Bash Shell scripts

Generalitats

Els “bash shell scripts” són fitxers de text on s'escriuen les comandes que es volen executar de forma automàtica, una rere l'altra. Aquests fitxers han de tenir activat el permís d'execució per l'usuari desitjat i la seva extensió, encara que pot ser qualsevol, sol ser “.sh”.

Per executar aquests fitxers si no estan guardats dins d'alguna carpeta dins del PATH., en el moment d'invocar-los des de la línia de comandes s'ha d'utilitzar la comanda especial "source", així: *source /ruta/nomscript.sh* o bé, de forma alternativa, indicar simplement la ruta absoluta a l'script (una ruta relativa també podria funcionar però per a què sigui així hauria de començar obligatòriament sempre per "./", així: *./ruta/nomscript.sh").*

Normalment, la primera línia d'un shell script és l'anomenat “shebang”, que indica -després dels símbols #!- la ruta de l'intèrpret que llegirà el codi allà escrit. Aquesta línia és imprescindible per a què la comanda *source* sàpiga quin shell ha d'escollir. Com que nosaltres escriurem script per a què els interpreti el shell bash, aquesta línia sempre serà per nosaltres #!/bin/bash.
NOTA: Veureu que a molts scripts el "shebang" és #!/bin/sh. Generalment, /bin/sh és un enllaç al shell per defecte del sistema, que sol ser /bin/bash però no sempre (hi ha molts shells diferents: dash, fish, ksh, zsh, etc i no tots ofereixen les mateixes funcionalitats i sintaxis). Vigileu amb això perquè pot ser que algun script no funcioni correctament si no és executat pel shell bash en concret.

Cada línia que pertanyi a un comentari ha de començar amb el símbol #, encara que no cal que sigui el primer caràcter de la línia

El símbol del punt i coma es pot fer servir per separar diferents sentències dins una mateixa línia en comptes d'escriure-les cadascuna en una línia diferent.

Una comanda molt comuna als scripts és la comanda *echo missatge*, la qual imprimeix per pantalla el missatge especificat. Cal tenir en compte que si s'escriu dins el missatge una expressió com $var1, les cometes dobles fan que aquesta expressió se sustitueixi pel valor de la variable corresponent, però si s'utilitzen cometes simples això no passa. Aquesta comanda té uns quants paràmetres interessants com són *-n* (que serveix per no afegir al final de la frase cap salt de línia) i *-e* (que permet, com ja sabem, introduir al missatge els símbols \t, \n, \r, \b, \xHH i els codis de colors -entre altres-).

Una altra comanda força important és *exit*, utilitzada per aturar i sortir de l'script

Exemple: Shell script que mostra un missatge de benvinguda tabulat, després fa un *ls*, després un *date* i seguidament surt sense haver executat la resta de línies (que en aquest cas només és una que només mostra un missatge de comiat)

```
#!/bin/bash
echo -e "Hola\tamic!"
ls
#Això és un comentari
date
exit
echo "Adéu"
```

Variables

No cal declarar les variables. Poden ser de tipus sencer o cadena (en aquest cas, el seu valor ha d'escriure's entre cometes).

Quan es vol utilitzar el VALOR d'una variable, s'ha de precedir el seu nom amb el signe $. Quan es vol utilitzar la variable en sí, no. Una regla "per a tontos" útil és recordar que si la variable s'escriu a la dreta d'una asignació, portarà $ i l'esquerra no.
*NOTA: Existeix la comanda let que serveix per assignar un valor a una variable (així: let x=32) amb l'avantatge de què permet realitzar operacions aritmètiques directament (per exemple:let a=x*2 ó let a=x+4) cosa que sense let no és possible. Compte perquè amb el let no s'utilitza el símbol $*

Exemple: Shell script que assigna el valor 32 a la variable anomenada *x* , seguidament asigna el VALOR de *x* (és a dir, 32) a una altra variable anomenada *y* i finalment assigna la cadena "x" a la variable *z* . Tot seguit, mostra per pantalla la cadena "x y z" i després els valors contecutius de les variables *x, y* i *z* (és a dir, "32 32 x").

```
#!/bin/bash
x=32
y=$x
z=x
echo x y z
echo $x $y $z
```

Per assignar un valor a una variable de forma interactiva, es pot fer servir la comanda *read nomvariable*. Es poden assignar varis valors a diferents variables a la vegada. Concretament: *read var1 var2* assignaria d'un sol cop (després de pulsar ENTER) un valor a var1 i un altre a var2, tots dos introduïts per l'usuari en el moment d'executar l'script (i separats entre sí per un espai en blanc...si hi ha més d'un espai en blanc, en aquest exemple var2 tindria tot el que resti). Aquesta comanda té uns quants paràmetres interessants:

-p "textamostrarabansdelcursor"
-t timeout : especifica temps màxim d'espera d'una resposta, en segons
-n nºchars: termina de rebre caràcters un cop s'ha arribat al número especificat
-s : mode ocult (no es veu a la pantalla el que s'escriu com a resposta)

NOTA: Cal dir que la comanda read ***no*** *es pot entubar (és a dir, no es pot fer echo valor | read var1 per exemple), perquè no és una comanda separada sinò que és un "builtin" del bash. Això vol dir que per funcionar inicia un shell fill intern i, per tant, encara que var1 sí que agafi el valor, no està accessible pel shell pare. Això es pot demostrar executant un echo $a dins la shell filla -posant-ho tot entre parèntesis- així: echo valor | (read var1; echo $var1) , mentre que si fem echo valor | read var1; echo $var1 veiem que no funciona. Per "solucionar-ho", hem de fer servir redireccionadors, per exemple: read a <<(echo valor) o bé agafar els valors de la primera línia d'un fitxer d'entrada, així: read a < unfitxer (en aquest darrer sentit, cal tenir en compte que la comanda read elimina els caràcters definits a la variable IFS (per defecte, l'espai, el tabulador i el salt de línia) perquè els tracta com caràcters separadors en el cas d'assignar varis valors d'una línia a diferents variables (així, read a b < unfitxer)*

