

Laboratorio de Arquitectura de Ordenadores

Shell Script

Jorge Blanco Rodríguez
Peter T. Breuer
Andrés Marín López
Pedro J. Muñoz Merino
Ralf E.D. Seepold



Departamento de
INGENIERIA
TELEMÁTICA
www.it.uc3m.es

Índice

- Introducción
- Tipos de Shell
- Estructura de un programa de Shell
- Ejecución de un script
- Aritmética del bash
- Estructuras de control
- Entrada/Salida

Introducción (Definiciones)

- Shell: Interfaz usuario / sistema
- Script: Fichero con comandos de Shell

Introducción (Caract. Espec.)

- No se interpretan literalmente dentro de un *string*
 - | & ; () < > \$!
- Para que sean interpretados han de escaparse mediante \
- Con 'string' todo el *string* se interpreta literalmente.
- Con "string" se preserva el significado de:
 - \$ ' \\$ \' \"
 - \retorno_de_carro \$@ \$*

Introducción (Variables)

- Definición: *nombre_de_var=valor*
- Valor por defecto: ""
- Valor de una var:
 - *\$nombre_de_var*
 - *\${nombre_de_var}*
- Longitud:
 - *\${#nombre_var}*

Introducción (Funciones)

- Definición:

```
function
nombre_de_funcion
{
    comandos_de_shell
}

nombre_de_funcion
()
{
    comandos_de_shell
}
```

Introducción (Funciones 2)

- Eliminar una función de la memoria:
 - `unset -f nombre_de_funcion`
- Variables locales:
 - `local nombre_de_var_local`
- Devolver valores:
 - `return [codigo]`
 - Detiene la ejecución de la función
 - Si no se especifica *codigo*, se devuelve como estado de salida el del último comando ejecutado

Introducción (Parám. Posic.)

- Almacenan los parámetros de entrada de un script
 - Nombre variables: `0 1 2`
 - Valores: `$0 $1 $2`
- `$0` → Nombre del script
- `$*` → Todos los argumentos separados por el separador por defecto (IFS)
- `$#` → Número de parámetros
- `$@` → N strings (se usa con *for*)

Introducción (Operadores)

- $\${nombre_var} :- string$ → Si la var no está definida devuelve string
- $\${nombre_var} := string$ → Si la var no está definida asigna string a la var
- $\${nombre_var} :? mensaje$ → Si la var no está definida muestra el mensaje de error y aborta la ejecución del script

Introducción (Operadores 2)

- $\${nombre_var} :+ string$ → Si la var existe y no vale *null* devuelve string, si no, devuelve *null*
- $\${nombre_var} :offset$ → Devuelve el string desde la posición *offset* hasta el final
- $\${nombre_var} :offset:longitud$ → Devuelve el string desde la posición *offset* hasta *offset+longitud*

Introducción (Patrones)

- $\${nombre_var \#patrón}$ → Devuelve el valor de la var quitando un **prefijo**, que es el acierto más **corto** del patrón
- $\${nombre_var ##patrón}$ → Devuelve el valor de la var quitando un **prefijo**, que es el acierto más **largo** del patrón

Introducción (Patrones 2)

- $\${nombre_var \%patrón}$ → Devuelve el valor de la var quitando un **sufijo**, que es el acierto más **corto** del patrón
- $\${nombre_var %%patrón}$ → Devuelve el valor de la var quitando un **sufijo**, que es el acierto más **largo** del patrón

Introducción (Patrones 3)

- `${nombre_var /patrón/string}` →
Devuelve el valor de la var sustituyendo el **acierto más largo** del patrón por string
- `${nombre_var //patrón/string}` →
Devuelve el valor de la var sustituyendo **todos los aciertos** del patrón por string

Introducción (Comandos)

- Ejecución de comandos:
 - `nombre_var=$(comando)`
 - `nombre_var=`comando``
- El valor de `nombre_var` será la salida del `comando`
- Ej: `fecha=$(date)`
lun ene 19 17:05:15 CET 2004

Tipos de Shell

- Existen varios intérpretes, cada uno con sintaxis diferente, pero con misma filosofía
 - sh → En todas las distribuciones UNIX
 - csh → Sintaxis un poco más sencilla que sh
 - Bourne Shell → También basado en sh
 - **bash** → Multitud de operadores nuevos (sh ++)
 - tcsh → Recuerda últimos comandos
- echo \$SHELL → Tipo de shell utilizado

Estructura de un prog. Shell

- Primera línea: Especifica tipo de Shell
 - #!/bin/bash
- Funciones de usuario
 - function check_param() {...}
- Conveniente devolver un código de error
 - exit 0

Ejecución de un script

- **source** *<nombre_script>*
- **sh** *<nombre_script>*
- **bash** *<nombre_script>*
 - Crea un nuevo proceso bash para correr el script
nombre_script
 - Antes dar permiso de ejecución con `chmod`

Ejecución de comandos

- **El operador ;**
 - Para ejecutar varios comandos en la misma línea, podemos usar el operador `;` como un separador. Por ejemplo: "comando1 ; comando2" ejecutará primero comando1, esperará hasta que termine, ejecutará comando2, esperará hasta que termine y entonces nos devolverá el control
- **El operador &**
 - Para ejecutar un comando en background, podemos usar el operador `&` después del comando. Esto nos devolverá el control inmediatamente en vez de esperar a que el programa termine. Por ejemplo: " comando1 & comando2 &" ejecutará comando1 e inmediatamente después comando2 e inmediatamente nos devolverá el control

Aritmética del bash

- Operaciones:

+ - * / %

== != <= >= < >

!

