1 Introdución á Shell

O **intérprete de comandos** ou ***Shell*** é a interfaz que permite ao usuario interactuar có sistema. O usuario introduce as súas órdenes, o intérprete as procesa e xera a saída correspondente.

> O intérprete de comandos é tanto unha interfaz de execución de ordes e utilidades como unha linguaxe de programación que admite crear novas ordes (denominadas "guións" ou *shellscripts*), utilizando combinacións de comandos e estruturas lóxicas de control, que contan con características similares ás do sistema e que permiten que os usuarios e grupos da máquina conten cun entorno personalizado.
> É dicir que, cando se inicia unha sesión coa ***Shell***, e se traballa cun contorno de texto (non gráfico), estase empregando esta fiestra como intérprete de comandos. Isto é, a fiestra *shell* mostra a liña de comandos, o usuario escribe un comando e a *shell* executa o comando e mostra de novo a liña de comandos.
>
> ◊ Tríptico de comandos

1.1 Sumario

1.2 Tipos de Shell

En Unix existen dúas familias principais de intérpretes de comandos:

- Os baseados no intérprete de Bourne: SH, KSH e BASH.
- Os baseados no intérprete C: CSH ou TCSH.

A continuación, móstrase unha pequena reseña de cada un deles:

- **Sh** (Bourne **Sh**ell). Foi desenrolado por Steve Bourne para AT&T, sendo durante moitos anos o "Shell patrón" do sistema operativo UNIX. É tamén coñecido como Standard Shell, xa que todas as distribucións Linux dispoñen del.
- **Ksh** (**K**orn **Sh**ell). Desenrolado por David Korn, é un superconxunto de sh e dispón, polo tanto, de todas as facilidades de sh e outras moitas mais agregadas. A compatibilidade con sh é total.
- **Bash** (**B**ourne **A**gain **Sh**ell). É o Shell mais moderno e probablemente o mais utilizado, debido a que se trata do Shell Estándar de GNU/Linux, ademais de ser moi intuitivo e flexible. É o axeitado para introducirse no mundo da programación Shell, pero é, ao mesmo tempo, unha potente ferramenta para usuarios avanzados. Considérase ademais un superconxunto do **sh**, polo que os comandos que funcionan en sh tamén o fan en bash, pero non ao contrario.
- **Csh** (**C Sh**ell). Desenrolado por Bill Joy da Universidade de Berkeley, é o Shell mais utilizado en sistemas BSD. A estrutura dos seus comandos é bastante similar á do linguaxe C. Non é unha Shell compatible sh.
- **Tcsh** (**T**ENEX **C Sh**ell). Trátase dun superconxunto de csh bastante mais rápido. Esta característica fai que sexa coñecido como Turbo Shell C.

Para coñecer os Shell dos que se dispón nun contorno Linux, débese consultar o ficheiro **/etc/shells**.

1.3 Que *shell* estamos a executar

A cada usuario se lle asigna un único Shell por defecto. Dita asignación atópase no ficheiro **/etc/passwd**.

```
usuario@pc:~$ cat /etc/passwd | grep 'usuario'
usuario:x:1000:1000:usuario,,,:/home/usuario:/bin/bash
```

Podemos identificar a *shell* que estamos a utilizar lendo o valor da variable **SHELL**:

```
$ echo $SHELL
/bin/bash
## Para ver a versión do shell bash que estamos empregando:
$ echo $BASH_VERSION
4.3.24(1)-release
```

Pódese cambiar a Shell por defecto coa orde **chsh** (change Shell). Os cambios feitos modifican o ficheiro */etc/passwd*, polo que será preciso reiniciar a sesión do usuario para ver o efecto. Un exemplo sería o seguinte:

```
$ chsh -s /bin/bash
```