Exemple: El codi és autoexplicatiu

```
#!/bin/bash
echo -n "Introdueix un valor: "
#El valor introduït s'assignarà a la variable anomenada X
read X
echo -n "Introdueix dos valors separats per espais: "
#Cada valor separat per espais s'assignarà a una variable diferent
#Si a la última variable encara queden valors, s'assignaran tots a aquesta
read Y Z
echo -n "Tens cinc segons per introduïr un valor secret: "
read -s -t 5 SECRET
echo -e "\nX val $X, Y val $Y i Z val $Z \nEl secret val $SECRET"
```

Una altra comanda interessant relacionada amb les variables és *expr*. Concretament, es pot usar així:
expr length $cadena : retorna la longitud de la cadena (una altra forma és fent ${#cadena})
expr index $cadenaonbuscar $cadenaabuscar : retorna la posició on es troba la cadena a buscar (o 0)
expr match $cadenaonbuscar exprregabuscar : el mateix, però usant expressions regulars
expr substr $cadenaonbuscar nºposinicial nºcaracters : retorna una subcadena

Exemple: El codi és autoexplicatiu

```
#!/bin/bash
cadena="Hola amic"
expr length "$cadena"        #Treurà 9 (la cadena té 9 caracters)
expr index  "$cadena" "la"  #Treurà 3 (la posició on comença "la" dins de la cadena, començant per 1)
expr substr "$cadena" 2 5   #Treurà "ola a"
```

D'altra banda, el bash ens ofereix un conjunt de variables predefinides amb un significat especial:

$0 : nom de l'script
$1-$9 : valor de cadascun dels paràmetres passats a l'script
Si es volen fer servir més de 9 paràmetres, el número s'ha d'escriure entre claus: ${10}, ${11},etc
$# : nº de paràmetres passats
$* : cadena formada per tots els paràmetres separats pel caràcter indicat a la variable IFS
$@ : similar a l'anterior, però aquí cada paràmetre és una cadena diferent
$$: PID de l'script
$? : estat de terminació de l'última comanda (val 0 si no hi ha errors).
$- : informa de la configuració actual del shell. En concret, el valor que més en interessa és "i": si apareix vol dir que el shell és interactiu (és a dir, no s'està executant un script).
NOTA: Per més informació sobre $-: https://www.chainsawonatireswing.com/2012/02/02/find-out-what-your-unix-shells-flags-are-then-change-them//?from=@

Exemple: Suposant que aquest codi el gravem en un arxiu anomenat “script.sh”, a l'hora d'executar-lo haurem d'escriure ./script.sh 34 hola 75 ,on el primer paràmetre ($1) contindrà el valor 34, el segon paràmetre ($2) contindrà el valor “hola” i el tercer paràmetre ($3) contindrà el valor 75.

```
#!/bin/bash
echo "L'script s'anomena: $0"
echo "El valor del primer paràmetre que has escrit és: $1"
echo "El valor del primer paràmetre que has escrit és: $2"
echo "El valor del primer paràmetre que has escrit és: $3"
echo "El número de paràmetres que has escrit és: $#"
echo "La llista completa de paràmetres que has escrit és: $*"
```

A més cal saber les possibilitats següents:

${variable} : és el mateix que $variable, però les claus ajuden al bash a saber on comença i on acaba el verdader nom de la variable. Amb això s'eviten posibles problemes com ara escriure: *$variablefoo*, quan en realitat es vol dir *${variable}foo.*

$(comanda) : substitueix la comanda (o línia de comandes) per la seva sortida. Se sol usar per fer que la sortida d'una comanda sigui el valor d'un paràmetre d'una altra, o d'una variable. És a dir, que fer *echo $(cat arch1|wc -w) o mivar=$(cat arch1|wc -w);echo $mivar* seria el mateix

$((expresio_matematica)) : permet realitzar sumes, restes, multiplicacions, divisions i mòduls (%) de números sencers (per números decimals caldrà fer servir eines externes com *bc*). És fàcil comprovar la seva utilitat si comparem el resultat de fer *z=$x+$y* amb *z=$(($x+$y))*. Els números que comencen per “0x” són tratats com hexadecimals. Com a dada curiosa, dir que dins dels parèntesis no cal escriure el símbol $ al davant de les variables involucrades (per tant, l'expressió z=$((x+y)) seria vàlida).
NOTA: Una alternativa obsoleta és escriure *$[expresio_matematica]*

**${variable:nº}** : pren el valor de *variable* a partir del caracter la posició del qual és nº, començant per 0. Per exemple, si x=1234567890, *${x:3}* serà 4567890. També es pot fer ${variable:nº:length}, i llavors *${x:3:4}* valdria 4567. La length pot ser negativa: en aquest cas, es conta des del final; per exemple *${x:3:-4}* valdria 456

${#variable} : retorna el número de caracters que té el valor de la variable indicada, o el número d'elements si és un array.

${variable:-valor}** : retorna el valor de variable sempre i quan no estigui buit: en aquest cas, retornarà el valor indicat després del "-" . La construcció **${variable:=valor} és similar però a més, assigna en el cas descrit el valor a variable; això és molt útil per exemple per assignar valors per defecte a variables en scripts on l'usuari no hagi indicat explícitament cap valor per elles.

Altres possibilitats més exòtiques són aquestes:

${variable##exprambcomodins}: elimina del valor de la variable l'expressió més llarga possible que sigui compatible amb l'expressió amb comodins indicada, començant des del principi. Per exemple, si $variable fos igual a "/ruta/un/fitxer.txt", ${variable##*/} serà igual a "fitxer.txt" perquè l'expressió "*/" vol dir "qualsevol cosa acabada en "/" -i que sigui la més llarga possible des del principi-; per tant, és equivalent a "/ruta/un/", expressió que és la que s'elimina. En realitat, en aquest exemple concret es podria haver fet el mateix amb la comanda *basename ${variable}* .

