

# Scripts de shell para Oracle

Álvaro González Sotillo

6 de mayo de 2022

## Índice

1. Introducción	1
2. Scripts de <i>shell</i>	2
3. Entorno inicial	9
4. Prerrequisitos	10
5. Ejecución de SQL desde la <i>shell</i>	12
6. Arranque y parada	13
7. Operaciones periódicas	16
8. Avanzado	17
9. Más avanzado: comandos desde PL/SQL	17
10. Referencias	18

## 1. Introducción

- Muchas tareas del mantenimiento de una base de datos **Oracle** se llevan a cabo desde la línea de comandos
- Por tanto, pueden automatizarse
  - Arranque y parada
  - Extracción de datos
  - Copia de seguridad de datos
  - Restauración de datos
- Para ello, se utilizan las facilidades de ejecución del sistema operativo aprendidas en otros módulos

---

## 2. Scripts de *shell*

### 2.1. *shebang*

- Los scripts empiezan con una línea indicando el intérprete que los ejecutará, con un comentario `# !`

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

```
#!/usr/bin/php
```

```
#!/usr/bin/python
```

### 2.2. Variables

```
# Variable local a esta shell
variable=valor

# Variable exportada a los hijos de esta shell
export variable_exportada=valor

# Variable definida solo para un comando
variable_para_un_comando=valor comando
```

### 2.3. Entrada/salida

- Los programas comienzan su ejecución con una salida y una entrada
- Son flujos de bytes
- Inicialmente:
  - La entrada es el teclado
  - La salida es la consola

#### 2.3.1. Redirigir entrada/salida a fichero

```
# La entrada sale de un fichero en vez del teclado
sort < fichero

# La entrada sale de un fichero, y la salida va a otro fichero
sort < fichero > fichero_ordenado

# La entrada sale de un fichero, y la salida se agrega al final de un fichero
sort < otro_fichero >> fichero_ordenado

# La salida del primer comando es la entrada del segundo
sort < fichero | less
```

---

### 2.3.2. *HEREDOCs*

```
# La entrada se especifica en el propio script
sort <<FINDEFICHERO
Maria
Pepe
Juan
Susana
Manolo
FINDEFICHERO
```

### 2.3.3. Salida como parámetro

- Se puede capturar la salida de un comando en una cadena
- Esa cadena se utiliza luego como otra cadena cualquiera en el script

```
# Defino una variable con los ficheros del directorio
variable=$(ls)
```

## 2.4. Parámetros del *script*

- \$0: El nombre del *script*
- \$1: Primer parámetro
- \$2: Segundo parámetro
- \$\*: Todos los parámetros a partir del primero
- \$#: Número de parámetros del *script*

## 2.5. Funciones

- Son conjuntos agrupados de órdenes con un nombre
- Tienen sus propios argumentos \$\*, \$1, \$2...

```
importante() {
    echo -----
    echo Aviso: $*
    echo -----
}

importante "Así se define una función en bash"
```

---

### 2.5.1. return en funciones

- Las funciones también tienen código de retorno
- Pueden simplificar `if` o bucles `while`

```
condicion() {  
  # AQUÍ SE PODRÍA DECIDIR EL RETORNO CON OTROS COMANDOS  
  # O CON IF'S ENCADENADOS, PERO COMO EJEMPLO DEVOLVEMOS TRUE  
  return 0  
}  
  
while condicion  
do  
  echo Esto es un bucle infinito  
done
```

## 2.6. Ejercicio

- Crea un *script* que reciba hasta 9 parámetros
- Su salida serán los parámetros, uno por línea, en orden alfabético inverso
- La salida no tendrá líneas en blanco
- Mejoras:
  - Más de 9 parámetros: `shift, $*`
  - Parámetros con espacios: `$@`