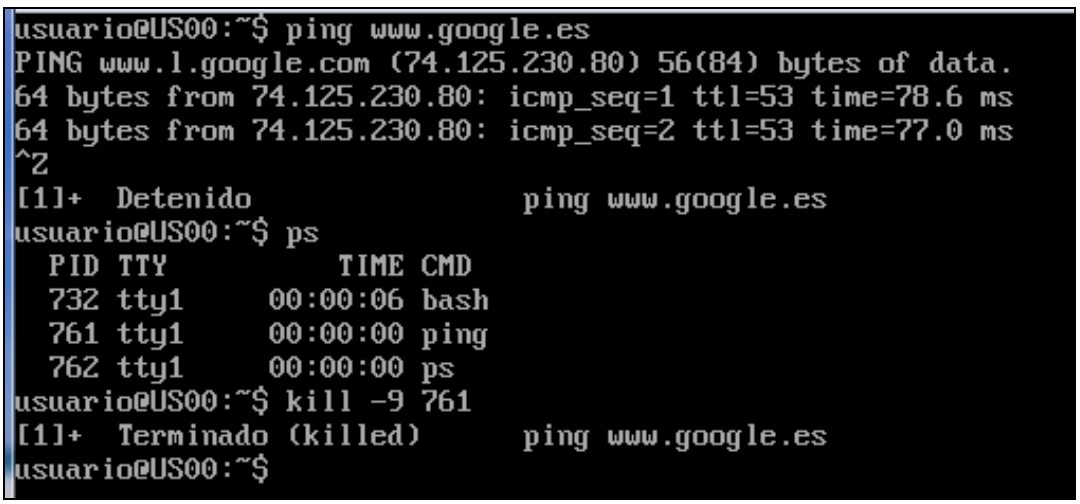
Tamén se pode obter un cambio temporal invocando directamente ao programa Shell dende a liña de ordes da Shell. Este cambio permanecerá activo mentres non se finalice a Shell coa orde **exit**.

```
## Un exemplo có shell tcsh pode ser o seguinte:
$ /bin/tcsh
## Se non está instalado no equipo, antes executar:
## $ apt-get install tcsh
#
## Para conocer a versión do tcsh que estamos a executar:
$ echo $version
tcsh 6.18.01 (Astron) 2012-02-14 (x86_64-unknown-linux) options wide,nls,dl,al,kan,rh,nd,color,filec
## Para saír do tcsh e voltar ao bash:
$ exit
exit
```

1.4 Corrección de erros

- Eliminar un carácter: **Supr**.
- Eliminar unha palabra: **Control-W**.
- Eliminar unha liña: **Control-U** ou **Control-X**.
- Abortar a execución:

  - Empregando a tecla de interrupción **Control-X**.
  - Empregando a tecla de suspensión **Control-Z**, logo o comando *ps* para verificar o número do traballo do programa, e utilizando o comando *kill* para abortar o programa:



- Repetición/Execución da liña de comandos: **Frecha arriba**.

1.5 Acceso de súper usuario

Cando utilicemos *root* como nome de usuario para iniciar a sesión, teremos permisos de súper usuario ou administrador. Podemos ler e escribir os arquivos do sistema, executar programas que os usuarios normais non poden, e mais. Cando instalemos unha distribución de Linux no noso equipo, asignaremos un contrasinal para o súper usuario.

- ***sudo:*** O programa *sudo* (abreviatura do inglés *superuser do* ou *substitute user do*) é unha utilidade dos sistemas operativos tipo Unix, como GNU/Linux, BSD, ou Mac OS X, que permite aos usuarios executar programas cós privilexios de seguridade de outro usuario (normalmente o usuario *root*) de maneira segura.

  Os usuarios deben confirmar a súa identidade ao executar *sudo* dando o seu propio contrasinal antes de executar o programa requirido. Unha vez autenticado o usuario, e se o arquivo de configuración /**etc**/**sudoers** permite dar ao usuario acceso ao comando requirido, o sistema o executa e o rexistra.

  - Axuda sobre configuración de usuarios.

Os sistemas operativos Ubuntu e Mac OS X forzan a facer todo acceso administrativo por medio de ***sudo***, pois o contrasinal de *root* está desactivada por defecto; aínda que se pode activar coa axuda do programa ***passwd*** en Ubuntu.

1.6 Onde atopar axuda?

A documentación en liña é a mais empregada en Linux, o manual (*man*) e as páxinas *info* están dispoñibles dende as primeiras versións do sistema operativo.

- **A opción *--help***

  Vexamos un exemplo:

  ```
  $ cat --help
  ```

  Se a información que mostra se sae da pantalla, utilice o modificador *less* utilizando unha tubería:

  ```
  $ cat --help | less
  ```

- ***man*: Mostra o manual do sistema**

  A utilidade *man* (de manual) mostra as páxinas da documentación de sistema. Esta documentación é útil cando se sabe que utilidade se que empregar pero xa nos esquecimos como facelo.

  ```
  $ man nano
  ```

  Instalación das páxinas *man* en español:

  ```
  $ apt-get install manpages-es
  $ apt-get install manpages-es-extra
  ```

- ***info*: Mostra información sobre as utilidades**

  A utilidade baseada en texto *info* é un sistema hipertexto baseado en menús desenrolado dentro do proxecto GNU e distribuído con Linux.