NOTA: Si s'usa # en comptes de ## s'eliminarà l'expressió més curta en comptes de la més llarga

${variable%exprambcomodins}: elimina del valor de la variable l'expressió més curta possible que sigui compatible amb l'expressió amb comodins indicada, començant des del final. Per exemple, si $variable fos igual a "/ruta/un/fitxer.txt", ${variable%/*} serà igual a "/ruta/un" perquè l'expressió "/*" vol dir "qualsevol cosa que comenci amb "/" -i que sigui la més curta possible, començant pel final-; per tant, és equivalent a "/fitxer.txt", expressió que és la que s'elimina. En realitat, en aquest exemple concret es podia haver fet el mateix amb l'ordre *dirname ${variable}*. Un altre exemple: si tenim un nom d'arxiu com a valor de la variable "arx", *${arx%.*}* treurà la seva extensió

NOTA: Si s'usa %% en comptes de % s'eliminarà l'expressió més llarga en comptes de la més curta. Un exemple seria *${CADENA#"${CADENA%%[![:space:]]*}"}* (per realitzar un "trim" per l'esquerra) *o ${CADENA%"${CADENA##*[![:space:]]}"}* (per realitzar-ho per la dreta)

D'altra banda, per canviar el valor d'una variable de mayúscules a minúscules o viceversa es pot fer *echo* **${mivar^}** (per convertir a majúscula la primera lletra) o bé *echo* **${mivar^^}** (per convertir-les totes) i *echo* **${mivar,}** (per convertir a minúscula la primera lletra) o bé *echo* ***${mivar,,}*** (per convertir-les totes).

També es pot canviar "al vol" una part del valor d'una variable per una altra fent **${mivar/partasustituir/partsustituta}** -si només volem canviar la 1ª ocurrència de "partasustituir"- o bé **${mivar//partasustituir/partsustituta}** -si volem canviar totes les ocurrències-, podent fer servir en qualsevol cas comodins a "partasustituir"

Per veure més possibilitats, recomano mirar la secció “Parameter expansion” del manual bash.

Exemple: El codi és autoexplicatiu

```
#!/bin/bash
a=$(ls)       #El texte que treuria la comanda ls per pantalla, en comptes de mostrar-lo allà, el grava a la variable a
b=$(date)   #El texte que treuria la comanda date per pantalla, en comptes de mostrar-lo allà, el grava a la variable b
sleep 5                  #M'espero 5 segons
echo $a $b               #Mostro el contingut de les variables a i b, un seguit de l'altre

num=345
echo $(($num + 1))       #Mostro el valor de num més 1 (346)
echo $((($num – 1)*2))  #Mostro el valor de num menys 1 multiplicat per 2 (688)
echo ${num:1}            #Mostro el valor 45 (és a dir, tot el que hi ha a partir de la posició 1 de num)
echo ${#num}             #Mostro un 3 perquè la variable num té tres caracters
echo ${d:=hola}          #Assigno el valor "hola" a la variable 'd' -no definida fins ara- i el mostra per pantalla
echo ${d}                #Confirmo que efectivament, s'ha assignat el valor "hola" a la variable 'd'

c=$(echo $b | sed "s/ //g") ; echo ${#c}  #Mostro el nº de caràcters que treu la comanda date sense comptar els espais
```

Condicionals

Podem fer servir la construcció *if/elif/else* :

```
if [[ cond ]]
then
        --
        --
elif [[ cond2 ]]
then
        --
        --
elif [[ cond3 ]]
then
        --
        --
(...)
else
        --
        --
fi
```

Tant els blocs *elif* com el bloc *else* són opcionals; així, podríem trobar estructures sense *elif*:

```
if [[ cond ]]
then
        --
        --
else
        --
        --
fi
```

o fins i tot sense ni *else*:

```
if [[ cond ]]
then
        --
        --
fi
```

NOTA: Les condicions també es podrien haver escrit d'una manera més "clàssica" : entre claudàtors simples, així *if [cond]* . No obstant, si es fa així llavors no es poden utilitzar algunes de les funcionalitats que sí estan disponibles amb els claudàtors dobles, com ara l'ús d'expressions regulars o comodins dins de les condicions, entre moltes altres que veurem. Per tant, es recomana sempre escriure dobles claudàtors.

La condició “cond” que s'ha de complir (és a dir, que ha de ser certa) per a què s'executi l'interior del bloc corresponent pot ser d'alguns d'aquests tipus (on es comparen de números sencers):

int1 -eq int2 : El número int1 és igual que el número int2
int1 -ge int2 : El número int1 és més gran o igual que el número int2
int1 -gt int2 : El número int1 és més gran que el número int2
int1 -le int2 : El número int1 és més petit o igual que el número int2
int1 -lt int2 : El número int1 és més petit que el número int2
int1 -ne int2 : El número int1 és diferent que el número int2

Exemple: El codi és autoexplicatiu. Per executar-lo es poden afegir paràmetres, com ara ./script.sh 234 qwerq afsd

```
#!/bin/bash
if [[ 1 -eq 1 ]]
then
        echo "Aquesta frase es mostrarà perquè la condició sempre és veritat"
elif [[ 2 -lt 3 ]]
then
        echo "Aquesta NO perquè encara que la condició és veritat, l'elif només es mira si l'if anterior era mentida"
fi

if [[ $1 -gt 4 ]]
then
        echo "Aquesta frase es mostrarà si el paràmetre introduït és major de 4"
else
        echo "Aquesta frase es mostrarà si el paràmetre introduït NO és major de 4"
fi

if [[ $# -ne 5 ]]
then
        echo "Aquesta frase es mostrarà si el número de paràmetres introduïts és diferent de 5"
fi
```

NOTA: Si es volen fer servir els símbols matemàtics "normals" (==, !=, >, <, >=, <=) en comptes de -eq, -ge, etc, hi ha una manera: substituir els dobles claudàtors per dobles parèntesis, així:

