# Programación en *shell script*

**Diseño y Administración de Sistemas y Redes**

***Juan Céspedes <cespedes@gsyc.es>***

***Curso 2005–2006***



## ¿Qué es una *shell*?

- Interfaz entre el usuario y el S.O.
- Doble función:
  - Intérprete de la línea de comandos (comunicación con el S.O.)
  - Lenguaje de programación

# Algunas *shell*s

- `sh` o *Bourne Shell*
- `csh` (*C SHell*)
- `tcsh` (*Trustedi C Shell*)
- `ksh` (*Korn SHell*)
- `bash` (*Bourne Again SHell*)

*Shell*s disponibles en el sistema: `/etc/shells`





# Tipos de *shell*

- *Shell* de *login*
  - Lanzada por un proceso de *login*
  - Especificada en el fichero `/etc/passwd`
  - Puede cambiarse con la orden `chsh`
- *Shell* ordinaria
  - Lanzada después del proceso de *login*, por otros motivos (directamente por el usuario, por un *script*, etc)

# Configuración en bash

- *Shell* de *login*
  - Al arrancar:
    - `/etc/profile`
    - `~/.bash_profile`, `~/.bash_login`, `~/.profile` (solamente uno de ellos, el primero que exista)
  - Al terminar:
    - `~/.bash_logout`
- *Shell* ordinaria
  - Al arrancar:
    - `/erc/bash.bashrc`
    - `~/.bashrc`





# Órdenes de la *shell*

- Internas: control de la *shell* y programación:
  `alias`, `bg`, `break`, `case`, `cd`, `continue`, `echo`, `exec`, `exit`, `export`, `fg`, `for`, `help`, `history`, `if`, `jobs`, `read`, `return`, `type`, `unalias`, `unset`, `until`...
- Órdenes externas: ejecución de otro programa:
  `cat`, `ls`, `mkdir`...

Para saber de qué tipo es una orden: "`type` *`orden`*"

Lista de órdenes internas y descripción de cada una: "`help`"

Las órdenes externas se buscan en la variable `$PATH`

```
blas:~# echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/bin/X11
```

# Primer plano, segundo plano

Hay 3 estados en la ejecución de un proceso:

- En primer plano (*foreground*)
  - Ejecución normal. La *shell* espera a que termine la ejecución para seguir con la siguiente orden.
- En segundo plano (*background*)
  - Se indica postponiendo en carácter **&** a la ejecución
  - La *shell* sigue ejecutándose simultáneamente y esperando más órdenes
- Parado, suspendido (*stopped*, *suspended*)
  - Se puede parar un proceso con **^Z** (*control* + **Z**)
  - O enviándole la señal **SIGSTOP**





# Control de trabajos en la *shell*

Podemos utilizar la orden interna `jobs` para ver la lista de procesos bajo control de la *shell*:

```
cespedes@blas:~$ sleep 3600 &
[1] 5954
cespedes@blas:~$ cat
       ←(control+Z)
[2]+  Stopped                 cat
cespedes@blas:~$ jobs
[1]-  Running                 sleep 3600 &
[2]+  Stopped                 cat
cespedes@blas:~$ kill%2
[2]+  Terminated              cat
cespedes@blas:~$ jobs
[1]+  Running                 sleep 3600 &
cespedes@blas:~$ fg
sleep 3600
       ←(control+Z)
[1]+  Stopped                 sleep 3600
cespedes@blas:~$ bg
[1]+ sleep 3600 &
cespedes@blas:~$
```

# Sustituciones

Antes de ejecutar una orden, la *shell* puede preprocesar la orden realizando algún número de sustituciones sobre ella:

- Sustitución de history
- Sustitución de alias
- Sustitución de variable
- Sustitución de orden
- Sustitución de nombres de fichero





# 1. Sustitución de history

Todas las órdenes recibidas por la `bash` se guardan, y se pueden consultar con la orden `history`. Sustituciones:

| | |
|---|---|
| `!n` | Línea número `n` |
| `!-n` | n-ésima última línea |
| `!!` | Última línea (`!-1`) |
| `!prefijo` | Última línea que comience por `prefijo` |
| `^xx^yy` | Cambia `xx` por `yy` en la última línea |
| `!*` | Todos los argumentos de la última línea |
| `!$` | Último argumento de la última línea |
| `!^` | Primer argumento de la última línea |
| `!:n` | n-ésimo argumento de la última línea |

# 2. Sustitución de alias

Los *alias* permiten sustituciones de una palabra a una cadena cuando se usa como primera palabra de una orden.