1.7 Historial da consola

Un atallo moi útil é o "historial" da consola, este garda un rexistro de todos os comandos escritos.

```
# Ver todo o historial almacenado:
$ history
...
513 echo $BASH_VERSION
514 clear
515 history
```

```
##
## Para executar un comando polo número de posición:
$ !514
##
## Aumentar o número de comandos gardados:
$ nano /etc/profile
# Engadimos unha liña ao final:
HISTSIZE=5000
##
## Para limpar o historial:
$ history -c
```

- O historial de bash almacénase no arquivo .bash_history do directorio principal do usuario. Trátase dun arquivo de texto convencional.

1.8  *'Configuración da consola bash'*

Os arquivos de configuración de bash son, realmente, scripts da consola bash:

- Principais arquivos de configuración xerais: **/etc/bash.bashrc** e **/etc/profile**.
- Principais arquivos de configuración dos usuarios: **~/.bashrc** e **~/.profile**.

- Para buscar un comando xa executado o mellor é empregar a combinación de teclas **Control+R**, deste xeito comézase a buscar comandos "cara atrás".

  · Os caracteres escritos a buscar no comando non teñen que ser os primeiros escritos, poden ser do medio, do final, ...
  · Logo pódese seguir premendo **Control+R** para realizar a búsqueda ata que se atope o comando desexado (sempre "cara atrás").
  · Para ir "cara adiante" emprégase **Control+S**.
  · Se queremos rematar a búsqueda empregaremos **Control+G**.

Para o manexo do historial de comandos

Comando	Descrición
<up-arrow>/<down-arrow>	Comando anterior/posterior
!!	Último comando ejecutado
!n	n-ésimo comando do historial
!-n	n comandos cara atrás
!cadea	Último comando executado que empeza por cadea
!?cadea	Último comando executado que contén por cadea
^cadea1^cadea2	Executa o último comando cambiando cadea1 por cadea2
Ctrl+r	Busca cara atrás no historial
fc	Permite ver, editar e reexecutar comandos do historial

1.9 Fin de sesión

Para finalizar a sesión, utilizaremos a combinación **Control-D** ou utilizaremos o comando *exit* na liña de comandos.

1.10 Uso de consolas virtuais

Cando se executa Linux nun ordenador persoal, normalmente utilizaremos a pantalla e o teclado conectados ao equipo. O uso dunha consola física, permítenos acceder ás 63 consolas virtuais (tamén chamados terminais virtuais). Algúns deles están configurados para permitir o inicio de sesión, mentres que outras actúan como pantallas gráficas. Para cambiar entre elas, utilizaremos a combinación **Control-Alt** e a tecla de función da consola

que nos interese, por exemplo, **Control-Alt-F5** mostra a quinta consola virtual.

1.11 Cambio de contrasinal

Para cambiar o contrasinal hai que empregar o comando *passwd* dende a liña de comandos:

```
$ passwd
```

1.12 Configuración do idioma

En modo texto nos atoparemos có problema dos acentos e das eñes, para solucionalo faremos o seguinte.

