

ADMINISTRACIÓ DE SISTEMES OPERATIUS

U1: SHELL SCRIPT, PART 1

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 INTRODUCCIÓ A SHELL SCRIPT

En el següent tema vorem una introducció bàsica a Scripts, per això són necessaris conceptes bàsics sobre les ordres més bàsiques a un terminal. Podem considerar els Script com una llista d'ordres Unix agrupats en un mateix fitxer amb una finalitat específica. Ens serveixen per automatitzar tasques i realitzar execucions estàtiques.

Això pot aplicar-se a tasques de manteniment, còpies de seguretat manuals, automatització d'execucions, entre d'altres. La finalitat d'el Script és permetre a l'administrador crear ordres més complexes a partir de les ordres bàsiques incorporades al sistema.

Per treballar amb Scripts, sols necessitem un intèrpret d'ordres (també conegut en anglés com CLI), i un editor de text, el que més vos agrade.

Cal dir que existeixen diferents intèrprets d'ordres, la gran majoria d'ells són evolucions de l'anterior amb millores en quant a funcions, entre els més famosos i utilitzats està:

- sh (shell)
- csh (shel)
- ksh (Korn Shell)
- bash (Bourne Again Shell, derivat de shell)
- I molts més...

Els més utilitzats al món laboral són per essència "bash" i "sh", "bash" per ser una de les últimes versions i junt amb ell totes les funcionalitats que aquest aporta i "sh" com que no té tantes utilitats és molt lleuger i s'utilitza a molts servidors sense entorn gràfic i contenidors (docker).

2 EL PRIMER SCRIPT

Els Scripts no són més que fitxers de text, que poden modificar-se amb qualsevol editor de text, tant a sistemes Linux (vi, vim, nano, etc), com a Windows (Editor de texto, Notepad++, etc). Per començar, necessitem una màquina pot ser física o virtualitzada amb un sistema Linux instal·lat (per exemple Ubuntu 20.04), després podem crear un fitxer, amb nom **primer.sh**, i dins amb el següent contingut:

```
#!/bin/bash  
  
echo "Hola mundo"
```

Aquest fitxer amb extensió ".sh" ja pot ser interpretat per la gran majoria de terminals que podem trobar als sistemes Linux.

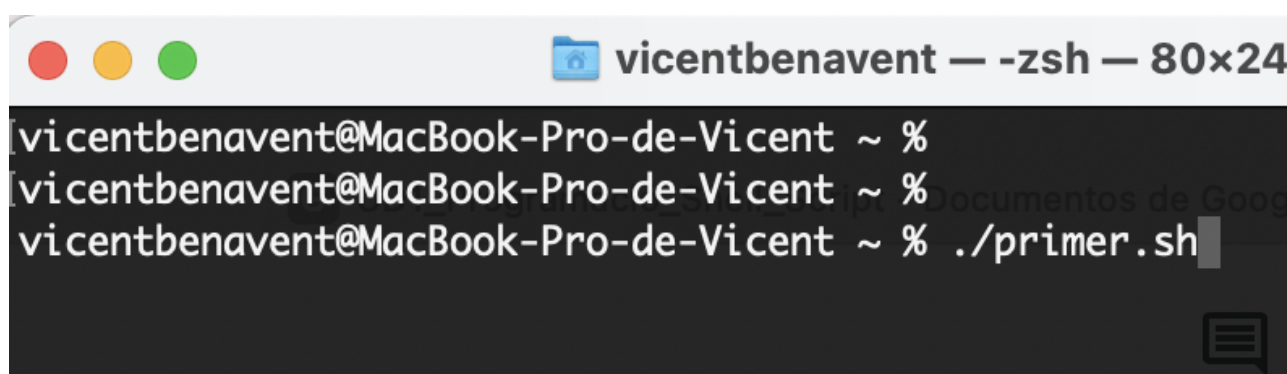
****Tot i que no és necessari donar-li extensió als fitxers en Linux (.sh) als Scripts per costum es manté, però no és necessari.***

Bé, coses a tenir en compte, qué indica la primera línia del fitxer? S'anomena **shebang**, aquest és l'indicador que reconeix el terminal només obrir el fitxer i que li està donant la informació de on es troba (la ruta) l'interpret que volem que s'utilitzi per l'execució del contingut. AQUESTA LÍNIA ÉS SEMPRE LA PRIMERA A TOTS ELS SCRIPTS.