Suelen estar en los ficheros de configuración de la bash: `.bashrc` o `.bash_login`

Uso:

- `alias [`*`nombre`*`[=`*`valor`*`]]`
- `unalias` *`nombre`*

Sin argumentos, `alias` muestra una lista de *alias*.
Con solo un nombre, indica si ese nombre es un *alias* o no.





# 3. Sustitución de variable

- Asignación de una variable a un valor:
  - `NOMBRE=valor`
- Uso de una variable: `$NOMBRE` en cualquier parte de una orden
- Lista de variables: `set`
- Borrar una variable: `unset VARIABLE`
- Se puede delimitar el nombre de una variable con llaves:
  - `echo una${variable}dentrodeunapalabra`

# 4. Sustitución de orden

- Ejecución de una orden antes de evaluar la línea, se inserta el resultado en la línea actual
- Se utilizan comillas invertidas **‘cmd‘**, o bien **$(cmd)**
- Ejemplos:

```
cespedes@blas:~$ AHORA=‘date‘
cespedes@blas:~$ echo $AHORA
Mon Feb 21 12:55:40 CET 2005
cespedes@blas:~$ DIR=$(pwd)
cespedes@blas:~$ echo $DIR
/home/cespedes
```





# 5. Sustitución de nombres de fichero

Caracteres comodín, expresiones regulares de la *shell*.

| | |
|---|---|
| * | Cualquier cadena de longitud 0 ó más |
| ? | Cualquier carácter |
| [ach] | Cualquier carácter de la lista |
| [a-f] | Cualquier carácter del rango |
| [!wxz] | Cualquier carácter que no esté en la lista |
| {uno,dos} | Cualquiera de las posibilidades indicadas |
| ~ | Mi directorio de inicio |
| ~user | Directorio de inicio del usuario “user” |

Ejemplo:

```
cespedes@blas:~$ cp *.[ch] practica1
cespedes@blas:~$ rm *.{gif,jpg}
cespedes@blas:~$ ls -l .??*
```

# Redireccion de entrada / salida

Se puede cambiar la entrada (`stdin`) / salida (`stdout`) estándar o la salida de error (`stderr`) de cualquier orden.

| | |
|---|---|
| `< fichero` | Toma `stdin` del fichero |
| `> fichero` | Envía `stdout` al fichero |
| `>> fichero` | Envía `stdout` al final del fichero (añade) |
| `2> fichero` | Envía el fd 2 (`stderr`) al fichero |
| `2>&1` | Envía `stderr` a donde vaya `stdout` |
| `&> fichero` | `stdout` y `stderr` al fichero |
| `<< cadena` | `stdin` de aquí en adelante hasta encontrar “`cadena`” |

Adicionalmente: “n<”, “n>”, “n>>”, “n>&m”





# Comillas

Se utilizan para agrupar caracteres y evitar sustituciones

- **`"Comillas dobles"`**

  Sustituye variables y órdenes, no sustituye nombres de fichero

```
cespedes@gizmo:~$ echo "$HOSTNAME, ‘pwd‘, *"
blas, /home/cespedes, *
```

- **`’Comillas simples’`**

  No se realiza ninguna sustitución

- **`‘Comillas invertidas‘`**

  Ya las hemos visto, son una “sustitución de orden”: se ejecuta la orden entre las comillas y se sustituye todo por el resultado de esa orden

# Carácter de escape \

Se puede utilizar el carácter \ para:

- Dividir una orden en varias líneas:

```
cespedes@gizmo:~$ ls \
> -la
```

- Obligar a la *shell* a que no interprete determinados caracteres (**'**, **"**, **$**, \), escribiéndolos precedidos precedidos del carácter de escape:

```
cespedes@gizmo:~$ echo "Cantidad total: \"25\$\""
Cantidad total: "25$"
```