## 2.7. Código de error (*exit code*)

- Al terminar, un programa devuelve un valor numérico
- Por convenio
  - 0: Todo ha funcionado correctamente
  - Distinto de 0: Ha sucedido algún tipo de error
- Se puede consultar con `$?` **inmediatamente** después de ejecutar el comando

```
grep cadena *  
exit_code_del_grep=$?  
echo grep ha devuelto: $exit_code_del_grep
```

## 2.8. Condicionales

- `if` utiliza los códigos de error de los programas
  - 0 se considera `true`
  - Cualquier otro valor se considera `false`

```
if grep cadena *  
then  
  echo grep ha encontrado algo sin errores  
else  
  echo grep no lo ha encontrado, o ha habido errores  
fi
```

---

### 2.8.1. Comando [

- [ es un comando externo que ayuda a hacer condiciones con if
  - Comparación de cadenas
  - Comparación de números
  - Existencia de ficheros

TEST(1)	User Commands	TEST(1)
NAME	<code>test</code> - check file types and compare values	
SYNOPSIS	<code>test</code> EXPRESSION <code>test</code>  [ EXPRESSION ] [ ] [ OPTION	
DESCRIPTION	Exit with the status determined by EXPRESSION.  --help display this help and <code>exit</code>  --version output version information and <code>exit</code>  An omitted EXPRESSION defaults to <code>false</code> . Otherwise, EXPRESSION is <code>true</code> or <code>false</code> and sets <code>exit</code> status. It is one of:  ( EXPRESSION ) EXPRESSION is <code>true</code>  ! EXPRESSION EXPRESSION is <code>false</code>  EXPRESSION1 -a EXPRESSION2 both EXPRESSION1 and EXPRESSION2 are <code>true</code>  EXPRESSION1 -o EXPRESSION2 either EXPRESSION1 or EXPRESSION2 is <code>true</code>  -n STRING the length of STRING is nonzero  STRING equivalent to -n STRING  -z STRING the length of STRING is zero  STRING1 = STRING2 the strings are equal  STRING1 != STRING2 the strings are not equal  INTEGER1 -eq INTEGER2 INTEGER1 is equal to INTEGER2  INTEGER1 -ge INTEGER2 INTEGER1 is greater than or equal to INTEGER2  INTEGER1 -gt INTEGER2 INTEGER1 is greater than INTEGER2  INTEGER1 -le INTEGER2 INTEGER1 is less than or equal to INTEGER2	

---

```
INTEGER1 -lt INTEGER2
    INTEGER1 is less than INTEGER2

INTEGER1 -ne INTEGER2
    INTEGER1 is not equal to INTEGER2

FILE1 -ef FILE2
    FILE1 and FILE2 have the same device and inode numbers

FILE1 -nt FILE2
    FILE1 is newer (modification date) than FILE2

FILE1 -ot FILE2
    FILE1 is older than FILE2

-b FILE
    FILE exists and is block special

-c FILE
    FILE exists and is character special

-d FILE
    FILE exists and is a directory

-e FILE
    FILE exists

-f FILE
    FILE exists and is a regular file

-g FILE
    FILE exists and is set-group-ID

-G FILE
    FILE exists and is owned by the effective group ID

-h FILE
    FILE exists and is a symbolic link (same as -L)

-k FILE
    FILE exists and has its sticky bit set

-L FILE
    FILE exists and is a symbolic link (same as -h)

-O FILE
    FILE exists and is owned by the effective user ID

-p FILE
    FILE exists and is a named pipe

-r FILE
    FILE exists and read permission is granted

-s FILE
    FILE exists and has a size greater than zero

-S FILE
    FILE exists and is a socket

-t FD  file descriptor FD is opened on a terminal

-u FILE
    FILE exists and its set-user-ID bit is set

-w FILE
    FILE exists and write permission is granted

-x FILE
    FILE exists and execute (or search) permission is
    granted
```

---

## 2.9. Ejercicio

- Haz un *script* que reciba un parámetro
- Si no recibe ningún parámetro, indica que lo necesita y acaba
- Si el parámetro es un fichero, informará de su longitud en bytes
- Si el parámetro es un directorio, informará del número de ficheros/directorios que contiene
- Si no es ni un fichero ni un directorio, informará de ello