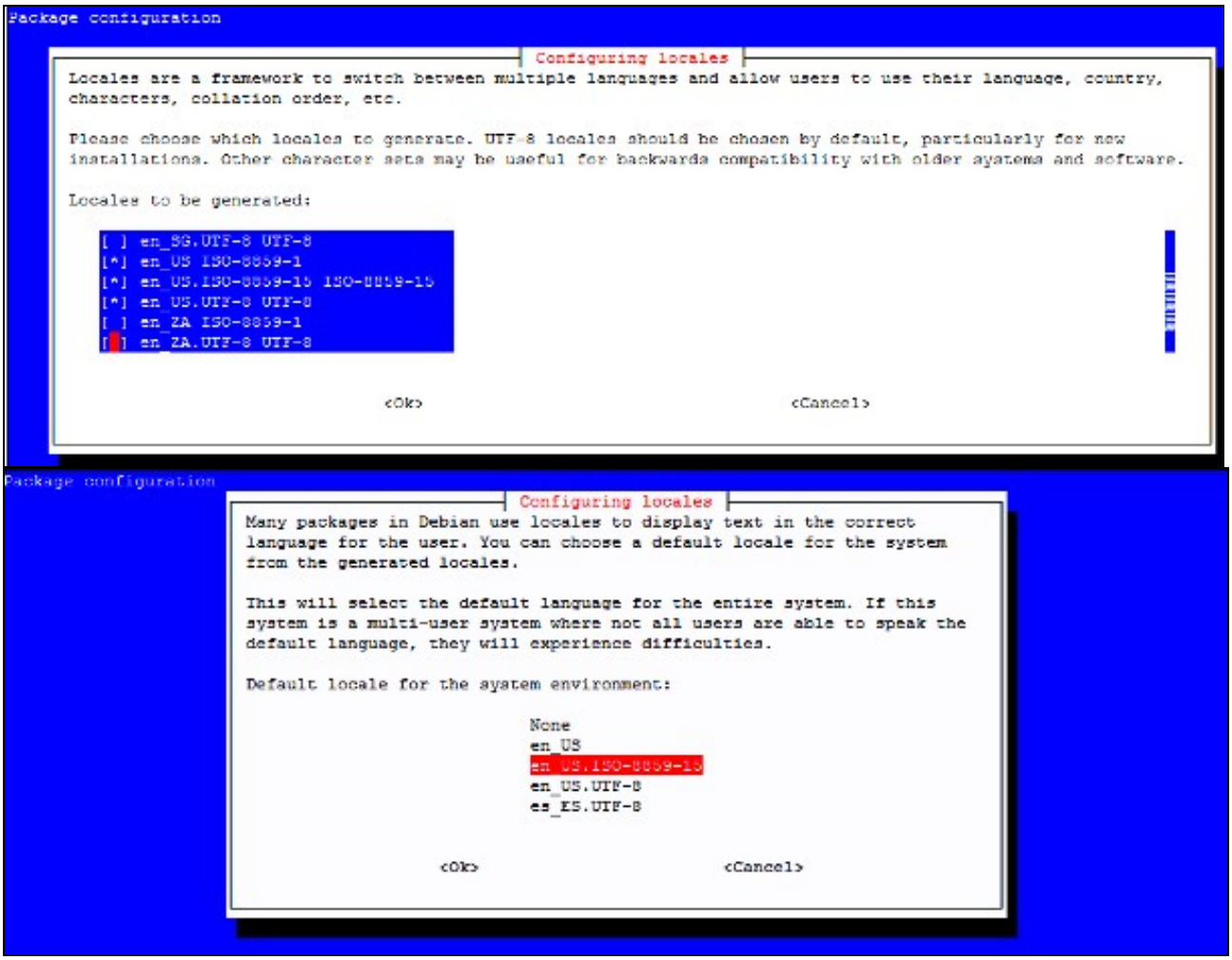
- O obxectivo é facer que a variable **LANG** teña o seguinte valor:

```
$ echo $LANG
en_US.ISO-8859-15
```

- Para conseguilo faremos o seguinte:

```
$ dpkg-reconfigure locales
```

E aparecerá a seguinte aplicación, onde debemos escoller as linguaxes:



Logo, só queda, cerrar sesión e volver a abrila.

1.13 Metacaracteres

Todos os Shells posúen un grupo de caracteres que, en diferentes contextos, teñen diferentes significados: os **metacaracteres**. Estes xogan un papel importante cando o Shell está analizando a liña de ordes, antes de executala. Os diferentes grupos afectan a aspectos separados do procesamento da liña de ordes.

1.13.1 Metacaracteis sintácticos

Utilízanse como caracteres especiais de puntuación entre ordes. Serven para combinar varias ordes có obxecto de construir unha única orde lóxica.