Exemple: El mateix codi que l'anterior exemple

```
#!/bin/bash
if (( 1 == 1 ))
then
        echo "Aquesta frase es mostrarà perquè la condició sempre és veritat"
elif (( 2 < 3 ))
then
        echo "Aquesta NO perquè encara que la condició és veritat, l'elif només es mira si l'if anterior era mentida"
fi

if (( $1 > 4 ))
then
        echo "Aquesta frase es mostrarà si el paràmetre introduït és major de 4"
else
        echo "Aquesta frase es mostrarà si el paràmetre introduït NO és major de 4"
fi

if (( $# != 5 ))
then
        echo "Aquesta frase es mostrarà si el número de paràmetres introduïts és diferent de 5"
fi
```

O bé, un d'aquests (on es comparen cadenes):

"cad1" == "cad2" : La cadena cad1 és igual que cad2 (en comptes de "==" també es pot usar "=". Dins de "cad2" poden haver comodins (*,?) però llavors no s'hauran d'escriure les cometes.
Fins i tot es poden fer servir expressions regulars (sense cometes, així: [[$var1 =~ *exprreg*]]
"cad1" != "cad2" : La cadena cad1 és diferent de cad2
-n "cadena" : La longitud de la cadena és més gran de 0 (no és una cadena buida)
-z "cadena" : La longitud de la cadena és 0 (és una cadena buida)

Exemple: El codi és autoexplicatiu. Per executar-lo es poden afegir un paràmetre, com ara ./script.sh pepe

```
#!/bin/bash
if [[ "hola" == "Hola" ]]
then
        echo "Aquesta frase NO es mostrarà perquè les minúscules i les majúscules són diferents"
fi

if [[ "$1" != "pepe" ]]
then
        echo "Aquesta frase es mostrarà si el primer paràmetre introduït és diferent de la paraula 'pepe'"
else
        echo "Aquesta frase es mostrarà si el primer paràmetre introduït és igual a la paraula 'pepe'"
fi

if [[ "$1" == [Yy]* ]]
then
        echo "Aquesta frase es mostrarà només si el 1r paràmetre comença per 'Y' o 'y'"
fi

#Si es volgués comparar números decimals, es podria utilitzar l'expr. reg següent: ^-?[0-9]+(\.[0-9]+)?$
if [[ $1 =~ ^-?[0-9]+$ ]]
then
        echo "El valor introduït és un número sencer"
else
        echo "El valor introduït NO és un número sencer"
fi

if [[ -z "$1" ]]
then
        echo "Aquesta frase es mostrarà si el primer paràmetre està buit (no s'ha introduït res)"
else
        echo "Aquesta frase es mostrarà si el primer paràmetre té algun valor"
fi

if [[ -z $(ls|grep "hola") ]]
then
        echo "No hi ha cap arxiu anomenat 'hola' a la carpeta actual"
fi
```

O bé, un d'aquests (on es comproven diferents característiques dels fitxers i carpetes del sistema):
"arx1" -ef "arx2" : es compleix si els dos arxius apunten al mateix inode (són enllaços forts)
"arx1" -nt "arx2" : es compleix si arx1 és més nou que arx2 (si sa data de modificació és posterior)
"arx1" -ot "arx2" : es compleix si arx1 és més vell que arx2 (si sa data de modificació és anterior)
-e "arxiu" : es compleix si existeix l'arxiu
-d "arxiu" : es compleix si existeix i és una carpeta
-f "arxiu" : es compleix si existeix i és un arxiu regular
-L "arxiu" : es compleix si existeix i és un enllaç dèbil. També serveix -h arxiu
-r "arxiu" : es compleix si existeix y té permís de lectura per l'usuari actiu
-w "arxiu" : es compleix si existeix i té permís d'escriptura " " " "
-x "arxiu" : es compleix si existeix i té permís d'execució " " " "
-O "arxiu": es compleix si existeix i el seu propietari és l'usuari actiu
-N "arxiu": es compleix si existeix i ha sigut modificat des del darrer cop que va ser llegit
-s "arxiu" : es compleix si existeix i el seu tamay és major que 0

Per saber més opcions, veure apartat "Conditional expressions" del manual del bash.

Exemple: El codi és autoexplicatiu. Per executar-lo es poden afegir un paràmetre, com ara ./script.sh /usr

```
#!/bin/bash
if [[ "/etc/passwd" -nt "/etc/shadow" ]]
then
        echo "L'arxiu 'passwd' és més nou (té una data de modificació més propera) que l'arxiu 'shadow'"
fi

if [[ -e "$1" ]]
then
        echo "El valor introduït com a primer paràmetre és la ruta d'un arxiu (o carpeta) real"
fi

if [[ -s "/etc/crontab" ]]
then
        echo "L'arxiu 'crontab' té un tamany major de 0"
fi

if [[ -x "/bin/ls" ]]
then
        echo "L'arxiu 'ls' té permisos d'execució"
fi
```

El símbol **!** (seguit d'un espai en blanc) davant d'una condició canvia el sentit de la comprovació. Els símbols **&&** i || serveixen per encadenar vàries condicions amb el sentit de “i” i de “o”, respectivament.
NOTA: Si es fan servir claudàtors simples, en comptes de && s'ha d'escriure "-a" i en comptes de "||" s'ha d'escriure "-o", però aquesta pràctica està desaconsellada per obsoleta

Exemple: Shell script que comprova si no existeix un arxiu (o carpeta) la ruta del qual s'ha hagut de passar com a paràmetre. Si no existeix, el crea en forma d'arxiu (i si ja existia, no fa res)

```
#!/bin/bash
#Aquesta comprovació prèvia és necessària per a què la comanda touch no "peti" en trobar-se un valor de $1 buit.
#Una altra condició similar podria ser [[ -z "$1" ]]
if [[ $# -eq 0 ]]
then
        echo "Has d'introduïr un paràmetre!!"
        exit  #Surto del programa
fi

if [[ ! -e "$1" ]]
then
        echo "No existeix l'element especificat. Es crearà com arxiu."
        touch $1
        echo "Ara ja existeix"
fi
```