## 2.10. Bucles

- Con `for` se pueden hacer bucles sobre una lista de parámetros
- Para bucles numéricos se puede usar el comando `seq`

```
for nombre in Maria Juan Pepe Susana Manolo
do
    echo Realizando una vuelta de bucle sobre $nombre
done

# CUIDADO CON LOS NOMBRES DE FICHERO CON ESPACIOS
for fichero in $(ls)
do
    echo El siguiente fichero es $fichero
done
```

### 2.10.1. Bucles y espacios

- Las cadenas se separan por la variable `IFS`, que suele ser `\t \n` (tabulador, nueva línea, espacio)
- Si no se quieren separar por espacios, hay que quitarlos de `IFS`

```
#!/bin/bash
ELEMENTOS="
elemento1
elemento 2
elemento 3
elemento4
"

echo "valor actual de IFS (en hexadecimal):" $(printf "%s" "$IFS" | xxd -p)
for ELE in $ELEMENTOS
do
    echo BUCLE: $ELE
done

IFS=$'\n'
echo "valor actual de IFS (en hexadecimal):" $(printf "%s" "$IFS" | xxd -p)
for ELE in $ELEMENTOS
do
    echo BUCLE: $ELE
done
```

---

### 2.10.2. Bucles sobre Ficheros

- Los nombres de fichero pueden contener caracteres de todo tipo (emojis, , espacios...)

```
touch "antes"$'\n'"despué s  "  
echo LS NORMAL  
ls  
echo LS LITERAL  
ls --literal --show-control-chars  
echo LS SIN CONTROLCHARS  
ls --literal --hide-control-chars
```

- La única forma segura es separar los ficheros por `$'\0'`, que no puede aparecer en la mayoría de sistemas de ficheros

```
find . -maxdepth 1 -print0 | while read -d $'\0' FILE  
do  
    echo FICHERO: $FILE  
done
```

### 2.11. Operaciones aritméticas

- Se utiliza `$((expresión))`

```
# INCREMENTAR LA VARIABLE contador  
contador=$((contador + 1))
```

### 2.12. Comandos útiles

- Enviar un email

```
echo Cuerpo del mensaje | mail -s "Asunto del mensaje" -a ficheroadjunto alvarogonzalezprofesor@gmail.  
→ com
```

- Operaciones matemáticas, si no se dispone de *bash*

```
echo 2+2 | bc
```

- *Text User Interface*

- `whiptail` o `dialog`

### 2.13. Ejercicio

- Usa `whiptail` / `dialog` para hacer un *script* que muestre información del sistema, con opciones para:
  - Mostrar el nombre del ordenador (`hostname`)
  - Mostrar las IPs (`hostname -I`)
  - Mostrar espacio libre en los discos (`df | grep`)
  - Mostrar usuarios con sesión abierta (`who`)
- Después de mostrar la información, se vuelve al diálogo, hasta que se elija la opción "salir".



