssh`
 - Usando `screen`, puedo abrir una única conexión `ssh` a una sesión `screen`, y en ella usar `n` ventanas
- Inconvenientes:
 - No es POSIX
 - Es necesario memorizar media docena de atajos de teclado

Screen maneja *sesiones*

- Una sesión de screen permite que un usuario se desasocie de ella (*dettach*). El usuario puede desconectarse y la sesión permanece (todos los procesos se siguen ejecutando). Cuando el usuario vuelva a conectarse (típicamente por ssh) puede reasociarse (*reattach*)

En cada sesión puede haber diferentes *ventanas*

- No son ventanas al estilo Windows / X Window ni incluso ncurses
- Se parece a tener varias pestañas en un gnome-terminal, o a diferentes sesiones en alt F1, alt F2

Uso típico

```
screen          Creamos una sesión de screen y nos asociamos a ella
screen -ls      Vemos listado de sesiones
screen -d       Nos desasociamos de la sesión actual
```

Desconectamos ssh o cerramos el terminal. Volvemos a conectarnos

```
screen -ls      Vemos listado de sesiones
screen -r nombre Nos reasociamos a la sesión
```


Uso de ventanas

Ordenes básicas para el uso de ventanas en screen

- `ctrl a "` Ver todas las ventanas de la sesión a las que estoy asociado, con los cursores elijo una.
- `ctrl a c` Crear una nueva ventana dentro de la sesión actual (c minúscula)

Otras órdenes para el uso de ventanas

- `ctrl a A` Cambiar el nombre a la ventana actual (A mayúscula)
- `ctrl a ?` Ayuda

Observaciones

Si queremos desconectarnos de la máquina manteniendo la sesión de screen para usarla en otro momento

- Nos desasociamos de la sesión (screen -d)
- Cerramos la shell (exit/Ctrl d)

Si queremos cerrar una sesión de screen

- Cerramos ordenadamente todas las ventanas (todas las shell, con exit o ctrl d). Esto cierra de forma definitiva todos los procesos

Si vemos una sesión como

```
koji@mazinger:~$ screen -ls
There is a screen on:
      4680.pts-3.mazinger      (18/01/11 12:54:05)      (Attached)
1 Socket in /var/run/screen/S-koji.
```

Esto significa que la sesión de screen 4680 ya tiene un terminal asociado, pero no sabemos si es el nuestro o es otro. Podemos saber si nuestro terminal está en alguna sesión de screen con `ctrl a "`

- Si estamos en screen, veremos un listado de sus ventanas
- Si no, veremos "

Multi display mode

- `screen -r -x <nombre_sesion>`

Nos asocia a una sesión de screen aunque ya haya otra sesión asociada

(Podremos usar ambos terminales)

Encontraremos un buen tutorial sobre screen *googleando*:

- *Jeff linux screen tutorial*

Una vez familiarizados con screen, podemos usar *byobu*, un recubrimiento de screen con algunas mejoras en el interfaz de usuario