- **Unindo ordes con ;**

  O uso do "punto e coma" (;) como separador permite escribir dúas ou mais ordes na mesma liña. As ordes se executan secuencialmente, coma se estivesen escritas en varias liñas.
  En programas Shell pódense utilizar por razóns estéticas, para agrupar ordes relacionadas.
  Na liña de ordes nos permite executar varias sen ter que esperar a que se complete unha antes de introducir a seguinte.
  Un exemplo sería o seguinte:

```
$ cd /var/log; ls error-*
```

- **Creando cauces con |**

  O contorno de execución dun programa utiliza varios arquivos no proceso:
  - A entrada estándar (*stdin*), con descritor de arquivo (*hadless*) **0**, e que corresponde por defecto có teclado.
  - A saída estándar (*stdout*), con descritor de arquivo (*hadless*) **1**, e que corresponde por defecto có monitor.
  - O erro estándar (*stderr*), con descritor de arquivo (*hadless*) **2**, e que corresponde por defecto cós arquivos de erro estándares.
  Pero, este comportamento pode cambiarse.
  Os cauces (*pipes*) son unha característica distintiva dos sistemas GNU/Linux. Un cauce conecta a saída estándar da orde que aparece á esquerda do símbolo | coa entrada estándar da orde que aparece á dereita. O fluxo de información entre ambas ordes realízase ao través do kernel do sistema operativo.
  No seguinte exemplo, no que empregamos a orde **who**, xérase un informe ordenado alfabéticamente dos usuarios que se atopan conectados actualmente no sistema:

```
$ who | sort
```

- **Combinando ordes con ()**

  En ocasións, necesítase aillar un cauce ou unha "secuencia de punto e coma" do resto dunha liña de ordes. Para ilustralo vexamos un exemplo no que empregamos a orde **date**, que informa da data e hora do sistema, e a orde **wc**, que mostra un número de liñas, palabras e caracteres dun arquivo que se pasa como argumento, xunto coa orde **who** empregada no exemplo anterior:

```
$ date ; who
vie jul 8 17:16:07 CEST 2011
gonzalo tty1 2011-07-08 16:54
gonzalo pts/0 2011-07-0 16:54 (:0.0)
$ (date ; who) | wc
   3   15   115
```

- **Executando ordes en segundo plano con &**

  GNU/Linux permite a execución de ordes dende o Shell, denominada execución en primeiro plano (*foreground*), ou desligadas do indicador, chamada execución en segundo plano (*background*). No primeiro caso, o indicador non aparece ata que finalizou a orde actual. No segundo, o indicador aparece inmediatemente, o que permite executar outras ordes ainda que a que se lanzou no segundo plano non rematara.
  Para executar unha orde en segundo plano, basta con finalizar a liña de ordes con **&**. O Shell responde a este carácter indicando o número de traballo que se lanzou (aparece entre corchetes), seguido do identificador do proceso que executa a orde.
  No seguinte exemplo lánzase, en segundo plano, o editor **gedit**:

```
$ gedit &
[1] 5650
```

- **Executando condicional de ordes con || e &&**

A Shell subministra dous metacaracteres que permiten a execución condicional de ordes baseada no estado de finalización. Separar dúas ordes con || ou con **&&** provoca que a Shell comprobe o estado de finalización da primeira e execute a segunda só si a primeira faia ou ten éxito.
Vexamos uns exemplos, onde sería interesante repasar a función grep:

```
$ who | grep root || echo "root non está conectado"
root non está conectado
$ who | grep gonzalo && echo "Usuario gonzalo conectado"
gonzalo tty1    2011-10-10 08:42
Usuario gonzalo conectado
```

1.13.2 Metacaracteis de nomes de arquivo

Estes metacaracteres utilízanse para formar patróns de igualación para a substitución de nomes de arquivos, co obxecto de poder referenciar de forma abreviada unha serie de arquivos con nomes que seguen un patrón.

- **Igualando un carácter simple con ?**

  É moi frecuente crear arquivos con algún patrón como parte do seu nome, sobre todo en procesamento por lotes. Así, por exemplo, se queremos obter información dos arquivos con **ls**, podemos empregar a orde:

```
$ ls -l arquivo?
... arquivo1
... arquivo2
... arquivo3
```

- **Igualando cero ou mais caracteres con ***

  Có metacarácter ***** pódense igualar cero o mais caracteres.

```
$ ls -l arq*
... arquivo1
... arquivo2
... arquivo3
... arquiv01
... arquiv02
```

- **Igualando cero ou mais caracteres con []**

  Os corchetes definen unha lista ou clase de caracteres, que se poden igualar con un único carácter. Vexamos algunhas formas de caracterizar grupos de arquivos:

Exemplo	Descrición
[A-Z]*	Iguala todos os arquivos con nomes que comecen por unha letra maiúscula.
*[aeiou]	Iguala calquera arquivo con nomes que finalicen cunha vogal.
tema.*[13579]	Iguala os temas con nomes que finalicen cun número impar.
tema.0[1-3]	Iguala tema.01, tema.02 e tema.03.
[A-Za-z][0-9]*	Iguala os arquivos que teñan nomes que comecen cunha letra (maiúscula ou minúscula) seguida dun díxito e cero ou mais caracteres.