Exemple: Shell script que comprova si existeix un arxiu la ruta del qual s'ha hagut de passar com a paràmetre. Si existeix, l'esborra (i si no, no fa res). Per fer el mateix amb carpetes caldria canviar la condició *-f* per *-d* i la comanda *rm* per *rm -rf*

```
#!/bin/bash
if [[ $# -eq 0 ]] ; then
        echo "Has d'introduïr un paràmetre!!" ;  exit
fi

if [[ -f "$1" ]]
then
        echo "Existeix l'arxiu especificat"
        rm $1 &&  echo "Ara ja no"   #M'asseguro abans d'escriure el missatge de què s'ha esborrat bé
fi
```

Exemple: Shell script que crea un fitxer (el nom del qual s'ha passat com a primer paràmetre) i seguidament comprova dues condicions (que s'han de complir a l'hora): si aquest fitxer té permisos d'escriptura i si s'ha introduït un segon paràmetre. Si ambdues condicions es compleixen, s'escriu dins del fitxer el valor del segon paràmetre i es mostra aquest recent contingut

```
#!/bin/bash
touch $1
if [[ -w $1 && ! -z $2 ]]
then
        echo $2 > $1
        cat $1
fi
```

Exemple: Shell script que presenta un menú amb dues opcions: si l'usuari escull la primera, el programa respondrà amb una frase i si escull la segona, respondrà amb una altra

```
#!/bin/bash
echo "Pulsi 'a' o 'b':"
echo "a) M'agrada el futbol"
echo "b) M'agrada el basket"
read opcio
if [[ "$opcio" == "a" ]]
then
        echo "Li agrada el futbol"
elif [[ "$opcio" == "b" ]]
then
        echo "Li agrada el basket"
else
        echo "No li agrada cap opció"
fi
```

Xul.leta amb regles d'escriptura general

*En una assignació de valors a variables, NO S'HAN DE DEIXAR espais entre el nom de la variable i el signe =, ni entre el signe = i el valor de la variable.

*Cada cop que s'escrigui claudàtors d'apertura o tancament ([[o]]) S'HA DE DEIXAR un espai en blanc DAVANT I DARRERA d'ells.

*En una comparació de cadenes (del tipus [[“cad1” == “cad2”]] S'HA DE DEIXAR un espai en blanc DAVANT I DARRERA del signe =

En qualsevol cas, si voleu comprovar la correcta sintaxis del vostre script, sempre podeu anar a http://www.shellcheck.net i provar al quadre online que apareix allà si qualsevol script està ben escrit (i si no, on es troba l'error). També es pot fer servir "shellcheck" sense tenir Internet si s'instal.la al sistema. Un altre truc és executar com a primera comanda del nostre script la comanda *set -ex* ; el que fa aquesta comanda és aturar l'script al primer error que trobi i mostrar la línia problemàtica.

Un altre condicional que podem fer servir és:

```
case $variable
in
        valor)
                --
                --
        ;;
        valor | unaltre | unaltremes )
                --
                --
        ;;
        *)
                --
                --
        ;;
esac
```

...on el valor ha avaluar pot contenir comodins (del tipus [1-6]*, por exemple) i, en comptes de ;; es pot escriure ;;& per continuar avaluant el següent valor del case.

Exemple: Shell script que fa el mateix que l'anterior, però substituïtn els *ifs/elifs/else* per un *case*:

```
#!/bin/bash
echo "Pulsi 'a' o 'b':"
echo "a) M'agrada el futbol"
echo "b) M'agrada el basket"
read opcio
case $opcio
in
        "a")
                echo "Li agrada el futbol"
        ;;
        "b")
                echo "Li agrada el basket"
        ;;
        *)
                echo "No li agrada cap opció"
        ;;
esac
```

Bucles

Podem fer servir:

while [[cond]]
do
--
--
done

on la condició “cond” és una del mateix tipus que un dels explicats anteriorment. El bucle *while* pot ser finit o no segons si la condició arriba a ser falsa alguna vegada o no, respectivament. Si cal, es poden utilitzar les paraules clau *true* i *false*.

Exemple: Shell script que mostra infinitament la paraula “Hola” (sense arribar a mostrar la paraula “Fi” mai) perquè la condició indicada mai deixa de ser veritat. Per aturar l'execució, cal prèmer CTRL + C

```
#!/bin/bash
while [[ 1 -eq 1 ]]
do
        echo "Hola"
done
echo " Fi"
```

Exemple: Shell script que directament mostra la paraula “Fi” sense executar el bucle perquè la condició indicada ja és mentida d'entrada.

```
#!/bin/bash
while [[ 1 -eq 2 ]]
do
        echo "Hola"
done
echo " Fi"
```

Exemple: Shell script que a cada repetició s'atura per demanar un nou valor a l'usuari, el qual el mostrarà a pantalla. Si aquest valor és diferent de “fi”, continuarà demanant un següent valor; si és igual a “fi”, el bucle es trenca i el programa es continua executant a partir de la línia següent

```
#!/bin/bash
echo "Introdueixi un valor"
read valor
while [[ "$valor" != "fi" ]]
do
        echo "Introdueixi un nou valor"
        read valor
done
echo "Fi"
```

Exemple: Shell script que utilitza un comptador (de valor inicial igual a 1) per mostrar “Hola”exactament 100 vegades

```
#!/bin/bash
contador=1
while [[ $contador -le 100 ]]
do
        echo "Hola"
        #Augmento en 1 el valor de contador. Arribarà un moment en què la condició del while ja no sigui veritat
        contador=$(($contador + 1))
done
echo "Fi"
```

Exemple: Shell script variació de l'anterior, on es mostra el valor del comptador a cada repetició. Com a novetat, hem fet que el comptador vagi de 100 a 1 de dos en dos.

```
#!/bin/bash
contador=100
while [[ $contador -ge 0 ]]
do
        echo $contador
        contador=$(($contador - 2))
done
echo "Fi"
```