# *shell scripts*

*shell script* = Programa en *shell*

- Separador de sentencias: cambio de línea o "`;`".
- Comentarios: "#" hasta el final de la línea.
- La primera línea indica qué *shell* interpretará el *script*:

```
#!/bin/sh
```

- Es necesario que el *script* tenga permisos de ejecución para el usuario que lo ejecute (`chmod +x nombre`).
- Ejemplo:

```
#!/bin/sh

echo "Hola, mundo"
```

# Variables

- Las predefinidas, especiales para la *shell*, están formadas por símbolos o letras mayúsculas (ejemplo: `$HOME`, `$SHELL`, `$USER`)
- Variables especiales:

| | |
|---|---|
| $# | Número de argumentos |
| $*n* | (*n=número*) Argumento *n*-ésimo del *script* |
| `$*` | Todos los argumentos. `"$*" = "$1 $2 $3..."` |
| $@ | Todos los argumentos. `"$@" = "$1" "$2" "$3"...` |
| $? | Resultado de la última orden (número entre 0 y 255) |
| $$ | PID de la *shell* |





# Pipes (filtros)

- Se especifican escribiendo dos órdenes en la misma línea, separadas por el carácter "**|**"
- Las dos órdenes se ejecutan simultáneamente, conectando la salida estándar de la primera a la entrada estándar de la segunda
- Ejecutar "**`orden1 | orden2`**" es similar a ejecutar "**`orden1 >/tmp/file`**" seguido de "**`orden2 </tmp/file`**", pero ejecutando las dos órdenes simultáneamente
- Ejemplo:

```
cespedes@blas:~$ ls -lrt /etc | tail -3
-rw-r--r--   1 root   root        106 2005-02-21 16:31 blkid.tab
-rw-r--r--   1 root   root         44 2005-02-22 19:53 resolv.conf
-rw-r--r--   1 root   cespedes    752 2005-02-22 19:59 mtab
```

# Ejecución de varias órdenes

- Ejecución condicional:
  - OR: "`orden1 || orden2`"

    ```
    grep nobody /etc/passwd > /dev/null || echo "No hay nobody"
    ```

  - AND: "`orden1 && orden2`"

    ```
    grep nobody /etc/passwd > /dev/null && echo "nobody existe"
    ```

- Agrupación de órdenes
  - Agrupación con paréntesis: se crea una nueva *shell*

    ```
    ( cd /tmp ; rm * ; )
    ```

  - Agrupación con llaves: lo gestiona la misma *shell* (se conservan variables creadas, directorio, etc)

    ```
    { ls /etc/ ; cat /etc/motd ; } | grep -i debian
    ```





# Sentencias de control

- Fin de la *shell*: **`exit`**
- Ejecución condicional: **`if`**
- Elección de opciones: **`case`**
- Bucles: **`for`**, **`while`**, **`until`**
- Ruptura de bucle: `continue`, **`break`**

El resultado de una orden se considera *verdadero* si su valor de retorno (**$?**) es 0, *falso* en cualquier otro caso

- Las órdenes **`true`** y **`false`** generan valores de retorno 0 y 1, respectivamente.

# La orden `test`

Las sentencias de ejecución condicional (`if`, `while`, etc) solo permiten ejecutar una orden para tomar la decisión, ellas mismas no son capaces de realizar comparaciones, etc.

"`test`" es una orden diseñada precisamente para esto: realizar comparaciones y comprobar tipos de ficheros

- "**`[`**" es el nombre de otro ejecutable que hace lo mismo que "**`test`**", pero requiere que la orden termine con el carácter "**`]`**"
- Uso: "**`test`** ***`expresión`***" o bien "**`[`** ***`expresión`*** **`]`**"





# Uso de `test`

Expresiones generales:

| | |
|---|---|
| `! expresión` | La expresión es falsa |
| `expr1 -a expr2` | AND |
| `expr1 -o expr2` | OR |

