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

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 cadena] [variable1 ...]
```

```
# input
#   lee teclado y asigna a variables
#   puede mostrarse un mensaje antes
#   si ninguna variable, REPLY = todo
```

```
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)
```

CONTROL DE PROCESOS

```
comando &
```

```
# ejecuta en segundo plano
```

```
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)
```