Exemple: Shell script variant de l'exemple anterior del menú d'opcions. La novetat està en que ara el menú torna a aparèixer després de què l'usuari hagi escollit una opció, de tal manera que hagi de tornar a escollir-la. I així constantment fins que premi l'opció “c”, que és sortir del programa

```
#!/bin/bash
echo "Pulsi 'a' o 'b':"
echo "a) M'agrada el futbol"
echo "b) M'agrada el basket"
echo "c) Sortir"
read opcio
while [[ $opcio != "c" ]]
do
        if [[ "$opcio" == "a" ]]
        then
                echo -e "Li agrada el futbol\n"
        elif [[ "$opcio" == "b" ]]
        then
                echo -e "Li agrada el basket\n"
        else
                echo -e "No li agrada cap opció\n"
        fi
        echo "Pulsi 'a' o 'b':"
        echo "a) M'agrada el futbol"
        echo "b) M'agrada el basket"
        echo "c) Sortir"
        read opcio
done
echo "Fi"
```

Exemple: Shell script que fa el mateix que l'anterior però on en comptes d'implementar un bucle finit s'ha escollit un bucle infinit acompanyat de la sentència exit. Les dues alternatives són vàlides

```
#!/bin/bash
echo "Pulsi 'a' o 'b':"
echo "a) M'agrada el futbol"
echo "b) M'agrada el basket"
echo "c) Sortir"
read opcio
while [[ true ]]
do
        if [[ "$opcio" == "a" ]] ;  then   echo -e "Li agrada el futbol\n"
        elif [[ "$opcio" == "b" ]] ; then  echo -e "Li agrada el basket\n"
        elif [[ "$opcio" == "c" ]] ;  then  exit
        else  echo -e "No li agrada cap opció\n"
        fi
        echo "Pulsi 'a' o 'b':"
        echo "a) M'agrada el futbol"
        echo "b) M'agrada el basket"
        echo "c) Sortir"
        read opcio
done ;  echo "Fi"
```

Exemple: Programa que genera un número aleatori entre 0 i 65000 (gràcies a la variable especial $RANDOM, la qual, cada cop que s'escriu, genera un número diferent) i proposa a l'usuari que l'endevini interactivament, indicant a cada intent si el número introduït és major o menor que la solució.

```
#!/bin/bash
#Guardo el número aleatori en una variable per no haver d'utilitzar més $RANDOM (si ho fes, cada cop estaria utilitzant un número diferent!!)
numerosecret=$RANDOM
#Defineixo un valor inicial qualsevol de numerousuari per poder fer la primera comparació del while correctament.
#Aquest valor inicial, això sí, ha de ser fora del rang posible de numerosecret per a què no pugui ser igual (i així
#evitar no poder entrar per primer cop dins del bucle while)
numerousuari=70000
while [[ $numerousuari -ne $numerosecret ]]
do
        echo -n "Introdueix un número: "
        read numerousuari
        if [[ $numerosecret -lt $numerousuari ]]
        then
                echo "La solució és menor que el número que has escrit"
        elif [[ $numerosecret -gt $numerousuari ]]
        then
                echo "La solució és major que el número que has escrit"
        else
                echo "MOLT BÉ!!"
        fi
done
echo "Fi"
```

Exemple: Shell script que fa el mateix que l'anterior però on en comptes d'implementar un bucle finit s'ha escollit un bucle infinit acompanyat de la sentència break. Aquí llavors ja no cal definir un valor inicial per numerousuari.

```
#!/bin/bash
numerosecret=$RANDOM
while [[ true ]]
do
        echo -n "Introdueix un número: "
        read numerousuari
        if [[ $numerosecret -lt $numerousuari ]] ; then echo "La solució és menor que el número que has escrit"
        elif [[ $numerosecret -gt $numerousuari ]] ; then  echo "La solució és major que el número que has escrit"
        else  echo "MOLT BÉ!!" ; break
        fi
done
echo "Fi"
```

NOTA: Per llegir línia a línia un fitxer, es pot utilitzar un bucle while amb la comanda read, perquè aquesta comanda retorna true sempre que ha llegit alguna cosa (fins arribar a un caràcter indicat per la variable IFS -els quals per defecte són l'espai en blanc, el tabulador i el salt de línia-) i false si no (que és quan el fitxer s'hauria acabat de llegir):

```
while read palabra1 palabra2 restolinea
do
        echo $palabra1 $palabra2 $restolinea
done < fitxer.txt
#Una manera equivalent seria: cat fichero.txt | while read palabra1 palabra2 restolinea; do ... done
```

NOTA: A partir de la nota anterior, ¿sabries dir què faria aquest bucle?

```
while IFS= read -n1 c
do
        echo $c
done <<< "$cadena"
```

Existeixen les paraules *break* i *continue* . La primera fa sortir del bucle i la segona interromp la iteració actual per continuar amb la següent.

Exemple: Shell script que només mostrarà els números del 1 al 3

```
#!/bin/bash
contador=1
while [[ $contador -le 10 ]]
do
        echo $contador
        if [[ $contador -eq 3 ]]
        then
                break
        fi
        contador=$(($contador + 1))
done
```

Un altre bucle similar que podem fer servir és:

```
until [[ cond ]]
do
        --
        --
done
```

La diferència amb el bucle *while* es troba en que el bucle *until* finalitza quan la condició passa a ser certa, i no al revés. En realitat, són el mateix bucle però escrivint la condició contrària. Si el que volem és, però, que una variable tingui un valor diferent a cada iteració, s'ha de fer servir:

```
for variable in valor1 valor2 valor3
do
        --
        --
done
```

Com es pot veure, cada valor que tindrà la variable (un a cada iteració, per ordre) s'especifica separat de la resta per un espai en blanc (o tabulador o salt de línia...en realitat és el que valgui la variable del shell IFS). Aquesta llista de valors pot venir definida tal qual o bé ser el resultat de l'execució d'alguna comanda, com es veu als exemples següents:

Exemple: Shell script que llegeix el fitxer /etc/passwd línia a línia i el mostra per pantalla separant cada línia amb ---

```
#!/bin/bash
for mivar in $(cat /etc/passwd)
do
        echo "--------"
        echo $mivar
done
```

Exemple: Shell script que mostra el tipus de tots els fitxers amb extensió "txt" presents a la carpeta actual

```
#!/bin/bash
for i in $(ls *.txt)
do
        file $i
done
```

Per repetir un número de vegades concret una acció es pot usar un format del *for* semblant al del C (fixeu-vos que el nom del contador dins del doble parèntesi no porta $ al davant i que s'utilitzen els símbols matemàtics habituals):

```
for ((contador=0;contador<3;contador++))
do
        _
done
```

Exemple: Shell script que torna a mostrar el valor d'un comptador d'1 a 100, però ara utilitzant el bucle *for*

```
#!/bin/bash
for ((contador=1;contador<=100;contador++))
do
        echo $contador
done
```

Funcions

Una funció és un tros de codi amb un nom, el qual es pot fer referència en un altre lloc del codi per executar aquest tros de codi. Una funció es declara així:

```
nomFuncio() {
        ---
  Dins d'aquest cos, $1, $2, $#, etc agafen el valor dels parámetres
  que s'hagin especificat a la crida de la funció
        ---
}
```

I s'executa així: ***nom param1 param2 …*** Es pot veure que en la crida a la funció no s'escriuen paràmetres, només els paràmetres separats per espais en blanc després del nom de la funció.

La declaració s'ha d'escriure dins el codi sempre abans de què s'executi la funció per primer cop, perquè si no el Bash no sabrà quina funció és la que es pretén executar.

Si una funció conté dins del seu codi la paraula ***return***, aquesta finalitza l'execució de la funció inmediatament. Opcionalment, es pot indicar un número després de la paraula "return" (així: *return 2*, per exemple) que serveix com codi d'estat de sortida (per defecte és 0, valor que per conveni significa "tot correcte")

Amb l'expressió *local var1=valor* es poden definir variables locals dins del cos de la funció(és a dir, que no existeixin fora d'ell, i en cas de què hi hagi dues variables amb el mateix nom, “guanyi” la local).

Si la funció mostra alguna cosa per la sortida estàndar, podem fer servir aquesta sortida en altres parts del nostres script com si fos una comanda qualsevol, així $(nomFuncio params).

Fixa't en els següents exemples bàsics i les seves diferències:

Exemple: Funció que suma dos números. En aquest script, els seus valors respectius són fixes

```
#!/bin/bash
function suma(){
        echo $(($1 + $2))
}
suma 3 5
```

Exemple: Funció que suma dos números. En aquest script, els seus valors respectius són passats com a segon i tercer paràmetre (el primer no s'usa per res)

```
#!/bin/bash
function suma(){
        echo $(($1 + $2))
}
suma $2 $3
```

Exemple: Funció que suma dos números. En aquest script, els seus valors respectius són fixes però no es mostra el resultat retornat directament per l'echo final sinò que es guarda dins de l'expressió $(...) com si fos una comanda estàndar.

```
#!/bin/bash
function suma(){
        echo $(($1 + $2))
}
echo "El resultat de la suma és $(suma 3 5)"
```

El següent exemple mostra el comportament i utilitat de les variables locals a una funció:

Exemple:

```
#!/bin/bash
function funcio(){
        echo "Valor de la variable global de l'script: $a" #Dins de la funció podem usar (i canviar) variables globals
        local a=3                       #Definint una variable local homònima a una global, "anul.lo" aquesta
        echo "Valor de la variable local de la funció: $a"
        a=$(($a+1))
        echo "Valor nou de la variable local de la funció: $a"
}
a=5
funcio
a=$(($a+1))  #La variable local no existeix fora de la funció on està definida
echo "Valor nou de la variable global de l'script: $a"
```

Acabarem amb un exemple més complet:

Exemple: Shell script que amplia la funcionalitat del joc d'endivinar el número secret amb l'afegit d'una funció que compta els intents, fent que si s'arriba als 10 intents s'aturi el joc

```
#!/bin/bash
#----Es declara la funció 'intents' .
#La variable contador és "global" (perquè es defineix fora de la funció); això vol dir que si el seu valor es modifica
#dins de la funció, afectarà també "a fora" (és a dir, a tot el program)
function intents(){
        echo "Intent nº: $contador"
        if [[ $contador -eq 10 ]]
        then
                echo "GAME OVER" ; break
        fi
        contador=$(($contador + 1))
}
#----Comença el programa
numerosecret=$RANDOM
contador=1
while [[ true ]]
do
        echo -n "Introdueix un número: "
        read numerousuari
        intents
        if [[ $numerosecret -lt $numerousuari ]]
        then
                echo -e "La solució és menor que el número que has escrit\n"
        elif [[ $numerosecret -gt $numerousuari ]]
        then
                echo -e "La solució és major que el número que has escrit\n"
        else
                echo "MOLT BÉ!!" ; break
        fi
done
echo "Fi"
```

A l'igual que passava amb les variables, el shell Bash té una sèrie de funcions predefinides (normalment dins de /etc/profile.d/). Algunes d'elles fins i tot s'executen automàticament en detectar-se algun tipus d'event (gràcies a l'ús del sistema DBUS). A l'igual que passava amb les variables, també podem redefinir aquestes funcions (als nostres scripts o al seu lloc original) i així adaptar-les a les nostres necessitats. Per exemple, la funció *command_not_found_handle()* s'executa cada cop que s'escriu una comanda que no es troba al PATH...si volem canviar el seu comportament, ja sabem com.