---

## 2.14. Solución

```
#!/bin/bash

dialogo() {
    whiptail --menu "Elige una opción" 20 50 12 \
        1 "Mostrar el nombre del ordenador"\
        2 "Mostrar las direcciones IP asignadas"\
        3 "Mostrar espacio libre en los discos"\
        4 "Mostrar usuarios con sesión abierta"\
        5 "Salir"
}

resultado_de_dialogo() {
    dialogo 3>&2 2>&1 1>&3
}

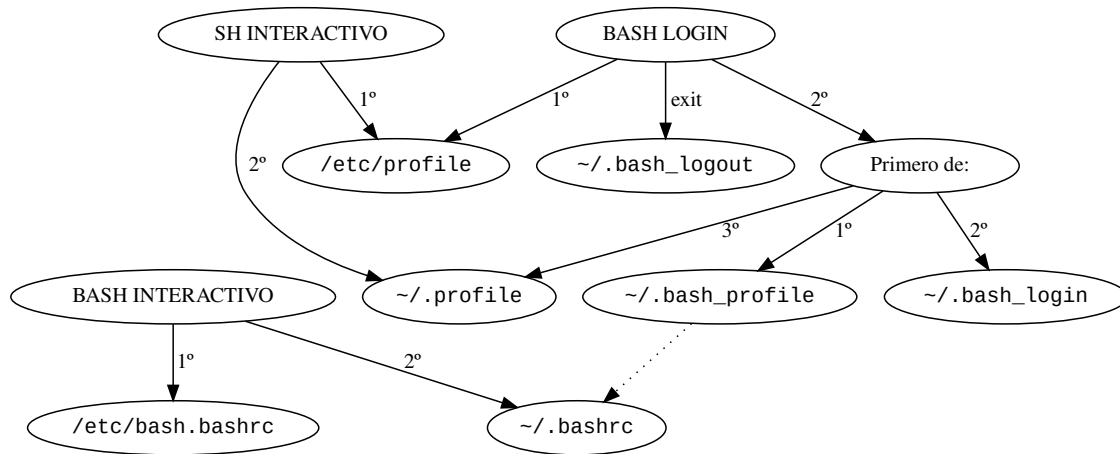
ejecuta_opcion() {
    case $1 in
        1)
            hostname
            ;;
        2)
            hostname -I
            ;;
        3)
            df | grep /dev | grep -v tmpfs
            ;;
        4)
            who
            ;;
        5)
            echo "Saliendo"
            ;;
    esac
}

respuesta=$(resultado_de_dialogo)
while [ $respuesta -ne 5 ]
do
    ejecuta_opcion $respuesta
    respuesta=$(resultado_de_dialogo)
    read -p "Pulsa intro para continuar"
done
```

## 3. Entorno inicial

- bash procesa los siguientes ficheros:
  - Si es un nuevo *login*: /etc/bash.bashrc (Debian), /etc/bashrc (Centos), ~/.bash\_login
  - Siempre: /etc/profile/ ~/.bashrc
- Así, se puede conseguir que
  - Se ejecuten comandos al entrar a un sistema
  - Se ejecuten comandos en cada nueva *shell*
  - Se ejecuten comandos para todos los usuarios

### 3.1. Ficheros de inicio de *shell*



### 3.2. Ejercicios

1. Edita los ficheros de inicio y añade una orden `echo ESTOY EN EL FICHERO XXXXXX`. Comprueba cuándo se invoca cada fichero.
2. Haz que el usuario `alumno` tenga las variables necesarias para utilizar **Oracle** cada vez que hace *login*.
3. Haz que el usuario `alumno` vea el estado del *listener* cada vez que inicia una *shell*.
4. Haz que todos los usuarios vean el mensaje Bienvenido a ASGBD cada vez que inician una *shell*.
5. Crea un *script* en `$HOME/estado.sh` que muestre el valor de las variables `HOME`, `USER`, `PATH`, `ORACLE_HOME`, `ORACLE_SID`, `ORACLE_BASE` y la hora actual.

## 4. Prerrequisitos

- Los comandos de **Oracle** necesitan conocer a qué instancia hacen referencia
- Para ello, necesitan las variables de entorno `ORACLE_HOME` y `ORACLE_SID`.
- También es conveniente añadir los comandos de **Oracle** al *path*
- El siguiente *script* puede utilizarse para tener estas variables (ejecutándolo con `source`)

```
#!/bin/sh
ORACLE_HOME=/var/oracle/product/12.1.0/asir_bbdd
ORACLE_SID=asir
PATH=$ORACLE_HOME/bin:$PATH
export ORACLE_HOME
export PATH
export ORACLE_SID
```

---

## 4.1. Autenticación de SQLPlus

- **SQLPlus** se autentica/autentifica de varias formas
  - Mediante **Oracle**: usuarios creados con `create user..`
  - Mediante el **sistema operativo**: Al instalar, se indica un grupo de usuarios que **Oracle** considera autenticados (grupo `wheel`)