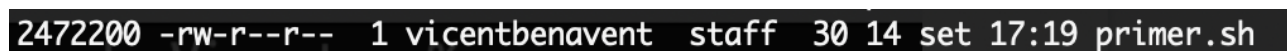
En segon punt, possiblement molts ja l'heu utilitzat, es tracta de l'ordre "echo" que el que fa és mostrar per l'eixida estàndard el contingut d'un vector o fitxer.

Amb això, ja podem executar el Script i veure com es comporta, per executar-lo només cal introduir "./" davant del nom del fitxer. Per què "./"? És per indicar-li que el fitxer que volem executar es troba a la carpeta actual on estem situats, en cas que estiguera a un altre directori no cal "./", amb la ruta completa del fitxer ens val.

Per exemple, el nostre fitxer "**primer.sh**":

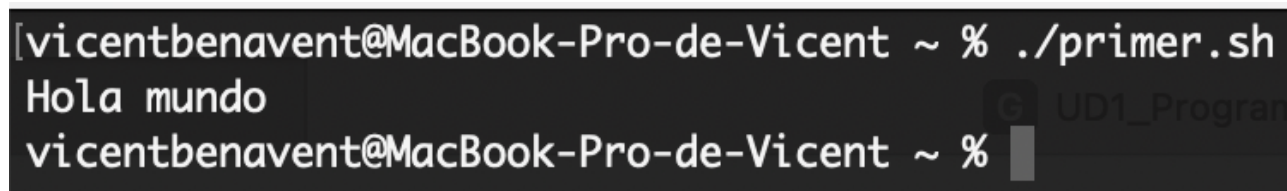


Seguint els passos comentats, intenteu executar el Script *primer.sh*, ha funcionat? Que vos mostra per pantalla? Per defecte, els fitxers creats a Linux tenen permisos "644" o millor dit, lectura i escriptura per al propietari i lectura per al grup i altres.



Per tant, és impossible que s'execute el fitxer, hem de donar-li permisos, mitjançant el mètode tradicional amb l'ordre i els binaris *chmod 755* o bé amb les abreviacions *chmod +x*.

Ara ja deuria funcionar sense cap problema i retornar el següent:



Per tant, queda clar quins són els requisits per poder executar un Script:

- Permisos d'execució per l'usuari que el va a executar
- Un terminal (o bé, a l'entorn gràfic, però no veuríem l'eixida)

Existeix una altra possibilitat per executar Scripts que no tenen permisos d'execució, és tracta de d'utilitzar l'ordre **"source"**, per exemple, al nostre fitxer seria: "source primer.sh". Compte, no utilitzar l'ordre "source" per executar un Script de forma professional, aquest sols es deu utilitzar per fer comprovacions del funcionament i proves. Una vegada el Script estiga complet i vaja a utilitzar-se en un entorn de "producció" cal assignar-li els permisos necessaris.

3 COMETES SIMPLES I DOBLES, VARIABLES, CARÀCTERS ESPECIALS I REDIRECCIÓ

3.1 CARÀCTERS ESPECIALS

Es tracta de diferents elements que podem utilitzar tant al terminal com als Script que permeten treballar més còmodament, els més característics són la **contrabarra "\"** i el **punt i coma ";"**. Estos permeten que el text que escrivim als Script actue de la mateixa forma encara que no tinga una estructura correcta.

Per exemple el punt i coma ";" indica un fi de comandament, d'aquesta forma, a una única línia podem escriure diferents ordres.

En canvi, la contrabarra "\" actua de forma similar a les cometes o cometes dobles, força al següent element a actuar de forma literal, actua com un valor, no com una funció.

En este exemple queda més clar fent ús d'un salt de línia:

```
#!/bin/bash
echo Aquest text apareix primer; pwd; echo Aquest després

echo En canvi, \
    aquest apareixerà\
        a la mateixa línia\
encara que no tinga el format
```

Actua d'aquesta forma, com veieu, el salt de línia és tractat com un literal i no actua com cal.:

```
vicent@vicent-MS-7C84:~$ vim especials.sh
vicent@vicent-MS-7C84:~$ source especials.sh
Aquest text apareix primer
/home/vicent
Aquest després
En canvi, aquest apareixerà a la mateixa líniaencara que no tinga el format
vicent@vicent-MS-7C84:~$
```