Arrays

Un array és una variable que conté un conjunt de valors. Es declara indicant aquests valors entre parèntesis i separats per espais, així: *nomarray=(46 "hola" 68)*, per exemple..

També es pot declarar un array sense haver de donar-li valors en aquell moment; així, els parèntesis estaran buits. En aquest cas, caldrà assignar llavors cada valor per separat a cada element de l'array que necessitem; això es fa seguint aquesta sintaxis: *nomarray[nº]=valor* (on el n.º a indicar entre els claudàtors és la posició -l'índex- que ocuparà aquest valor dins de l'array -la primera posició és la 0- i el valor haurà d'estar entre cometes si és una cadena). Si, un cop un determinat element amb un índex concret ja té assignat un valor, més endavant tornés a patir una altra assignació, el valor anterior se sobrescriuria.

Per utilitzar als nostres scripts el valor ja assignat a un determinat element de l'array cal seguir aquesta sintaxis: *${nomarray[nº]}*. Les claus s'han d'escriure quan volem obtenir el valor d'un element de l'array perquè si no els claudàtors es podrien interpretar malament. És a dir...:

Exemple: Shell script que declara inicialment un array amb tres valors, després n'afegeix un altre al final i finalment només mostra aquest per pantalla

```
#!/bin/bash
unarray=(46 "hola" 68)
unarray[3]="nou"
echo ${unarray[3]}
```

Al següent codi es pot veure un exemple del seu ús, i a més també es pot veure com generar números aleatoris entre un rang concret, com ara 0 i 15):

Exemple: Deixo al lector que dedueixi per sí mateix per a què serviria aquest script

```
#!/bin/bash
DIGIT=( 0 1 2 3 4 5 6 7 8 9 A B C D E F )
i=$(($RANDOM%16))
n1=${DIGIT[$i]}
i=$(($RANDOM%16))
n2=${DIGIT[$i]}
i=$(($RANDOM%16))
n3=${DIGIT[$i]}
i=$(($RANDOM%16))
n4=${DIGIT[$i]}
MAC="00:1f:c6:32:$n1$n2:$n3$n4"
echo $MAC
ifconfig eth1 down hw ether $MAC
```

Opcionalment, es pot utilitzar la comanda *declare* per declarar l'array, així: *declare -a nomarray=(46 "hola" 68)* o bé, si es vol declarar sense donar-li valors: *declare -a nomarray* . L'assignació posterior de valors (i el seu ús) es realitzaria de la mateixa manera que ja hem vist als paràgrafs anteriors.
NOTA: La comanda *declare*, de fet, es pot utilitzar per declarar qualsevol tipus de variable. La seva gràcia, no obstant, està en les seves opcions:: *declare -r mivar* fa que la variable sigui de només lectura (constant); *declare -i mivar* declara la variable com entera (per defecte és cadena), entre altres.

L'expressió *${nomarray[@]}* retorna tota la llista de valors en forma de múltiples cadenes (una per valor). L'expressió *${nomarray[*]}* retorna tota la llista de valors en forma d'una única cadena.

Per saber el número d'elements que té un array es pot consultar el valor de *${#nomarray[@]}*

El primer element té l'índex 0, però es pot especificar un índex concret a qualsevol element de l'array, ja sigui en el moment de crear-lo, així: *nomarray = ([1]=unvalor, unaltre, [5]=unaltremés)* com posteriorment assignant valors a elements no consecutius.

Per destruir un array sencer: *unset nomarray*. Per destruir un element de l'array: *unset nomarray[1]*

Si volem assignar valors a un array de forma interactiva, o podem fer igualment amb la comanda read, però hem de fer servir el seu paràmetre -a, així: *read -a nomarray* (els valors introduÏts per l'usuari han de ser escrits separats per espais, fins que es pulsi enter).

Bash també admet el concepte de parells clau<->valor (o "hashes"), el qual no deixa de ser un array on, en comptes d'utilitzar-se un índex numèric per identificar cada element, s'utilitza una paraula -o "clau"- definida per nosaltres. Per crear aquest tipus d'arrays (que a vegades s'anomenen "associatius") cal fer servir la comanda *declare* així: *declare -A nomarray=([clau1]=46 [clau2]="hola" ...)* o bé, si es vol crear inicialment buit, *declare -A nomarray*. Per donar valor a un determinat element, caldrà fer: *nomarray[clau1]=46.* Per utilitzar el valor d'un determinat element, caldrà fer *${nomarray[clau2]},* per exemple. Per veure tots els valors de cop, es fa igual que sempre: *${nomarray[@]}* , i per veure totes les claus de cop, es fa *${!nomarray[@]}* (útil per posar en un for, per exemple)

Exemple: A continuació es presenta un exemple de joc interactiu "pedra-paper-tissora" contra l'ordinador

```
#!/bin/bash
pet=("paper" "pedra" "tissores")
#Torn de jugador
echo -n "Escull ( ${pet[@]} ): "
read resposta
case $resposta in
        "paper"|"p") resposta=0;;
        "pedra"|"e")  resposta=1;;
        "tissores"|"t") resposta=2;;
        *) echo "No facis trampa!" ; exit;;
esac
echo -e "Has escollit:\e[32m ${pet[$resposta]} \e[0m"
#Torn de la computadora
contratac=$(( $RANDOM % 3 ))
echo -e "La computadora ha escollit:\e[31m ${pet[$contratac]} \e[0m"
#Còmput del resultat
if (( $resposta == $contratac ))
then
        echo -e "\e[33m Empat! \e[0m"
else
        case "$resposta$contratac" in
                01) guanyador="\e[32m Tu! \e[0m";;
                02) guanyador="\e[31m La computadora, oooh \e[0m";;
                10) guanyador="\e[31m La computadora, oooh \e[0m";;
                12) guanyador="\e[32m Tu! \e[0m";;
                20) guanyador="\e[32m Tu! \e[0m";;
                21) guanyador="\e[31m La computadora, oooh \e[0m";;
        esac
        echo -e "El guanyador és: $guanyador "
fi
```