- **Abreviando nomes de arquivos con {}**

  O uso das chaves ({}), solas ou combinadas cós anteriores caracteres especiais (?, *, []), permite formar expresións de nomes de arquivos mais complexos. As chaves conteñen unha lista de un ou mais caracteres separados por comas. Cada ítem da lista utilízase en turnos para expandir un nome de arquivo que iguala a expresión completa na que están inmersas as chaves.
  Por exemplo, a{f,d,e}b expándese en afb, adb e aeb.

- **Abreviación do directorio home con ~**

Cando se utiliza o tilde (~), o Shell substitúea polo camiño absoluto do directorio /home. O tilde pode ser seguido do nome dun usuario do sistema. Nese caso, o Shell substitúea polo camiño do directorio home dese usuario.

1.13.3 Metacaracteis de citación

Existen tres metacaracteis de citación, que se utilizan para controlar cando deben protexerse o resto de metacaracteres da expansión ou interpretación do Bash. Denomínase **expansión** ao procedemento polo que a Shell substitúe por unha lista a ocorrencia dun carácter especial. A utilización correcta destes caracteres permite a construción de scripts mais complexos.

- **Eludir metacaracteis con** \

  Cando a Shell procesa unha orde, o primeiro que fai é expandir os metacaracteis, de forma que se substitúen polos seus respectivos valores para que, na execución, aparezan os valores, non os metacaracteis. "Eludir" metacaracteis significa evitar a súa interpretación pola Shell, de forma que estes permanezan na liña de ordes para ser procesados na execución.
  Vexamos uns exemplos:

```
$ sincomas="Sin comas"
$ echo $sincomas
Sin comas
$ concomas=\"Comas\"
$ echo $concomas
"Comas"
$ echo *
Descargas Documentos Escritorio Imágenes
$ echo \*
*
```

- **Protexendo metacaracteis con "**

  A barra invertida só protexe o carácter que lle segue inmediatamente, polo que si se queren protexer varios caracteres, cada un debería ir precedido dunha barra invertida (\), o que non é nada cómodo para protexer cadeas longas. Para protexer a cadea completa, utilízanse as comiñas dobres, as cales desactivan o significado especial dos caracteres entre elas, salvo os de **!evento** e **$var**.

```
$ usuario=Patricia
$ echo "**** Hola $usuario ****"
**** Hola Patricia ****
```

- **Protexendo ordes, variables e metacaracteis con ?**

  As comiñas simples son parecidas as comiñas dobres. Desactivan o significado especial de algúns caracteres, salvo a expansión da historia de ordes (!evento).

```
$ texto="primeira variable"
$ dentro="valor da $texto"
echo $dentro
$ dentro='valor da $texto'
$ echo $dentro
$ '!ola'
!ola: orden no encontrada
```

1.13.4 Metacaracteis de entrada/saída ou de redirección

Á hora de crear cauces, Unix herda tres ficheiros especiais da linguaxe de programación C, que representan as funcións de entrada e saída de cada programa. Estes son:

- **Entrada estándar.** Por defecto, procede do teclado; abre o ficheiro descritor **0** (*stdin*) para lectura.
- **Saída estándar.** Por defecto, diríxese cara a pantalla; abre o ficheiro descritor **1** (*stdout*) para escritura.
- **Saída de erro.** Por defecto, diríxese cara a pantalla; abre o ficheiro descritor **2** (*stderr*) para escritura e control das mensaxes de erro.

O proceso de redirección permite facer unha copia destes ficheiros especiais cara ou dende outro ficheiro normal. Tamén poden asignarse os descritores de ficheiros do 3 ao 9 para abrir outros tantos arquivos, tanto de entrada como de saída.

O ficheiro especial /**dev**/**null** emprégase para descartar algunha redirección e ignorar os seus datos.

Ademais, é posible misturar os fluxos de información da saída e erro estándares para que saian xuntos (polo mesmo descritor de arquivo, o 1).