**SQLPlus** con autenticación de sistema operativo

```
SHOW PARAMETER os_authent_prefix; -- Por defecto es 'ops$'  
CREATE USER ops$usuario IDENTIFIED EXTERNALLY;
```

```
su usuario  
sqlplus /
```

**SQLPlus** con autenticación de **Oracle**

```
sqlplus sys/alumno as sysdba
```

## 4.2. Conexiones de SQLPlus

- Hasta ahora
  - todas las conexiones de **SQLPlus** son locales, sin utilizar la red
  - todas las conexiones de **SQLDeveloper** son por red
- Para conectar por red con **SQLPlus** se utiliza un descriptor de conexión
  - Los descriptors están en el fichero `tnsnames.ora`

```
sqlplus sys/alumno@CONEXION as sysdba
```

## 4.3. `tnsnames.ora`

```
MYSID=  
(DESCRIPTION =  
  (ADDRESS_LIST =  
    (ADDRESS = (PROTOCOL = TCP) (HOST = mydnshostname) (PORT = 1521))  
  )  
  (CONNECT_DATA =  
    (SERVICE_NAME = MYSID)  
  )  
)
```

- Situado en `$ORACLE_HOME/network/admin/`
- Indica las formas de conexión a instancias de base de datos
  - Protocolo de conexión: TCP
  - Dirección IP
  - Puerto
  - SID
- Equivalen a la lista de conexiones de **SQLDeveloper**

---

#### 4.4. Conexiones sin `tnsnames.ora`

- No es necesario cambiar el fichero `tnsnames.ora` para conectarse a un servidor remoto
- Aunque puede ser recomendable
  - Ejemplo: varios scripts usan un nombre de conexión, de forma que pueda cambiarse fácilmente

```
sqlplus username/password@host:port/sid
```

### 5. Ejecución de SQL desde la *shell*

- El comando `sqlplus` puede ejecutarse desde la *shell*
- Lee las órdenes SQL desde la entrada estándar.
  - Se puede redirigir de un fichero
  - Se puede usar un *heredoc*

#### 5.1. Ejemplo *heredoc*

```
sqlplus -S alumno/alumno <<HEREDOC
set autocommit off
create table prueba(un_atributo int);
insert into prueba values(1);
insert into prueba values(2);
select * from prueba;
rollback;
HEREDOC
```

#### 5.2. Consultas a fichero

- Puede enviarse la salida a un fichero

```
sqlplus -S alumno/alumno <<HEREDOC
set autocommit off
insert into prueba values(1);
insert into prueba values(2);
spool prueba.txt
select * from prueba;
spool off
rollback;
HEREDOC

less prueba.txt
```

#### 5.3. Formateo básico de la salida

- Tiene algunas facilidades para formatear la salida (por ejemplo, para generar ficheros **CSV**)

```

sqlplus -S sys/alumno as sysdba <<HEREDOC
set colsep ',' -- separate columns with a comma
set pagesize 0 -- No header rows
set trimspool on -- remove trailing blanks
set headsep off -- this may or may not be useful...depends on your headings.
set linesize 1000 -- X should be the sum of the column widths

spool tablas.csv

select table_name, tablespace_name
  from all_tables
 where owner = 'SYS'
    and tablespace_name is not null;

spool off
HEREDOC

```

- Ver también `set markup csv on`

## 5.4. Scripts SQL para sqlplus

- sqlplus también puede leer scripts de SQL con @

```

sqlplus -S sys/alumno as sysdba <<HEREDOC
@camino/al/fichero.sql
HEREDOC

