

Sistemas Operativos

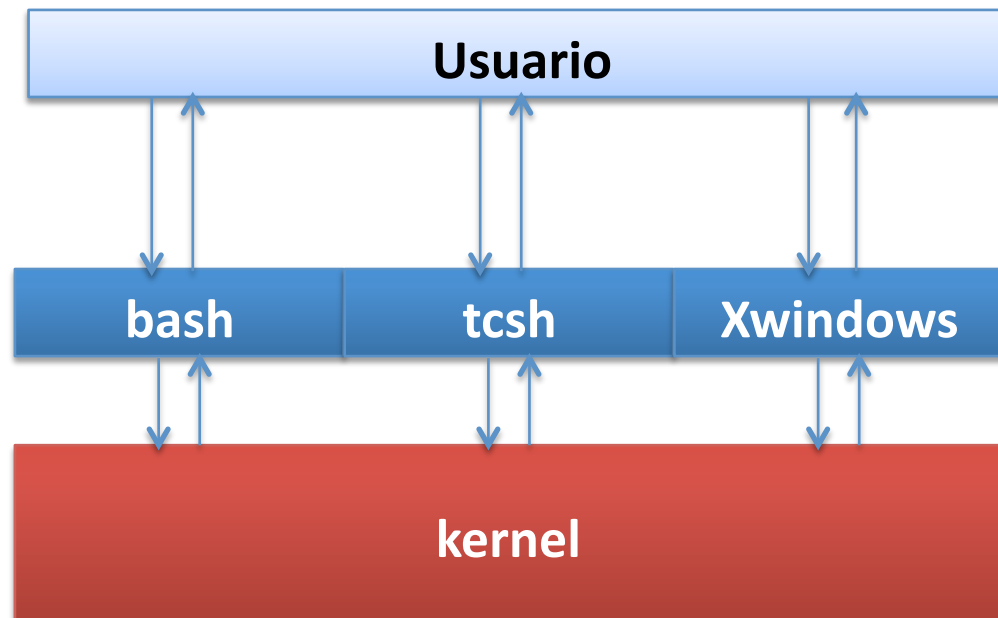
Curso
2014-2015

Introducción al shell *Bash*

Shell



- Programa que actúa como interfaz entre el usuario y el SO



Intérprete de comandos Bash

- Bash (Bourne-again shell) es una shell Unix escrita por Brian Fox para el proyecto GNU como una alternativa libre a la shell Bourne
- Bash es el intérprete predeterminado en muchos sistemas UNIX: GNU/Linux, Solaris y Mac OS X
- También se ha portado a Microsoft Windows (proyecto Cygwin)
- Otros intérpretes:
 - **sh**: Es el shell Bourne
 - **tcsh** o TENEX C shell: Derivado de csh, es un shell C
 - **ksh** o Korn shell: en ocasiones usado por usuarios con experiencia en UNIX

Ejecución Bash

- Comportamiento del shell:
 - Cuando un shell interactivo que no es un login shell arranca, Bash lee y ejecuta órdenes desde `~/ .bashrc`, si existiese.
 - Dispone de prefijos o “prompts” (`PS1` y `PS2`).
 - Los mandatos se leen en línea (`readline`) y se ejecutan tras su lectura.
 - La historia de mandatos se guarda en fichero (`HISTFILE`) y es posible realizar búsquedas en el historial (`CTRL+R`)
 - Se permite la expansión de alias.
 - Se pueden modificar los manejadores de señal (`Ctrl+C`).
 - Se puede controlar la acción a tomar cuando el interprete de comandos recibe un carácter EOF (`ignoreeof.`)

Ejecutando comandos

- Tipos de comandos
 - **Comandos internos** (*built-in commands*): Forman parte del repertorio del propio shell
 - **Comandos externos**: Programas externos al shell instalados en el sistema (ficheros binarios ejecutables o scripts)
- Las secuencias de comandos pueden incluirse en un fichero denominado **guión** o Script Bash
 - Cuando el programa es un guión, Bash creará un nuevo proceso usando `fork()`

Comandos propios

■ Bourne Shell built-ins ...