Després tenim el coixinet "#" o "almohadilla" en castellà, este caràcter està indicant un comentari, quan

apareix a una línia del Script eixa no serà interpretada quan s'execute, normalment els comentaris s'utilitzen com a guia per als companys de treball, o bé per deixar anotacions de com actua el Script.

```
#!/bin/bash
echo Aquest text apareix primer; pwd; echo Aquest després
#PER EXEMPLE, ACÍ INTRODUÏM UN COMENTARI QUE NO APAREIXERÀ A CAP LLOC
echo En canvi, \
    aquest apareixerà\
        a la mateixa línia\
encara que no tinga el format
```

A més, disposem de una variable que sempre existeix al terminal. Es tracta de "\$?", aquesta variable conté un nombre, o bé és el 0 o bé és l'1. Per què? Doncs este nombre indica si l'última ordre executada s'ha executat correctament o no. En cas de haver anat tot bé valdrà 0, en cas de haver fallat valdrà 1. Exemple:

```
vicent@vicent-MS-7C84:~$ echo $?
0
vicent@vicent-MS-7C84:~$ mv noexisteix Documentos/
mv: no se puede efectuar `stat' sobre 'noexisteix': No existe el archivo o el di
rectorio
vicent@vicent-MS-7C84:~$ echo $?
1
vicent@vicent-MS-7C84:~$ cat if.sh
#!/bin/bash

echo "Els dies de la setmana són: "
for nombre in 1 2 3 4 5 6 7 8 9 10; do
    echo "La variable nombre conté el valor $nombre"
done
vicent@vicent-MS-7C84:~$ echo $?
0
vicent@vicent-MS-7C84:~$
```

3.2 COMETES SIMPLES I DOBLES

Dins dels Scripts, tenim gran varietat de signes per indicar el tipus de contingut, com podeu observar al Script "primer.sh" el contingut que imprimeix per l'eixida estàndard està entrecomillat doble, el que indiquem amb les " " i ' ' és que el contingut entre estes mantindrà el seu valor literal, no s'interpretarà de cap manera. La diferència entre este dos està en que les cometes dobles " " sí que interpreten alguns caràcters especials com és el cas de les variables "\$var"

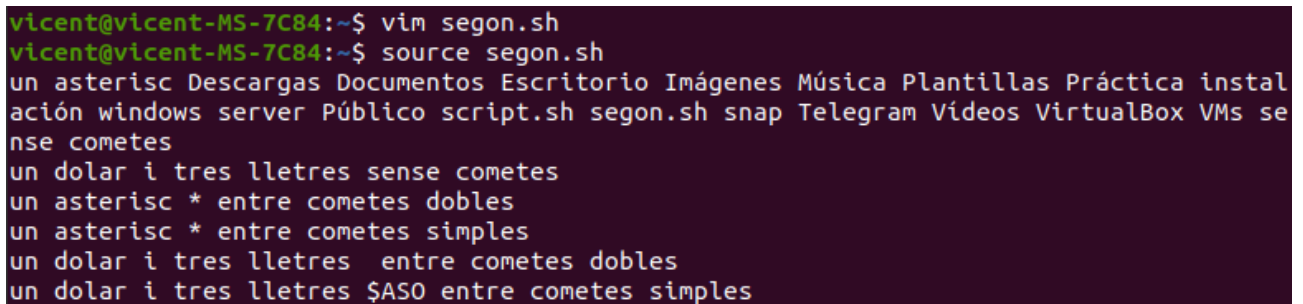
COMPTE: No confondre les cometes simples ' ' amb ' ' ni les dobles " " amb " ", ja que provocarà un error, hem d'utilitzar la topografia on apareixen rectes, no curvades.

Que vol dir això? Entenem per literal a que el contingut no s'interpretarà ni es modificarà, apareixerà exactament igual que està escrit. Res millor per entendre-ho que provar-ho. Creem un nou Script al que nombrarem "segon.sh" amb el següent contingut:

```
#!/bin/bash
echo un asterisc * sense cometes
echo un dolar i tres lletres $ASO sense cometes
echo "un asterisc * entre cometes dobles"
echo 'un asterisc * entre cometes simples'
echo "un dolar i tres lletres $ASO entre cometes dobles"
echo 'un dolar i tres lletres $ASO entre cometes simples'
```

Prova a executar-lo al teu ordinador i analitza com l'interpret bash actua amb les diferents línies del Script.