```

# 6. Arranque y parada

## 6.1. dbstart y /etc/oratab

- Oracle proporciona el *script* dbstart para arrancar instancias de base de datos
- Se guía por el contenido de /etc/oratab
- Por alguna razón,
  - no levanta el *listener* :(
  - no hace startup open, así que no se registra en el *listener* :(
  - Se puede modificar el *script* para que lo haga

```

# This file is used by ORACLE utilities.  It is created by root.sh
# and updated by either Database Configuration Assistant while creating
# a database or ASM Configuration Assistant while creating ASM instance.

# A colon, ':', is used as the field terminator.  A new line terminates
# the entry.  Lines beginning with a pound sign, '#', are comments.
#
# Entries are of the form:
#   $ORACLE_SID:$ORACLE_HOME:<N|Y>:
#
# The first and second fields are the system identifier and home
# directory of the database respectively.  The third field indicates
# to the dbstart utility that the database should, "Y", or should not,
# "N", be brought up at system boot time.
#

```

```
# Multiple entries with the same $ORACLE_SID are not allowed.
#
#
asir:/var/oracle/product/12.1.0/asir_bbdd:Y
```

## 6.2. Ejecutar Oracle al iniciar el sistema

- Cada sistema operativo tiene sus formas de arrancar servicios/demonios al inicio
  - **Windows:** Servicios
  - **Linux:**
    - **systemd:** Ficheros en el directorio `/etc/systemd/system`. Se controla con la orden `systemctl`
    - **rc init:** Se basaba en *scripts* en los directorios `/etc/rc.*`. Se está reemplazando por **systemd**

## 6.3. systemd

- Los servicios se crean con ficheros en `/etc/systemd/system` (entre otros)
  - Dependen de otros servicios (After)
  - Otros servicios dependen de ellos (WantedBy)
  - Se puede elegir el usuario que lo lanza (User)

```
[Unit]
Description=Oracle
After=network.target

[Service]
Type=forking
User=alumno
ExecStart=/home/alumno/oracle-al-inicio.sh
ExecStop=/home/alumno/oracle-al-final.sh

[Install]
WantedBy=multi-user.target
```

Más información con `man systemd.service` y `man systemd.unit`

### 6.3.1. Dependencias de la sección Unit

Si el servicio A ...	A intenta arrancar B	Si se apaga B, A también	A arranca si B está inicialmente apagado	Si falla arranque de B, arranca A
<code>BindsTo=B</code>	Sí	Sí	Sí	No
<code>Requires=B</code>	Sí	No*	Sí	No
<code>Requisite=B</code>	No	No	No	No
<code>Wants=B</code>	Sí	No	Sí	Sí

Fuente: [freedesktop](#)

[\*] Si B se apaga explícitamente, se apaga A. Si el servicio B termina sin error, no se apaga A

---

### 6.3.2. Dependencias de la sección **Install**

Si el servicio A		
WantedBy=B		Equivalente a que B indique Wants=A
RequiredBy=B		Equivalente a que B indique Required=A

### 6.3.3. Controlar el arranque de servicios

- Habilitar/Deshabilitar un servicio al inicio del sistema

```
systemctl enable SERVICIO
systemctl disable SERVICIO
```

- Arrancar o parar un servicio

```
systemctl start SERVICIO
systemctl stop SERVICIO
```

### 6.3.4. Recargar ficheros

- Si se crean nuevas unidades, no se tienen en cuenta automáticamente
- Se necesita `systemctl daemon-reload`

### 6.3.5. Servicios de usuario

- El comando `systemctl` se ejecuta con el parámetro `--system` por defecto
  - Servicios de sistema
- Si se ejecuta con `--user` se utilizan los servicios del usuario llamante
  - Sus unidades están en `$HOME/.config/systemd/user/`
- Es una opción que no existe en Centos.
  - Es posible que los quiten de todas las distribuciones linux

### 6.3.6. *Runlevels y targets*

Runlevel	Target	Alias
0	poweroff.target	runlevel0.target
1	rescue.target	runlevel1.target
3	multi-user.target	runlevel3.target
5	graphical.target	runlevel5.target
6	reboot.target	runlevel6.target