Os metacaracteis de entrada/saída ou redirección podémolos ver na seguinte táboa:

Metacarácter	Descrición da función
< ficheiro	Redirecciona a entrada dunha orde para ler do ficheiro.
> ficheiro	Redirecciona a saída dunha orde para escribir no ficheiro. Se o ficheiro existe, o sobreescribe.
>| ficheiro	Como no caso anterior, pero o ficheiro debe existir previamente.
>& ficheiro	A saída de *stderr* combínase con *stdout* e se escriben no ficheiro.
>> ficheiro	A saída da orde se engade ao final do ficheiro.
>>& ficheiro	Engade a saída *stderr* combinada con *stdout* ao final do ficheiro.
[N]<> ficheiro	Redirección de entrada/saída entre o ficheiro e o arquivo con descritor **N**.
<<[-]delimitador Texto ... delimitador	Emprega o propio Shell *script* como entrada estándar, ata a liña onde se atopa o delimitador.
|	Crea un cauce entre dúas ordes. A saída estándar da orde da esquerda do símbolo conéctase á entrada estándar da orde da dereita do símbolo.
|&	Crea un cauce entre dúas ordes, coas saídas *stderr* e *stdout* da orde da esquerda combinadas e conectadas na entrada da orde da dereita.

- **Combinación de redireccións**

  Pódese combinar mais dunha redirección sobre a mesma orde ou grupo de ordes, interpretándose sempre de esquerda a dereita.
  Sería interesente interpretar as seguintes ordes:

```
$ ls -al /usr /tmp /noexiste >ls.sal 2>ls.err
$ find /tmp -print >find.sal 2>/dev/null
```

  Outra forma de combinar redireccións é realizar copias de descritores de ficheiros de entrada ou saída. A seguinte táboa mostra os formatos para duplicar descritores.

Metacarácter	Descrición da función
[N]<&M	Redirecciona a entrada dunha orde para ler do ficheiro.
[N]<&-	Redirecciona a saída dunha orde para escribir no ficheiro. Se o ficheiro existe, o sobreescribe.
[N]<&M-	Como no caso anterior, pero o ficheiro debe existir previamente.
[N]>&M	A saída de *stderr* combínase con *stdout*e se escriben no ficheiro.
[N]>&-	A saída da orde se engade ao final do ficheiro.
[N]>&M-	Engade a saída *stderr* combinada con *stdout* ao final do ficheiro.

  Convén facer notar que, seguindo as normas anteriores, as dúas liñas seguintes son equivalentes e ambas serven para almacenar as saídas normal e de erro no ficheiro indicado:

```
$ ls -al /var/* &>ls.txt
$ ls -al /var/* &>ls.txt 2>&1
```

  Con todo, o seguinte exemplo mostra dúas ordes que **NON teñen que dar o mesmo resultado,** xa que as redireccións se procesan de

esquerda a dereita, tendo en conta os posibles duplicados de descritores feitos nas liñas anteriores.

```
$ ls -al * >ls.txt 2>&1
# Saída normal e de erro a "ls.txt"
$ ls -al * 2>&1 >ls.txt
# Asigna a saída de erro á normal anterior
# e logo manda a saída estándar a "ls.txt"
```

1.14 Variables de entorno

Ao igual que calquera outro proceso, a Shell mantén un conxunto de variables que informan sobre o seu propio contexto de operación. O usuario (ou un *script*) pode actualizar e engadir variables, exportando os seus valores ao contorno do intérprete (comando **export**), o que afectará tamén a todos os procesos fillos xerados por el. O administrador pode definir variables de entorno estáticas para usuarios do sistema.

Na seguinte táboa aparecen as variables de entorno que ten o sistema.

