Parámetros de entrada posicionales

- Es posible pasar parámetros adicionales en el momento en que invocamos al *script* en la línea de comandos para ejecutarlo.
- Dentro del *script* se pueden recuperar los parámetros que se pasaron al invocarlo con `$1`, `$2`, `$3`, etc. El número indica la posición del parámetro a recuperar.
- `$0` representa el propio nombre con el que se invocó al *script*.
- `$#` representa el número de parámetros (sin contar el 0).
- `$*` devuelve todos los parámetros posicionales (en una sola cadena).
- ”`$*`” devuelve ”`$1 $2`” ... (en una sola cadena).
- `$@` representa los parámetros posicionales (igulal que `$*` pero separados en varias cadenas).
- ”`$@`” devuelve ”`$1`” ”`$2`” ...
- `shift` desplaza los parámetros. Por ejemplo, `$3` pasa a ser `$2` y se actualiza el valor de `$#`.

Sentencia `if`

- Permite decidir si se ejecuta un comando o grupo de comandos según el status de salida con el que acabe el comando de la condición.
- Sintaxis parecida a muchos lenguajes de programación.
  - Primer bloque condicional con `if`.
  - Sucesivos bloques alternativos (va probando si falla el primero) con `elif`.
  - Bloque por defecto (si no entra en ninguno de los anteriores) con `else`.
  - La estructura acaba con `fi`.
- Si el comando de la condición acaba con éxito (status 0) entonces es verdadero, si devuelve fallo es falso.

Sentencia `if`

```
if comando
then
    comando1
    comando2
elif otrocomando
then
    comando3
    comando4
else
    comando 5
    comando 6
fi
```

Sentencia `if`

- Se puede negar la condición del resultado de un comando con el carácter `!`

```
if ! comando
then
    comando1
    comando2
fi
```

Sentencia case

- Ejecución condicional en función de si localiza un patrón.

```
case palabra in
patrón1)
    comandos
    ;;
patrón2 | patrón3)
    comandos
    ;;
*)  # este es el default
    comandos
    ;;
esac
```

Bucles

- Sólo hay `while` y `for`.

```
# Ejemplo de bucle while
while comando
do
    comandos
done
# Ejemplo de bucle for
for variable in palabra1 palabra2 palabraN
do
    comandos
done
```

Sentencias de control en comandos

- Se puede poner el carácter ; al final de una sentencia para continuar escribiendo en la misma línea.
- Esto es útil en la terminal interactiva, puesto que el carácter de final de línea acabaría el comando

```
while ls|egrep '\.z$'; do
    comandos
done
```

Comando read

- El comando `read` permite leer una línea (cadena de caracteres) de su entrada estándar y guardarla en la variable que se le pasa como argumento.
- Por ejemplo, esto permite procesar la entrada línea por línea mediante un bucle.
- Por supuesto, si ya tenemos otro comando, filtro o *pipeline* de comandos que ya hace lo que necesitamos es mejor usar esa opción en lugar de `read`.
  - Los comandos ya disponibles son muchísimo más eficientes.

Comando read: ejemplo

```
# En la terminal, creamos un fichero con dos líneas y dos letras en cada línea
$ echo 'a b
  c d' > /tmp/e
# Ahora, creamos y ejecutamos un shell script para leer cada línea
# y repetirla por pantalla. Fichero de entrada: /tmp/e
while read line
do
    echo $line
done <  /tmp/e
# Sin embargo, este otro shell script itera 4 veces (procesa letra a letra).
for x in 'cat /tmp/e'
do
    echo $x
done
```

Variable IFS

- Es una variable que contiene los caracteres reconocidos como separadores entre campos.
- Por defecto, contiene el tabulador, el espacio en blanco y el salto de línea.
- Cuidado: si tocamos indebidamente el valor de esta variable podemos hacer que todo deje de funcionar como es debido.
  - Por ejemplo, el espacio en blanco ya no serviría para separar los argumentos que paso detrás de un comando.

```
$ export IFS=-
$ for i in $(echo uno dos tres) ; do echo $i ; done
uno dos tres
$
```

Funciones

- Se pueden definir funciones, accediendo a sus parámetros como hacemos con los parámetros posicionales del *script* principal.

```
hello () {
    echo hola $1
    shift
    echo adios $1
}
```

- Ahora ejecutamos la función.

```
$ hello uno dos
hola uno
adios dos
```

El comando `alias`

- Define una etiqueta para invocar un comando o conjunto de comandos de forma directa.
- Sin argumentos, `alias` muestra los que hay definidos.
- El comando `unalias` elimina un alias previamente definido.

```
$ alias hmundo='echo hola mundo'
$ hmundo
hola mundo
$ unalias hmundo
$ hmundo
hmundo: command not found
$ alias
alias la='ls -A'
alias ll='ls -alF'
alias ls='ls --color=auto'
$
```

Comando `test`

- Para comprobar diferentes tipos de condiciones.
- Con ficheros:
  - `-f fichero`
    Comprueba si existe el fichero.
  - `-d dir`
    Comprueba si existe el directorio.

Comando `test`

- Con cadenas de caracteres:
  - `-n String1`
    Si la longitud de la string no es cero.
  - `-z String1`
    Si la longitud de la string es cero.
  - `String1 = String2`
    Si son iguales.
  - `String1 != String2`
    Si las cadenas en `String1` y `String2` no son idénticas.
  - `String1`
    . Si la cadena en `String1` no es nula.

Comando `test`

- Con enteros:
  - `Integer1 -eq Integer2`
    Si los enteros `Integer1` e `Integer2` son iguales.
- Otros posibles operadores con enteros:
  - `-ne` : no igual (comprueba si son distintos).
  - `-gt` : mayor que.
  - `-ge` : mayor o igual que.
  - `-lt` : menor que.
  - `-le` : menor o igual que.

Sintaxis alternativa para `test`

- El comando:

```
[ $a -eq $b ]
```

- Es equivalente a este otro:

```
test $a -eq $b
```

Operaciones aritméticas con números enteros

- La Shell permite realizar operaciones básicas que involucren números enteros.
- Otra alternativa es usar el comando bc.

```
$ echo $((5 + 7))
12
$
```