

ADMINISTRACIÓ DE SISTEMES OPERATIUS

U1: SHELL SCRIPT, PART 2

CFGS
ASIX

DPT INF

SHELL SCRIPT

```
vicent@vicent-MS-7C84: ~  
vicent@vicent-MS-7C84:~$ chmod +x script.sh  
vicent@vicent-MS-7C84:~$ ./script.sh  
Hola mundo, este es mi primer script  
vicent@vicent-MS-7C84:~$
```

Vicent Benavent

CFGS ASIX

Mòdul: Administració de Sistemes Operatius

UD1: Shell Script



Tot el temari el pots trobar a Github: [ASO](#)

1 FUNCIONS

Les funcions ens aprofiten per preparar una series d'ordres que es van a repetir constantment, d'aquesta forma evitem escriure cada vegada el mateix. Com funciona? Bàsicament, cal declarar la funció (function) i assignar-li un nom.

```
#!/bin/bash

function nomDeLaFuncio {

    contingut

}
```

Com per exemple, imaginem que volem duplicar un valor d'un nombre:

```
#!/bin/bash
function doblar {
    echo "Doblem el nombre guardat a la variable NOMBRE"
    let NOMBRE=NOMBRE*2
}

NOMBRE=5
echo '$NOMBRE val: ' $NOMBRE
doblar #ara s'executa la funció
echo '$NOMBRE ara val: ' $NOMBRE
```

Amb el resultat:

```
vicent@vicent-MS-7C84:~$ source funcio.sh
$NOMBRE val: 5
Doblem el nombre guardat a la variable NOMBRE
$NOMBRE ara val: 10
vicent@vicent-MS-7C84:~$
```

Encara que a l'exemple estem declarant la funció abans que la variable té cap valor, no importa, ja que eixa part del codi no s'executa fins que en algun punt del programa cridem la funció.

Per defecte totes les variables són globals, és a dir, que totes les funcions i el Script les comparteixen, poden modificar el valor, llegir-les amb les modificacions i etc.

Però en certs casos ens interessa que les variables siguin locals a la funció, estan iniciades dins de la mateixa funció i si per algun motiu la funció modifica eixe valor, el Script no s'entera. Per exemple:

```
#!/bin/bash

function salutacio {
    NOM="Josep Peiró"
    echo "Un plaer haver-lo conegut $NOM"
}

NOM="Ana Climent"
salutacio
echo "En el script principal, el meu nom és $NOM"
```

Actuant de la següent forma:

```
vicent@vicent-MS-7C84:~$ source funcio2.sh
Un plaer haver-lo conegut Josep Peiró
En el script principal, el meu nom és Josep Peiró
vicent@vicent-MS-7C84:~$
```

Com observeu, encara que hem modificat el nom de la variable, com que el valor l'hem donat dins la funció, sempre que s'execute aquesta després de fer cap canvi a \$NOM el canviarà. Si eliminem la crida a la funció el resultat és este:

```
vicent@vicent-MS-7C84:~$ source funcio2.sh
En el script principal, el meu nom és Ana Climent
vicent@vicent-MS-7C84:~$
```

En estos casos, aquesta variable \$NOM sempre ha estat actuant com a variable GLOBAL, en canvi, si la mateixa variable li forcem que actua com una local, aquesta solament té en compte les modificacions quan estem dins de la funció, fora actua igual. Exemple:

```
vicent@vicent-MS-7C84:~$ source funcio2.sh
Un plaer haver-lo conegut Josep Peiró
En el script principal, el meu nom és Ana Climent
```

Com que el primer missatge si que està modificat dins la funció apareix que el nom és Josep, però quan acaba l'execució d'aquesta torna a tindre el seu valor original.

Compte, si volem utilitzar els paràmetres que passem al script, cal introduir-los de nou quan executem una funció, a l'igual, si dins de la funció volem cridar a \$1 \$2 \$3, cal indicar els valors quan es crida la funció:

```
$ test.sh
1  #!/bin/bash
2  read -p "ejem: " valor
3
4  function sumar {
5      let resultat=$1 # $1 contindrà el valor que
6      #passem quan executem la funció
7      echo $resultat
8  }
9
10  sumar $valor #<- $valor serà el valor que rebrà com $1 la funció
```

Un altre exemple, cridem al script passant-li els valors 3, 2 i 3, fixeuvos a l'eixida, \$1 fora de la funció té un valor, però \$1 dins de la funció recull el valor que hem passat quan l'executem (cridem):

```
$ test.sh
1  #!/bin/bash
2  echo "Els valors que arribem per paràmetre són: $1, $2 i $3"
3
4  let suma12=$1+$2
5  let suma23=$2+$3
6  let suma31=$3+$1
7
8
9  function sumar {
10     echo "Els valors que han entrat a la funció sumar són: $1, $2 i $3"
11     let resultat=$1+$2+$3 # $1 contindrà el valor que
12     #passem quan executem la funció
13     echo $resultat
14 }
15
16  sumar $suma12 $suma23 $suma31
```

PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN

vicent@vicent-MS-7C84:~/Visual Code\$ source test.sh 3 2 3
Els valors que arribem per paràmetre són: 3, 2 i 3
Els valors que han entrat a la funció sumar són: 5, 5 i 6
16
vicent@vicent-MS-7C84:~/Visual Code\$

2 ESTRUCTURES CONDICIONALS

2.1 CONDICIÓN SIMPLE (IF)

La principal estructura que utilitzem als Scripts és el famós “if, elif, else”, es tracta d’una de les estructures més utilitzades a la programació de Scripts, per això cal que quede molt clar el seu funcionament, esta es la que ens fa la funció de comparar diferents valors, ja siguin numèrics o cadenes (paraules, frases, lletres).

Donats uns certs valors que volem comparar, es fa una comparació, si aquesta resulta ser certa, executa el codi que té assignat, en cas contrari revisa si hi ha més casos.

El “if” (sí...) és la condició mínima, totes les estructures condicionals comencen amb “if”, este ens permet comprovar una condició, en cas de ser certa s’executa el contingut que hi ha rere el “then” i acaba obligatòriament amb un “fi”.

```
1  #!/bin/bash
2  num=$1
3  if [ $num -lt 5 ]; then
4      echo "Estàs suspès"
5  fi
```

PROBLEMS OUTPUT TERMINAL CONSOLA DE DEPURACIÓN

```
vicent@vicent-MS-7C84:~$ source if.sh 4
Estàs suspès
vicent@vicent-MS-7C84:~$ ~
```

Compte amb els espais en blanc quan utilitzem esta sintaxi, ja que si no respectem els espais en blanc, pot donar errors que de vegades ens fan perdre hores buscant i revisant el codi. Els exemples que tractem al principi són molt curts, però un Script real pot tindre centenars de línies.

2.2 ELIF

Si es dona el cas on volem comprovar més d'una opció, podem utilitzar **"elif"** (en altre cas), si la condició **"if"** és falsa passa al següent **"elif"**. Poden haver tants **"elif"** com vullgam a un Script. Per exemple:

```
1  #!/bin/bash
2  num=$1
3  if [ $num -lt 5 ]; then
4      echo "Estàs suspès"
5  elif [ $num -ge 9 ]; then
6      echo "Excel·lent"
7  elif [ $num -gt 6 ]; then
8      echo "Ben fet"
9  fi
```

PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPU

```
vicent@vicent-MS-7C84:~$ source if.sh 4
Estàs suspès
vicent@vicent-MS-7C84:~$ source if.sh 7
Ben fet
vicent@vicent-MS-7C84:~$
```

"Elif" segueix la mateixa estructura que **"if"**, després de la condició cal un **"then"**, a més tenir en compte que el **"fi"** solament es troba al final de la condició, si el col·loquem abans provocaria un error de sintaxi.

2.3 ELSE

En cas de voler una resposta per a un cas que no es dona a ninguna de les condicions anteriors fem ús del **"else"** (en cap dels casos). Com per exemple

```
#!/bin/bash
num=$1
if [ $num -lt 5 ]; then
    echo "Estàs suspès"
elif [ $num -ge 9 ]; then
    echo "Excel·lent"
elif [ $num -gt 6 ]; then
    echo "Ben fet"
else
    echo "Has aprovat l'examen"
fi
```

Esta condició segueix la següent estructura i deu de complir eixe ordre:

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

```
if condició #PRIMERA COMPROVACIÓ
```

```
then
```

```
    ordres o accions
```

```
elif condició #EN CAS QUE LA PRIMERA NO SIGA VERTADERA, 2ON COMPROVACIÓ
```

```
then
```

```
    més ordres
```

```
else #SI CAP DE LES 2 ANTERIORS ÉS VERTADERA ENTRA AL ELSE
```

```
    més ordres
```

```
fi
```

Per què ens aprofita este tipus d'estructura? Per què al Script pugam decidir fer una acció o un altra depenent del valor o la condició resultant d'algun tipus d'operació al moment.

TAMBÉ TENIM L'OPCIÓ DE FER-HO TOT A UNA LÍNIA

```
if condició; then
```

```
    ordres o accions
```

```
elif condició; then
```

```
    més ordres
```

```
else
```

```
    més ordres
```

```
fi
```

2.4 OPERADORS

Tenim diferents operadors per analitzar aquests valors, per a les cadenes utilitzem:

Operadors condicionals per tractar cadenes alfanumèriques	
v1 = v2	Vertader si les dos cadenes són exactament iguals
v1 != v2	Vertader si les dos cadenes no són iguals
v1 < v2	Si el valor de la cadena v1 és menor que el de v2
v1 > v2	Si el valor de la cadena v1 és major que el de v2
-n v1	Vertader si la cadena no és nula
-z v1	Vertader si la cadena és nula

Per les operacions amb nombres utilitzem les següents:

Operadors condicionals per tractar valors numèrics	
v1 -eq v2	Vertader si els dos nombres són exactament iguals (eq = equal)
v1 -ne v2	Vertader si els dos nombres no són iguals (ne = not equal)
v1 -lt v2	Si el valor del nombre v1 és menor que el de v2 (lt = less than)
v1 -gt v2	Si el valor del nombre v1 és major que el de v2 (gt = greater than)
v1 -le v2	Si el valor del nombre v1 és menor o igual que el de v2 (le = less or equal)
v1 -ge v2	Si el valor del nombre v1 és major o igual que el de v2 (ge = great or equal)

Per condicions amb fitxers i directoris:

Operadors condicionals especials	
-a fitxer	Vertader si el fitxer existeix
-d directori	Vertader si el directori existeix
-f fitxer	Vertader si el fitxer regular existeix (fitxers de imatge, vídeo, etc)
-r fitxer	Vertader si el fitxer té permisos de lectura
-s fitxer	Vertader si el fitxer no està buit

-w fitxer	Vertader si el fitxer té permisos de escriptura
-x fitxer	Vertader si el fitxer té permisos d'execució
-z variable	Comprova si la variable té cap contingut
fitxer1 -nt fitxer2	Vertader si el fitxer1 és més nou que el fitxer2
fitxer1 -ot fitxer2	Vertader si el fitxer 1 és més vell que el fitxer2

Uns exemples perquè quede més clar:

```
#!/bin/bash
num1=5
num2=15

if [ $num1 -lt $num2 ]; then #EN PRIMER CAS COMPROVE SI ÉS MENOR
    echo '$num1 és més menut que $num2'
elif [ $num1 -gt $num2 ]; then #EN SEGON CAS COMPROVE SI ÉS MAJOR
    echo '$num1 és més gran que $num2'
else #EN QUALEVOL ALTRE CAS
    echo '$num1 i $num2 són iguals'
fi
```

En aquest cas, tenint en compte que el valor de \$num1 és 5, es dona com a certa la primera condició, per tant el que obtenim al intèrpret d'ordres és:

```
vicent@vicent-MS-7C84:~$ source condicions.sh
$num1 és més menut que $num2
```

Si posem un cas per comparar cadenes, podem provar el següent:

```
#!/bin/bash
dia=$(date +%A)

if [ $dia = "viernes" ]; then #EN PRIMER CAS COMPROVE SI ÉS DIVENDRES
    echo 'De luxe, ja és divendres'
elif [ $dia = "lunes" ]; then #EN SEGON CAS COMPROVE SI ÉS dilluns
    echo 'És el primer dia de la setmana encara...'
else #EN QUALEVOL ALTRE CAS
    echo 'Encara no és divendres'
fi
```

On el resultat és (coincideix que el dia d'esta prova era dilluns):

```
vicent@vicent-MS-7C84:~$ source condicions.sh
És el primer dia de la setmana encara...
```

COMPTE, NO UTILITZAR ELS OPERADORS DE COMPARACIÓ NUMÈRICS AMB CADENES, JA QUE EL

SISTEMA ENS TORNARÀ UN ERROR GREU. ACÍ UN EXEMPLE:

```
vicent@vicent-MS-7C84:~$ source condicions.sh
bash: [: lunes: se esperaba una expresión entera
bash: [: lunes: se esperaba una expresión entera
Encara no és divendres
```

Ara, arribat a este punt, podem comparar tot tipus de valor, ja siga cadena o nombres, però si ens donem compte, solament estem parlant de un valor per cada “if” o “elif”.

Realment és el màxim que podem arribar a fer? NO. Podem encadenar condicions amb les expressions && (and) , || (or) i ! (not).

Bàsicament, per no liar-se, el && ve a ser un “i”, el || un “o” i el ! el contrari del que siga, si es vertader, és fals, si “1 -lt 3” seria vertader, si utilitzem ! seria fals.

Un exemple amb els operadors especials podria ser:

```
1  #!/bin/bash
2
3  if ! [ -d /media/vicent/Seagate/temporal ]; then
4      mkdir /media/vicent/Seagate/temporal
5  else
6      echo "Ja existeix aquest directori"
7  fi
```

PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN

```
vicent@vicent-MS-7C84:~$ source for3.sh
vicent@vicent-MS-7C84:~$ source for3.sh
Ja existeix aquest directori
vicent@vicent-MS-7C84:~$
```

Utilitze el “!” indicant que volem la negació, comprove si existeix el directori “/media/vicent/Seagate/temporal”, en cas de no existir executarà el mkdir, en cas d’existir (com és veu a la 2on execució) mostrarà el missatge.

De la mateixa forma que utilitzem el "if" per comparar un únic valor, anem a veure com podem fer-ho amb dos valors, per a l'exemple de la nota d'examen:

```
1  #!/bin/bash
2  num=$1
3  if [ $num -ge 9 ]; then
4      echo "Enhorabona, ben treballat"
5  elif [ $num -gt 7 ] && [ $num -lt 9 ]; then
6      echo "Ben fet"
7  elif [ $num -ge 5 ] && [ $num -le 7 ]; then
8      echo "Molt bé, però millorable"
9  elif ! [ $num -ge 5 ]; then
10     echo "Has suspès l'examen"
11 fi
```

I un altre exemple on podem fer ús del || (or) , seria en cas que acceptarem 2 opcions diferents per una mateixa resposta:

```
#!/bin/bash
nom=$1
if [ $nom = "Carla" ] || [ $nom = "carla" ]; then
    echo "Benvinguda Carla"
elif [ $nom = "Joan" ] || [ $nom = "Carlos" ]; then
    echo "Benvingut"
else
    echo "Persona no identificada"
fi
```

2.5 CONDICIONS NIUADES

Que són les condicions niuades? (En castellà, “anidadas”) Es tracta de una condició que en cas de ser certa, conté dins d’ella més condicions possibles. **Arribats a aquest punt, és molt important mantenir una estructura correcta dins els Scripts per no tornar-se boig.** Que vull dir amb això, tabular molt bé cadascuna de les condicions perquè quede molt clar si estem a una condició niuada o a la principal.

Per exemple, seguint l’exemple anterior dels noms, afegint una condició niuada al primer “if”:

```
#!/bin/bash
nom=$1
cognom=$2

if [ $nom = "Carla" ] || [ $nom = "carla" ]; then
if [ $cognom = "Bañon" ]; then
echo "Benvinguda Carla"
else
echo "Persona no identificada"
fi
elif [ $nom = "Joan" ] || [ $nom = "Carlos" ]; then
echo "Benvingut"
else
echo "Persona no identificada"
fi
```

Com podeu observar, no és el millor exemple per comprendre que està passant al Script, no és gens intuïtiu, en canvi, si li donem el format correcte mitjançant les tabulacions... :

```
#!/bin/bash
nom=$1
cognom=$2
if [ $nom = "Carla" ] || [ $nom = "carla" ]; then
|   if [ $cognom = "Bañon" ]; then
|       echo "Benvinguda Carla"
|   else
|       echo "Persona no identificada"
|   fi
elif [ $nom = "Joan" ] || [ $nom = "Carlos" ]; then
|   echo "Benvingut"
else
|   echo "Persona no identificada"
fi
```

L’actuació és: si el nom és Carla o carla entrarà a la condició, i passarà a revisar si el cognom coincideix.

Queda molt més fàcil d’entendre tant per al desenvolupador com per al company que ho pot revisar en un futur.

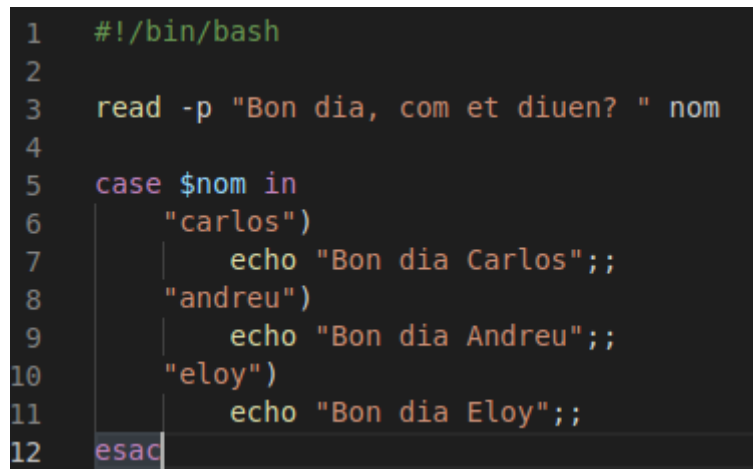
2.6 CASE

A més del “if/elif/else” disposem d’un altra ferramenta més còmoda per a situacions on ens donen moltíssimes possibilitats diferents, es tracta del “case”, aquesta estructura ens permet executar igual que el “if” unes ordres si la condició que es proposa és certa. Segueix esta estructura:

```
#!/bin/bash

case VARIABLE in
    valor1)
        echo “el que siga”
        ;;
    valor2)
        echo “altra cosa”
        ;;
esac #Per tancar el case (igual que el if -> fi)
```

Un exemple, per validar el nom introduït:



```
1  #!/bin/bash
2
3  read -p "Bon dia, com et diuen? " nom
4
5  case $nom in
6      "carlos")
7          echo "Bon dia Carlos";;
8      "andreu")
9          echo "Bon dia Andreu";;
10     "eloy")
11         echo "Bon dia Eloy";;
12 esac
```

Com podeu observar, no és necessari posar els “;;” a la línia de baix, poden anar al final del que vullgam executar.

A més, per als diferents casos també podem fer ús dels caràcters especials com “*” o “?” per substituir amb elements que desconeguem, per exemple:

```
1  #!/bin/bash
2
3  read -p "Bon dia, com et diuen? " nom
4
5  case $nom in
6      carl*)
7          echo "Cas amb asterisc";;
8      andre?)
9          echo "Cas amb interrogant";;
10     ?ar*)
11         echo "Cas amb asteris i interrogant";;
12     esac
```

PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN

```
vicent@vicent-MS-7C84:~$ source if.sh
Bon dia, com et diuen? carlos
Cas amb asterisc
vicent@vicent-MS-7C84:~$ source if.sh
Bon dia, com et diuen? carles
Cas amb asterisc
vicent@vicent-MS-7C84:~$ source if.sh
Bon dia, com et diuen? andreu
Cas amb interrogant
vicent@vicent-MS-7C84:~$ source if.sh
Bon dia, com et diuen? andrea
Cas amb interrogant
vicent@vicent-MS-7C84:~$ source if.sh
Bon dia, com et diuen? carmen
Cas amb asteris i interrogant
vicent@vicent-MS-7C84:~$ source if.sh
Bon dia, com et diuen? carmela
Cas amb asteris i interrogant
vicent@vicent-MS-7C84:~$
```

També un cas per detectar una vocal, o qualsevol idea que pugau interpretar amb els operadors :

```
1  #!/bin/bash
2
3  read -p "Introdueix una lletra? " lletra
4
5  case $lletra in
6      a|e|i|o|u)
7          echo "Eixa lletra és una vocal";;
8      *)
9          echo "No és una vocal";;
10     esac
```

PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN

```
vicent@vicent-MS-7C84:~$ ./if.sh
Introdueix una lletra? o
Eixa lletra és una vocal
vicent@vicent-MS-7C84:~$ ./if.sh
Introdueix una lletra? andorra
No és una vocal
```

També es poden indicar rangs de nombres i lletres de la següent forma:

```
$ test.sh
1  #!/bin/bash
2  read -p "nombre: " val
3  echo $val
4  case $val in
5      [0-9]) echo "De 0 a 9";;
6      [1][0-9]) echo "De 10 a 19";;
7      [2][0-9]) echo "De 20 a 29";;
8      [a-z]) echo "minúscules";;
9      [A-Z]) echo "majúscules";;
10 esac
11
```

A més, totes les combinacions que podem utilitzar al terminal, com: **[0-9] | 19 | 29** -> De 0 a 9, o 19 o 29.

O bé per a qualsevol element, a l'igual que al terminal fem ús del **asterisc ***

3 DEBUGGING

Concepte de debugging (depuració del progrma):

La depuració de programes és el procés d'identificar i corregir errors de programació. En anglès es coneix com debugging, perquè s'assembla a l'eliminació d'insectes (bugs), manera en què es coneix informalment als errors de programació.

Tot i que als Scripts no podem fer un “debug” com a la gran majoria de llenguatges de programació, tenim una opció que ens permet mostrar els passos que fa el Script d'un en un, mostrant els valors de les variables i els possibles camins (en cas d'existir condicions) que ha triat.

Per això únicament cal afegir al “SHEBANG” el valor “-x”: `#!/bin/bash -x`

D'esta forma quan executem el Script, abans de veure els resultats per l'eixida estàndard, apareixerà el recorregut que ha fet i els valors de les variables, exemple dels noms i cognoms:

```
vicent@vicent-MS-7C84:~$ ./if.sh carla Perez
+ nom=carla
+ cognom=Perez
+ '[' carla = Carla ']'
+ '[' carla = carla ']'
+ '[' Perez = Bañon ']'
+ echo 'Persona no identificada'
Persona no identificada
vicent@vicent-MS-7C84:~$
```

Recomane encaridament l'ús d'aquesta opció si vos trobeu bloquejats a cap Script que no actua com esperaveu, pot estalviar-vos molt de temps.

4 BUCLES

Unes de les principals estructures iteratives que podem utilitzar dins dels Script són el for, while, until i select. Estos ens permeten repetir de forma llimitada una serie d'ordres per poder atendre múltiples respostes a un únic Script.

4.1 FOR

El cas for és un dels bucles que itera tantes vegades com element o conjunts d'elements existisquen, per ell, interpreta tants elements apareguen separats per espais en buit. L'estructura és la següent:

```
#!/bin/bash

for VARIABLE in VALOR VALOR VALOR; do
    echo $VARIABLE #Conté un valor diferent per cada iteració
done
```

Per començar, fem un exemple amb nombres i així podem veure com actua el bucle:

```
1  #!/bin/bash
2
3  echo "Els dies de la setmana són: "
4  for nombre in 1 2 3 4 5 6 7 8 9 10; do
5      echo "La variable nombre conté el valor $nombre"
6  done
```

PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN

```
vicent@vicent-MS-7C84:~$ ./if.sh
Els dies de la setmana són:
La variable nombre conté el valor 1
La variable nombre conté el valor 2
La variable nombre conté el valor 3
La variable nombre conté el valor 4
La variable nombre conté el valor 5
La variable nombre conté el valor 6
La variable nombre conté el valor 7
La variable nombre conté el valor 8
La variable nombre conté el valor 9
La variable nombre conté el valor 10
```

Com que hem introduït 10 elements, el bucle es repeteix 10 vegades, i la variable \$nombre cada iteració conté un valor diferent (en ordre de com estan introduïts).

Suposem el cas de voler imprimir tots els dies que formen la setmana, seguiria este format:

```
1  #!/bin/bash
2
3  echo "Els dies de la setmana són: "
4  for dies in dilluns dimarts dimecres dijous divendres dissabte diumenge; do
5  |   echo $dies
6  done
```

PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN

```
vicent@vicent-MS-7C84:~$ ./if.sh
Els dies de la setmana són:
dilluns
dimarts
dimecres
dijous
divendres
dissabte
diumenge
```

Per cada element que hi ha després del “in” executa el “echo” amb la variable \$dies, es repetirà tantes vegades com dies hi hagen, i per cada iteració la variables \$dies té un valor.

Anem a donar-li un poc més de complexitat al bucle, en compte de agafar valors que hem de indicar a mà, anem a utilitzar un llistat de fitxers. Primer que res, tenim un document que es diu “dades” amb aquest contingut:

```
vicent@vicent-MS-7C84:~$ cat dades
hamburguesa
cafe
amanida
creilles
vaca
bou
cavall
tomaca
pebrot
cogombrets
```

Ara, el que volem fer es utilitzar els diferents elements del document, com a entrades per al bucle. Com podem fer això? Si recordem la teoria de la part 1, sabem que podem volcar dins una variable el contingut d'un fitxer, en aquest cas, com que el bucle actua igual que si volcarem les dades dins una variable, podem indicar que els elements són la resposta de l'execució d'un comandament.

Que vull dir amb això, si observeu el resultat del “cat anterior”, envia a l'eixida estàndard cada un d'ells, aquesta eixida ens aprofita per al bucle, per tant podem crear el nostre bucle for amb les dades obtingudes per l'execució d'una ordre.

Ací l'exemple:

```
1  #!/bin/bash
2  echo "Per preparar el menú del restaurant necessitem: "
3  for valors in $( cat dades ); do
4  |   echo $valors
5  done
6
```

PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN

vicent@vicent-MS-7C84:~\$./fordades.sh
Per preparar el menú del restaurant necessitem:
hamburguesa
cafe
amanida
creilles
vaca
bou
cavall
tomaca
pebrot
coqombrets

O bé, imagineu que el que volem és que el Script ens mostre per pantalla tots els elements que tenim a un directori específic, per exemple a /bin . Podem fer ús del mateix sistema:

```
1  #!/bin/bash
2  echo "Dins del directori /bin podem trobar: "
3  for valors in $( ls /bin ); do
4  |   echo $valors
5  done
6
```

PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN

zoom
vicent@vicent-MS-7C84:~\$ ^C
vicent@vicent-MS-7C84:~\$./fordades.sh | more
Dins del directori /bin podem trobar:
[
2to3-2.7
4lltoppm
7z
7za
7zr
aa-enabled
aa-exec

La resposta del comandament “ls” l’utilitzem per introduir valors al bucle, com és lògic la resposta dels elements era molt més llarga, he retalla la imatge.

A més, tenim una ordre a Linux que és “seq” la què ens permet generar automàticament un nombre de nombres de forma automàtica. La forma de treballar i les diferents opcions les podeu consultar a “man seq”

a un terminal.

Bàsicament, permet generar nombres d'un punt inici a un punt final amb increments d'un en un, o bé amb el que nosaltres decidim:

Imprimirà per l'eixida estàndard des del nombre 1 fins al 100:

```
vicent@vicent-MS-7C84:~$ seq 1 100
```

Imprimirà del nombre 1 fins al 100 amb un increment de 5:

```
vicent@vicent-MS-7C84:~$ seq 1 5 100
```

*Per tant, podem fer: **for NOMBRE in \$(seq 1 100); do***

Altra forma de indicar un rang de nombres és amb les claus:

Format: {VALORINICIAL..VALORFINAL} o bé {VALORINICIAL..VALORFINAL..INCREMENT}

*Una forma simple és: **for NOMBRE in {1..10}; do** anirà des de l'1 fins al 10*

*Exemple: **for NOMBRE in {1..50..5}; do** Aquest actua com el "seq" però l'increment que volem va al final, farà del nombre 1 fins al 50 amb increment de 5.*

El bucle "FOR" ens permet utilitzar un altra estructura que és compartida amb els llenguatges de programació com C, Java i similars. Es tracta de dividir les expressions en 3 parts:

```
for (( variableInicial; condicio; expressiodecanvi ))
```

o millor

```
for (( contador=0; contador < 10; contador=contador+1 ))
```

Contador té el valor 0, mentre contador siga menor que 10, executarà el contingut del "for" i després incrementarà el valor de \$contador

Cal tindre en compte que si fem ús d'aquest mode, la condició per comprobar si és major, mejor, etc no seran els operadors numèrics, sinó els alfanumèrics (<, >, <=, >=, =, !=)

Després, moltes vegades podem caure en bucles infinits per diferents motius, com que mai arriba a donar-se la condició necessària per parar romanen fins que l'interrompem. L'única forma que tenim per cancel·lar l'execució és amb la drecera "Ctrl+C".

També podem fer utilització de la funció "**break**" si el que volem és parar el script de forma manual, si en cap moment del script desitgem parar-lo podem fer una condició on s'executa un "break" i este es para

instantàniament. Esta forma no està ben vista, ja que es tracta d'una para "brusca".

Per exemple, l'ús del break i un bucle infinit en un cas un poc més realista de programa:

```
#!/bin/bash
function menu {
    echo "1. Compra un vol"
    echo "2. Revisa el teu vol"
    echo "3. Cancel·la el teu vol"
    echo "4. Veure l'estat de l'espai aeri"
    echo "5. Ofertes especials d'última hora"
    echo "6. Mostrar el menú"
    echo "0. Per eixir"
    #LA FUNCIO MENU IMPRIMEIX PER PANTALL EL LLISTAT D'OPCIONS
}

echo "Benvinguts al script de cerca de vols"

menu #EXECUTEM LA FUNCIO LA PRIMERA VEGADA QUE ARRANCA EL SCRIPT PER MOSTRAR LES OPCIONS

for (( ; ; )); do #AQUESTA ES UNA FORMA DE DECLARAR UN BUCLE INFINIT
    read -p "Introdueix el nombre del 1 a 6 per accedir al menú (0 per eixir): " nombre
    #EL BUCLE PREGUNTARÀ DE FORMA ININTERROMPUDA CANVIANT EL VALOR DE $nombre
    case $nombre in
        1) echo "Per comprar un vol entra directament a la web";;
        2) echo "Per revisar el teu vol entra directament a la web";;
        3) echo "Per cancel·lar el teu vol, entra a la web";;
        4) echo "Per veure l'estat aeri, revisa https://www.seguridadaerea.gob.es";;
        5) echo "Per revisar les ofertes del moment, entra a la web";;
        6) menu;;
        0) break;; #EN CAS DE TRIAR EL 0 ES PARA EL SCRIPT
    esac
done
```