Variable de entorno	Descrición	Valor por omisión
DISPLAY	Donde aparecen as saídas de X-Windows.	
EDITOR	Editor empregado por defecto.	
FUNCNAME	Nome da función que se está executando.	O modifica a Shell
HOME	Directorio persoal da conta.	O define *root* (superusuario)
HOSTNAME	Nome da máquina.	
IFS	Engade a saída *stderr* combinada con *stdout* ao final do ficheiro.	
LANG	Redirección de entrada/saída entre o ficheiro e o arquivo con descritor **N**.	
LINENO	Emprega o propio Shell *script* como entrada estándar, ata a liña onde se atopa o delimitador.	
LOGNAME	Crea un cauce entre dúas ordes. A saída estándar da orde da esquerda do símbolo conéctase á entrada estándar da orde da dereita do símbolo.	
OLDPWD	Crea un cauce entre dúas ordes, coas saídas *stderr* e *stdout* da orde da esquerda combinadas e conectadas na entrada da orde da dereita.	
PATH	Crea un cauce entre dúas ordes, coas saídas *stderr* e *stdout* da orde da esquerda combinadas e conectadas na entrada da orde da dereita.	
PPID	Crea un cauce entre dúas ordes, coas saídas *stderr* e *stdout* da orde da esquerda combinadas e conectadas na entrada da orde da dereita.	
PS1 ... PS4	Crea un cauce entre dúas ordes, coas saídas *stderr* e *stdout* da orde da esquerda combinadas e conectadas na entrada da orde da dereita.	
PWD	Crea un cauce entre dúas ordes, coas saídas *stderr* e *stdout* da orde da esquerda combinadas e conectadas na entrada da orde da dereita.	
SHELL	Crea un cauce entre dúas ordes, coas saídas *stderr* e *stdout* da orde da esquerda combinadas e conectadas na entrada da orde da dereita.	
TERM	Crea un cauce entre dúas ordes, coas saídas *stderr* e *stdout* da orde da esquerda combinadas e conectadas na entrada da orde da dereita.	
USER	Crea un cauce entre dúas ordes, coas saídas *stderr* e *stdout* da orde da esquerda combinadas e conectadas na entrada da orde da dereita.	

1.15 O comando echo

1.15.1 Un toque de cor

Ollo a este script:

```
#!/bin/bash
#
# Este ficheiro saca por pantalla un montón de códigos de cor
# para demostrar que hai dispoñible. Cada liña é unha cor
# fondo negro e gris, có código en medio. Funciona sobre
# fondos brancos, negros e verdes
#
echo "  Sobre gris claro:        Sobre negro:"
echo -e "\033[47m\033[1;37m  Blanco
    \033[0m\  1;37m \
    \033[40m\033[1;37m  Blanco
    \033[0m"
echo -e "\033[47m\033[37m  Gris Claro
    \033[0m\  37m \
    \033[40m\033[37m  Gris Claro
    \033[0m"
echo -e "\033[47m\033[1;30m  Gris
     \033[0m\  1;30m \
     \033[40m\033[1;30m  Gris
      \033[0m"
echo -e "\033[47m\033[30m  Negro
      \033[0m\  30m \
      \033[40m\033[30m  Negro
      \033[0m"
echo -e "\033[47m\033[31m  Rojo
      \033[0m\  31m \
      \033[40m\033[31m  Rojo
      \033[0m"
echo -e "\033[47m\033[1;31m  Rojo Claro
      \033[0m\  1;31m \
      \033[40m\033[1;31m  Rojo Claro
      \033[0m"
echo -e "\033[47m\033[32m  Verde
      \033[0m\  32m \
      \033[40m\033[32m  Verde
      \033[0m"
echo -e "\033[47m\033[1;32m  Verde Claro
      \033[0m\   1;32m \
      \033[40m\033[1;32m  Verde Claro
      \033[0m"
echo -e "\033[47m\033[33m  Marrón
      \033[0m\   33m \
      \033[40m\033[33m  Marrón
      \033[0m"
echo -e "\033[47m\033[1;33m  Amarillo
      \033[0m\   1;33m \
      \033[40m\033[1;33m  Amarillo
      \033[0m"
echo -e "\033[47m\033[34m  Azul
      \033[0m\  34m \
      \033[40m\033[34m  Azul
      \033[0m"
echo -e "\033[47m\033[1;34m  Azul Claro
      \033[0m\  1;34m \
      \033[40m\033[1;34m  Azul Claro
      \033[0m"
echo -e "\033[47m\033[35m  Púrpura
       \033[0m\   35m \
       \033[40m\033[35m  Púrpura
       \033[0m"
echo -e "\033[47m\033[1;35m  Rosa
       \033[0m\  1;35m \
       \033[40m\033[1;35m  Rosa
       \033[0m"
echo -e "\033[47m\033[36m  Cyan
```

```
        \033[0m\   36m \
        \033[40m\033[36m  Cyan
        \033[0m"
echo -e "\033[47m\033[1;36m  Cyan Claro
        \033[0m\  1;36m \
        \033[40m\033[1;36m  Cyan Claro
        \033[0m"
```

E a súa saída é a seguinte: