REDIRECCIONES

output (salida estándar)

tee fichero	# output a fichero y a pantalla
> fichero	# output a fichero
>> fichero	# output al final del fichero
> /dev/null	# descarta output

error

2>&1	# error a output
2> fichero	# error a fichero
2>> fichero	# error al final del fichero
2> /dev/null	# descarta error

output y error

2>&1 | tee fichero	# ambos a fichero y a pantalla
&> fichero	# ambos a fichero
&>> fichero	# ambos al final del fichero

VARIABLES

variables de entornobrack

$PWD	# directorio de trabajo actual
$OLDPWD	# directorio de trabajo anterior
$PPID	# identificador del proceso padre
$HOSTNAME	# nombre del ordenador
$USER	# nombre del usuario
$HOME	# directorio del usuario
$PATH	# rutas búsqueda de comandos
$LANG	# idioma para los mensajes
$FUNCNAME	# nombre función en ejecución
$LINENO	# número de línea actual (del script)
$RANDOM	# número aleatorio

variables especiales

$0		# nombre del script
${N}		# parámetro N
$$		# identificador del proceso actual
$!		# identificador del último proceso
$@ (como array)	**$*** (como string)	# todos los parámetros recibidos
$#		# número de parámetros recibidos
$? (0=normal, >0=error)		# código de retorno del último comando
shift		# $1=$2, $2=$3, ... ${N-1}=${N}

ARRAYS

declare –a ARRAY	# declaración array
ARRAY**=(**valor1 ... valorN**)**	# asignación compuesta
ARRAY**[N]=**valorN	# asignación simple
ARRAY**=([N]=**valorN valorM **[P]=**valorP**)**	# asigna celdas N, M y P
${ARRAY**[N]}**	# valor celda N
${ARRAY**[*]}**	# todos los valores

Autor: Adolfo Sanz De Diego (asanzdiego - Blog|GitHub|LinkedIn|SlideShare|Twitter)

OPERADORES

operadores aritméticos

+	# suma
-	# resta
*****	# multiplicación
/	# división
%	# resto
++	# incremento
--	# decremento

operadores comparaciones numéricas

numero1 **–eq** numero2	# numero1 igual que numero2
numero1 **–ne** numero2	# numero1 distinto que numero2
numero1 **–lt** numero2	# numero1 menor que numero2
numero1 **–le** numero2	# numero1 menor o igual que numero2
numero1 **–gt** numero2	# numero1 mayor que numero2
numero1 **–ge** numero2	# numero1 mayor o igual que numero2

operadores lógicos

!	# NOT
&& , -a	# AND
|| , -o	# OR

operadores de ficheros

-e fichero		# existe
-s fichero		# no está vacío
-f fichero		# normal
-d fichero		# directorio
-L fichero	**-h** fichero	# enlace simbólico
-r fichero		# permiso de lectura
-w fichero		# permiso de escritura
-x fichero		# permiso de ejecución
-O fichero		# propietario
-G fichero		# pertenece al grupo
f1 **-ef** f2		# f1 y f2 enlaces mismo archivo
f1 **-nt** f2		# f1 más nuevo que f2
f1 **-ot** f2		# f1 más antiguo que f2

operadores de cadenas

-n cadena	**-z** cadena	# no vacía	# vacía
cadena1 **==** cadena2		# cadena1 igual a cadena2	
cadena1 **!=** cadena2		# cadena1 distinta a cadena2	

EXPRESIONES

if [condición **]**		# alias comando test (más portable)
if [c1 **] && [** c2 **]**	**if [** c1 **] || [** c2 **]**	# juntar expresiones and y or
if [[condición **]]**		# [] mejorado pero no sh (solo bash/ksh)
if ((expresión aritmética **))**		# no sh (solo bash/ksh)
if (comando **)**		# si exit comando = 0 -> true

Autor: Adolfo Sanz De Diego (asanzdiego - Blog|GitHub|LinkedIn|SlideShare|Twitter)

ENTRECOMILLADO

#! RUTA	# ruta al interprete (/bin/bash)
\carácter	# valor literal del carácter
linea1 \ linea2	# para escribir en varias lineas
'cadena'	# valor literal cadena
“cadena”	# valor literal cadena, excepto $ ' \

EXPANSIÓN

[prefijo]{cad1,[,...],cadN}[sufijo]	# = precad1suf ... precadNsuf
${VARIABLE:-valor}	# si VARIABLE nula, retorna valor
${VARIABLE:=valor}	# si VARIABLE nula, asigna valor
${VARIABLE:?mensaje}	# si VARIABLE nula, mensaje error y fin
${VARIABLE:inicio}	# recorta desde inicio hasta el final
${VARIABLE:inicio:longitud}	# recorta desde inicio hasta longitud
${!prefijo*}	# nombres de variables con prefijo
${#VARIABLE}	# número de caracteres de VARIABLE
${#ARRAY[*]}	# elementos de ARRAY
${VARIABLE#patrón}	# elimina mínimo patrón desde inicio
${VARIABLE##patrón}	# elimina máximo patrón desde inicio
${VARIABLE%patrón}	# elimina mínimo patrón desde fin
${VARIABLE%%patrón}	# elimina máximo patrón desde fin
${VARIABLE/patrón/reemplazo}	# reemplaza primera coincidencia
${VARIABLE//patrón/reemplazo}	# reemplaza todas las coincidencias
$((expresión))	# sustituye expresión por su valor
$[expresión]	# sustituye expresión por su valor

EJECUCIÓN

./comando		# ejecuta desde directorio actual
$RUTA/comando		# ejecuta desde cualquier sitio
comando		# ejecuta si está en el $PATH
. script	source script	# ejecuta exportando variables (=import)
$(comando p1 ... pN)	`comando p1 ... pN`	# ejecuta en subselect
comando &		# ejecuta en segundo plano
c1 | c2		# redirige salida c1 a entrada c2
c1 ; c2		# ejecuta c1 y luego c2
c1 && c2		# ejecuta c2 si c1 termina sin errores
c1 || c2		# ejecuta c2 si c1 termina con errores

ARGUMENTOS DE LÍNEA DE COMANDOS

```
while getopts "hs:" option ; do
  case "$option" in
    h) DO_HELP=1 ;;
    s) argument=$OPTARG ; DO_SEARCH=1 ;;
    *) echo "Invalid" ; return ;;
  esac
done
```

```
# getops + "opciones disponibles"
#   mientras haya argumentos
#     seleccionamos
#       -h sin opciones
#       -s con opciones en $OPTARG
#        * error
```

ESTRUCTURAS DE CONTROL

```
if expresión1; then
      bloque1
elif expresión2; then
      bloque2
else
      bloque3
fi
```

```
# condicional
#   si expresión1 entonces
#      bloque1
#   sino y expresión2 entonces
#      bloque2
#   si ninguna entonces
#      bloque2
```

```
case VARIABLE in
      patrón11|...|patrón1N)
            bloque1 ;;
      patrón21|...|patrón2N)
            bloque2 ;;
      *)
            bloqueDefecto ;;
esac
```

```
# selectiva
#   si VARIABLE coincide con patrones1
#      entonces bloque1
#   si VARIABLE coincide con patrones2
#      entonces bloque2
#   si ninguna
#      entonces bloqueDefecto
```

```
for VARIABLE in LISTA; do
      bloque
done
```

```
# iterativa con lista
#   ejecuta bloque sustituyendo
#   VARIABLE por cada elemento de LISTA
```

```
for ((expr1; expr2; expr3; )); do
      bloque
done
```

```
# iterativa con contador
#   primero se evalúa exp1
#   luego mientras exp2 sea cierta
#   se ejecutan el bloque y expr3
```

```
while expresión; do
      bloque
done
```

```
# bucle “mientras”
#   se ejecuta bloque
#   mientras expresión sea cierta
```

```
until expresion; do
      expresion
done
```

```
# bucle “hasta”
#   se ejecuta bloque
#   hasta que expresión sea cierta
```

```
[function] expresion () {
   ... [ return [valor] ] ...
}
```

```
# función
#   se invoca con
#   nombreFunción [param1 ... paramN]
```

INTERACTIVIDAD

```
read [-p mensaje] [variable1 ...]
```

```
# input lee teclado y asigna a variables
```

```
echo cadena
  -n no hace salto de linea
  -e interpreta caracteres con \
```

```
# output
#   manda el valor de la cadena
#   a la salida estándar
```

```
printf
```

```
# output formateado (igual que C)
```

```
cat << NOMBRE_ETIQUETA
texto con
  varias lineas y con tabuladores
NOMBRE_ETIQUETA
```

```
# output
#   saca por pantalla el testo con
#   varias lineas y con tabuladores
#   que está entre NOMBRE_ETIQUETA
```

CONTROL DE PROCESOS

```
bg númeroProceso
```

```
# continúa ejecución en segundo plano
```

```
fg númeroProceso
```

```
# continúa ejecución en primer plano
```

```
jobs
```

```
# muestra procesos en ejecución
```

```
kill señal PID1|númeroProceso1
```

```
# mata proceso(s) indicado(s)
```

```
exit código
```

```
# salir con código de retorno
# (0=normal, >0=error)
```

```
trap [comando] [código1 ...]
```

```
# ejecuta comando cuando señal(es)
```

```
wait [PID1|númeroProceso1]
```

```
# espera hasta fin proceso(s) hijo(s)
```

```
nice -n prioridad comando
```

```
# ejecuta con prioridad -20(MAX) 19(MIN)
```

```
renice -n prioridad comando
```

```
# modifica prioridad -20(MAX) 19(MIN)
```

Autor: Adolfo Sanz De Diego (asanzdiego - Blog|GitHub|LinkedIn|SlideShare|Twitter)