- `:`, `.`, `break`, `cd`, `continue`, `eval`, `exec`, `exit`, `export`, `getopts`, `hash`, `pwd`, `readonly`, `return`, `set`, `shift`, `test`, `[`, `times`, `trap`, `umask` and `unset`.

■ + Bash built-in commands:

- `alias`, `bind`, `builtin`, `command`, `declare`, `echo`, `enable`, `help`, `let`, `local`, `logout`, `printf`, `read`, `shopt`, `type`, `typeset`, `ulimit` and `unalias`.

Comandos básicos

Comando	Descripción
<code>ls</code>	Lista los ficheros del directorio actual
<code>pwd</code>	Muestra en qué directorio nos encontramos
<code>cd directorio</code>	Cambia de directorio
<code>man comando</code>	Muestra la página de manual para el comando dado
<code>apropos string</code>	Busca la cadena en la base de datos whatis
<code>file fichero</code>	Muestra el tipo de fichero dado
<code>cat fichero</code>	Muestra el contenido del fichero en pantalla
<code>exit/logout</code>	Abandona la sesión
<code>grep</code>	Busca en archivos líneas que contengan un patrón de búsqueda dado
<code>echo</code>	Muestra una línea de texto
<code>env</code>	Guarda información en el entorno
<code>export</code>	Cambia el valor de una variable de entorno

Variables y operadores

■ Variables

a=5

#asignación

echo \$a

#expansión

b=\$((\$a+3))

#aritmética entera

b=\$((\$a<<1))

#operadores de bits

■ Operadores aritméticos y de bits:

+ - / * % & | ^ << >>

Redirecciones

- Tres descriptores de ficheros predeterminados:

stdin (0) **stdout** (1) **stderr** (2)

- Redirección de la salida estándar:

`orden > fichero`

- Redirección de la salida de error:

`orden 2> fichero`

- Redirección de la entrada estándar:

`orden < fichero`

Ejemplos



■ Redirecciones:

```
ls -l > listado
```

```
ls -l /etc >> listado
```

```
ls /bin/basha 2> error
```

```
find / -name 'lib*' -print > librerias 2>&1
```

Cauces, tuberías o Pipes

- La salida estándar de una orden sirve como entrada estándar de otra:

```
ls -l | more
```

- Se combinan cauces y redirecciones:

```
ps aux | grep -v root > ps.out
```

Listas de órdenes

- Variable `$?`
 - *status* de la última orden ejecutada
- **`orden1 ; orden2`**
 - `orden2` se ejecuta cuando acaba `orden1`.
 - `$?` es el *status* de `orden2`
- **`orden1 && orden2`**
 - `orden2` sólo se ejecuta si *status* de `orden1` == 0 (éxito)
- **`orden1 || orden2`**
 - `orden2` sólo se ejecuta si *status* de `orden1` != 0 (fallo)

Ejecución en primer y segundo plano



■ *foreground y background*

- En modo interactivo los procesos se ejecutan en primer plano (*foreground*): la *shell* no muestra el *prompt* hasta que no finaliza la ejecución de la última orden introducida.
- Si queremos dejar el proceso en segundo plano (background) se añade **&**:

```
$ xeyes &
```

```
[2] 7584
```

← Job_ID ← PID

Comodines

- Permiten referirnos a un conjunto de ficheros con características comunes en sus nombres.
 - ***** corresponde con cualquier conjunto de caracteres.
 - **?** corresponde con cualquier carácter individual
 - **[conjunto]** corresponde con cualquier carácter dentro de **conjunto**.
- Ejemplo:
 - **?[a-c]*.h** cualquier fichero cuyo nombre comience por un carácter cualquiera seguido de las letras **a**, **b** ó **c** y que acabe en **.h**

Expansión de órdenes

- Podemos guardar en una variable la salida estándar de una orden o lista de órdenes.

