

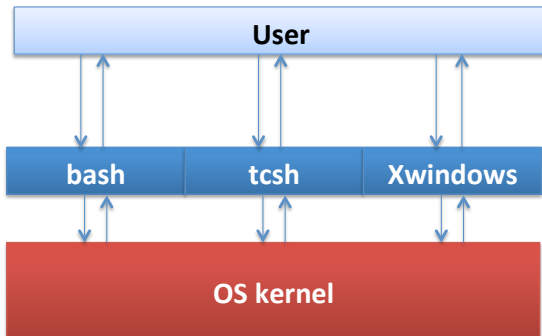
Sistemas Operativos

Universidad Complutense de Madrid
2016-2017

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: la mayoría de sistemas 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)

Scripts de BASH

- Es posible incluir una secuencia de comandos del shell en un fichero de texto, que se denomina **guión** o *script*
- Los comandos del *script* pueden ejecutarse como un todo (secuencialmente)
 - Al lanzar un script de BASH, el SO crea un nuevo proceso bash

Comandos internos

Bourne Shell built-ins:

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

BASH built-in commands:

`alias, bind, builtin, command, declare, echo, enable, help, let, local, logout, printf, read, shopt, type, typeset, ulimit, 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 directory</code>	Cambia de directorio
<code>man command</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 filename</code>	Muestra el tipo de fichero dado
<code>cat textfile</code>	Muestra el contenido del archivo en pantalla
<code>exit</code> or <code>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>	Muestra el conjunto de variables de entorno
<code>export</code>	Cambia el valor de una variable de entorno

Variables y operadores

Variables

<code>a=5</code>	<code>#asignación</code>
<code>echo \$a</code>	<code>#expansión</code>
<code>b=\$((\$a+3))</code>	<code>#aritmética entera</code>
<code>b=\$((\$a<<1))</code>	<code>#operadores a nivel de bit</code>

Operadores aritméticos y de bits:

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

Redirecciones

- Tres descriptores de ficheros predeterminados:
 - `stdin(0)`, `stdout(1)` and `stderr(2)`
- Comportamiento por defecto cuando se lanza un programa (binario o script) desde una consola de texto o terminal:
 - `stderr` y `stdout` se asocian al terminal
 - `stdin` se asocia al teclado
- Redirección de la salida estándar:
 - `comando > fichero`
- Redirección de la salida de error:
 - `comando 2> fichero`
- Redirección de la entrada estándar:
 - `comando < fichero`

Redirecciones

Ejemplos

```
ls -l > listing
ls -l /etc >> listing
ls /bin/basha 2> error
find / -name 'lib*' -print > libraries 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 `$?`
 - Código de terminación (*status*) de la última orden ejecutada
 - Convenio: `0` \rightarrow éxito; `!=0` \rightarrow fallo
- `cmd1 ; cmd2`
 - `cmd2` se ejecuta cuando acaba `cmd1`.
 - `$?` es el *status* de `cmd2`
- `cmd1 && cmd2`
 - `cmd2` sólo se ejecuta si *status* de `cmd1` = `0` (éxito)
- `cmd1 || cmd2`
 - `cmd2` sólo se ejecuta si *status* de `cmd1` \neq `0` (fallo)

Ejecución en primer y segundo plano

- En modo interactivo los procesos se ejecutan en primer plano (*foreground*)
 - Modo de ejecución por defecto
 - La shell no muestra el prompt hasta que no finaliza la ejecución de la última orden introducida
- Las aplicaciones con interfaz gráfica (GUI) se lanzan típicamente en segundo plano (*background*)
 - Si queremos dejar el proceso en segundo plano se añade '&' al final:

```
$ 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 de conjunto: `[abc]`
- Ejemplo de patrón con comodines: `?[a-c]*.h`
 - cualquier fichero cuyo nombre acabe en `“.h”` y comience por un carácter cualquiera seguido de las letras `‘a’`, `‘b’` or `‘c’`
- Los patrones con caracteres comodín pueden usarse tanto en scripts de BASH como en sesiones interactivas con el shell
 - Ejemplo: `rm ?[a-c]*.h`
 - ¿Qué hace ese comando?