& |

Aritmética del bash (2)

- Valor aritmético de una expresión:

`$((expresion))`

```
var1=5
```

```
var2=7
```

```
var3=$var1+$var2      ➔ "5+7"
```

```
var4=$( (var1+var2) ) ➔ "12"
```

Aritmética del bash (3)

- Otra forma: **expr**

```
i=1  
i=$(expr $i + 1) → "2"  
i=$((i+1)) → "3"
```

- **expr** evalúa una expresión aritmética para dar su resultado

Control de flujo

- **if / else**
- **for**
- **case**
- **select**
- **while**
- **until**

if / else

```
if condicion ; then
    codigo
[elif condicion ; then
    codigo]
[else
    codigo]
fi
```

if / else (2)

- ***condicion*** no es un valor booleano, sino el **estado de salida** de un comando
 - ÉXITO → 0 (cierto)
 - ERROR → >0 (falso)
- **let \$(*expresion*)** da como estado de salida el valor de la evaluación aritmética de la expresión
- **[]** da como estado de salida el resultado de aplicar un operador de comparación a un string
 - Equivalente a **test**

if / else (3)

- Anidamiento de condiciones:
 - and → `$$`
 - or → `||`
 - not → `!`
- Para comprobar otro tipo de condiciones de forma indirecta (ej: existe archivo??):
 - Operadores de comparación, aritméticos, etc.
 - `[]`

if / else (4)

- Operadores de comparación de strings
 - `string==string` lexicográficamente igual
 - `string!=string` lexicográficamente distinto
 - `string<string` lexicográficamente menor
 - `string>string` lexicográficamente mayor

if / else (5)

- Más operadores:

| | | |
|----|---------|-------------------------------|
| -d | archivo | existe y es un directorio |
| -e | archivo | existe |
| -f | archivo | existe y es un fichero normal |
| -r | archivo | permisos de lectura |
| -s | archivo | existe y no está vacío |
| -w | archivo | permisos de escritura |
| -x | archivo | permisos de ejecución |

if / else (6)

- Más operadores:

| | | |
|----|----------|-------------------------------|
| -b | archivo | es un dispositivo de bloque |
| -c | archivo | es un dispositivo de carácter |
| -n | variable | no es null |
| -O | archivo | propietario de archivo |
| -G | archivo | grupo con acceso |

- Algunos más:

man bash

if / else (7)

- Más operadores:

| | |
|-------------------------------------|---------------|
| <code>expresion -a expresion</code> | → AND |
| <code>expresion -o expresion</code> | → OR |
| <code>archivo1 -nt archivo2</code> | → más nuevo |
| <code>archivo1 -ot archivo2</code> | → más antiguo |

if / else (8)

- Operaciones sobre strings interpretadas aritméticamente:

| | |
|------------------|---------------------|
| <code>-lt</code> | → menor que |
| <code>-le</code> | → menor o igual que |
| <code>-eq</code> | → igual que |
| <code>-ge</code> | → mayor o igual que |
| <code>-gt</code> | → mayor que |
| <code>-ne</code> | → distinto que |

if / else (9)

```
echo "Diga si o no: "  
read var  
if [ $var == "si" ]  
then  
    echo "Ha dicho -si-"  
else  
    echo "No ha dicho -si-, ha dicho -$var-"  
fi
```

if / else (10)

```
if [ -d correo ] ; then  
    echo "El archivo -correo- existe y es un directorio"  
fi  
  
if [ -e correo ] && [ ! -d correo ] ; then  
    echo "El archivo -correo- existe y NO es un directorio"  
fi
```

for

```
for variable_del_bucle [in lista]
do
    codigo
done
```

- Repite *codigo* un número fijo de veces
- *lista*: string con varias palabras separadas por IFS, que por defecto es ' ' (si se omite → \$@)

for (2)

```
notas_de_lao="9:9:10:9,5:10:9:8:10"
IFS=:

for nota in $notas_de_lao
do
    echo "$nota"
done
```

case

```
case expression in
  patron1 )
    codigo ;;
  patron2 )
    codigo ;;
  patron3 )
    codigo ;;
  ...
esac
```

case (2)

