Algo sobre bash scripting. 15 recetas.

1. Todo script debe comenzar con el "shebang", que indica al sistema el intérprete que debe interpretar las líneas:

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

2. Los comandos que hay en el script, son comandos que se podrían ejecutar en una línea de comandos. La única diferencia, es que se pueden ejecutar secuencias de comandos más complejas:

```
#!/bin/bash
echo "Un script de prueba"
```

3. Para que un script pueda ejecutarse, debe tener permisos de ejecución:

```
$ chmod 750 script.sh
$ ./script.sh
```

NOTA: Los permisos 750 (rwxr-x---) indican que el archivo solo puede ser ejecutado por el propietario y por los pertenecientes al grupo del propietario.

4. Las variables almacenan temporalmente un valor para poder procesarlo durante la ejecución del script.

```
#!/bin/bash
variable="78345aas2wo2332"
echo "El contenido de la variable es el siguiente: "$variable
```

5. Recogida de variables: el comando "read nombre_variable" introduce en la variable $nombre_variable lo que el usuario introduzca por teclado:

```
#!/bin/bash
echo "Introduce tu nombre: "
read nombre_persona
echo "Tu nombre es $nombre_persona"
```

6. Se pueden crear funciones dentro de un script. Una función es un bloque de instrucciones que se pueden invocar con solo indicar el nombre de la función. La variables que se definen dentro de una función, mueren al terminar la función:

```
#!/bin/bash
function ambito {
     a="Variable local de la función ámbito"
     echo "La variable \$a contiene: $a"
}

a="Variable global"
echo "La variable \$a contiene: $a"
echo "Ahora vamos a invocar la función ámbito:"
ambito
```

7. Con bash scripting se puede hacer toma de decisiones, con "if". Por ejemplo, "si existe el archivo llamado 'miarchivo' entonces hacer esto o lo otro":

```
#!/bin/bash
echo "Escribe un nombre de archivo: "
read archivo
if [ -f $archivo ]; then
     echo "El archivo existe"
else
     echo "El archivo no existe"
fi
```

NOTA: Algunos operadores sobre archivos son los siguientes:

Operador	Verdad si
-d archivo	El archivo existe y es un directorio
-e archivo	El archivo existe
-f archivo	El archivo existe y es un archivo regular
- r archivo	Tenemos permiso de lectura en el archivo
-s archivo	El archivo existe y no está vacío
-w archivo	Tenemos permiso de escritura
arch1 -nt arch2	El archivo "arch1" es más nuevo que "arch2"
arch1 -ot arch2	El archivo "arch1" es más antiguo que "arch2"

8. Se pueden hacer varias comparaciones en una sentencia de control, con operadores lógicos "and" (&&), "or"(||) y "not" (!):

```
#!/bin/bash
echo "Escribe un nombre de archivo: "
read archivo
if [ -r $archivo ] && [ -w $archivo ]; then
     echo "El archivo es completamente accesible"
elif [ -r $archivo ]; then
     echo "El archivo solo es accesible en lectura"
elif [ -w $archivo ]; then
     echo "El archivo solo es accesible en escritura"
fi
```

9. Se pueden hacer comparaciones con números. Por ejemplo:

```
#!/bin/bash
if [ $x -eq $y ]; then
echo "son iguales"
fi
```

Comparación	Comando
x = y	$x -eq $y
x != y	$x -ne $y
x < y	$x -lt $y
x <= y	$x -le $y
x > y	$x -gt $y
x >= y	$x -ge $y
x no vacía	-n $z
x vacía	-z $x

10. Multiselección. Cuando hay varias opciones sobre las que tomar una decisón, podemos utilizar "case":

```
#!/bin/bash
echo "Escribe un número"
read valor;
case $valor in
        0) echo "valor es cero" ;;
        1) echo "valor es uno" ;;
        2) echo "valor es dos" ;;
        3) echo "valor es tres" ;;
           *)  echo "valor es cuatro"
esac
```

11. Argumentos en la línea de comandos: Los argumentos acompañan a la ejecución del comando:

```
linux-$  comando argumento1 argumento2 argumento3
```

Variable	Significado
$1	Primer argumento
$2	Segundo argumento
$3	Tercer argumento
$#	Contiene el número de argumentos.
$0	Nombre del script

```
#!/bin/bash

function mostrar_uso {
echo "INSTRUCCIONES DE USO:  $0 dir_origen dir_destino"
exit 1
}
# Main program starts here
if [ $# -ne 2 ]; then
mostrar_uso
else # There are two arguments
if [ -d $1 ]; then
dir_origen=$1
else
echo 'Directorio origen no válido'
mostrar_uso
fi
if [ -d $2 ]; then
dir_destino=$2
else
echo 'Directorio destino no válido'
mostrar_uso
fi
fi
echo "El directorio origen es $dir_origen"
echo "El directorio destino es $dir_destino"
```

12. Tuberías y redirección

Canal	**Descriptor de archivo**
STDIN	0
STDOUT	1
STDERR	2

Símbolo de redirección	Significado
<	Redirigir el contenido de un archivo a un comando.
>	Redirigir el resultado de un comando a un archivo (machacando su contenido anterior).
>>	Añadir el resultado de un comando al final de un archivo.

Los canales STDIN, STDOUT y STDERR son vistos por el sistema como archivos, por lo que los símbolos "0<", "1>", "2>" y ">&" permiten redirigir los contenidos de STDIN, STDOUT y STDERR. Por ejemlo:

`find / -name core 2> /dev/null` → Este comando no mostraría los mensajes "permiso denegado".

13. Bucles for:

```
#!/bin/bash
sufijo=BACKUP--`date +%Y%m%d-%H%M`
for script in `ls *.sh`; do
nuevonombre="$script.$sufijo"
echo "Copiando $script a $nuevonombre..."
cp $script $nuevonombre
done
```

otra forma de utilizar el for, es mediante un contador:

```
for ((i=0; i<$maximo;i++));do
```

14. Bucles while:

```
#!/bin/bash
exec 0<$1
### con la instrucción anterior, el STDIN proviene del primer
argumento.
contador=1
while read linea; do
     echo "$contador: $linea"
     ((contador++))
done
```

NOTA: ((expresión)) obliga la evaluación numérica de "expresión".

15. Arrays. Podemos tener vectores llenos de cosas:

```
#!/bin/bash

mi_array=(valor1 valor2 valor3 valor4)
echo "El array mi_array contiene "${#mi_array[@]}" elementos"

contador = 1
for elem in ${mi_array[@]}; do
     echo " Elemento $contador = "$elem
     ((contador++))
done

echo "Referencia al tercer elemento: "${mi_array[2]}
```