Expansión de órdenes

- Las construcciones de expansión de órdenes permiten convertir la salida de un comando en una cadena de caracteres
 - Al hacer esto, los saltos de línea de la salida se eliminan
 - La cadena resultante se puede usar como argumento de otra orden o almacenarse en una variable de Bash

- Ejemplo:

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

- Sintaxis alternativa:

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

Guiones o scripts de BASH

- Un *script* es un fichero de texto que contiene una secuencia de comandos del shell
 - Los comentarios comienzan con #
- Al lanzar un script, se crea un proceso bash que interpreta las líneas (*subshell*)

my-script.sh

```
#!/bin/bash
mkdir tmp
cd tmp
echo hi > file
cd ..
```

- ¿Cómo se ejecuta el script?:
 - \$./my-script.sh
- El fichero debe tener permisos de ejecución
 - \$ chmod +x my-script.sh

Guiones o scripts de BASH

Gestión de argumentos de línea de comandos

- Los argumentos se referencian mediante las siguientes variables especiales : \$1, \$2, \$3 ... \$9
- \$0 almacena la ruta del script, especificada en la línea de comandos
- \$# indica el número de argumentos
- El comando interno `shift builtin` elimina el primer parámetro y desplaza el resto hacia la izquierda
 - \$2 será \$1, \$3 será \$2, y así sucesivamente.

Sentencias condicionales (I)

if-then-else

```
if condition ; then
    THEN_BLOCK
else
    ELSE_BLOCK
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 has execute permissions"
else
    echo "/bin/bash does not have execute permissions"
fi
```

Sintaxis alternativa...

```
if [ -x /bin/bash ]; then
    echo "/bin/bash has execute permissions"
else
    echo "/bin/bash does not have execute permissions"
fi
```

Condiciones (I)

Cadenas de caracteres

```
str1 = str2    # Verdadero si son iguales
str1 != str2   # Verdadero si no son iguales
-n str        # Verdadero cadena no nula
-z str        # Verdadero si cadena nula
```

Ficheros

```
-d file    # es un directorio
-e file    # existe
-f file    # es un fichero regular
-r file    # tiene permisos de lectura
-s file    # es un fichero no vacío
-w file    # tiene permisos de escritura
-x file    # tiene permisos de ejecución
```

Condiciones (II)

Aritméticas

```
exp1 -eq exp2      # ambas expresiones son iguales
exp1 -ne exp2      # ambas expresiones son diferentes
exp1 -gt exp2       # exp1 > exp2
exp1 -ge exp2       # exp1 >= exp2
exp1 -lt exp2       # exp1 < exp2
exp1 -le exp2       # exp1 <= exp2
! exp               # exp es falsa
```

Bucles for

Sintaxis #1

```
for variable in values
do
    LOOP_BODY
done
```

Sintaxis #2

```
for (( i=0 ; $i<N; i++ ))
do
    LOOP_BODY
done
```

Ejemplo #1

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

Ejemplo #2

```
for (( i=0 ; $i<10; i++ ))
do
    echo $i
    sleep 1
done
```

Bucles while

Sintaxis

```
while condition; do
    LOOP_BODY
done
```

Ejemplo

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



Expresiones regulares (I)

- Son un mecanismo muy potente para la búsqueda de patrones en cadenas de caracteres
- Comandos externos como `grep`, `sed` o `awk` están equipados con soporte de expresiones regulares

Bloques básicos

- `char`: coincide con un carácter concreto (p.ej., 'a')
- `.` : coincide con cualquier carácter
- `^` : principio de línea
- `$`: final de línea
- `[set]` : cualquier carácter en set
- `[^set]`: cualquier carácter que no esté en set

Expresiones regulares (II)

Operadores de repetición (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 .

Expresiones regulares (III)

Ejemplos

- `a`: cualquier cadena que contenga 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`.