- Utilizado para comprobar valores simples, como enteros o caracteres

```
case "$#" in
  0 )
    echo "Nigún argumento" ;;
  * )
    echo "Algún argumento" ;;
esac
```

select

```
select nombre_de_variable [in lista] ; do  
    codigo  
done
```

- Genera un menú para cada objeto de la lista
- Pregunta al usuario por un n^o (prompt=**PS3**)
- Guarda la opción seleccionada en **REPLY (n^o)**
- Repite el proceso de forma continua → Hay que salir con **break**

select (2)

```
PS3="Que quiere comer? "  
IFS=:  
comidas="paella:tortilla:salir"  
select eleccion in $comidas ; do  
    case $eleccion in  
        paella ) echo "Va a comer paella" ;;  
        tortilla ) echo "Va a comer el plato $REPLY" ;;  
        salir ) echo "Saliendo..."  
                break ;;  
        * ) echo "Comida no disponible. Elija otra" ;;  
    esac  
done
```

while

```
while estado_de_salida
do
    codigo
done
```

- Ejecuta el bucle mientras el estado_de_salida sea 0

while (2)

```
i=1
while true
do
    echo $i
    i=$(expr $i + 1)
    if [ $i == 10 ] ; then
        exit 0
    fi
done
```

until

```
until comando
```

```
do
```

```
    codigo
```

```
done
```

- Corre *codigo* hasta que *comando* se ejecute con éxito (el estado de salida sea 0)

until

```
until command
```

```
do
```

```
    code
```

```
done
```

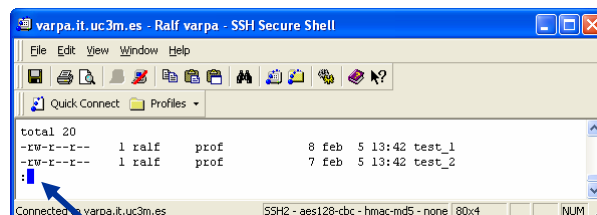
- The loop executes *code* as long as the *condition* is false

Redirección de entrada/salida

- <
 - Redirección del flujo de entrada
 - Ejemplo: `cat < mifichero`
- >
 - Redirección del flujo de salida
 - Ejemplo: `cat mifichero > miotrofichero`
- >>
 - Añadir algo al final del fichero (cf. operador >)

Pipe

- |
 - Operador pipe
 - Ejemplo `ls -l | less`



```
varpa.it.uc3m.es - Ralf varpa - SSH Secure Shell
File Edit View Window Help
Quick Connect Profiles
total 20
-rw-r--r-- 1 ralf prof 8 feb 5 13:42 test_1
-rw-r--r-- 1 ralf prof 7 feb 5 13:42 test_2
:
```

Muestra sólo tres líneas y espera para evitar el desbordamiento de la ventana

Control de permisos

- 3 roles: dueño (owner) – grupo (group) – mundo(world)
- 3 tipos de acceso: read, write execute => rwx
- 3 veces 3 tipos de acceso : rwx rwx rwx
- chmod – cambia los permisos
- chmod 751 directory ???
- chmod g=u-w

Programación en shell: teclas

- **C-c** Aborta el programa actual. No deberíamos tener que usarlos, pues cada programa debe ser capaz de terminar por sí mismo. Pero si el programa se cae o no puede terminar por sí mismo, necesitaremos abortarlo.
- **C-d** Envía un carácter EOF (end of file). Este carácter está presente al final de cada fichero. Si un programa espera un fichero como entrada, a veces se puede teclear manualmente, y se termina con este carácter.
- **Flecha arriba/abajo** Se mueve por la historia de comandos.
- **Flecha izquierda/derecha** Mueve el cursor en la línea actual.
- **Tab** Completa automáticamente la línea actual hasta que surja una ambigüedad. Funciona tanto con comandos como con nombres de ficheros. Si se vuelve a pulsar Tab, lista todas las posibles terminaciones.

...y otros comandos útiles

• FLAGS

```
ls -l
```

– Ejemplo:

```
varpa:~/Test> ls
test_1 test_2
varpa:~/Test> ls -l
total 8
-rw-r--r--  1 ralf  prof      8 feb  5 13:42 test_1
-rw-r--r--  1 ralf  prof      7 feb  5 13:42 test_2
varpa:~/Test>
```

necesitados con frecuencia:

- **awk**: para procesamiento de texto, más completo que **sed**
- **chsh**: para cambiar el tipo de shell por defecto
- **cat**: concatena ficheros y los imprime por la salida estándar
- **cd**: cambia el directorio actual
- **cut -d separator -f column**: corta columnas
- **date**: informa sobre el día y la hora
- **dd**: convierte y copia un fichero
- **echo**: ECHO, Echo, echo...
- **exit**: Provoca una terminación de la shell con estado n
- **grep**: cosas como: "toma todas las líneas que empiecen por ..."
- **head -n X**: muestra las primeras X líneas de la entrada estándar
- **mail**: pues eso
- **paste**: pega
- **read**: lee de la entrada estándar y se guarda en una variable
- **sed**: para procesado de texto, sustitución, borrado...
- **sort**: ordena alfabética o numéricamente [-n]
- **tail -n X**: muestra las últimas X líneas de la entrada estándar
- **touch**: cambia la hora de modificación
- **who**: informa acerca de quién está en el sistema

Bibliografía

- `info bash`
 - Información de GNU Linux sobre bash
- `man bash`
 - Páginas de manual de bash