Aquest deu ser el resultat de l'execució, com podeu veure cada línia actua de forma totalment diferent, i això ja ens dona pistes clares de què fa cadascuna de les cometes.



```
vicent@vicent-MS-7C84:~$ vim segon.sh
vicent@vicent-MS-7C84:~$ source segon.sh
un asterisc Descargas Documentos Escritorio Imágenes Música Plantillas Práctica instal
ación windows server Público script.sh segon.sh snap Telegram Videos VirtualBox VMs se
nse cometes
un dolar i tres lletres sense cometes
un asterisc * entre cometes dobles
un asterisc * entre cometes simples
un dolar i tres lletres entre cometes dobles
un dolar i tres lletres $ASO entre cometes simples
```

- Primer que res, per imprimir text amb l'ordre "echo" no és necessari fer ús de les cometes, però, com es veu, la primera línia està imprimint al terminal la frase que hem escrit amb alguna modificació, ha substituït l'asterisc * i està mostrant tot el contingut que existeix a la carpeta on s'està executant. Per què? Com recordareu, el caràcter * al terminal Linux fa referència a tots els elements que existeixen a un directori determinat, per tant, el comandament "echo" com que el text no està entre cometes interpreta cada paraula com un element independent, fins que arriba a l'asterisc, on és interpretat que volem que mostre per l'eixida estàndard tots els elements existents al directori on estem.
- La segona línia, el text apareix exactament igual, a excepció que no apareix el \$ASO. A què es deu això? Als Scripts, el "\$" s'interpreta amb el valor d'una variable, per tant, com que al Script no apareix cap variable amb nom "ASO", el valor apareix buit.
- La tercera línia, com que es troba entre cometes dobles, tot el contingut entre estos ja no es interpretat, apareix com un literal, per això mateix podem veure l'asterisc.
- La quarta línia, actua aparentment exactament igual que la tercera, però ara veurem que no és així.
- La quinta línia, com que són cometes dobles, interpreta el contingut com un literal, però si ens fixem, veurem que sí que interpreta les variables, per això no apareix "\$ASO".

- La sexta línia, actua com la quinta, però no interpreta les variables, tot actua com un literal.

Ara, aquestes cometes es poden combinar entre elles perfectament, si el que volem mostrar pel terminal són cometes dobles, el que hem de fer és utilitzar les simples per la frase i al contrari.

Per exemple:

```
vicent@vicent-MS-7C84: ~  
#!/bin/bash  
echo "Este text 'deuria' estar entre cometes simples"  
echo 'Este text "deuria" estar entre cometes dobles'
```

D'aquesta manera el que aconseguim és el següent:

```
vicent@vicent-MS-7C84:~$ vim cometes.sh  
vicent@vicent-MS-7C84:~$ source cometes.sh  
Este text 'deuria' estar entre cometes simples  
Este text "deuria" estar entre cometes dobles  
vicent@vicent-MS-7C84:~$
```

3.3 VARIABLES

Les variables als Scripts son molt simples, no cal definir-les abans d'utilitzar-les ja que es tracta d'un llenguatge no tipat, no cal indicar quin tipus de valor contindrà (nombres, lletres, frases, boolean, etc). Per tant l'únic requisit d'aquestes és escriure el seu nom i assignar-li un valor. Després, per obtindre el seu valor cal fer ús del "\$". Per exemple:

```
#!/bin/bash  
missatge="Bon dia pel matí"  
  
echo $missatge  
  
echo $missatge "un bon café ens dona la vida"
```

Com podeu observar he creat una variable anomenada "missatge" que conté el missatge "Bon dia pel matí", per fer ús d'ella al comandament echo cal utilitzar el \$. A més, he creat un altra línia exactament igual, però afegint-li un missatge seguit. Com penseu que actuarà este Script?

```
vicent@vicent-MS-7C84:~$ vim variables.sh  
vicent@vicent-MS-7C84:~$ source variables.sh  
Bon dia pel matí  
Bon dia pel matí un bon café ens dona la vida
```

L'ordre "echo" fa una concatenació automàtica del contingut de la variable i del literal creant un únic missatge final.

Les variables ens donen un joc molt important al món dels Script, ja que no sols podem guardar valors numèrics, paraules, frases, i més, podem guardar ordres directament, de forma que podem cridar una variables perquè faja una acció.

3.4 REDIRECCIÓ D'ENTRADA I EIXIDA

La redirecció és la forma que podem utilitzar per guardar els resultats i errors de l'execució d'un Script directament a un fitxer, per exemple, tenim els següents operadors:

- `<` : Utilitzar el contingut d'un fitxer com a entrada.
- `>` : Redirecciona l'eixida estàndard del Script a un fitxer, en cas de no existir, el crearà, si existeix, tot el contingut que contenia s'esborrarà.
- `>>` : Redirecciona l'eixida igual que `>` però amb la diferència que no esborra el contingut, escriurà a continuació de l'última línia.
- `2>` : Redirecciona l'eixida d'errors, actua exactament igual que `>`, si el fitxer on volem redirigir els errors conté cap contingut s'esborrarà, en cas contrari, crearà el fitxer i guardarà el valor.
- `2>>` : Redirecciona l'eixida d'errors del Script a un fitxer, en cas de no existir, el crearà, s'escriurà a continuació de l'última línia.

A continuació vos mostre els possibles casos:

```
#!/bin/bash
echo "El resultat de echo anirà al fitxer" > echoresult
echo "El resultat aquest anirà junt amb el echo d'abans" >> echoresult
echo "Si hi ha algun error es guardarà a errorresult" 2> errorresult
```

Executem el Script i veiem com els 2 primers echo no han aparegut, directament s'han redirigit al fitxer, per això al fer un cat sobre "echoresult" veiem les dues línies. En canvi, el "echo" amb la redirecció de l'error sí que apareix ja que no hi ha cap error, i si revisem "errorresult" veurem com està buit.

```
vicent@vicent-MS-7C84:~$ vim redireccio.sh
vicent@vicent-MS-7C84:~$ source redireccio.sh
Si hi ha algun error es guardarà a errorresult
vicent@vicent-MS-7C84:~$ cat echoresult
El resultat de echo anirà al fitxer
El resultat aquest anirà junt amb el echo d'abans
vicent@vicent-MS-7C84:~$ cat errorresult
```

Ara, si provoquem un error intencionadament, podem revisar el fitxer, per vore què conté:


```
cd ls 2> errorresult
```

I el resultat guardat és:

```
vicent@vicent-MS-7C84:~$ cat errorresult  
bash: cd: ls: No existe el archivo o el directorio
```

4 EXECUCIÓ D'ORDRES I OPERACIONS ARITMÈTIQUES

4.1 EXECUCIONS

Tot seguit al punt anterior, l'execució d'ordres dins d'un Script és una de les funcionalitats més útils que podem trobar a la programació en Shell. Per poder executar una ordre solament hem de fer ús de **\$()**, tot el que es trobe dins dels parèntesis s'executarà i tornarà una eixida, com que ens trobem al terminal, l'eixa és l'estandard, el mateix terminal per pantalla, però això pot canviar.

Tenim l'opció de recollir el resultat de l'operació. Per guardar una ordre dins d'una variable, cal utilitzar el símbol del dolar "\$" juntament a un parèntesi però a més, afegirem davant una variable on es guarda. Més tard, per obtindre el valor d'aquesta també farem ús del dolar. Un exemple amb l'ordre "date":

```
#!/bin/bash  
  
data=$(date +%d-%m-%Y)  
  
echo $data;
```

On obtindriem:

```
vicent@vicent-MS-7C84:~$ vim data.sh  
vicent@vicent-MS-7C84:~$ source data.sh  
16-09-21
```

4.2 OPERACIONS ARITMÈTIQUES

Per fer operacions dins dels Script existeixen 3 possibilitats:

- Fer ús de **[\$variable + X]**
- Fer ús de **\$((variable + X))**
- Fer ús de l'ordre "let", que també interpreta que el resultat a guardar és una operació aritmètica (necessita una variable on guardar-lo), exemple: "let resultat=variable + X".

No és necessari utilitzar variables per guardar el resultat de les anteriors operacions i la resta d'ordres, es poden fer totes en una única línia, però sí que és una bona pràctica, intentar no complicar massa el fitxer i fer que siga el més fàcil d'entendre possible.