4.2 WHILE (MENTRE) I UNTIL (FINS QUE)

Quan no volem recorre un conjunt d'elements (com al for), ens val amb repetir indefinidament fins que es complisca una condició o fins que deixi de complir-se, podem fer ús del "while" i "until".

L'estructura d'aquest dos és esta:

```
while [ expressió ]; do
    estes línies es repetiran fins que l'expressió canvie
done

until [ expressió ]; do
    estas línies es repetiran fins que l'expressió canvie
```

done

Ambdós fan exactament el mateix, repetir-se fins que canvie l'expressió, cada iteració del bucle comprova si l'expressió és certa, si ho és, torna a executar el codi. Ací uns exemples:

```
#!/bin/bash
read -p "Introdueix un nombre: (0 per eixir) " nombre
while [ $nombre -ne 0 ]; do
    echo "El doble de $nombre és:" $((nombre*2))
    read -p "Introdueix un nombre: (0 per eixir) " nombre
done

echo
echo "Entrem al until"
echo

read -p "Introdueix un nombre: (0 per eixir) " nombre
until [ $nombre -eq 0 ]; do
    echo "El triple de $nombre és:" $((nombre*3))
    read -p "Introdueix un nombre: (0 per eixir) " nombre
done
```

Al “while” mentre nombre no tinga valor 0 continuarà rodant el bucle, en canvi al “until” fins que nombre no valga 0 (que és el mateix que al while) continuarà rodant.

4.3 SELECT

Esta és l'última estructura iterativa que anem a veure, el que ens permet el select és fer un bucle però al més estil menú. Que vull dir amb això? El que es repeteix és el menú. L'estructura és la següent:

```
select VARIABLE in options; do
```

Ací la variable té un valor de les opcions

```
done
```

Aquesta estructura és molt similar a la del “for”, però la diferència és que no hi ha una condició que pare la iteració, l'única opció que tenim és fer ús del “break” o bé de “exit” per parar el script sencer. Un exemple

paregut a l'anterior del menú dels vols:

```
#!/bin/bash

select opcio in comprar revisar cancelar veure eixir; do
    case $opcio in
        comprar)
            echo "Compra un vol";;
        revisar)
            echo "Revisa el teu vol";;
        cancelar)
            echo "Cancela el teu vol";;
        veure)
            echo "Veure l'estat de l'espai aeri";;
        eixir)
            break
    esac
done
```

Un altre exemple, pot ser un menú dins un altre menú, en aquest cas llista tots els fitxers del directori on s'executa el Script i després mostra diferents opcions a fer amb els fitxers disponibles:

```
#!/bin/bash
select fitxer in $( ls ); do
    echo "Has seleccionado el fichero $fitxer"
    select opciones in vore borrar copiar cancelar; do
        case $opcions in
            vore)
                cat $fitxer
                break;;
            borrar)
                rm $fitxer
                break;;
            copiar)
                read -p "On vols moure el fitxer? (ruta completa) " ruta
                mv $PWD/$fitxer $ruta
                echo "Fitxer $fitxer mogut a $ruta"
                break;;
            cancelar)
                break;;
        esac
    done
done
```

5 VECTORS (ARRAY)

Per acabar amb els Script, cal entendre que a més de tractar variables, conjunts de valors, condicions i expressions, també podem treballar amb vectors per emmagatzemar dades. Un vector al cap i a la fi no és més que una variable però que conté X nombre d'elements i té tantes posicions com elements conté el

vector.

Per declarar un vector podem fer-ho amb:

declare -a nomVariable

L'estructura d'un vector no és més que una variable acompanyada d'un claudàtor on s'indica la posició on volem accedir:

vector[0] -> Posició 0 del vector, la primera posició sempre

vector[1] -> Posició 1 del vector

vector[2] -> Posició 2 del vector

Un vector pot ser inicialitzat mitjançant l'assignació d'un valor, un conjunt de valors o la lectura (read), per exemple:

vector[2]=0

vector=(Element1 Element2 Element3)

read a[2] -> Esta forma assignarà el primer valor que introduïm a la posició 2

Per recollir els valors de l'eixida d'una ordre canvia, no és com una variable estàndard, cal utilitzar el signe d'accentuació obert per rodejar l'ordre, i redirigir a una variable, exemple:

vector=(`cat fitxer.txt`)

Un exemple del vector, per fer-nos una imatge mental, es tracta d'un llistat d'elements ordenats per posició, imaginem una variable que conté l'edat dels membres d'un club de golf. La variable es diu "edatMembres":

Contingut	32	19	32	34	51	63	20
Posició	0	1	2	3	4	5	6

És el mateix que dir:

- edatMembres[0]=32
- edatMembres[1]=19
- edatMembres[2]=32
- edatMembres[3]=34
- edatMembres[4]=51
- edatMembres[5]=63
- edatMembres[6]=20

Compte, per fer ús dels valors que conté un vector és necessari accedir als elements amb `${ }`, en cas de no fer-ho, interpretarà el la variable com una variable estàndard i mostrarà únicament el primer valor del vector. Si intentem accedir al valor de `$vector[0]`, interpretarà la variable com dos elements diferents, “`$vector`” com una variable estàndard que mostrarà el primer valor del vector i “[0]” com un altre element contigu.

