1. ¿Qué son?

Un programa, macro o script, tal y como lo vamos a estudiar aquí es un conjunto de comandos que se ejecutan con una sola orden, de forma que nos ahorramos tener que ir tecleándolos uno por uno. Así podemos automatizar trabajos.

Las macros bash, que son las que vamos a ver, tienen bastante potencia, pero en este curso, nos restringiremos a sólo una parte menor. Es decir, que dejamos bastantes cosas fuera, y todos los que quieran aumentar la eficiencia están invitados a buscar en la documentación de bash todo lo que nos ha quedado fuera del curso.

2. ¿Cómo se hacen funcionar?

Para activar nuestra macro, lo hacemos desde la consola de comandos, con el nombre que le hallamos dado. Esto se denomina "llamada" Si, por ejemplo, se llama `macro1.sh` y está en la carpeta actual, lo pondríamos en marcha con:

```
./macro1.sh
```

3. Argumentos

Muchas veces, nuestros programas recibirán datos: sobre qué actuar, con qué, etc. Se llaman argumentos y dentro del programa se usan con `$1, $2, $3`, etc. según su posición en la llamada. Por ejemplo si la llamada es:

```
./macro1.sh pruebas txt
```

Entonces en el programa `$1` sería `pruebas` y `$2` sería `txt`.

Un ejemplo tonto de macro que lo único que haga es escribir sus dos argumentos por orden inverso al que se le mandan sería:

```
#!/bin/bash
#Macro que escribe argumentos por orden inverso
echo $2 $1
```

Cuando hemos procesado ya los primeros argumentos y queremos borrarlos usamos `shift` n siendo n la cantidad de argumentos iniciales a borrar. Por ejemplo: `shift 2` borraría el primero y el segundo

Además, podemos usar los siguientes:

`$#` el número de argumentos

`$@` son todos los argumentos

También tenemos los patrones ya vistos al hablar de comandos, como * y ?

4. Organización general

La primera línea será:

```
#!/bin/bash
```

Después pondremos la secuencia de comandos.

Se puede hacer con funciones, pero no lo vamos a ver en este curso.

5. Comentarios

Cualquier línea más allá de la primera que empiece por # se considerará un comentario, algo que ponemos para aclararnos nosotros pero que no hay que hacer.

6. Resultados intermedios. Variables

Podemos guardar cualquier resultado intermedio poniéndole un nombre. Eso crea una "variable". La forma de hacerlo es:

```
nombre = valor
```

Si el valor lo queremos pedir por pantalla pondremos

`read -p "mensaje en pantalla" nombre`

Por ejemplo: `read -p "¿Cuántos son?" cuantos`

Cuando queremos usar el valor que sale de ejecutar un comando, ponemos `$(comando)`

Comoquiera que hayamos cargado la variable, su valor se puede usar más adelante poniendo `$nombre`

7. Salida en pantalla

El comando echo nos pone en pantalla lo que le pasemos. Por ejemplo:

```
echo "El primer argumento es " $1
```

8. Condiciones

Si ejecutar algunos comandos depende de alguna condición, la plantilla a usar es:

```
if ....
then
    comandos
elif ...
then
```

```
    comandos
...
else
    comandos
fi
```

Ahí tenemos varias condiciones, asociadas cada una a unos comandos. Las condiciones van detrás de `if` o `elif` Se ponen entre corchetes, que siempre irán separados con espacios de lo elementos. Por ejemplo:

```
if [ "$1" = "prueba" ]
```

Para combinar condiciones están los operadores Y (`&&`) y O (`||`); por ejemplo:

```
if [ "$1" = "prueba" ] && [ "$2" = "txt" ]
```

Los operadores más corrientes que se usan (también se deben separar con espacios) son:

Operador	Significado	Ejemplo
Para texto		
=	igual que	"$1" = "prueba"
!=	distinto de	"$2" != "txt"
Para ficheros		
-a	existe	-a x.log
-d	es un directorio	-d "$1"

9. Comandos que se ejecutan dependiendo del resultado de otros

Comentamos dos casos:

1. Cuando el segundo comando sólo debe hacerse si el primero ha funcionado normal: ponemos el primero, luego && y luego el segundo. Por ejemplo:
   `mv $fich $fich.log && echo "Renombrado con éxito"`

2. Cuando el segundo comando sólo debe hacerse si el primero ***no*** ha funcionado normal: ponemos el primero, luego || y luego el segundo. Por ejemplo:
   `mv $fich $fich.log || echo "No he podido renombrarlo"`

10. Parada

En cualquier punto en que nos convenga parar una macro lo podemos hacer con el comando `exit`

11. Repeticiones

Para procesar una serie de elementos, de uno en uno, por ejemplo, una serie de ficheros, usamos la plantilla:

```
for var in lista
do
    comandos
done
```

De esta forma, `var` va tomando sucesivamente cada uno de los valores que hay en `lista`, a los que se les va aplicando los comandos que hay dentro del ciclo.

Por ejemplo, ir cambiando el nombre de todos los argumentos a como eran antes con un `.log` al final:

```
for fich in ‘‘$@’’
do
    mv $fich $fich.log
done
```