Els operadors aritmètics que podem fer servir per fer operacions són:

- Resta (-)
- Suma (+)
- Divisió (/)
- Multiplicació (*)
- Mòdul o Resta (%) (el “resto” d’una divisió)

Després, tenim l’opció d’acotar a un nombre X de decimals amb “echo” i “bc”, aquest ens permeten imprimir tants decimals com desitgem dels resultats de les operacions. Per fer un exemple, fent ús de la primera opció per operar $235/3$, prova al teu terminal el següent:

```
echo $[ scale=2; 235/3 ] | bc -l
```

Comprova quina resposta obtenim per l’eixida estàndard, i torna a executar l’ordre però augmentant el valor de “scale”.

Repasant un poc les tuberíes: El que està fent el “echo”, es imprimir el resultat de l’operació que volem per l’eixida estàndard, aquesta s’introdueix a l’ordre “bc” mitjançant la tubería. “bc” es tracta de una calculadora científica integrada al terminal de linux, com que obté la resposta de “echo”, la primera ordre és configurar l’escala i després executa l’operació.

Perquè quede més clar, si arranquem bc i executem els mateixos passos el resultat deu ser el mateix:

```
vicentbenavent@MacBook-Pro-de-Vicent ~ % bc
bc 1.06
Copyright 1991-1994, 1997, 1998, 2000 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
scale=4
235/3
78.3333
scale=8
235/3
78.33333333
```

5 RECOLLIDA DE DADES

Ja tenim unes bases per poder treballar amb els Script i fer operacions senzilles, però fins ara, tot el que tractem és bastant estàtic, tots els paràmetres eren interns (posats a mà) o els obtenim com resultat d'alguna ordre. Per introduir valors dins del nostre Script tenim 2 opcions:

- Entrada per paràmetres, l'execució dels Script ens permeten passar diferents valors al moment de l'execució, això ens dona certa flexibilitat per a tractar dades. Com ho fem? Molt senzill, hem d'introduir els valors quan s'executa i dins de l'escript els recollim com una variable.

Cada variable és la posició en la què l'hem introduït:

- \$0 -> La posició 0 és el nom del script que estem utilitzant
- \$1 -> El primer paràmetre
- \$2 -> El segon paràmetre
- \$3 -> ... etc
- \$* -> Conjunt de tots els paràmetres (tots ells junts separats per espais)
- \$# -> Nombre de paràmetres que has passat

Exemple:

```
vicentbenavent@MacBook-Pro-de-Vicent ~ % vim parametres.sh
vicentbenavent@MacBook-Pro-de-Vicent ~ % source parametres.sh
El nom del script és parametres.sh
El paràmetre en posició 1 és
El paràmetre en posició 2 és
El paràmetre en posició 3 és
vicentbenavent@MacBook-Pro-de-Vicent ~ % source parametres.sh Josep Antoni Miguel
El nom del script és parametres.sh
El paràmetre en posició 1 és Josep
El paràmetre en posició 2 és Antoni
El paràmetre en posició 3 és Miguel
vicentbenavent@MacBook-Pro-de-Vicent ~ %
```

On el codi és:

```
#!/bin/bash
echo "El nom del script és $0"
echo "El paràmetre en posició 1 és $1"
echo "El paràmetre en posició 2 és $2"
echo "El paràmetre en posició 3 és $3"
```

- Recollida mitjançant l'ordre "read", aquesta ens permet mostrar un text i recollir el text que s'introdueix al terminal fins que li donem a "Intro", aquesta és perfecta per quan configurem un

Script per crear usuaris (per exemple), cada vegada que l'executem li donem els valors que ens demana.

```
#!/bin/bash
echo "Script de creació d'usuaris"

read -p "Introdueix el teu nom: " nom
read -p "Introdueix el teu cognom: " cognom
read -p "Introdueix la teua edat: " edat

echo "$nom $cognom, el teu usuari '$nom$cognom' s'ha creat correctament"
```

L'execució del fitxer seria esta:

```
vicent@vicent-MS-7C84:~$ vim read.sh
vicent@vicent-MS-7C84:~$ source read.sh
Script de creació d'usuaris
Introdueix el teu nom: Vicent
Introdueix el teu cognom: Benavent
Introdueix la teua edat: 23
Vicent Benavent, el teu usuari 'VicentBenavent' s'ha creat correctament
vicent@vicent-MS-7C84:~$
```

Amb el que hem vist fins ara ja tenim coneixements per començar a treballar en alguns exercicis, per tant, vos deixo a Aules uns exercicis per entregar. Recordar-vos que l'entrega està disponible fins el dia 30/31 del mes següent.

2. BIBLIOGRAFIA

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

Vicent Benavent

Paula Grau



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