- Ejemplo:

```
num=$( ls a* | wc -w )
```

- Forma equivalente:

```
num=`ls a* | wc -w`
```

Scripts Shell

- Un guión o script *shell* es un fichero que contiene una secuencia de órdenes *shell*.
 - Los comentarios comienzan por el carácter #
- Se crea un proceso *shell* que interpreta las líneas (*subshell*)
- Ejemplo:


```
#!/bin/bash
mkdir tmp
cd tmp
touch hola
cd ..
```
- Ejecución del script: \$./mi-script.sh
 - El fichero ha de tener permisos de ejecución


```
$ chmod +x mi-script.sh
```


Scripts Shell

- Un script es más versátil si su ejecución depende de parámetros.
 - Los parámetros posicionales se denotan por `$1`, `$2`, `$3` ... `$9`
 - Pueden usarse como si fueran variables normales pero además:
 - `$#` es el número total de parámetros.
 - `shift` desplaza a la izquierda los parámetros.

Sentencias condicionales

- Estructura **if-then-else**:

```
if condicion ; then  
    bloque then  
else  
    bloque else  
fi
```

- Nota importante: en la condición, 0 significa “verdadero”, otro valor significa “falso”

Sentencias condicionales II



■ Ejemplo:

```
if test -x /bin/bash ; then
    echo "/bin/bash es ejecutable"
else
    echo "/bin/bash no es ejecutable"
fi
```

■ También:

```
if [ -x /bin/bash ] ; then...
```

Condiciones



■ Cadenas:

`cadena1 = cadena2`

`cadena1 != cadena2`

`-n cadena`

`-z cadena`

Verdadero si son iguales

Verdadero si no son iguales

Verdadero cadena no nula

Verdadero si cadena nula

■ Ficheros

`-d fichero`

`-e fichero`

`-f fichero`

`-r fichero`

`-s fichero`

`-w fichero`

`-x fichero`

es un directorio

existe

es un fichero regular

tiene permisos de lectura

tiene longitud > 0

tiene permisos de escritura

tiene permisos de ejecución

Condiciones II

■ Aritméticas

<code>expresión1 -eq expresión2</code>	ambas expresiones son iguales
<code>expresión1 -ne expresión2</code>	ambas expresiones no son iguales
<code>expresión1 -gt expresión2</code>	<code>expresión1 > expresión2</code>
<code>expresión1 -ge expresión2</code>	<code>expresión1 ≥ expresión2</code>
<code>expresión1 -lt expresión2</code>	<code>expresión1 < expresión2</code>
<code>expresión1 -le expresión2</code>	<code>expresión1 ≤ expresión2</code>
<code>! expresión</code>	expresión es falsa

Bucles for



■ Bucles

■ Bucle for (I):

```
for variable in valores
do
    cuerpo del for
done
```

■ Ejemplo

```
for i in `seq 0 1 9`
do
    echo $i
done
```

■ Bucle for (II):

```
for (( i=0 ; $i<10; i++ ))
do
    echo $i
done
```

Bucles While

- Bucle **while**:

```
while condición ; do
    cuerpo del while
done
```

- Ejemplo:

```
while [ $# -gt 0 ] ; do
    echo $1 ; shift
done
```

Expresiones regulares

- Son un mecanismo muy potente para la búsqueda de patrones en cadenas de caracteres.
- Bloques básicos:
 - **carácter**: coincide con un carácter concreto. P.e. a
 - **.** : (punto) coincide con cualquier carácter.
 - **^** : principio de línea.
 - **\$** : final de línea.
 - **[lista]** : cualquier carácter dentro de lista
 - **[^lista]** : cualquier carácter fuera de lista

Expresiones regulares II



- Operadores de repetición. El elemento precedente concuerda:
 - $?$: como mucho una vez (puede ser ninguna).
 - $*$: cero o más veces.
 - $\{n\}$: exactamente n veces.
 - $\{n, \}$: n o más veces.
 - $\{, m\}$: como mucho m veces.
 - $\{n, m\}$: al menos n veces y no más de m .

Ejemplos

- a** : cualquier cadena que contenga al menos una a.
- ab*** : cualquier cadena que contenga al menos una a.
- ab** : cualquier cadena que contenga la subcadena “ab”
- a . b** : cualquier cadena que tenga una a y una b separadas por un carácter cualquiera.
- ^[abc]** : cualquier línea que comience por **a**, **b** ó **c**.
- [^abc]** : cualquier cadena que contenga cualquier carácter distinto de **a**, **b** ó **c**.