L'accés correcte és `${vector[0]}`. Ací un exemple:

```
1  #!/bin/bash
2  vector[0]="Element 0"
3  vector[1]="Element 1"
4  echo $vector
5  echo $vector[0]
6  echo ${vector[0]}
```

PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURA

```
vicent@vicent-MS-7C84:~$ source vectors.sh
Element 0
Element 0[0]
Element 0
```

Un exemple:

```
1  #!/bin/bash
2  vector[0]="Element 0"
3  vector[1]="Element 1"
4  vector[2]="Element 2"
5
6  read -p "Introdueix un valor: " vector[3]
7
8  echo ${vector[0]}
9  echo ${vector[1]}
10 echo ${vector[2]}
11 echo ${vector[3]}
12
```

PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN

```
vicent@vicent-MS-7C84:~$ source vectors.sh
Introdueix un valor: 12324
Element 0
Element 1
Element 2
12324
vicent@vicent-MS-7C84:~$ source vectors.sh
Introdueix un valor: 3123
Element 0
Element 1
Element 2
3123
```

En un possible cas, que volem omplir un vector amb l'ordre "read" no és necessari fer un "read" per línia:

read	a[0]
read	a[1]
read a[2]	

NO

Podem fer ús de l'opció "-a" que permet introduir tots els elements introduïts al "read" amb una única ordre com a elements d'un vector:

```

1  #!/bin/bash
2  echo "Introdueix els elements separats per espais en blanc: "
3  read -a vector
4
5  echo "Elements del vector: "
6  echo ${vector[0]}
7  echo ${vector[1]}
8  echo ${vector[2]}

```

PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN

```

vicent@vicent-MS-7C84:~$ source vectors.sh
Introdueix els elements separats per espais en blanc:
primer segon tercer
Elements del vector:
primer
segon
tercer

```

Per comprovar la grandària o el nombre d'elements que conté un vector, tenim la característica especial:

`${#variable[@]}` -> Ens retorna un nombre

Per revisar tots els elements que conté un vector amb una única instrucció tenim la característica:

`${variable[@]}`

Ací una prova:

```

vicent@vicent-MS-7C84:~$ variable=( andres jose paco josep )
vicent@vicent-MS-7C84:~$ echo ${variable[@]}
andres jose paco josep
vicent@vicent-MS-7C84:~$ echo ${#variable[@]}
4
vicent@vicent-MS-7C84:~$ █

```

Per què ens val això? Podem aprofitar tant els elements com el nombre d'elements per fer un bucle amb "for" (per exemple) i recorre cadascuna de les posicions del vector. Un exemple utilitzant cada element del vector:

```
1  #!/bin/bash
2  echo "Introdueix els elements separats per espais en blanc: "
3  read -a vector
4
5  for element in ${vector[@]}; do
6      echo "Aquest element té un valor de: " $element
7  done
8
```

PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN

```
vicent@vicent-MS-7C84:~$ source vectors.sh
Introdueix els elements separats per espais en blanc:
josep andrea carla antoni
Aquest element té un valor de:  josep
Aquest element té un valor de:  andrea
Aquest element té un valor de:  carla
Aquest element té un valor de:  antoni
```

Un altre exemple utilitzant el nombre d'elements que hi han:

```
1  #!/bin/bash
2  echo "Introdueix els elements separats per espais en blanc: "
3  read -a vector
4
5  #echo  ${ seq 0 ${#vector[@]} }
6  for element in $( seq 0 $(( ${#vector[@]} - 1 )) ); do
7      echo "Aquest element té un valor de: " ${vector[$element]}
8  done
9
```

PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN

```
vicent@vicent-MS-7C84:~$ source vectors.sh
Introdueix els elements separats per espais en blanc:
Andreu Arnau Eloy Carlos Ian
Aquest element té un valor de:  Andreu
Aquest element té un valor de:  Arnau
Aquest element té un valor de:  Eloy
Aquest element té un valor de:  Carlos
Aquest element té un valor de:  Ian
```

Amb això donem per completat el Tema 1 - Part 2, amb els coneixements bàsics necessaris per la programació amb Shell Script. Ara sols queda per la vostra part completar les diferents activitats que aniré deixant-vos a AULES.

2. BIBLIOGRAFIA

Temari original baix llicència Creative Commons Reconeixement-NoComercial-CompartirIgual 4.0 Internacional:

Vicent Benavent

Javier Martínez (IES María Enríquez)



**Reconocimiento-NoComercial-
CompartirIgual 4.0 Internacional
(CC BY-NC-SA 4.0)**