Expresiones con ficheros:

| | |
|---|---|
| `-e fichero` | El fichero existe |
| `-f fichero` | El fichero existe y es un fichero ordinario |
| `-d fichero` | El fichero es un directorio |
| `-r fichero` | El fichero tiene permiso de lectura |
| `-w fichero` | El fichero tiene permiso de escritura |
| `-s fichero` | El fichero tiene tamaño no nulo |

# Uso de `test` (2)

Comparación de cadenas:

| | |
|---|---|
| `-z cadena` | La cadena está vacía |
| `-n cadena` | La cadena no está vacía |
| `cadena1 = cadena2` | Las cadenas son iguales |

Comparación de enteros:

| | |
|---|---|
| `a -eq b` | Los enteros son iguales |
| `a -ne b` | Los enteros son distintos |
| `a -lt b` | a es menor que b |
| `a -gt b` | a es mayor que b |
| `a -le b` | a es menor o igual que b |
| `a -ge b` | a es mayor o igual que b |





# Sitaxis de `if`

```
if órdenes1; then órdenes2;
[ elif órdenes3; then órdenes4; ]...
[ else órdenes5; ] fi
```

Usos habituales:

```
if cmd1
then cmd2
fi
```

```
if cmd1; then
   cmd2
fi
```

```
if cmd1
then
   cmd2
else
   cmd3
fi
```

```
if cmd1; then
   cmd2
elif cmd3; then
   cmd4
else cmd5
fi
```

# Sitaxis de case

```
case palabra in
[ patrón [| patrón]...) órdenes ;;]...
esac
```

patrón: uso de los mismos caracteres comodín que en la expansión de nombres de ficheros.

Si hay varios patrones que coinciden, se ejecuta el conjunto de órdenes del primero que lo haga.

Si no coincide con ninguno, no se ejecuta nada.





# Sitaxis de for

```
for var [in palabras ... ;]
do órdenes; done
```

Ejecuta órdenes tantas veces como palabras haya; en cada ejecución, la variable var toma el valor de la palabra correspondiente.

Si no incluimos "in palabras ... ;", es equivalente a escribir "in "$@"".

No hay manera de especificar, como en otros lenguajes, bucles que vayan "de un número a otro". Para eso es útil la órden "seq".

# Sitaxis de `while`

```
while órdenes1 ; do órdenes2; done
```

1. Ejecuta “`órdenes1`”
2. Si el resultado es verdadero, ejecuta “`órdenes2`” y vuelve al punto 1.

Dentro de “`órdenes2`”, se puede usar “`continue`” para volver al comienzo del bucle, y “`break`” para salir de él.





# Sitaxis de `until`

```
until órdenes1 ; do órdenes2; done
```

“`until`” es idéntico a “`while`”, pero negando la condición de salida:

1. Ejecuta “`órdenes1`”
2. Si el resultado es falso, ejecuta “`órdenes2`” y vuelve al punto 1.

Al igual que en los bucles “`for`” y “`while`”, se pueden usar “`continue`” y “`break`” dentro de “`órdenes2`”.

# Uso de funciones

Sintaxis:

```
nombre () { órdenes ; }
```

- Ejecución de la función: "`nombre arg1 arg2 ...`"
- Dentro de la función, se puede acceder a su nombre y a los argumentos como se accede normalmente a los argumentos de un *script*: usando `$*`, `$@`, `$#`, `$1`, `$2`...
- La función puede devolver un valor de retorno (entre 0 y 255) usando "`return valor`".
- La parte que ha llamado a la función puede usar ese valor como una condición o consultando `$?`





# Operaciones aritméticas

En *Bourne Shell* no hay manera de realizar operaciones aritméticas internamente (suma, resta, etc). Para ello, existen dos órdenes externas: `expr` y `bc`.

- **`expr`**: operaciones sencillas con enteros (`+`, `-`, `*`, `/`, `%`, `()`)
  - **`resultado=‘expr $1 + $2‘`**
  - Cuidado con los `*`: la *shell* los sustituye
- **`bc`** es una calculadora de precisión arbitraria
  - Se puede usar interactivamente
  - **`resultado=‘echo "$1+$2"| bc‘`**
- Solo en **`bash`**: se puede usar `$(( .. ))` para operaciones con enteros