### 6.3.7. Utilidades `systemd`

- `systemd-analyze plot`: Tiempo de carga de cada servicio
- `systemd-analyze dot`: Fichero con todas las dependencias. Puede convertirse en SVG con `dot -Tsvg -o nombre-dibujo.svg`
- `systemctl list-dependencies`: Servicios necesarios para cargar un servicio. Se pueden listar a la inversa con `--reverse`

---

## 7. Operaciones periódicas

- Los sistemas operativos aportan formas para ejecutar tareas periódicamente
  - **Windows** tiene las **tareas programadas**
  - **Linux** tiene el sistema `cron` y `systemd`

### 7.1. `cron`

- Es un servicio que
  - Lee el fichero `/etc/crontab`
  - Ejecuta las órdenes descritas en ese fichero
  - Más información [en la Wikipedia](#)
- Suele utilizar el comando `run-parts`
  - Este comando ejecuta todos los comandos de un directorio
  - Más información con `man run-parts`

### 7.2. `systemd`

Más información en `man systemd.timer` y `man systemd.time`

```
[Unit]
Description=Prints date into /tmp/date file

[Service]
Type=oneshot
ExecStart=/usr/bin/sh -c '/usr/bin/date >> /tmp/date'
```

Listado 1: `/etc/systemd/system/date.service`

```
[Unit]
Description=Run date.service every 10 minutes

[Timer]
OnCalendar=*:0/10
```

Listado 2: `/etc/systemd/system/date.timer`

### 7.3. Operaciones periódicas manuales

- Se puede crear un bucle infinito con `sleep`
- El bucle se interrumpe con alguna condición externa
  - Por ejemplo, que exista o deje de existir un fichero

```
#!/bin/bash

rm $HOME/elbucledebeparar
hay_que_seguir(){
    if [ -e $HOME/elbucledebeparar ]
    then
        return 1
    fi
}
```



```

    else
        return 0
    fi
}

SEGUNDOS=3
sleep $SEGUNDOS
while hay_que_seguir
do
    echo han pasado $SEGUNDOS segundos
    sleep $SEGUNDOS
done

```

## 8. Avanzado

### ■ Literales

- Comillas dobles: sustituyen variables
- Comillas simples: no se sustituyen variables

### ■ Heredocs

- **Indentados:** «-EOF

»• Sin sustitución de variables: «'EOF'

»

»• **Process substitutions:** Trata una orden como un fichero temporal

```

» diff <(ls $first_directory) <(ls $second_directory)
»

```

Listado 3: Compara los contenidos de un directorio con otro

»• mktemp para ficheros temporales

»• **traps** para finalizar un *script*

»• **awk**, **sed**, **whiptail**

## 9. Más avanzado: comandos desde PL/SQL

- [http://www.dba-oracle.com/t\\_execute\\_operating\\_system\\_os\\_command\\_oracle.htm](http://www.dba-oracle.com/t_execute_operating_system_os_command_oracle.htm)
- Se necesita el privilegio CREATE\_EXTERNAL\_JOB
- Solo he conseguido escribir ficheros en \$ORACLE\_HOME

```

begin
    -- BORRAR EL TRABAJO POR SI YA EXISTE
    dbms_scheduler.drop_job( job_name => 'myjob' );

    -- CREAR TRABAJO Y PONER ARGUMENTOS
    dbms_scheduler.create_job (job_name      => 'myjob',
                              job_type      => 'executable',
                              job_action    => '/bin/touch',
                              number_of_arguments => 1,

```

---

```
                                auto_drop    => true);
dbms_scheduler.set_job_argument_value ('myjob', 1, '/opt/oracle/product/18c/dbhomeXE/fichero.nuevo
↵ ');

-- EJECUTAR TRABAJO
dbms_scheduler.run_job ('myjob');
end;
/
```

## 10. Referencias

- Formatos:
  - ◊ [Transparencias](#)
  - ◊ [PDF](#)
  - ◊ [EPUB](#)
- Creado con:
  - ◊ [Emacs](#)
  - ◊ [org-re-reveal](#)
  - ◊ [Latex](#)
- Alojado en [Github](#)
- [oracle-base](#)